Aprendizaje supervisado para la detección de amenazas web mediante clasificación basada en árboles de decisión

Aplicación de técnicas de machine learning a la ciberseguridad

José María Dueñas Quesada
Máster Universitario en Seguridad de las Tecnologías de la Información y de las Comunicaciones
Área de Análisis de datos
Trabajo Final de Máster

Dirigido por **Helena Rifà Pous**
Tutorizado por **Enric Hernández Jiménez**
2 de Junio de 2020

Universitat Oberta de Catalunya

FICHA DEL TRABAJO FINAL DE MÁSTER

Titulo del trabajo	Aplicación de técnicas de machine learning a la ciberseguridad: Aprendizaje supervisado para la detección de amenazas web mediante clasificación basada en árboles de decisión
Nombre del autor	José María Dueñas Quesada
Nombre del director	Enric Hernández Jiménez
Fecha de entrega	09/06/2020
Titulación	Máster Universitario en Seguridad de las Tecnologías de la información y de las Comunicaciones.
Universidad	Universitat Oberta de Catalunya, Universitat Autònoma de Barcelona y Universitat Rovira i Virgili
Área del Trabajo Final	Análisis de datos
Idioma del trabajo	Español
Palabras claves	Aprendizaje automático, Machine Learning, Seguridad informática, Ciberseguridad, Análisis de datos, Aprendizaje supervisado, Árboles de decisión, Python, scikit-learn

Resumen

El objetivo del TFM es desarrollar un modelo predictivo de machine learning basado en árboles de decisión cuya tarea sea clasificar un conjunto de peticiones HTTP en peticiones normales y anómalas. El TFM incluye un estudio del estado del arte sobre las aplicaciones en ciberseguridad del machine learning, la implementación con Python y Scikit-learn de un modelo clasificador basado en aprendizaje supervisado con árboles de decisión y el análisis de los resultados de aplicar dicho modelo sobre el dataset CSIC-2010. El modelo propuesto en este TFM consigue hasta un 100% de exactitud (*accuracy*) en la clasificación de las peticiones HTTP.

Abstract

The purpose of this Master thesis is to develop a decision tree based predictive machine learning model which main task is to classify a set of HTTP requests in normal and anomalous ones. The thesis includes a state-of-the-art survey on the main applications of the machine learning in the cybersecurity field, the implementation of the proposed decision tree classifier model with the Python language and the Scikit-learn library and the analysis of the results obtained on the application of the implemented model against the CSIC-2010 dataset. The proposed model in this thesis achieved a 100% of accuracy in classifying the HTTP requests.

Índice de contenidos

Índice de figuras

Índice de tablas

1 Introducción

Este apartado sirve de introducción al trabajo final de máster e incluye el plan de trabajo, describe el problema que se pretende resolver, la propuesta para resolverlo y la descomposición en tareas y metas temporales.

1.1 Situación actual

Con el desarrollo de Internet, los ciberataques han evolucionado constantemente tanto en frecuencia como en complejidad, lo que lleva a una situación en la que conseguir un alto grado de ciberseguridad es cada vez más complicado.

Como muestra del volumen y peligrosidad, en su último informe del 2019 sobre ciberamenazas y tendencias [1], el CCN-CERT como CERT Gubernamental Nacional, informa que "*durante el año 2018 gestionó un total de 38.029 incidentes de ciberseguridad, de los cuales, el 2,7% tenían una peligrosidad "muy alta" o "crítica". Un año, en el que el Centro Criptológico Nacional volvió a constatar cómo los Estados y los grupos patrocinados por ellos siguen representando la ciberamenaza más significativa del panorama internacional*". El mismo informe destaca, además, que el *machine learning*, como mecanismo de protección proactiva, "*será vital para vencer a los ciberdelincuentes, especialmente a aquellos que usan malware con técnicas de ocultación*".

Son muchos los métodos de *machine learning* que pueden aplicarse en el campo de la ciberseguridad en aras de mitigar las diferentes amenazas y también son muchas las formas de aplicar cada uno de esos métodos, lo cual hace casi imposible dar una fórmula magistral que nos indique qué combinación usar dependiendo del tipo de ataque a detectar. Por otra parte, es de gran importancia en el *machine learning* la elección del *data set* para el entrenamiento del modelo.

1.2 Motivación

La propuesta elegida para la realización del Trabajo Final de Máster ha sido la aplicación de técnicas de *machine learning* a la Seguridad. En concreto se centra en la clasificación de peticiones HTTP con aprendizaje supervisado y árboles de decisión para la detección de amenazas a servidores web.

La oportunidad de conocer y estudiar la aplicación en el ámbito de la seguridad TIC de una disciplina como el *machine learning* es sin duda una gran motivación que me permitirá aplicar parte de lo aprendido durante mis estudios del Máster, comprender las bases de esta disciplina y emplearla en mi vida laboral.

1.3 Objetivos

Teniendo en cuenta la situación actual descrita en el apartado 1.1 Situación actual, el problema que se pretende resolver es el de la detección de ataques a servidores web mediante la aplicación de técnicas de *machine learning*.

Por lo tanto, el objetivo principal del TFM ha sido desarrollar un modelo predictivo de *machine learning* con aprendizaje supervisado para la detección de amenazas web mediante clasificación basada en árboles de decisión que permita clasificar un conjunto de peticiones HTTP en peticiones normales y anómalas.

Otros objetivos a alcanzar con la realización de este TFM, se indican a continuación:

- Estudiar y analizar la aplicación de las técnicas y algoritmos de *machine learning* para resolver problemas en el ámbito de la seguridad TIC.
- Estudiar y analizar, en concreto, la aplicación de algoritmos supervisados de *machine learning* basados en árboles de decisión en el ámbito de la seguridad TIC en general, y de la detección de intrusiones web en particular.
- Conocer las técnicas empleadas para la selección de características.
- Conocer las técnicas empleadas en la optimización de la parametrización del entrenamiento para conseguir la mayor exactitud posible.
- Conocer las técnicas empleadas en la partición de un data set para entrenamiento, *testing* y aplicación del algoritmo de *machine learning*.
- Desarrollo de un software basado en aplicación de algoritmos supervisados de *machine learning* basados en árboles de decisión que tome por entrada una petición web HTTP y detecte si se trata de una amenaza o no.
- Presentación y análisis de los resultados obtenidos en la aplicación del software desarrollado a un data set concreto.

1.4 Enfoque metodológico

Para la realización del TFM se ha seguido un proceso consistente en cuatro fases principales:

1. Investigación sobre el estado del arte del *machine learning* aplicado a la seguridad.
2. Desarrollo del software detector de amenazas web. Se ha empleado el lenguaje de programación Python y la librería scikit-learn, principalmente.
3. Ejecución, parametrización y optimización del software desarrollado sobe el conjunto de datos seleccionado (dataset) y análisis de los resultados.
4. Documentación, presentación y análisis de los resultados.

1.5 Descomposición de tareas

El trabajo realizado se descompone en varias tareas que se describen a continuación y que han sido entregadas siguiendo la planificación del TFM.

1.5.1 Plan de trabajo

Esta primera tarea consiste en realizar un plan de trabajo. El plan de trabajo es un elemento clave del funcionamiento del TFM ya que describe el problema que se pretende resolver, el trabajo concreto que se llevará a cabo y la descomposición de este trabajo en tareas y metas temporales. De esta manera, este plan de trabajo contiene:

- La explicación detallada del problema a resolver.
- La enumeración de los objetivos que se quieren alcanzar con la realización del TFM.
- La descripción de la metodología que se seguirá durante el desarrollo del TFM.

- El listado de las tareas a realizar para alcanzar los objetivos descritos.
- La planificación temporal detallada de estas tareas y sus dependencias.
- Una pequeña revisión del estado del arte.

1.5.2 Investigación sobre el estado del arte del machine learning aplicado a la seguridad

Para esta segunda tarea se ha realizado un estudio del estado del arte del *machine learning* y su aplicación a la seguridad.

- Se han analizado los distintos algoritmos para árboles de decisión disponibles en la librería *scikit-learn* de Python y se ha seleccionado el óptimo para cumplir con los objetivos marcados en 1.3 OBJETIVOS.
- Se han seleccionado los *datasets*.

1.5.3 Desarrollo del software detector de amenazas web.

Para esta tercera tarea se han realizado las siguientes subtareas:

- Diseño y desarrollo del software.
- Aplicación sobre el *dataset*.
- Ejecución y análisis de resultados.

1.5.4 Memoria final

Entrega de la memoria del Trabajo Final de Máster y del software resultante. La memoria sintetiza el trabajo realizado durante el TFM y muestra cómo se han alcanzado los objetivos propuestos.

Para realizar la memoria se han seguido las indicaciones y normativa disponibles en el aula, así como las indicaciones del director del trabajo. Se estimó que la memoria tuviera entre 50 y 70 páginas.

1.5.5 Presentación Virtual

La presentación virtual consiste en un vídeo de 15 minutos de duración máxima en el que se presenta mediante unas diapositivas una síntesis del trabajo realizado con el soporte una presentación de diapositivas. Además, la presentación virtual incluye una demostración del funcionamiento del software desarrollado.

1.6 Planificación temporal

A continuación, se presenta la planificación temporal del TFM en un diagrama de Gantt.

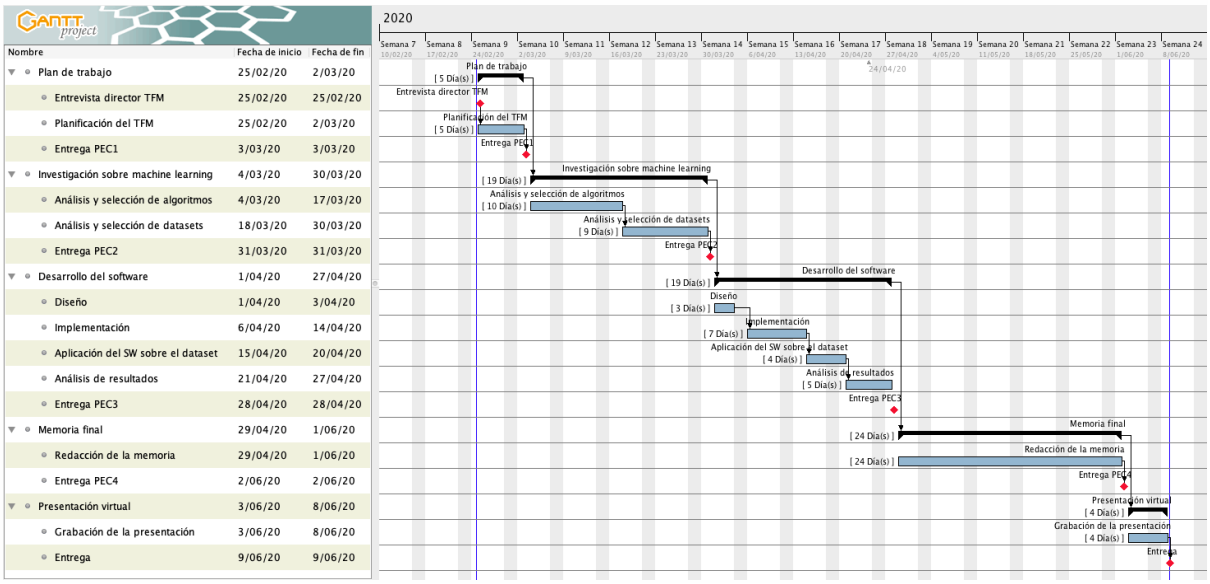

Figura 1. Diagrama de Gantt con la planificación temporal del TFM

2 Estado del arte

2.1 El machine learning

El *machine learning,* o aprendizaje automático en español, es una disciplina de la Inteligencia Artificial que, mediante el uso de algoritmos, dota a los ordenadores de la capacidad de identificar patrones en grandes volúmenes de datos con el objetivo de realizar predicciones. Esto les permite a los ordenadores realizar una serie de tareas de forma automática sin necesidad de ser programados.

Otra definición, desde un punto de vista de la ingeniería informática, es la dada por Tom Mitchell, científico de la computación y profesor de la Universidad Carnegie Mellon:

> *A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.*

En resumen, se puede decir que el objetivo del *machine learning* es que el sistema desarrollado aprenda de un conjunto de datos (*dataset*) y a través de un determinado entrenamiento con esos datos pueda tomar decisiones sobre nuevos datos que se le van pasando como entrada. Además, el desempeño del sistema en cuanto a su aprendizaje irá mejorando con la experiencia que vaya adquiriendo.

En realidad, el *dataset* o conjunto de datos puede estár formado a su vez por:

- ***training-set***: es el conjunto de datos que sirve para entrenar al sistema de machine learning desarrollado.
- ***test-set***: es el conjunto de datos independientes de los anteriores, pero que siguen una misma distribución de probabilidad y que sirven para comprobar que el sistema funciona.
- ***validation-set***: es similar al test-set solo que se reserva para hacer una validación final de nuestro sistema permitiendo ajustar los parámetros del modelo empleado y conseguir así mejores resultados.

Hay diferentes **tipos de sistemas de *machine learning***, pudiendo ser clasificados dependiendo de si:

- el entrenamiento del sistema se ha realizado con la supervisión de un humano o no: **aprendizaje supervisado, no supervisado, semisupervisado** y **aprendizaje reforzado**.
- el sistema analiza el conjunto completo de datos (***batch learning***) o los va a adquiriendo de forma secuencial conforme van estando disponibles (***online learning***).
- el funcionamiento del sistema se basa en comparar ciertos parámetros de nuevos datos con los parámetros de datos ya existentes (***instance-based learning***) o por el contrario detectar patrones de los datos ya existentes para así crear modelos de predicción (***model-based learning***)

2.2 Aplicaciones del machine learning a la ciberseguridad

La aplicación del *machine learning* al campo de la ciberseguridad está cada vez más extendido. El uso de herramientas cuyo objetivo es la detección de diversos tipos de amenazas se basan cada vez más en sistemas de aprendizaje automático.

Las principales aplicaciones de *machine learning* [2] al ámbito de la seguridad de los sistemas de la información abarcan: la detección y prevención de intrusiones en sistemas, la detección de *phising*, la preservación de la privacidad de usuario, la detección de spam, el análisis de riesgos, la detección de *malware* y el *testing* de las propiedades de seguridad de un sistema.

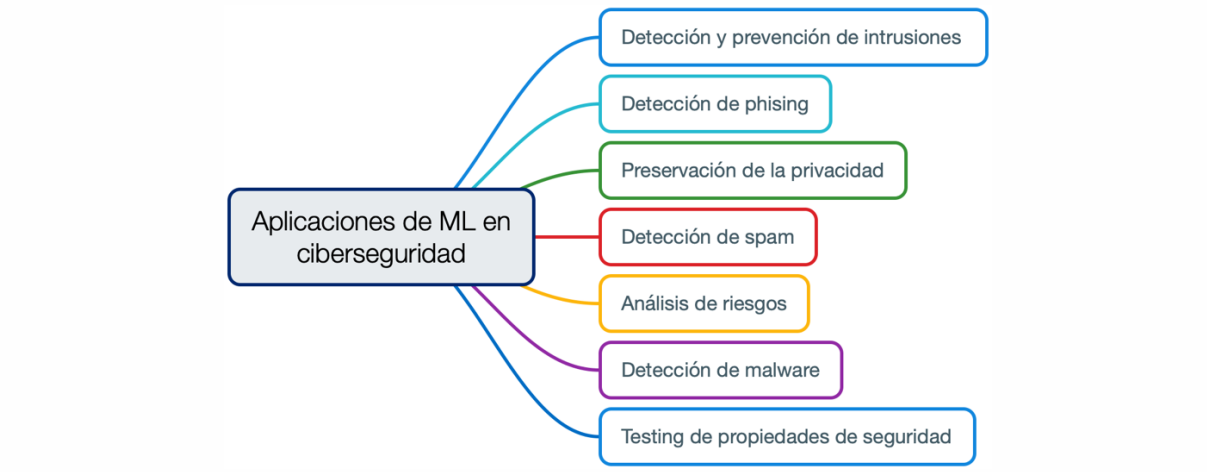

Figura 2. Aplicaciones de Machine Learning a la ciberseguridad

A continuación, se realiza un recorrido por las principales investigaciones llevadas a cabo en los ámbitos de aplicación del aprendizaje automático que acabamos de identificar, focalizando en la detección de intrusiones que será el objeto de estudio del TFM.

2.2.1 Detección y prevención de intrusiones

Uno de los principales sistemas informáticos cuya finalidad es la detección de acceso no autorizados a una red, servidor o sistema en general son los IDS (*Intrusion Detection System*). El IDS, al tratarse de un sistema pasivo, recopila muchos datos a través de unos sensores virtuales, normalmente *sniffers*, sobre los que realiza un análisis para detectar intrusiones.

También, más relacionado con el objeto de estudio de este TFM, existe otro sistema informático, el WAF (Web Application Firewall) que protege contra ataques/intrusiones en servidores y aplicaciones web.

En ambos casos, para la realización de este análisis y poder discriminar entre las que son una conexión legítima y otras que son un intento de intrusión, se pueden tomar diferentes aproximaciones aplicando machine learning, siendo las principales la basada en firmas y la basada en anomalías.

2.2.1.1 Aproximación basada en firmas

Los IDS basados en firmas recopilan todos los paquetes que circulan por la red o hacía un servidor y los comparan con una base de datos de firmas, que son un conjunto de patrones de ataque preconfigurados y predeterminados. Si alguna de las firmas de la base de datos concuerda con el paquete de datos analizado, entonces lo marcará como intento de intrusión.

Estos sistemas resultan ser clasificadores probabilísticos, es decir, clasifican los paquetes que leen de la red en paquete malicioso o paquete no malicioso. Por lo tanto, atendiendo al tipo o tipos de algoritmos

de clasificación que se usen para esta técnica podemos distinguir entre: clasificador único, clasificador híbrido y clasificador combinado.

a) Clasificador único.

En este tipo de técnica de intrusión se usa solo un algoritmo de machine learning para detectar la intrusión. Los modelos algorítmicos mas empleados por su eficacia son: algoritmos de árboles de decisión, redes neuronales artificiales (ANN) y máquinas de vectores de soporte (SVN).

b) Clasificador híbrido.

En este tipo de técnica se emplea una mezcla de algoritmos distintos para cubrir todas las fases del aprendizaje automático, desde la fase de normalización de datos hasta la fase de decisión (clasificación). Por ejemplo [3], podría emplearse un algoritmo clasificador Naïve Bayes que marcara con un *flag* los paquetes de red sospechosos y se los pasara como entrada a otro algoritmo que use el modelo oculto de Markov (HMM) y que se encargaría de incluir las direcciones IP de procedencia de esos paquetes en una lista negra.

c) Clasificador combinado.

Se trata de crear un clasificador combinado a partir de varios clasificadores débiles. Por clasificadores débiles entendemos aquellos que su exactitud en clasificar es bastante mediocre pero aun así es bastante más preciso que si se intentara clasificar por especulación. Esta técnica hace uso, principalmente, de algoritmos de boosting combinando los resultados de varios clasificadores débiles para así obtener un clasificador robusto. Un ejemplo sería combinar algoritmos basados en redes neuronales profundas (DNN) y agrupamiento espectral.

2.2.1.2 Aproximación basada en anomalías

Esta aproximación se basa en monitorizar un sistema y detectar aquellas desviaciones que tienen lugar con respecto a un comportamiento normal de este.

Se resume, a continuación, un análisis [2] de las técnicas de detección de intrusiones basadas en anomalías.

a) Clasificador único.

Los algoritmos y técnicas empleadas por algunos investigadores [2] son *deep learning*, árboles de decisión, Naïve Bayes y máquinas de vectores de soporte. La ventaja de usar *deep learning* es que se consigue un porcentaje muy alto de exactitud en la detección de las intrusiones, aunque para ello es necesario emplear un tiempo considerablemente elevado para el proceso de aprendizaje. En el caso de los árboles de decisión y Naïve Bayes [4], el porcentaje de falsos positivos es mayor comparado con en el caso anterior aunque por otro lado la exactitud aumenta notablemente cuando se toman como parte de su *dataset* de entrenamiento casos reales de intrusiones en otros sistemas. Por último, el empleo de máquinas de vectores de soporte parece dar muy buen resultado [5], incluso un 100% de exactitud siempre que el dataset esté compuesto por datos 'no homogéneos'[1].

b) Clasificador híbrido.

[1] un tipo de conjunto que se caracteriza por agrupar en él elementos, que pese a ser considerados como pertenecientes a la misma naturaleza, no pueden ser tenidos como elementos de igual género.

Entre estos clasificadores podemos citar algunas investigaciones [6] realizadas sobre IDS que emplean *k-means* modificado y el algoritmo C4.5, que permiten procesar de forma eficiente grandes volúmenes de tráfico real de red aunque la exactitud conseguida no suele ser alta.

c) Clasificador combinado.

Para este tipo de clasificadores, algunos estudios realizados [7] han obtenido resultados más precisos (99,3%) cuando se emplean *Bagged trees* y *GentleBoost*.

2.2.2 Detección de phising.

Según Panda Security [8], la empresa española especializada en la creación de soluciones de seguridad informática y recientemente adquirida por WatchGuard, el phishing se refiere al envío de correos electrónicos que tienen la apariencia de proceder de fuentes de confianza (como bancos, compañías de energía etc.) pero que en realidad pretenden manipular al receptor para robar información confidencial.

Ante este tipo de ataque también se aplican técnicas de *machine learning* para la detección de correos fraudulentos y cuentas de redes sociales falsas con respecto a los que son realmente legítimos.

En el ámbito de la detección de correos fraudulentos, Hamid et al. [9] proponen emplear técnicas y algoritmos como AdaBoost y optimización mínima secuencial (SMO), aunque obtienen una tasa de error del 18%, mientras que en otros estudios como los de Basnet et al. [10] que propone el uso de Naïve Bayes, consiguen disminuir la tasa de error hasta el 1.6%.

Para el caso de la detección de sitios web fraudulentos, Li et al. [11] proponen una aproximación basada en el empleo de un subtipo de máquinas de vectores de soporte conocidas como TSVM (*transductive support vector machine*).

2.2.3 Preservación de la privacidad.

Desde el punto de vista de la privacidad y para el cumplimiento de normas como GDPR[2], uno de los objetivos principales a la hora de aplicar técnicas de *machine learning* sobre datos de usuario es extraer toda la información posible que sea de utilidad para el análisis sin menoscabar la privacidad del usuario, para ello es necesario encontrar una regla de compromiso que satisfaga los dos extremos.

Técnicas como la de máquinas de vectores de soporte [12] o de agrupamiento de k-medias [13] se aplican en ese ámbito.

2.2.4 Detección de spam.

El envío mensajes no solicitados de correo electrónicos, SMS, redes sociales o comentarios en blogs, es otro de los ámbitos de aplicación del aprendizaje automático que persigue detectar ese tipo de mensajes.

Entre los estudios más recientes en el ámbito de la detección de correo no deseado, destacan el realizado por Sutta et al. [14] donde se analizan y comparan el rendimiento de varios sistemas clasificadores de correo electrónico: diferentes tipos de máquinas de vector de soporte, redes neuronales artificiales, k vecinos más cercanos, Naïve Bayes, regresión logística, árboles de decisión y Random Forest.

En cuanto a detección de spam en redes sociales, se puede citar el realizado por Chen et al. [15] donde se estudia el empleo de Random Forest, C4.5, Naïve Bayes, KNN Bayes Network y máquinas de vectores de soporte para la detección en tiempo real de mensajes de spam en Twitter.

[2] Regulation (EU) 2016/679 (General Data Protection Regulation), en español: Reglamento General de Protección de Datos.

2.2.5 Análisis de riesgos.

El análisis de riesgos según *la* ISO 9001:2015 es el procedimiento que debemos llevar a cabo para identificar los elementos o los factores que tienen la posibilidad de generar riesgos (u oportunidades) que afecten en forma negativa (*o* positiva) la operación de la organización.

También en estos procesos de análisis es posible aplicar técnicas de *machine learning* sobre alguna de las actividades de este, como por ejemplo el cálculo del nivel del riesgo (crítico, alto, medio, bajo) dentro del análisis cualitativo de estos. En Eminagaoglu et al. [16] se aplica un técnica de clasificación para los riesgos que los categoriza en dos niveles: riesgo y sin riesgo. De un total de 68 algoritmos de clasificación diferentes, el que mejor resultados obtuvo fue REPtree, un tipo de árbol de decisión.

2.2.6 Detección de malware.

Según Kaspersky, el *malware*, abreviatura de "*malicious software*" (software malicioso), se refiere a un tipo de programa informático diseñado para infectar el ordenador de un usuario legítimo e infligir daño en este de varias formas. El *malware* puede infectar ordenadores y dispositivos de varias maneras, y adopta a su vez varias formas, entre las cuales se incluyen virus, gusanos, troyanos, *spyware* y mucho más. Es vital que todos los usuarios sepan cómo reconocer el *malware* y protegerse de él en todas sus formas.

Son muchas las aplicaciones de *machine learning* aplicadas para la detección de *malware* que toman una de las tres aproximaciones posibles: detección basada en firmas, basada en anomalías y basada en heurística. La mayoría de ellas son aplicaciones software, pero merece la pena destacar la propuesta realizada por Das et al. [17], que presenta una solución software apoyada por hardware (microprocesador y FPGA) para la detección online de malware. Haciendo uso de técnicas y algoritmos como J48, Naïve Bayes, máquinas de vectores de soporte, regresión lineal, optimización mínima secuencial (SMO) y perceptrón multicapa, consigue detectar durante el primer 30% de la ejecución de la muestra observada hasta un 46% de los *malware*, y el 97% de los *malware* al ejecutarlo completamente. Todo con un índice de falsos positivos del 3%.

2.2.7 Testing de propiedades de seguridad.

El testing de las propiedades de seguridad se refiere a las pruebas que se realizan para comprobar la correcta implementación de un determinado protocolo, normalmente criptográfico, dentro de un sistema distribuido como los que se utilizan en infraestructuras militares, sistemas críticos, comunicaciones móviles ad-hoc, etc.

Varias aproximaciones como pruebas en tiempo real, pruebas de caja negra, etc. se suelen llevar a cabo para la comprobación de la correcta implementación de estos protocolos. Aquí también puede aplicarse el *machine learning*. Como muestra, se puede citar el estudio llevado a cabo por Shu et al. [18] que propone utilizar el concepto de pruebas de caja negra junto a algoritmos de aprendizaje supervisado para probar de forma automática y sistemática la implementación de un protocolo de mensajería de un sistema distribuido con el objetivo de garantizar que se cumple la confidencialidad en las comunicaciones.

2.3 Python y la librería *scikit-learn*

Python [19] es un lenguaje de programación poderoso, versátil y muy fácil de aprender. Es un lenguaje de programación interpretado, de código legible y multiparadigma, que permite la programación orientada a objetos, la programación imperativa y en menor medida la programación funcional. La elegante sintaxis de Python y su tipado dinámico junto con su naturaleza interpretada hacen de éste un lenguaje ideal para scripting y desarrollo rápido de aplicaciones en diversas áreas y sobre la mayoría de las plataformas.

El intérprete de Python y la extensa biblioteca estándar están a libre disposición en forma binaria y de código fuente para las principales plataformas desde el sitio web de Python, https://www.python.org/, y

puede distribuirse libremente. El mismo sitio contiene también distribuciones y enlaces de muchos módulos libres de Python de terceros, programas y herramientas, y documentación adicional.

Scikit-learn [20] es un módulo para Python que integra una gran variedad de algoritmos de *machine learning* para resolver problemas de hasta una envergadura media tanto con aprendizaje supervisado como no supervisado.

Este módulo permite hacer uso, de una forma sencilla, de técnicas de *machine learning* usando un lenguaje de alto nivel y de propósito general como es Python. El módulo se caracteriza por su facilidad de uso, rendimiento, documentación y consistencia en su API. Además, apenas presenta dependencias de otros módulos. Se distribuye bajo la licencia BSD, fomentando su uso tanto en el ámbito académico como empresarial. El código fuente, binarios y la documentación está disponible en http://scikit-learn.sourceforge.net.

2.4 Algoritmos de *machine learning*

2.4.1 Tipos de algoritmos

Son muchos los algoritmos de machine learning disponibles, así como las formas de agruparlos. Por ejemplo, podríamos agruparlos según cómo es su método de aprendizaje: supervisado o no supervisado (ver FIGURA 3. ALGORITMOS DE MACHINE LEARNING SEGÚN SU MÉTODO DE APRENDIZAJE) o también atendiendo a su similitud en cuanto a cómo es su funcionamiento (ver FIGURA 4. ALGORITMOS DE MACHINE LEARNING POR SIMILITUD).

A continuación, se realizará un repaso no exhaustivo por los principales algoritmos de machine learning [21] [22], agrupándolos por su similitud. El objetivo no es explicar la base matemática de cada uno de ellos sino mostrar qué problema intentan resolver para así tener una noción básica de en qué situaciones pueden aplicarse.

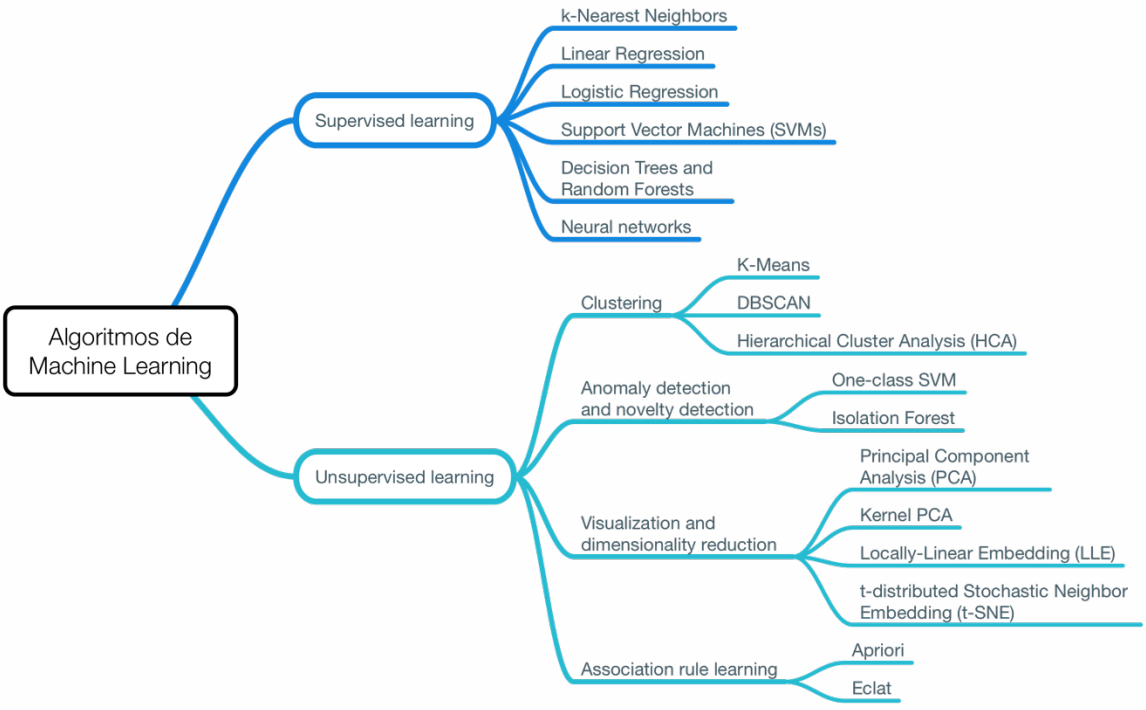

Figura 3. Algoritmos de Machine Learning según su método de aprendizaje

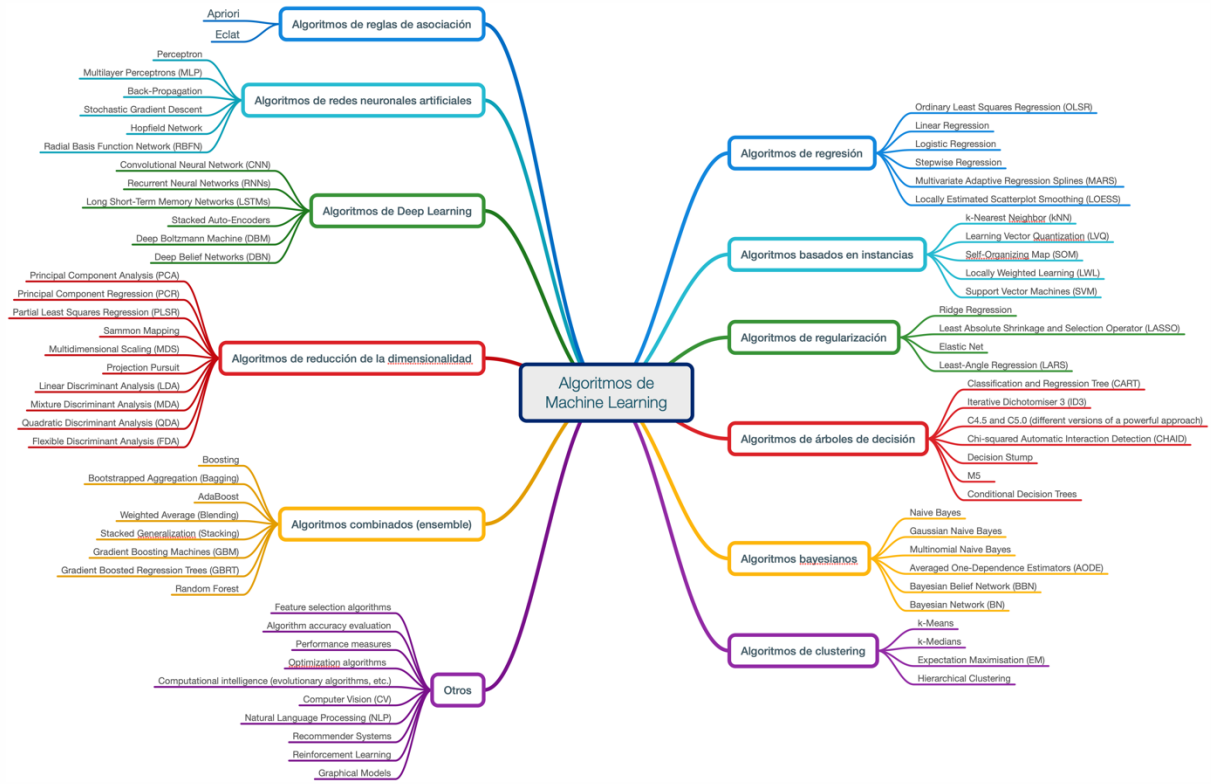

Figura 4. Algoritmos de Machine Learning por similitud

2.4.1.1 Algoritmos de regresión

Este tipo de algoritmos se aplican en modelos de aprendizaje automático supervisado que buscan estimar y determinar la existencia de relaciones entre variables que forman parte del objeto de estudio. Para ello, se establece como dependiente una variable y se estudia su comportamiento en relación con otra serie de variables independientes y/o cambiantes.

Con los algoritmos de regresión podemos realizar un proceso de aprendizaje automático que facilite la predicción de resultados y pronósticos.

Los principales algoritmos de este tipo son:

- *Ordinary Least Squares Regression (OLSR)*
- *Linear Regression*
- *Logistic Regression*
- *Stepwise Regression*
- *Multivariate Adaptive Regression Splines (MARS)*
- *Locally Estimated Scatterplot Smoothing (LOESS)*

2.4.1.2 Algoritmos basados en instancias

Una instancia es cada uno de los datos de los que se disponen para hacer un análisis.

Estos algoritmos se emplean en modelos de aprendizaje para resolver problemas de decisión con instancias de datos de entrenamiento que son importantes o requeridos por el modelo. Memoriza las instancias de entrenamiento y su clase. Las instancias constituyen el conocimiento del sistema.

Realiza una clasificación en base a la similitud de la instancia a clasificar con las instancias memorizadas.

Los principales algoritmos de este tipo son:

- *k-Nearest Neighbor (kNN)*
- *Learning Vector Quantization (LVQ)*
- *Self-Organizing Map (SOM)*
- *Locally Weighted Learning (LWL)*
- *Support Vector Machines (SVM)*

2.4.1.3 Algoritmos de regularización

En el proceso de entrenamiento de un modelo, se intenta "encajar" (*fit*, en inglés) los datos de entrada entre ellos y con la salida. Si se entrena mucho ese modelo puede provocar un sobreajuste (*overfitting*, en inglés), lo cual es un problema. La regularización es un método usado habitualmente para tratar este problema frecuente en machine learning del *overfitting* de los modelos.

Los principales algoritmos para realizar esta regularización son:

- *Ridge Regression*
- *Least Absolute Shrinkage and Selection Operator (LASSO)*
- *Elastic Net*
- *Least-Angle Regression (LARS)*

2.4.1.4 Algoritmos de árboles de decisión

A través de estos algoritmos es posible construir modelos de decisiones basados en los valores de los datos analizados, en sus atributos para ser mas concretos.

En estos modelos cada nodo del árbol representa una prueba sobre una variable específica y las ramas muestran el resultado que resulta de dicha prueba.

Estos algoritmos suelen emplearse para resolver problemas de clasificación de datos y también para regresión. Suelen ser bastante rápidos y precisos en su ejecución y fácilmente representables gráficamente.

Los principales algoritmos de este tipo son:

- *Classification and Regression Tree (CART)*
- *Iterative Dichotomiser 3 (ID3)*
- *C4.5 and C5.0 (different versions of a powerful approach)*
- *Chi-squared Automatic Interaction Detection (CHAID)*
- *Decision Stump*
- *M5*
- *Conditional Decision Trees*

2.4.1.5 Algoritmos bayesianos

Estos algoritmos aplican el Teorema de Bayes y son empleados tanto para resolver problemas de clasificación como de regresión.

Los principales algoritmos de este tipo son:

- *Naive Bayes*
- *Gaussian Naive Bayes*
- *Multinomial Naive Bayes*
- *Averaged One-Dependence Estimators (AODE)*
- *Bayesian Belief Network (BBN)*

- *Bayesian Network (BN)*

2.4.1.6 Algoritmos de clustering

Son un tipo algoritmos de machine learning para aprendizaje no supervisado. Gracias a ellos podemos agrupar datos no etiquetados en varias categorías (*clusters*), es decir, dado un conjunto de datos desordenados (bajo ninguna categoría) permite agruparlos en varias categorías.

Estos algoritmos realizan búsquedas dentro del conjunto de datos que estamos estudiando, estableciendo una variable que los representa dentro del grupo y posteriormente de forma iterativa va asignando en cada punto de datos esa variable, según las características que determinemos.

Los principales algoritmos de este tipo son:

- *k-Means*
- *k-Medians*
- *Expectation Maximisation (EM)*
- *Hierarchical Clustering*

2.4.1.7 Algoritmos de reglas de asociación

Las reglas de asociación se utilizan para descubrir hechos que ocurren en común dentro de un determinado conjunto de datos. Estos algoritmos extraen esas reglas de asociación entre los datos observados.

Los principales algoritmos de este tipo son:

- *Apriori algorithm*
- *Eclat algorithm*

2.4.1.8 Algoritmos de redes neuronales artificiales

El funcionamiento de estos algoritmos pretende emular cómo se comporta el cerebro humano en cuanto al procesamiento de información. Estos comprenden unidades dispuestas en capas. Cada una de estas capas posee una conexión con las capas anexas.

Todos estos elementos de procesamiento de datos están íntimamente interconectados y trabajan de forma conjunta para darle solución a los problemas específicos que deban analizar. Los algoritmos de redes neuronales artificiales son normalmente empleados para establecer modelos de relaciones no lineales o donde la relación de las variables de ingreso a un sistema con un alto nivel de complejidad y su comprensión suele ser difícil de entender.

Los principales algoritmos de este tipo son:

- *Perceptron*
- *Multilayer Perceptrons (MLP)*
- *Back-Propagation*
- *Stochastic Gradient Descent*
- *Hopfield Network*
- *Radial Basis Function Network (RBFN)*

2.4.1.9 Algoritmos de Deep Learning

Se trata de una evolución de las redes neuronales artificiales que ya se ha comentado. Estos algoritmos se aprovechan de que el proceso de cómputo es relativamente barato (uso de GPU de tarjetas gráficas principalmente) para poder aplicar el modelo sobre redes neuronales mucho más grandes y complejas, con un conjunto de datos enorme.

Los principales algoritmos de este tipo son:

- *Convolutional Neural Network (CNN)*
- *Recurrent Neural Networks (RNNs)*
- *Long Short-Term Memory Networks (LSTMs)*
- *Stacked Auto-Encoders*
- *Deep Boltzmann Machine (DBM)*
- *Deep Belief Networks (DBN)*

2.4.1.10 Algoritmos de reducción de la dimensionalidad

El objetivo de estos algoritmos es explotar la estructura existente de forma inherente en los datos a través de un aprendizaje no supervisado para simplificarlos y reducirlos o comprimirlos.

Estos algoritmos reducen del número de variables que deben considerarse para lograr una solución concreta. Así mismo, ayudan a mejorar la eficiencia de los procesos de aprendizaje automático devolviendo resultados precisos en menor tiempo.

Son útiles para visualizar datos o para simplificar el conjunto de variables que luego serán usados por un algoritmo supervisado.

Los principales algoritmos de este tipo son:

- *Principal Component Analysis (PCA)*
- *Principal Component Regression (PCR)*
- *Partial Least Squares Regression (PLSR)*
- *Sammon Mapping*
- *Multidimensional Scaling (MDS)*
- *Projection Pursuit*
- *Linear Discriminant Analysis (LDA)*
- *Mixture Discriminant Analysis (MDA)*
- *Quadratic Discriminant Analysis (QDA)*
- *Flexible Discriminant Analysis (FDA)*

2.4.1.11 Algoritmos combinados (ensemble)

Estos algoritmos implementan un clasificador combinado a partir de varios clasificadores débiles. Por clasificadores débiles entendemos aquellos cuya exactitud en clasificar es bastante mediocre pero aun así son bastante más precisos que si se intentara clasificar por simple especulación. Esta técnica hace uso, principalmente, de algoritmos de *boosting*, combinando los resultados de varios clasificadores débiles para así obtener un clasificador robusto. Un ejemplo sería combinar algoritmos basados en redes neuronales profundas (DNN) y agrupamiento espectral.

Los principales algoritmos de este tipo son:

- *Boosting*
- *Bootstrapped Aggregation (Bagging)*

- *AdaBoost*
- *Weighted Average (Blending)*
- *Stacked Generalization (Stacking)*
- *Gradient Boosting Machines (GBM)*
- *Gradient Boosted Regression Trees (GBRT)*
- *Random Forest*

2.4.2 Elección del algoritmo adecuado

Hay que tener claro que un mismo problema puede resolverse siguiendo varias aproximaciones distintas. Para problemas muy específicos para los que hay algoritmos claramente identificados para ese propósito no hay dudas, pero para otros problemas no tan específicos el conjunto de posibles aproximaciones a tomar está mucho mas abierto y quizás requieran de un proceso de prueba y error para dar con los resultados óptimos a través del mejor algoritmo o una combinación de estos.

Para la correcta elección de qué algoritmo de machine learning es el más adecuado para resolver el problema, se debe empezar por:

- Categorizar el problema, teniendo claro que objetivo queremos conseguir
- Revisar los datos de los que disponemos (analizarlos, procesarlos y transformarlos).

Una vez revisados estos dos puntos y teniendo un conocimiento de los distintos algoritmos de machine learning existentes, se estará preparado para empezar a analizar qué aproximación y algoritmo son los que mejor se adaptan al problema a resolver.

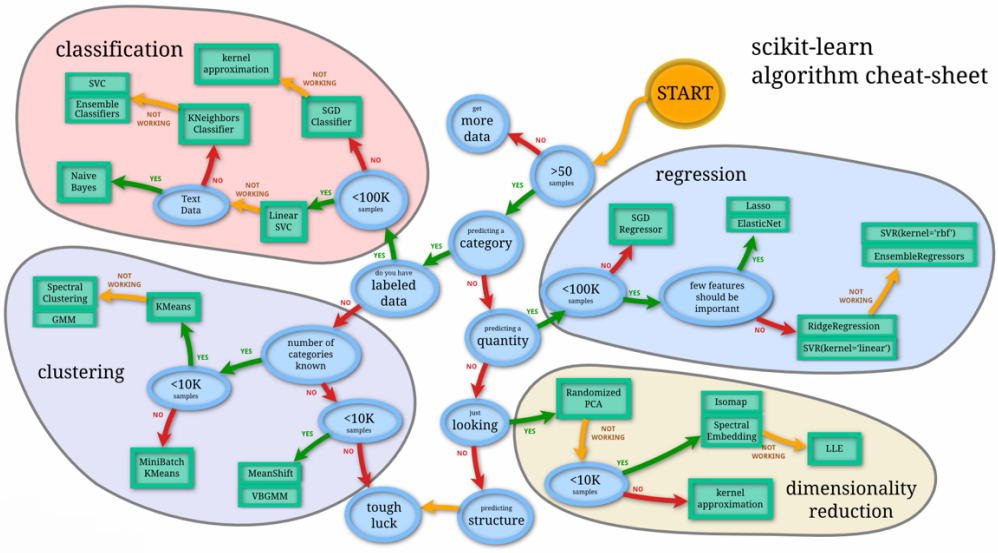

Figura 5. Flujograma para la elección del algoritmo adecuado

2.4.2.1 Categorizar el problema

Para categorizar nuestro problema [23] podemos hacernos las siguientes preguntas: ¿nuestros datos están etiquetados? En caso afirmativo, estaremos ante un problema de aprendizaje supervisado, en caso contrario ante uno de aprendizaje no supervisado. ¿El resultado que obtendremos al resolver el problema es un número? En caso afirmativo estaremos ante un problema de regresión, en caso de que el resultado

sea una categoría o clase será un problema de clasificación. Por otro lado, si el resultado esperado es un conjunto de grupos, el problema será de *clustering* o agrupamiento.

Por el tipo de datos que se usarán en este TFM, como se mostrará más adelante en 2.5.1 SELECCIÓN DEL DATASET, los datos están etiquetados y el resultado que devolverá el modelo es una categoría, por lo que respondiendo a las preguntas anteriores podemos categorizar el problema como de aprendizaje supervisado y de clasificación.

2.4.2.2 Revisar los datos

2.4.2.2.1 Analizar los datos

El tamaño del *dataset* y el tipo de estos (cualitativos, numéricos, etiquetados, etc.) son clave [24] a la hora de la elección del algoritmo adecuado. Una herramienta fundamental que ayudará a conocer los datos es la estadística descriptiva, mediante la representación y cálculos de correlaciones, medias, medianas, percentiles, etc. se adquirirá un conocimiento adecuado de la clase de datos de los que se disponen.

2.4.2.2.2 Procesar los datos

Este proceso incluye el pre-procesado, el *profiling* y la limpieza de los datos. En ocasiones, también supone extraer datos de diferentes fuentes y unificarlos en un solo conjunto.

- El pre-procesado consiste en preparar los datos para que tengan un formato adecuado que pueda ser pasado como entrada al algoritmo de machine learning.
- El *profiling* se refiere a las técnicas empleadas para extraer todo el conocimiento posible de los datos en bruto del *dataset* para así tener una visión clara de lo que podremos obtener de estos datos una vez lo pasemos como entrada al algoritmo.
- La limpieza de los datos es la técnica que se emplea para identificar aquellas partes de datos que son incorrectos, incompletos, poco precisos o simplemente inexistentes para así modificarlos, reemplazarlos o eliminarnos.

2.4.2.2.3 Transformar los datos

Este proceso, también conocido como *feature engineering*, se centra en las características o propiedades de los datos que forman parte del *dataset*. Se realiza un proceso sobre las características de los datos para transformarlas en características o propiedades que representen de una forma más adecuada de cara al problema que queremos resolver, aumentado la exactitud del modelo de machine learning.

2.4.2.3 Elegir el algoritmo

En este punto ya se debería de tener un conocimiento amplio de qué tipo de problema hay que resolver, qué datos hay disponibles y qué nos pueden ofrecer estos. Por lo tanto, sería el momento de identificar el algoritmo adecuado en cuanto a aplicabilidad y tiempo de implementación.

En la selección del algoritmo para la construcción del modelo de *machine learning*, algunos factores para tener en cuenta son:

- La **exactitud** y **precisión** del modelo a la hora de identificar las relaciones y patrones entre las variables de los datos.
- La **interpretabilidad** del modelo: qué características están incluidas en ese modelo y el contexto del problema.
- La **complejidad** del modelo: numero de características de los datos, complejidad algorítmica, etc.

- La **escalabilidad** del modelo: cómo de fácil sería aplicar el modelo a un conjunto de datos muchos más grande.
- El **tiempo** que llevaría **construir, entrenar y probar** el modelo.
- El **tiempo** que llevaría al modelo a **realizar las predicciones**.
- Que el modelo **satisfaga el objetivo perseguido** para la resolución del problema planteado.

Como uno de los objetivos de este TFM es la aplicación de algoritmos supervisados de machine learning basados en árboles de decisión en el ámbito de la seguridad TIC en general, y de la detección de intrusiones web en particular, **nos centraremos en los algoritmos que implementan árboles de decisión**. No obstante, existe literatura científica [25] que avala el uso de estos tipos de algoritmos con esa finalidad, presentando un alto porcentaje de exactitud en cuanto a la detección de los intentos de intrusión.

Los árboles de decisión tienen por objetivo crear un modelo que sea capaz de predecir el valor de una variable objetivo a través de aprender reglas sencillas de decisión inferidas de las características o propiedades de los datos. En este caso, a través de los datos (peticiones HTTP a un servidor web) deberá predecir si este es un intento de intrusión (conexión anómala) o no (conexión normal).

Son muchas las **ventajas que caracterizan a los árboles de decisión**. A continuación, se indican las principales:

- Son simples de entender e interpretar. Además, se pueden visualizar de forma gráfica.
- Necesita de un menor procesamiento de los datos.
- Pueden manejar tanto datos de tipo numérico como categórico.
- Usan una aproximación de caja blanca para la resolución del problema por lo que los resultados son más fáciles de interpretar (lo de caja negra son más complicados de interpretar), permitiendo mostrar en la propia representación gráfica del árbol las decisiones que el modelo va tomando para la clasificación de los datos.

Sin embargo, también presenta algunas **desventajas**:

- Se pueden obtener árboles demasiados complejos que resulten en un problema común como el *overfitting*.
- En ocasione pueden ser algo inestables, en el sentido que cambios mínimos en los datos pueden resultar en árboles completamente diferentes.
- Pueden presentar algo de sesgo en los casos en los que existen categorías predominantes, para ello es necesaria balancear el *dataset* para evitarlo.

Los principales algoritmos que implementan árboles de decisión son:

- **ID3** (Iterative Dichotomiser 3). Desarrollado en 1986 por Ross Quinlan, este algoritmo crea un árbol n-ario formado por nodos de decisión y nodos hojas. Cada nodo de decisión dispone de dos o más ramas, cada una representando las posibilidades de esa decisión, es decir, los valores de la característica que representa, mientras que los nodos hoja representan la clasificación correspondiente en relación con la propiedad que se pretende clasificar, representado por el nodo raíz del árbol. Las características representadas solo pueden ser categóricas. Además, el recorrido del árbol se realiza de forma voraz, sin *backtracking*, de arriba abajo.

- C4.5 es el sucesor del ID3 y a la vez una extensión de este. Carece de la restricción de que las características empleadas solo puedan ser categóricas, definiendo para ello una característica discreta (basada en variables numéricas) que realice una partición del valor de la característica continua en un conjunto discreto de intervalos. C4.5 toma como entrada la salida de ID3 (el árbol entrenado) y los convierte a un conjunto de reglas *if-then*. Se realiza un cálculo para saber como de precisas son estas reglas para así establecer el orden en el que tienen que ser aplicadas. El podado del árbol se realiza eliminando una precondición de la regla en aquellos casos en los que la precisión de la regla mejore sin esa precondición. En cada nodo del árbol, C4.5 elige un atributo de los datos que más eficazmente dividan el conjunto de muestras en subconjuntos enriquecidos en una clase u otra. Su criterio [26] es el normalizado para ganancia de información (diferencia de entropía) que resulta en la elección de un atributo para dividir los datos. El atributo con la mayor ganancia de información normalizada se elige parámetro de decisión. El algoritmo C4.5 va dividiendo recursivamente los datos en subconjuntos más pequeños.

- C5.0 es la última versión del algoritmo desarrollado por Quinlan que usa menos memoria y construye conjuntos mas pequeños de reglas en comparación con C4.5 pero con la salvedad de que tiene una licencia propietaria.

- CART (Classification and Regresion Trees) también es muy similar a C4.5 pero difiere en que soporta variables numéricas objetivo (para regresión) y no calcula conjuntos de reglas. Crea árboles binarios en los que, para cada iteración, selecciona la variable predictiva y el punto de partición que mayor ganancia de información genere.

La librería *scikit-learn* que se empleará en este TFM usa una versión optimizada del algoritmo CART. Como la versión de CART implementada en la librería no soporta las variables categóricas, y como el tipo de datos que se usarán, como se mostrará más adelante en 2.5.1 SELECCIÓN DEL DATASET, contienen variables categóricas, el *dataset* a emplear tendrá que ser transformado a variables numéricas.

En conclusión, **el algoritmo que se empleará será la implementación de CART proporcionada por la librería *scikit-learn* de Python**.

2.5 Datasets

2.5.1 Selección del dataset

Teniendo en cuenta que el objetivo de este TFM es la aplicación de algoritmos supervisados de machine learning basados en árboles de decisión en el ámbito de la seguridad TIC en general, y de la detección de intrusiones web en particular, se ha realizado una búsqueda de *datasets* relacionados con la detección de intrusiones 'IDS' y peticiones 'HTTP' a servidores web que contuvieran intentos de intrusión o ataques web.

Hay tanto literatura científica [27] [28] que realiza un repaso al estado del arte en cuanto a los *datasets* disponibles para IDS, como recursos web especializados [29] [30] [31] [32] que los recopilan, actualizan y los van corrigiendo. Incluso hay buscadores especializados de *datasets* como el de Google [33], Kaggle [34], y UCI [35].

De todos los recursos indicados anteriormente, son varios los *datasets* que se han encontrado y que a continuación se agrupan en dos categorías:

1. para intrusiones en un sistema/red en general: KDD-99, DARPA 1999, DARPA 2000, NSL-KDD, ISCX 2012, UNB-CIC, CSE-CIC-IDS2018, etc.

2. para intrusiones en servidores/aplicaciones web. No son muchos los *datasets* encontrados que están acotados a intentos de conexión HTTP a servidores o aplicaciones web. Los principales son dos: CSIC-2010 [36] y ECML/PKDD 2007 [37].

Dado que el objetivo es centrarse en los ataques a través de peticiones HTTP únicamente, se descarta el primer grupo. Del segundo grupo, **se selecciona el CSIC-2010** y se deja en reserva, por si hiciera falta el ECML/PKDD 2007.

El *dataset* CSIC-2010 contiene el tráfico generado hacia una aplicación web de comercio electrónico expresamente desarrollada para tal efecto. En esta aplicación web los usuarios pueden comprar artículos usando un carrito de compra y pueden registrarse proporcionando su información personal. Como la aplicación web está en español, el *dataset* contiene caracteres latinos.

CSIC-2010 contiene 36,000 peticiones normales y más de 25,000 peticiones anómalas. Cada petición HTTP está etiquetada como normal o anómala.

El *dataset* CSIC-2010 contiene tres tipos de peticiones anómalas:

1. Ataques estáticos que intentan realizar peticiones sobre recursos ocultos o inexistentes. Estas peticiones incluyen ficheros obsoletos, identificadores de sesión codificadas mediante reescritura de la URL, ficheros de configuración, ficheros por defecto, etc.

2. Ataques dinámicos que modifican los argumentos de peticiones válidas: inyecciones SQL, inyecciones CRLF, *cross site scripting*, *buffer overflows*, etc.

3. Peticiones ilegales no intencionadas. Estas peticiones, aunque no tienen un propósito malintencionado, no siguen los parámetros de comportamiento normal de la aplicación web y no tienen la misma estructura que los valores de los parámetros válidos (por ejemplo, un número de teléfono compuesto por letras).

3 Desarrollo de la propuesta

3.1 Instalación del entorno

Este capítulo describe el proceso de diseño e implementación del modelo y analiza los resultados obtenidos.

3.1.1 Descripción del entorno

La realización del proyecto se ha realizado íntegramente en un portátil con sistema operativo macOS y sobre un sistema operativo GNU/Linux Ubuntu virtualizado. Por consiguiente, se identifican dos entornos:

- Entorno de desarrollo, que comprende las fases de investigación, diseño e implementación del modelo y documentación. Estas fases se han llevado a cabo sobre el sistema operativo anfitrión del portátil macOS.

- Entorno de ejecución del modelo. Se ha realizado en un sistema operativo virtualizado GNU/Linux Ubuntu.

Las características del portátil son las siguientes:

Nombre del modelo:	MacBook Pro
Identificador del modelo:	MacBookPro11,1
Sistema Operativo:	macOS Catalina versión 10.15.4
Nombre del procesador:	Dual-Core Intel Core i5
Velocidad del procesador:	2,4 GHz
Cantidad de procesadores:	1
Cantidad total de núcleos:	2
Caché de nivel 2 (por núcleo):	256 KB
Caché de nivel 3:	3 MB
Tecnología Hyper-Threading:	Activado
Memoria RAM:	8 GB

Las características de la máquina virtual:

Software de virtualización:	Oracle VirtualBox versión 6.0.18 r136238 (Qt5.6.3)
Sistema Operativo virtualizado:	GNU/Linux Ubuntu versión 18.04
Procesador emulado:	Intel(R) Core(TM) i5-4258U CPU @ 2.40GHz
Cantidad de procesadores:	1
Cantidad total de núcleos:	1
Memoria RAM:	5623 MB

3.1.2 Software instalado del entorno de desarrollo

A continuación, se lista el software que ha sido necesario instalar sobre el sistema operativo macOS y se describe su funcionalidad:

- ATOM versión 1.45.0. Se trata de un editor de texto altamente personalizable y configurable mediante paquetes de terceros que añaden nuevas funcionalidades. Se ha usado para la implementación en Python del modelo de machine learning.

- **Git** versión 2.21.0. Es un sistema de control de versiones distribuido, gratuito y de código abierto diseñado para manejar proyectos tanto grandes como pequeños con gran velocidad y eficiencia.
- **GitHub Desktop** versión 2.4.1, como *frontend* para Git. Ha permitido manejar de forma visual todos los cambios en el proyecto e interactuar con otros repositorios que se han usado en el proyecto y sobre los que se le ha realizado algún *pull-request*.
- **Oracle Virtual Box**, versión 6.0.18. Utilizado para virtualizar el sistema de ejecución sobre el que se ha probado el modelo de *machine learning* desarrollado, un GNU/Linux Ubuntu versión 18.04.
- **Zotero** versión 5.0.86. Es un gestor bibliográfico con el que se ha gestionado todos los *papers* académicos, recursos web y demás documentación consultados y posteriormente referenciados en esta memoria.

3.1.3 Software instalado del entorno de ejecución

A continuación, se realiza una enumeración no exhaustiva de los principales componentes software instalados en el GNU/Linux Ubuntu versión 18.04, así como su parametrización principal:

- En Oracle Virtual Box ha habilitado la compartición de directorios *guest-host*.
- Instalación de **Ubuntu 18.04:**
  - Servicio **ssh** para conexión desde MacOS, ejecución y revisión de resultados del modelo.
  - Instalación de **Anaconda** para la gestión de varios entornos Python.
  - El dataset se encuentra originalmente comprimido con rar, así que se ha instalado paquetería extra en Ubuntu para poder descomprimirlo desde Python. Adicionalmente es necesario instalar el módulo **pyunpack** de Python.
- Instalación de **Python versión 3.7.7**.
- Principales módulos Python empleados:
  - **Scikit-learn**. Es una librería para aprendizaje automático de software libre para el lenguaje de programación Python. Incluye varios algoritmos de clasificación, regresión y análisis de grupos entre los cuales están máquinas de vectores de soporte, árboles de decisión, *Random Forest*, *Gradient boosting*, *K-means* y *DBSCAN*.
  - **Urllib**. Ha proporcionado métodos útiles de gestión y manejo de URL, tanto para la descarga del dataset como el tratamiento de las peticiones HTTP.
  - **Pandas** y **Numpy**. Permiten un manejo y operación sencilla sobre los *datasets*.
  - **matplotlib.pyplot, seaborn, pydotplus y graphviz**, para la generación de los gráficos del proyecto (árboles de decisión, matriz de correlaciones, curvas de aprendizaje, curvas de validación, etc.)
  - A continuación, el listado completo de módulos Python que han sido necesario instalar:
    - backcall==0.1.0
    - certifi==2019.11.28
    - cycler==0.10.0
    - decorator==4.4.2
    - EasyProcess==0.2.10
    - entrypoint2==0.2
    - graphviz==0.13.2
    - ipython==7.13.0

- ipython-genutils==0.2.0
- jedi==0.16.0
- joblib==0.14.1
- kiwisolver==1.2.0
- matplotlib==3.2.1
- numpy==1.18.2
- pandas==1.0.3
- parso==0.6.2
- patool==1.12
- pexpect==4.8.0
- pickleshare==0.7.5
- prompt-toolkit==3.0.5
- ptyprocess==0.6.0
- pydotplus==2.0.2
- Pygments==2.6.1
- pyparsing==2.4.6
- python-dateutil==2.8.1
- pytz==2019.3
- pyunpack==0.1.2
- scikit-learn==0.22.2.post1
- scipy==1.4.1
- seaborn==0.10.0
- six==1.14.0
- skfeature==1.0.0
- sklearn==0.0
- traitlets==4.3.3
- validate-email==1.3
- wcwidth==0.1.9

- Instalación de la librería **sklearn-features**.
    - Es un conjunto de utilidades y algoritmos programados en Python para la selección de features. Para su utilización, se ha realizado un clonado del repositorio de GitHub https://github.com/jundongl/scikit-feature, se ha corregido un error que no permitía el correcto funcionamiento y se ha publicado un *fork* en https://github.com/valvasor/scikit-feature. Se ha realizado un *pull-request* al desarrollador original, para que acepte los cambios propuestos de resolución.

3.2 Implementación del modelo

3.2.1 Introducción

Como ya se ha indicado anteriormente, el objetivo del TFM es desarrollar un modelo predictivo de *machine learning* basado en árboles de decisión cuya tarea sea clasificar un conjunto de peticiones HTTP en peticiones normales y anómalas.

Se ha seguido una aproximación estándar que ha contemplado tres grandes fases: estudiar el problema, entrenar el modelo con un conjunto de datos y evaluar los resultados.

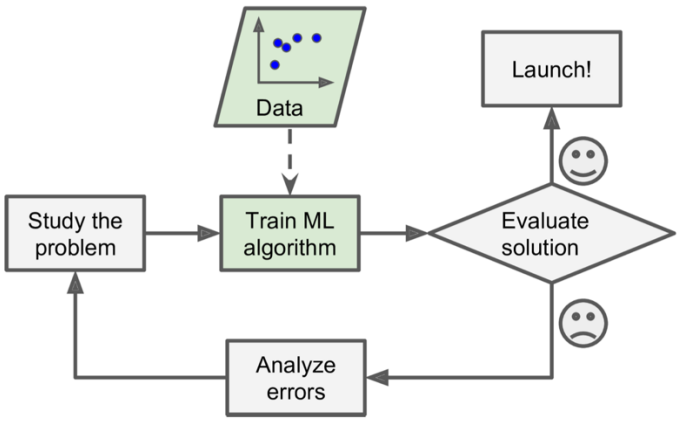

Figura 6. Aproximación para el desarrollo del modelo

En cuanto a su aprendizaje, las principales características del modelo propuesto en este TFM son:

- **Aprendizaje supervisado**. El conjunto de datos de entrenamiento ya contiene la solución buscada, es decir, cada una de las peticiones HTTP del conjunto de datos que usa el modelo como entrenamiento se encuentran etiquetadas como: petición normal o anómala.

- **Aprendizaje en lote** (*offline learning* o *batch learning*). El modelo se alimenta con todo el conjunto de datos disponible de una sola vez, se entra con él, y a continuación se pone en producción para que las nuevas peticiones HTTP vayan siendo clasificadas, sin inferir más conocimiento de ellas de lo que ya se ha aprendido al entrenar el modelo.

- **Aprendizaje basado en modelo.** Sobre un conjunto de datos de entrenamiento se construye un modelo predictivo que será usado posteriormente sobre otro conjunto de datos para realizar las predicciones.

- **Aprendizaje con partición del conjunto de datos (*splitting train/test/validation datasets)*.** Aunque también se ha implementado un ejemplo de modelo basado en *cross-validation* con *k-folding*, el modelo propuesto ha sido entrenado, probado y mejorado a través de tres conjuntos de datos para cada uno de los fines.

En cuanto a la tipología de IDS, según la clasificación dada en 2.2.1 DETECCIÓN Y PREVENCIÓN DE INTRUSIONES, el modelo sigue una aproximación basada en anomalías.

Teniendo en cuenta todo lo anterior se ha seleccionado *Python* y *scikit-learn* como las principales tecnologías para implementar el modelo descrito anteriormente. El código *Python* sigue la guía de estilo PEP8 [38].

A continuación, se describen secuencialmente las fases que forman parte de la implementación realizada.

3.2.2 Descarga y parseado del conjunto de datos

En esta fase, se realiza la descarga, *parseado* del conjunto de datos y transformación a formato CSV.

El conjunto de datos utilizado ha sido el HTTP-DATASET-CSIC-2010 [39], que está disponible en la web del Instituto de Seguridad de la Información (ISI) del Consejo Superior de Investigaciones Científicas (CSIC).

El conjunto de datos está compuesto por tres ficheros de texto: normalTrafficTraining.txt, normalTrafficTest.txt y anomalousTrafficTest.txt, que suman un total de 97,065 peticiones HTTP (de tipo GET, POST y PUT). Los dos primeros ficheros contienen peticiones clasificadas como normales y el tercero las que se han clasificado como anómalas.

En esta fase, se implementa la descarga de los tres ficheros, su parseado para leer el contenido y extraer las características (features) y, finalmente, transformarlo a formato CSV con las diferentes *features* separadas por una barra vertical ("|") y unirlo todo en un solo fichero temporal sobre el que se seguirá trabajando para transformarlo y adecuarlo a nuestro modelo.

Se parte del formato siguiente:

```
001: GET
002: http://localhost:8080/tienda1/publico/anadir.jsp?id=2&nombre=Jam%F3n+Ib%E9rico&precio=85&cantidad=%27%3B+DROP+TABL
     E+usuarios%3B+SELECT+*+FROM+datos+WHERE+nombre+LIKE+%27%25&B1=A%F1adir+al+carrito HTTP/1.1
003: User-Agent: Mozilla/5.0 (compatible; Konqueror/3.5; Linux) KHTML/3.5.8 (like Gecko)
004: Pragma: no-cache
005: Cache-control: no-cache
006: Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
007: Accept-Encoding: x-gzip, x-deflate, gzip, deflate
008: Accept-Charset: utf-8, utf-8;q=0.5, *;q=0.5
```

```
009: Accept-Language: en
010: Host: localhost:8080
011: Cookie: JSESSIONID=B92A8B48B9008CD29F622A994E0F650D
012: Connection: close
013:
014:
015: POST http://localhost:8080/tienda1/publico/anadir.jsp HTTP/1.1
016: User-Agent: Mozilla/5.0 (compatible; Konqueror/3.5; Linux) KHTML/3.5.8 (like Gecko)
017: Pragma: no-cache
018: Cache-control: no-cache
019: Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
020: Accept-Encoding: x-gzip, x-deflate, gzip, deflate
021: Accept-Charset: utf-8, utf-8;q=0.5, *;q=0.5
022: Accept-Language: en
023: Host: localhost:8080
024: Cookie: JSESSIONID=AE29AEEBDE479D5E1A18B4108C8E3CE0
025: Content-Type: application/x-www-form-urlencoded
026: Connection: close
027: Content-Length: 146
028:
029: id=2&nombre=Jam%F3n+Ib%E9rico&precio=85&cantidad=%27%3B+DROP+TABLE+usuarios%3B+SELECT+*+FROM+datos+WHERE+nombre+LI
     KE+%27%25&B1=A%F1adir+al+carrito
```

Y se obtiene este otro, donde la primera fila corresponde al nombre de las características y el resto de filas a cada una de las peticiones HTTP:

```
001: feature-Method|feature-URL|feature-Protocol|feature-User-Agent|feature-Pragma|feature-Cache-Control|feature-
     Accept|feature-Accept-Encoding|feature-Accept-Charset|feature-Accept-Language|feature-Host|feature-Cookie|feature-
     Content-Type|feature-Content-Length|feature-Query|feature-Connection|Label
002: GET|http://localhost:8080/tienda1/index.jsp|HTTP/1.1|Mozilla/5.0 (compatible; Konqueror/3.5; Linux) KHTML/3.5.8
     (like Gecko)|no-cache|no-
     cache|text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5|x-gzip,
     x-deflate, gzip, deflate|utf-8, utf-8;q=0.5,
     *;q=0.5|en|localhost:8080|JSESSIONID=EA414B3E327DED6875848530C864BD8F|notpresent|notpresent|notpresent|close|norma
     l
003: GET|http://localhost:8080/tienda1/publico/anadir.jsp?id=1&nombre=Jam%F3n+Ib%E9rico&precio=39&cantidad=41&B1=A%F1ad
     ir+al+carrito|HTTP/1.1|Mozilla/5.0 (compatible; Konqueror/3.5; Linux) KHTML/3.5.8 (like Gecko)|no-cache|no-
     cache|text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5|x-gzip,
     x-deflate, gzip, deflate|utf-8, utf-8;q=0.5,
     *;q=0.5|en|localhost:8080|JSESSIONID=54E25FF4B7F0E4E855B112F882E9EEA5|notpresent|notpresent|id=1&nombre=Jam%F3n+Ib
     %E9rico&precio=39&cantidad=41&B1=A%F1adir+al+carrito|close|normal
004: POST|http://localhost:8080/tienda1/publico/anadir.jsp|HTTP/1.1|Mozilla/5.0 (compatible; Konqueror/3.5; Linux)
     KHTML/3.5.8 (like Gecko)|no-cache|no-
     cache|text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5|x-gzip,
     x-deflate, gzip, deflate|utf-8, utf-8;q=0.5,
     *;q=0.5|en|localhost:8080|JSESSIONID=788887A0F479749C4CEEA1E268B4A501|application/x-www-form-
     urlencoded|74|id=1&nombre=Jam%F3n+Ib%E9rico&precio=39&cantidad=41&B1=A%F1adir+al+carrito|close|normal
005: GET|http://localhost:8080/tienda1/publico/autenticar.jsp?modo=entrar&login=caria&pwd=egipciaca&remember=off&B1=Ent
     rar|HTTP/1.1|Mozilla/5.0 (compatible; Konqueror/3.5; Linux) KHTML/3.5.8 (like Gecko)|no-cache|no-
     cache|text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5|x-gzip,
     x-deflate, gzip, deflate|utf-8, utf-8;q=0.5,
     *;q=0.5|en|localhost:8080|JSESSIONID=54ECD5EE8EF7EFE4BB26C701B150ED7B|notpresent|notpresent|modo=entrar&login=cari
     a&pwd=egipciaca&remember=off&B1=Entrar|close|normal
006: POST|http://localhost:8080/tienda1/publico/autenticar.jsp|HTTP/1.1|Mozilla/5.0 (compatible; Konqueror/3.5; Linux)
     KHTML/3.5.8 (like Gecko)|no-cache|no-
     cache|text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5|x-gzip,
     x-deflate, gzip, deflate|utf-8, utf-8;q=0.5,
     *;q=0.5|en|localhost:8080|JSESSIONID=23391DBBADEC19FE01E02D201F278C6A|application/x-www-form-
     urlencoded|60|modo=entrar&login=caria&pwd=egipciaca&remember=off&B1=Entrar|close|normal
007: GET|http://localhost:8080/tienda1/publico/caracteristicas.jsp?id=2|HTTP/1.1|Mozilla/5.0 (compatible;
     Konqueror/3.5; Linux) KHTML/3.5.8 (like Gecko)|no-cache|no-
     cache|text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5|x-gzip,
     x-deflate, gzip, deflate|utf-8, utf-8;q=0.5,
     *;q=0.5|en|localhost:8080|JSESSIONID=98D39BB41546D2A0C7E89F02B01A97F8|notpresent|notpresent|id=2|close|normal
008: GET|http://localhost:8080/tienda1/publico/anadir.jsp?id=2&nombre=Jam%F3n+Ib%E9rico&precio=85&cantidad=%27%3B+DROP+
     TABLE+usuarios%3B+SELECT+*+FROM+datos+WHERE+nombre+LIKE+%27%25&B1=A%F1adir+al+carrito|HTTP/1.1|Mozilla/5.0
     (compatible; Konqueror/3.5; Linux) KHTML/3.5.8 (like Gecko)|no-cache|no-
     cache|text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5|x-gzip,
     x-deflate, gzip, deflate|utf-8, utf-8;q=0.5,
     *;q=0.5|en|localhost:8080|JSESSIONID=B92A8B48B9008CD29F622A994E0F650D|notpresent|notpresent|id=2&nombre=Jam%F3n+Ib
     %E9rico&precio=85&cantidad=%27%3B+DROP+TABLE+usuarios%3B+SELECT+*+FROM+datos+WHERE+nombre+LIKE+%27%25&B1=A%F1adir+
     al+carrito|close|anomalous
009: POST|http://localhost:8080/tienda1/publico/anadir.jsp|HTTP/1.1|Mozilla/5.0 (compatible; Konqueror/3.5; Linux)
     KHTML/3.5.8 (like Gecko)|no-cache|no-
     cache|text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5|x-gzip,
     x-deflate, gzip, deflate|utf-8, utf-8;q=0.5,
     *;q=0.5|en|localhost:8080|JSESSIONID=AE29AEEBDE479D5E1A18B4108C8E3CE0|application/x-www-form-
     urlencoded|146|id=2&nombre=Jam%F3n+Ib%E9rico&precio=85&cantidad=%27%3B+DROP+TABLE+usuarios%3B+SELECT+*+FROM+datos+
     WHERE+nombre+LIKE+%27%25&B1=A%F1adir+al+carrito|close|anomalous
010: GET|http://localhost:8080/tienda1/publico/anadir.jsp?id=2%2F&nombre=Jam%F3n+Ib%E9rico&precio=85&cantidad=49&B1=A%F
     1adir+al+carrito|HTTP/1.1|Mozilla/5.0 (compatible; Konqueror/3.5; Linux) KHTML/3.5.8 (like Gecko)|no-cache|no-
     cache|text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5|x-gzip,
     x-deflate, gzip, deflate|utf-8, utf-8;q=0.5,
     *;q=0.5|en|localhost:8080|JSESSIONID=F563B5262843F12ECAE41815ABDEEA54|notpresent|notpresent|id=2%2F&nombre=Jam%F3n
     +Ib%E9rico&precio=85&cantidad=49&B1=A%F1adir+al+carrito|close|anomalous
```

Para aquellas características que no contienen ningún valor, se rellenan con la cadena de texto '*notpresent*'

El listado de características que contiene cada muestra (petición HTTP), es el siguiente:

feature-Method	Cabecera HTTP. Método de petición http. Posibles valores: GET, POST y PUT
feature-URL	Cabecera HTTP. URL de la petición
feature-Protocol	Cabecera HTTP. Protocolo de comunicación y versión de este
feature-User-Agent	Cabecera HTTP Agente de usuario. Información de la aplicación cliente que se conecta al servidor web
feature-Pragma	Cabecera HTTP que normalmente se sigue usando por compatibilidad para http/1.0
feature-Cache-Control	Cabecera HTTP que especifica las directivas que se deben cumplir por los mecanismos de caché durante la cadena request/response
feature-Accept	Cabecera HTTP que indica los formatos soportados para las respuestas del servidor
feature-Accept-Encoding	Cabecera HTTP que indica las codificaciones aceptadas
feature-Accept-Charset	Cabecera HTTP que indica el conjunto de caracteres soportados
feature-Accept-Language	Cabecera HTTP que indica los idiomas aceptados
feature-Host	Cabecera HTTP. Nombre de dominio y Puerto desde el que está accesible el servidor
feature-Cookie	Cabecera HTTP. Cookie previamente enviada al servidor y que le sirve para consultar actividad previa del cliente en el servidor
feature-Content-Type	Cabecera HTTP. El tipo de contenido de la petición en POST o PUT
feature-Content-Length	Cabecera HTTP. El tamaño del contenido de la petición en bytes
feature-Query	Query string (cadena de consulta) es la parte de una URL que contiene los datos que deben pasar a aplicaciones web
feature-Connection	Cabecera http. Opciones de control de la conexión cliente/servidor.
Label	Este campo indica si la petición HTTP es: normal o anómala

Tabla 1.Características originales del dataset

Son un total de **16 características**, más una variable (*Label*) que identifica el tipo de petición.

3.2.3 Pre-procesado y partición del conjunto de datos

En esta fase de implementación es donde tiene lugar la ingeniería de características, más conocido por su termino en ingles *feature engineering*, que es el proceso de usar el conocimiento del dominio, la seguridad TIC en este caso, para extraer características (*feature extracting*) de datos sin procesar a través de técnicas de minería de datos. Estas características se pueden usar para mejorar el rendimiento de los algoritmos de aprendizaje automático empleados por el modelo predictivo. Se han implementado diversas funciones en el código para este proceso de *feature engineering* [40] que han facilitado en gran medida la manipulación de URL.

De este modo, tomando como punto de partido el fichero CSV anterior, que contiene todo el conjunto de datos, y a partir de las dos características más representativas: feature-URL y feature-Query se crean otras 15 nuevas.

feature-Query-Length	Longitud del query
feature-Query-Longest-Value	Longitud del parámetro más largo del query
feature-Query-token_avg	Longitud media de todos los tokens de la query, entendiendo token como el nombre del parámetro y su valor
feature-Query-token_count	Número de tokens del query
feature-Query-token_largest	Longitud del token más largo del query

feature-URL-Param-Value-Count-Dot	Número de ocurrencias del caracter punto "." en el query
feature-URL-Param-Value-Count-Percentage	Número de ocurrencias del caracter porcentaje "." en el query
feature-URL-Param-Value-Count-SpecialChars	Número de ocurrencias de caracteres especiales (=,$,&,%, etc.) en el query
feature-URL-Parameters-Names	Cadena de texto con todos los nombres de los parámetros del query concatenados
feature-URL-Parameters-Num	Número de parámetros en el query
feature-URL-Parameters-Type	Cadena de texto que representa los tipos de cada uno de los valores de los parámetros del query. Ejemplo: nombreVar1=2,nombreVar2=Jose →[DIGIT][ALPHABET]
feature-URL-Path	Path de la URL. Ejemplo: /tienda1/imagenes/2.gif
feature-URL-Path-Length	Longitud del path de la URL
feature-URL-Path-ResourceExtension	Extensión del fichero del recurso del path de la URL (Ejemplo: gif)
feature-URL-Path-ResourceExtension-Length	Longitud de la extensión del fichero del recurso del path de la URL

Tabla 2. Nuevas características creadas

Los algoritmos de *scikit-learn*, librería de machine learning empleada en este TFM, no soporta las características en formato texto, por lo que el próximo paso de esta fase es convertir esas características a números, asignando a cada cadena de texto concreta el mismo valor numérico para todas las ocurrencias de esta en todo el conjunto de datos.

Así, por ejemplo, pasamos de tener algo como esto:

```
GET|localhost:8080|notpresent|notpresent|notpresent|0|0|/asf-logo-
wide|14|0|0|notpresent|notpresent|notpresent|0|0|0|10.0|1|10|anomalous
```

A esto otro:

```
0|1|2|2|2|3|3|10480|58|3|3|2|2|2|3|3|3|8|9|8|anomalous
```

Por último, en esta fase, se realiza la partición del conjunto completo en tres conjuntos de datos:

- *training dataset*
- *test dataset*
- *validation dataset*

Se han empleado para entrenar, probar y mejorar el modelo, respectivamente. El tamaño de estos conjuntos de datos representa el 80%, 10% y 10% respectivamente del total. Se emplea un muestreo aleatorio para que los conjuntos de datos resultantes tengan muestras heterogéneas.

Tras haber ejecutado versiones previas del modelo final, tras el análisis de resultados se han identificado dos características que añaden ruido al conjunto: feature-Cookie y feature-URL. Aunque incluyéndolas permitían lograr al modelo un porcentaje de acierto en las predicciones del 100%, han sido eliminadas. La razón es que al contener identificadores únicos como el id de la cookie o la URL, que no generalizan adecuadamente el conocimiento del modelo y por tanto pueden producir *overfitting, además de* árboles de decisión poco profundos (se obtenían árboles con profundidad 2) y el peso de decisión recaía íntegramente en estas dos características.

3.2.4 Selección de características

En esta fase se realizó la selección de las mejores características que fueron las que finalmente formaron parte de los *datasets* finales que fueron empleados por el modelo para su entrenamiento, mejora y cálculo de la exactitud, entre otras métricas.

Para el modelo se ha optado por una selección automática de las características, aunque también se fijaron otras mediante selección manual, para comparar al final el porcentaje de exactitud en la predicción del modelo.

Para la selección automática, se ha empleado el *validation dataset*, cuyos datos no fueron empleados en ningún momento por el proceso de aprendizaje del modelo. El código de la función que se encarga de la selección automática se presenta a continuación junto a una explicación de este:

```
001: def auto_feature_selection(X_validation, y_validation):
002:     # to decrease the time elapsed by CFS, autopreselection of features by Variance and SelectKBest
003:     sel = VarianceThreshold(threshold=(.6 * (1 - .6)))
004:     sel.fit(X_validation)
005:     X_validation = X_validation[X_validation.columns[sel.get_support(indices=True)]]
006:     sel2 = SelectKBest(chi2, k=10).fit(X_validation, y_validation)
007:     X_validation = X_validation[X_validation.columns[sel2.get_support(indices=True)]]
008:     # get all the headers (features) in the dataframe
009:     features = list(X_validation.columns.values)
010:     X_validation_np = X_validation.to_numpy()
011:     y_validation_np = y_validation.to_numpy()
012:     n_samples = X_validation_np.shape[0]
013:     print("\n\nAuto selected features by Correlation Feature Selection (CFS) method:")
014:     # Aplication of CFS algorithm to select the best features
015:     features_selected_idx = CFS.cfs(X_validation_np, y_validation_np)
016:     selected_features = []
017:     for idx in features_selected_idx:
018:         # cfs returns a 6 elements list, if less than 6 features are selected, -1 are used to complete the list
019:         if idx >= 0:
020:             selected_features.append(features[idx])
021:             print('    - %s' % features[idx])
022:     return selected_features
```

Se realizan tres pasos en la selección automática:

1. Se eliminan las características con una varianza inferior al 60% (líneas 002-005). Que tengan varianza 0% indica que todas las muestras tienen el mismo valor para esa característica, un valor el 100% indica que cada muestra tiene un valor diferente. En el *validation dataset* no hay ninguna característica con varianza inferior al 60%, por lo que no se elimina ninguna.

2. De las resultantes, se seleccionan las 10 mejores con la función *SelectKBest* de *scikit-learn* calculando la distribución chi-cuadrado entre cada una de las características y la variable de clase (*Label*). El algoritmo selecciona como las diez mejores estas:

   feature-Content-Length, feature-Host, feature-Query, feature-Query-Length, feature-Query-Longest-Value, feature-Query-token_avg, feature-URL-Parameters-Names, feature-URL-Parameters-Type, feature-URL-Path, feature-URL-Path-ResourceExtension.

3. Por ultimo, de estas diez, se seleccionan las mejores, pero esta vez mediante el método de selección de características basado en correlaciones (CFS, por sus siglas en ingles). Este método, objeto de la tesis presentada en [41], parte de una hipótesis principal que establece que un buen conjunto de características poseen una correlación alta con la variable de clase (*Label*, en este caso) y a la vez poca correlación entre ellas mismas. El algoritmo usado y que implementa este método ha sido *CFS()*, que forma parte de la librería **sklearn-features**, comentada en el apartado 3.1.3 SOFTWARE INSTALADO DEL ENTORNO DE EJECUCIÓN. Con la ejecución de este algoritmo sobre el conjunto de características resultantes del apartado anterior, se seleccionaron las siguientes seis características, que fueron las que finalmente se emplearon en el modelo predictivo:

feature-Query	Query string
feature-URL-Path-ResourceExtension	Extensión del fichero del recurso del path de la URL (Ejemplo: gif)
feature-Query-Length	Longitud del query

feature-Host	Nombre de dominio y Puerto desde el que está accesible el servidor
feature-URL-Path	Path de la URL. Ejemplo: /tienda1/imagenes/2.gif
feature-Content-Length	Tamaño del contenido de la petición en bytes

Tabla 3. Características finales seleccionadas automáticamente por el modelo

Con *scikit-learn* es posible calcular la importancia de cada característica y representarla gráficamente, el resultado, que se muestra en la FIGURA 7. IMPORTANCIA DE LAS CARACTERÍSTICAS, indica que la más importante es feature-Query, seguida de feature-URL-Path y de feateure-Host.

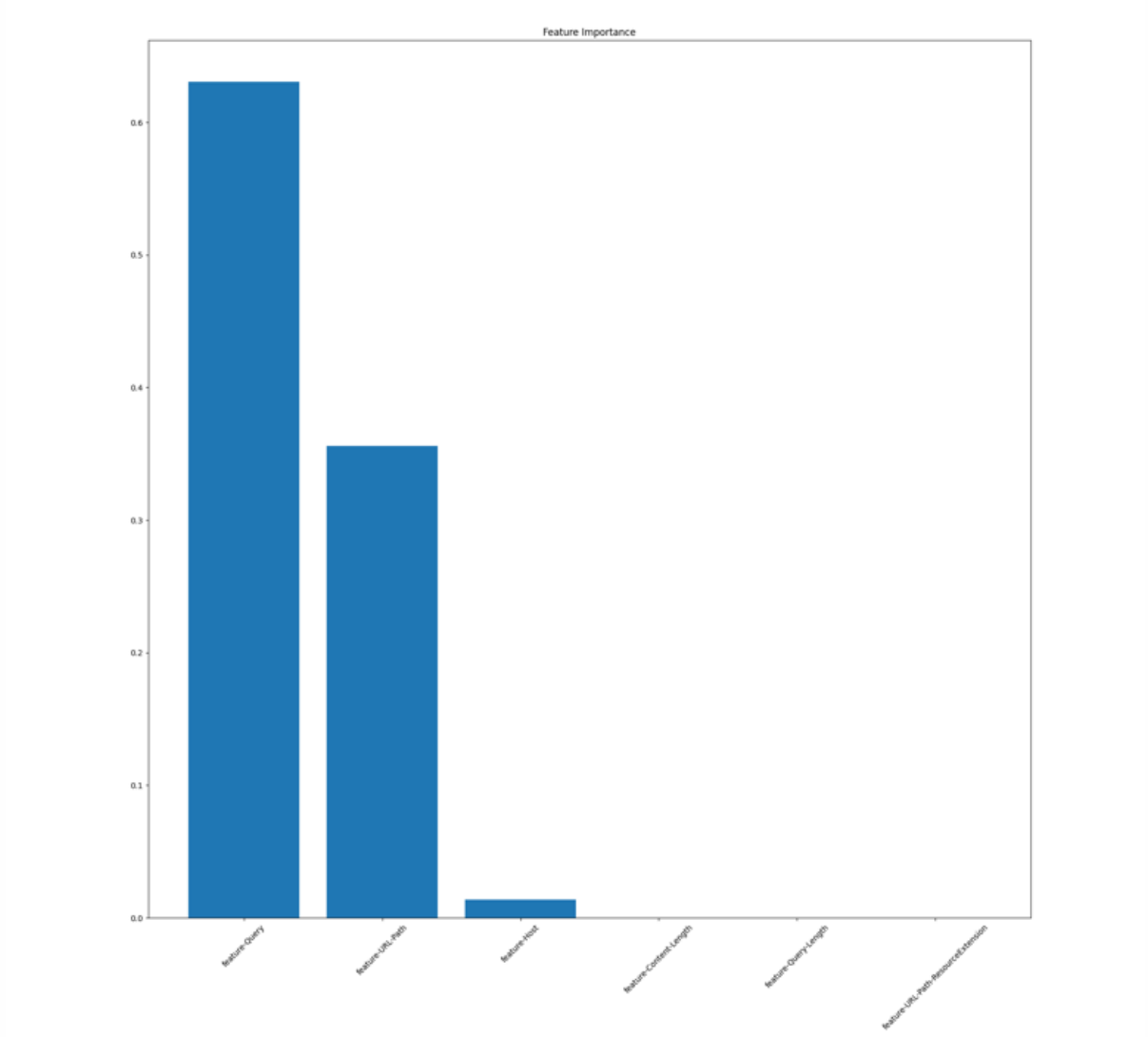

Figura 7. Importancia de las características

La correlación entre las características y la variable de clase (*Label*), como parte esencial en la selección automática de características, puede comprobarse gráficamente al calcular un mapa de calor (ver FIGURA 8. MAPA DE CALOR DE LA MATRIZ DE CORRELACIONES), que en este caso ha sido realizado mediante la librería *seaborn* [42] de Python. En este mapa de calor, si nos fijamos en la última columna, correspondiente a *Label*, podemos ver las intersecciones entre esta y las características, donde se ve que las características de la TABLA 3. CARACTERÍSTICAS FINALES SELECCIONADAS AUTOMÁTICAMENTE POR EL MODELO, tienen colores más fríos (amarillos/verdes), con una correlación mayor.

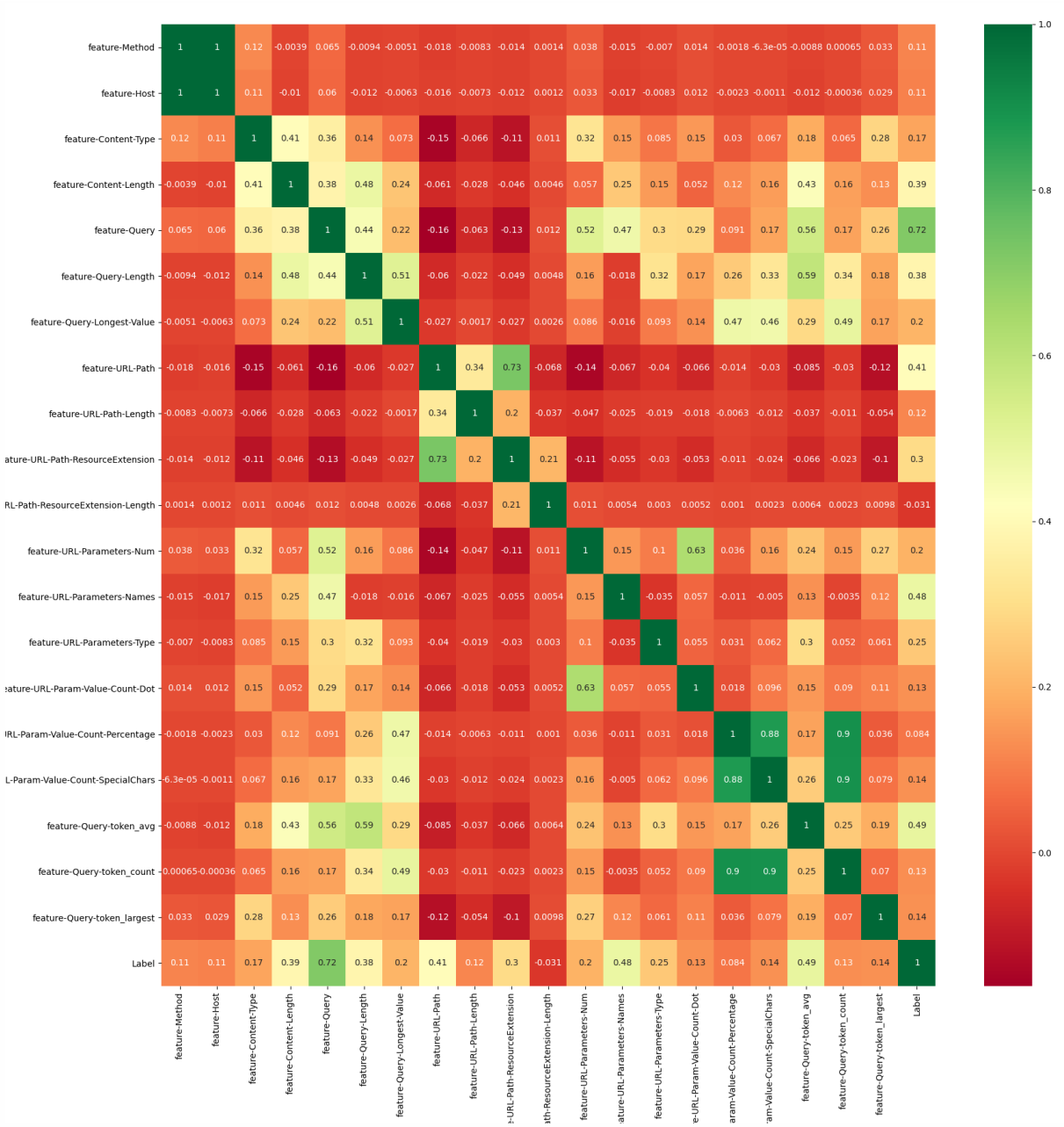

Figura 8. Mapa de calor de la matriz de correlaciones

3.2.5 Selección automatizada de los parámetros del modelo

Para la implementación del modelo de clasificación se usará un árbol de decisión. La librería *scikit-learn* dispone de la siguiente clase *DecisionTreeClassifier*:

```
class sklearn.tree.DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=None, min_samples_split=2,
min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_i
mpurity_decrease=0.0, min_impurity_split=None, class_weight=None, presort='deprecated', ccp_alpha=0.0
```

Invocar al constructor de esta clase con los argumentos por defecto suele resultar en un fenómeno conocido como *overfitting*, en el que el árbol se ha ajustado tantísimo a los datos de entrenamiento que provocará que no se realicen predicciones correctas sobre datos no entrenados. Es decir, los resultados de clasificación de las peticiones HTTP serán muy buenos, con un alto porcentaje de aciertos, pero si aplicamos ese árbol a otros datos diferentes a los de entrenamiento, los resultados serán mucho peores.

Por lo tanto, el objetivo de esta fase es seleccionar de forma automatizada, a través de una búsqueda exhaustiva, los mejores argumentos para pasarlos al constructor de esta clase, evitando así un posible *overfitting*. La acción de buscar los mejores parámetros se le conoce por su término en ingles: *hyperparameters tuning*.

Esto se conseguirá con el empleo de la función *GridSeachCV() [43]* de *scikit-learn* (línea 016).

```
023: def auto_hyperparameters_selection(X_validation, y_validation):
024:     ''' Tuning the hyper-parameters of the DecisionTree '''
025:     criterion = ['gini', 'entropy']
026:     splitter = ['best', 'random']
027:     max_depth = np.arange(2, 6)
028:     min_samples_split = [2, 6, 13, 20, 27, 34, 40] # best 2 to 40
029:     min_samples_leaf = [1, 5, 9, 13, 17, 20] # best: 1 to 20
030:     class_weight = ['balanced', None]
031:     param_grid = {'criterion': criterion,
032:                   'splitter': splitter,
033:                   'min_samples_split': min_samples_split,
034:                   'max_depth': max_depth,
035:                   'min_samples_leaf': min_samples_leaf,
036:                   'class_weight': class_weight}
037:     print("\n\nAuto selected hyperparameters (DecisionTreeClassifier) by Exhaustive Grid Search: ")
038:     clf = GridSearchCV(DecisionTreeClassifier(), param_grid, cv=5)
039:     clf.fit(X_validation, y_validation)
040:     #print(clf.best_score_)
041:     #print(clf.best_params_)
042:     for idx in clf.best_params_:
043:         print(f'    - {idx}: {clf.best_params_[idx]}')
044:     return clf.best_params_
```

En este caso, se usará el *validation dataset*, que no será empleado durante el proceso de aprendizaje del modelo, sobre el que se aplicará una validación cruzada (*cross-validation*) [44] para la partición del conjunto de datos en entranamiento/test. La función *GridSearchCV()* realiza una búsqueda exhaustiva, es decir, va probando una a una todas las combinaciones de los parámetros y calcula los resultados obtenidos por cada una de esas combinaciones, para ver que parámetros producen el mejor resultado. Al tratarse de un proceso exhaustivo, el tiempo de ejecución de esta función puede elevarse exponencialmente, así que se ha optado por fijarse en los parámetros más representativos de DecisionTreeClassifier (*criterion, splitter, max_depth, min_samples_split, min_samples_leaf y class_weight*) y realizar la búsqueda de los valores óptimos de solo esos parámetros.

Además, como realizar la búsqueda sobre todos los posibles valores que pueden tomar esos parámetros seguiría teniendo un tiempo de ejecución muy elevado (horas en la máquina de ejecución utilizada), se ha realizado un estudio [45] previo de cuáles serían los rangos de valores más plausibles para aquellos parámetros que pueden tomar un amplio rango de valores (*max_depth, min_samples_split* y *min_samples_leaf*). Los resultados de ese estudio previo son que:

- *max_depth*: un árbol con una profundidad muy grande puede producir *overfitting* (el modelo intentaría describir demasiado cada característica) mientras que una profundidad demasiado pequeña podría producir el efecto contrario conocido como underfitting. Teniendo en cuenta que se dispone de 6 características seleccionadas, la profundidad adecuada debería estar entre 2 y 6, que será el rango de valores que se le pasará a *GridSearchCV()* (ver línea 005).

- *min_samples_split*: según '*An empirical study on hyperparameter tuning of decision trees*' [46] el rango ideal para este parámetro suele estar entre 1 y 40 para el algoritmo CART, que es el implementado por *scikit-learn* para *DecisionTreeClassifier*. Este parámetro suele controlar que no se produzca *overfitting*. Por lo tanto, le pasamos una lista de valores dentro del rango 1 a 40.

- *min_samples_leaf*: en [46] también se concluye que el rango óptimo para evitar un *overfitting* es 1 a 20.

Para el resto de los parámetros, al tener pocos valores posibles, se los pasamos todos. También se le pasa como argumento el parámetro cv (*cross-validation*) con un valor de 3, para la validación cruzada del *validation dataset*, por lo que empleará una estrategia recursiva de ir partiendo el *validation dataset* hasta en tres ocasiones.

A continuación, se muestra la salida que proporciona la función *auto_hyperparameters_selection()*, indicando los mejores parámetros para *DecisionTreeClassifier*:

```
Auto selected hyperparameters (DecisionTreeClassifier) by Exhaustive Grid Search:
    - class_weight: balanced
    - criterion: gini
    - max_depth: 3
    - min_samples_leaf: 1
    - min_samples_split: 2
    - splitter: best
```

Se han seleccionado automáticamente los parámetros:

- **class_weight: balanced**. Es curioso que *GridSearchCV()* haya seleccionado este valor, porque el *dataset* solo contiene una variable de clase (*Label*), por lo que realmente no hay "pesos" que asignar a cada una de las variables de clase, la única tendrá un peso de 1. Se hubiera esperado que se seleccionara automáticamente el valor 'None', pero tampoco tiene mayor importancia porque no influiría en el resultado final.

- **criterion: gini**. Indica el criterio a tomar para decidir la calidad de las particiones del nodo del árbol. '*gini*' es uno de los dos (el otro es '*entropy*') posibles métodos de partición de los nodos del árbol. En este caso ha seleccionado que se realicen las particiones de acuerdo con la medida de impureza conocida como *Gini*, en detrimento de la ganancia de información proporcionada por el método de entropía.

- **max_depth: 3.** Al parecer, la profundidad óptima del árbol, para este caso, sería 3.

- **min_samples_leaf: 1.** El mínimo número de muestras (peticiones http) que habrá en cada nodo del árbol será de 1.

- **min_samples_split: 2.** el mínimo número de muestras necesarias para poder realizar la partición de un nodo será de 2.

- **splitter: best**. Aunque tiene mayor carga computacional que la otra opción para este parámetro (random), ya que realiza cálculos más exigentes en lugar de una simple selección aleatoria, permite elegir el mejor criterio de partición de cada nodo, basándose en la medida de impureza.

Gráficamente, la selección de *max_depth = 3* puede comprobarse si representamos su curva de validación [47] (ver FIGURA 9. CURVA DE VALIDACIÓN PARA MAX_DEPTH), es decir, cómo varía el *accuracy* del modelo para los diferentes valores que puede tomar el parámetro *max_depth*. Se ve claramente como el *accuracy* llega al 100% cuando el *max_depth* llega al valor 3, conservándose ese *accuracy* hasta valores superiores.

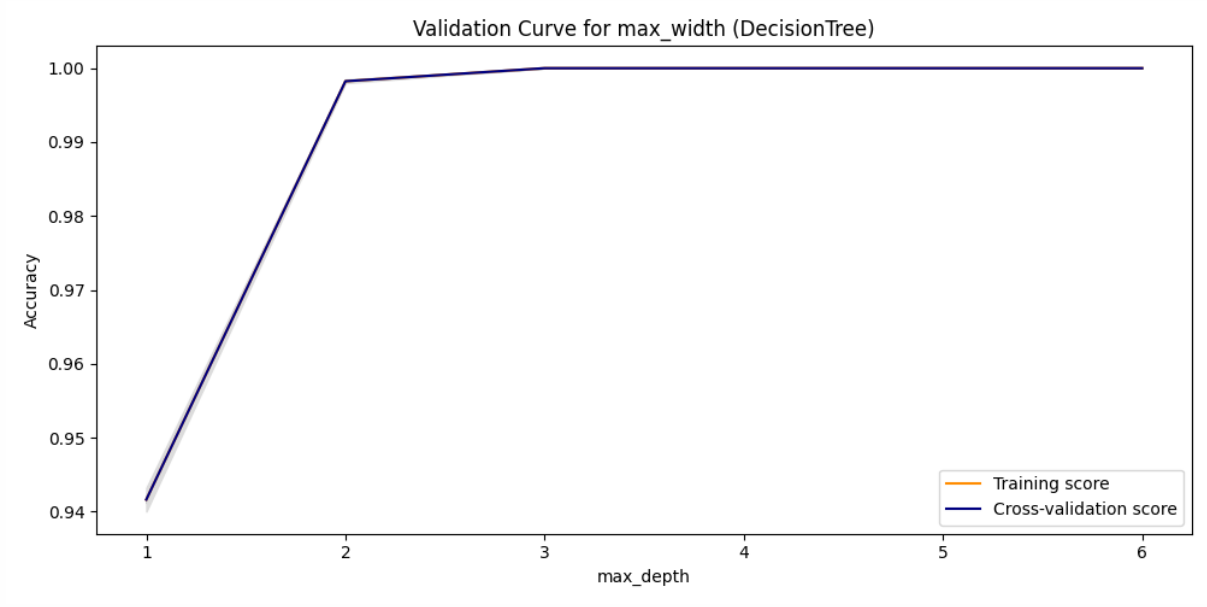

Figura 9. Curva de validación para max_depth

En cuanto al parámetro *min_samples_split*, si calculamos su curva de validación, (ver FIGURA 10. CURVA DE VALIDACIÓN PARA MIN_SAMPLES_SPLIT), podemos comprobar que para todos los valores del rango 1 a 40 se obtendría una *accuracy* del 100%.

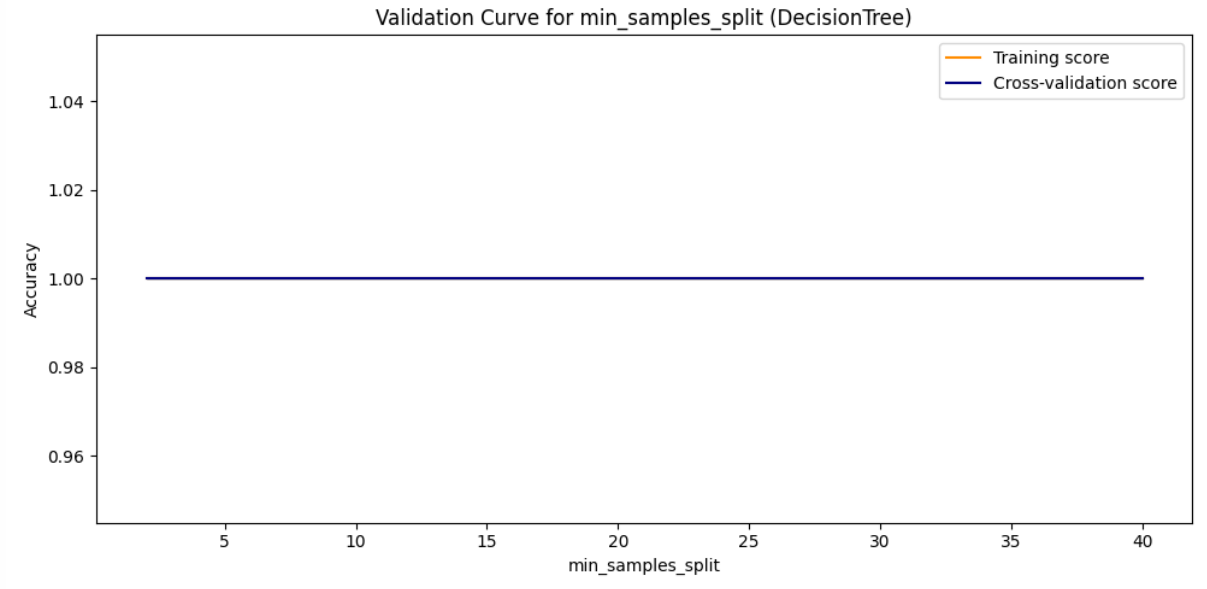

Figura 10. Curva de validación para min_samples_split

Lo mismo ocurre para el parámetro *min_samples_leaf*. Se puede comprobar en su curva de validación (ver FIGURA 11. CURVA DE VALIDACIÓN PARA MIN_SAMPLES_LEAF).

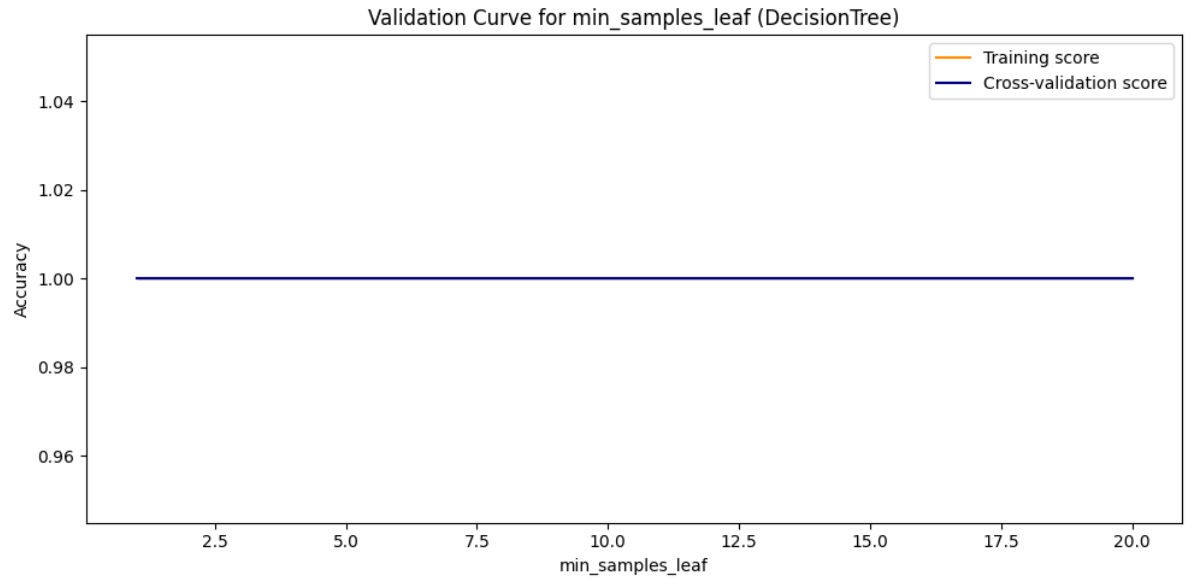

Figura 11. Curva de validación para min_samples_leaf

En este punto ya se tienen todos los elementos necesarios: algoritmo de clasificación *DecisionTreeClassifier()*, sus parámetros y el conjunto de datos (*training dataset*) para poder ejecutar la fase de entrenamiento del modelo.

3.2.6 Entrenamiento y verificación del modelo

En esta fase es donde tiene lugar el proceso más importante, el entrenamiento del modelo con el conjunto de datos de entrenamiento *training dataset* y su posterior verificación con el conjunto de datos de prueba *test dataset*.

```
001:    '''
002:    ====== TRAINING AND TESTING THE MODEL ================================
003:    '''
004:    t0 = time()
005:    classifier = DecisionTreeClassifier(**best_hyperparameters)
006:    # Train Decision Tree classifier
007:    classifier = classifier.fit(X_train[auto_selected_features],y_train)
008:    times['training'] = time() - t0
009:    # Predict the response for test dataset
010:    t0 = time()
011:    predictor = classifier.predict(X_test[auto_selected_features])
012:    total_time = "%0.3fs" % (time() - t0)
013:    times['predicting'] = time() - t0
```

Se instancia un árbol de decisión para clasificación *DecisionTreeClassifier()* con los mejores hiperparámetros posibles, previamente seleccionados (línea 005), se entrena con los datos de entrenamiento (línea 007) y se verifica con los datos de prueba (línea 011).

Todo el proceso de entrenamiento sobre un total de 77.652 peticiones HTTP, y clasificación de un total de 9707 peticiones HTTP (las correspondientes al *test dataset*) lleva algo más de 1 segundo. En cualquier caso, más adelante se analizarán con más detalle los resultados obtenidos.

3.2.7 Generación de gráficos del modelo

Durante esta fase tiene lugar la generación de todas las gráficas que darán apoyo en las decisiones que se tomen en el resto de las fases y también para analizar los resultados obtenidos por el modelo.

Se han utilizado las librerías de Python: *seaborn, IPython, matplotlib* y las propias de *scikit-learn* para la generación de todas las gráficas del modelo.

En concreto, durante esta fase se han generado las siguientes gráficas:

- Mapa de calor con la matriz de correlaciones entre las características y variable de clase (*correlation_heatmap.png*)
- Gráfica que muestra la importancia de cada una de las características seleccionadas para la clasificación de la petición HTTP (*feature_importances.png*)
- Árbol de decisión del modelo (*http-traffic-tree.png*)
- Curvas de aprendizaje del modelo (*learning-curves.png*)
- Curva de validación del modelo para la característica *max_width* (*validation-curve-max_width.png*)
- Curva de validación del modelo para la característica *min_samples_leaf* (*validation-curve-min_samples_leaf.png*)
- Curva de validación del modelo para la característica *min_samples_split* (*validation-curve-min_samples_split.png*)

Todas ellas, junto con otras generadas en otras fases, serán de apoyo más adelante para explicar los resultados obtenidos por el modelo,

3.2.8 Generación de informes de resultados

Durante esta penúltima fase, se han calculado todas las métricas necesarias para la generación de los informes de resultados obtenidas a través de la ejecución del modelo.

Se han calculado las más representativas [48]:

tiempos de ejecución	Tiempos de ejecución para cada una de las fases: - gathering datasets - processing datasets - selecting features - selecting hyper-parameters - training the model - predicting the test set - plotting graphs
correctamente clasificadas	Número de peticiones HTTP correctamente clasificadas como normal o anómala
incorrectamente clasificadas	Número de peticiones HTTP incorrectamente clasificadas como normal o anómala
verdaderos negativos	son las peticiones HTTP anómalas correctamente clasificadas
falsos negativos	son las peticiones HTTP normales clasificadas como anómalas
verdaderos positivos	son las peticiones HTTP normales clasificadas correctamente
falsos positivos	peticiones HTTP anómalas clasificadas como normales
Precision	Proporción entre las peticiones correctamente clasificadas y el total de peticiones
Recall	proporción entre los verdaderos positivos que se han clasificado correctamente y el total de verdaderos positivos
f1-score	A menudo es conveniente combinar precision y recall en una sola métrica conocida como f1-score, permitiendo comparar dos modelos con una sola métrica. Es la media harmónica de precision y recall. A diferencia de la media aritmética que trata a todos los valores por igual, la harmónica le da más peso a

	los valores bajos. Como resultado, el modelo clasificador solo obtendrá un valor f1-score alto si tanto precision como recall son altos.
accuracy	Es la proporción entre las peticiones correctamente clasificadas y el total de peticiones del conjunto de datos. Se trata quizás de la métrica más intuitiva y sobre la que basaremos la decisión de si el modelo desarrollado es bueno o no

Tabla 4. Métricas calculadas

3.2.9 Resultados usando otros modelos

Esta última fase, simplemente ejecuta otros modelos de clasificación, con los parámetros por defecto, para comparar la exactitud (*accuracy*) de estos con la del modelo propuesto. Aunque la comparativa no es muy rigurosa debido a que no se han ajustado los hiperparámetros de los distintos modelos, nos proporciona una ligera idea sobre cuán exactos podrían llegar a ser en la clasificación de peticiones http.

3.2.9.1 Árbol de decisión con selección manual de características

Además, también tiene lugar en esta fase la ejecución del modelo propuesto, pero con una selección manual de las características, para comparar el *accuracy* con el modelo propuesto, cuyas características han sido seleccionadas automáticamente mediante algoritmos especialmente implementados para esa tarea.

Con la selección manual de características, obtiene un 86% de exactitud en la clasificación de las peticiones que forman parte del *test dataset*. Las características, manualmente seleccionadas, han sido:

- feature-Query-Length
- feature-Query-Longest-Value
- feature-URL-Parameters-Num
- feature-URL-Path-Length
- feature-URL-Param-Value-Count-SpecialChars

El árbol de decisión obtenido de este modo es el que se muestra a continuación en la FIGURA 12. ÁRBOL DE DECISIÓN CON SELECCIÓN MANUAL DE CARACTERÍSTICAS.

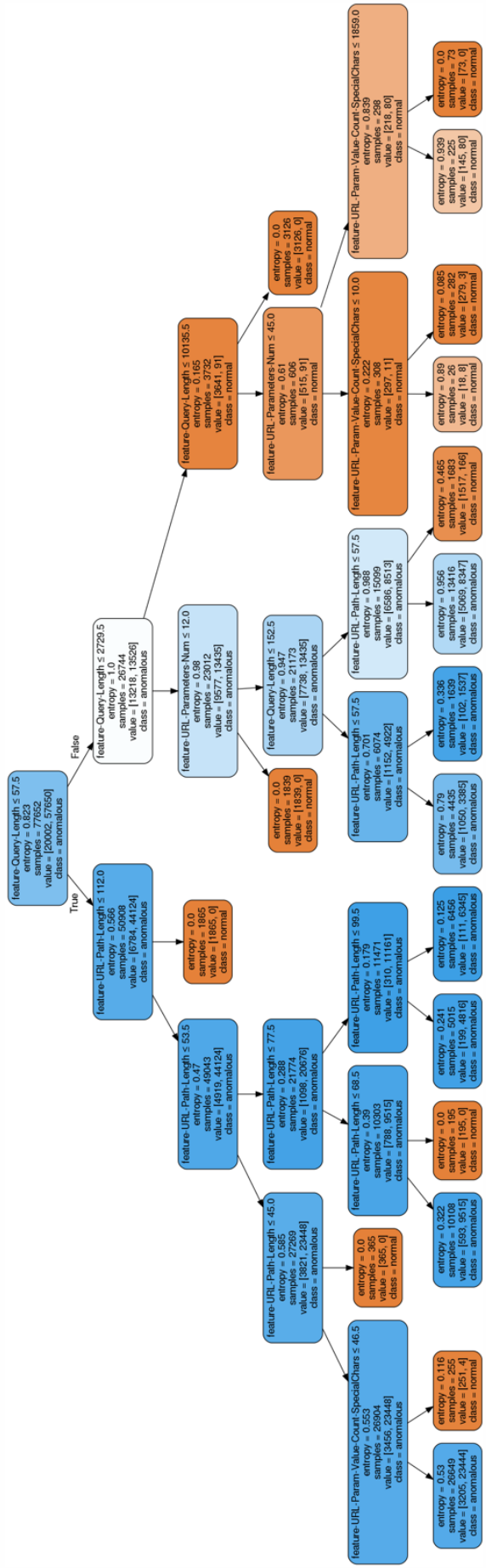

Figura 12. Árbol de decisión con selección manual de características

3.2.9.2 Árbol de decisión con cross-validation y selección automática de características

Se ha probado un árbol de decisión que emplee solamente el conjunto de datos de entrenamiento, tanto para el entrenamiento como para la validación del modelo. Esta técnica suele emplearse habitualmente cuando no se tiene un conjunto lo suficientemente grande como para partirlo en *training, test* y *validation datasets*. En este caso, se han usado las características seleccionadas automáticamente mediante el método de correlación de características (CFS) empleado en el modelo propuesto en este TFM. Se ha probado este tipo de técnica, a propuesta del director del TFM, con el objetivo de conocer su funcionamiento y comportamiento, teniendo en cuenta que es una técnica estándar en machine learning. Se ha logrado conseguir un 100% con esta técnica.

El árbol de decisión obtenido de este modo es el que se muestra a continuación, en la FIGURA 13. ÁRBOL DE DECISIÓN CON CROSS-VALIDATION.

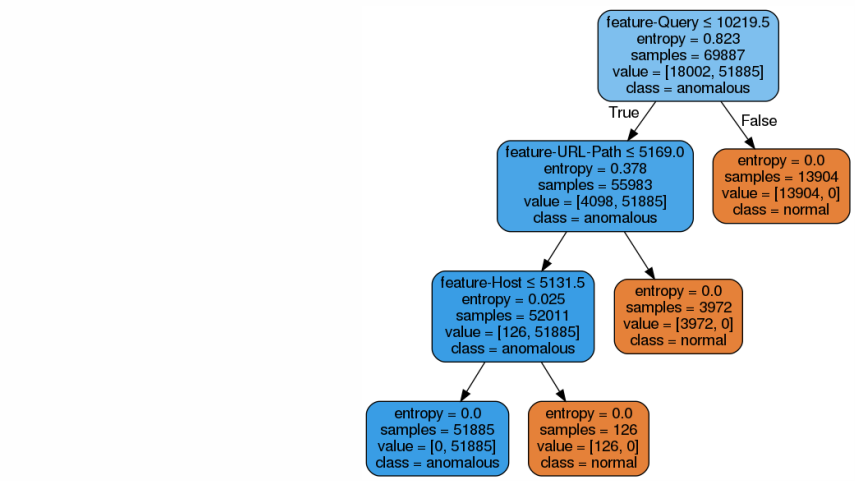

Figura 13. Árbol de decisión con cross-validation

3.2.9.3 Otros modelos

A continuación, se muestran, junto con los anteriores, los resultados obtenidos tras la ejecución de estos otros modelos:

```
RESULTS for other models:
================================================================================

    Accuracy for DecisionTree model with k-fold cross-validation training: 1.00
    Accuracy for DecisionTree model with manually selected features: 0.86
    Accuracy for RandomForest model: 1.00
    Accuracy for AdaBoost model: 1.00
    Accuracy for SVM model: 1.00
    Accuracy for Gaussian Naive Bayes model: 0.87
```

El modelo Gaussiano Naïve-Bayes ha obtenido un resultado similar, 87% de exactitud en la clasificación.

El resto de los modelos, AdaBoost y el de Máquina de Vectores de Soporte (SVM) han obtenido un 100% de exactitud.

4 Ejecución y análisis de resultados

4.1 Ejecución

La ejecución del clasificador propuesto devuelve la siguiente salida:

```
(tfm_env) jose@uoc:~/uoc/tfm$ ./src/http_classifier.py

Splitting the dataset into train, test and validation sets...
    - train-set: 80%  (77652 samples)
    - test-set: 10%  (9707 samples)
    - validation-set: 9%  (9706 samples)


Auto selected features by Correlation Feature Selection (CFS) method:
    - feature-Query
    - feature-URL-Path-ResourceExtension
    - feature-Query-Length
    - feature-Host
    - feature-URL-Path
    - feature-Content-Length


Auto selected hyperparameters (DecisionTreeClassifier) by Exhaustive Grid Search:
    - class_weight: balanced
    - criterion: gini
    - max_depth: 3
    - min_samples_leaf: 1
    - min_samples_split: 2
    - splitter: best


Plotting graphs...


 RESULTS for DecisionTree classifier:
================================================================================

         Time:
                  gathering datasets:         11.821s
                  processing datasets:        16.160s
                  selecting features:         8.091s
                  selecting hyper-parameters: 0h:06m:39s
                  training the model:         0.512s
                  predicting the test set:    0.024s
                  plotting graphs:            0h:00m:57s

         Total Instances: 9707
                  Correctly classified:   9707 (100.000%)
                  Incorrectly classified: 0 (0.000%)

         Confusion Matrix values:
                  True Negatives  (TN):   7129 (73.442%)
                  False Positives (FP):   0 (0.000%)
                  False Negatives (FN):   0 (0.000%)
                  True Positives  (TP):   2578 (26.558%)
                  (Negative='Normal request', Positive='Anomalous request')


         Scoring report:

              precision    recall  f1-score   support

      normal       1.00      1.00      1.00      2578
   anomalous       1.00      1.00      1.00      7129

    accuracy                           1.00      9707
   macro avg       1.00      1.00      1.00      9707
weighted avg       1.00      1.00      1.00      9707


 RESULTS for other models:
================================================================================

         Accuracy for DecisionTree model with k-fold cross-validation training: 1.00
         Accuracy for DecisionTree model with manually selected features: 0.86
         Accuracy for RandomForest model: 1.00
         Accuracy for AdaBoost model: 1.00
         Accuracy for SVM model: 1.00
         Accuracy for Gaussian Naive Bayes model: 0.87
```

4.2 Análisis de resultados

En cuanto a cada una de las fases, sus tiempos de ejecución han sido los siguientes:

```
gathering datasets:          11.821s
processing datasets:         16.160s
selecting features:          8.091s
selecting hyper-parameters: 0h:06m:39s
training the model:          0.512s
predicting the test set:     0.024s
plotting graphs:             0h:00m:57s
```

Se puede apreciar que la mayor parte del tiempo total, con diferencia, se la ha llevado la selección automática de los parámetros adecuados del árbol de decisión, unos 6 minutos. La fase de creación de gráficas ha llevado casi 1 minuto en completarse. Le sigue el tiempo dedicado al procesamiento del *dataset* para convertirlo en un formato adecuado y añadirle nuevas características, con unos 16 segundos, y la descarga y unión de todos los conjuntos de datos en uno solo, con casi 12 segundos. La selección de características ha consumido un total de 8 segundos, mientras que el entrenamiento y validación del modelo apenas ha llevado 1 segundo.

La escalabilidad del modelo en cuanto al tiempo de entrenamiento respecto del número de peticiones HTTP, representada en la FIGURA 14. ESCALABILIDAD DEL MODELO, sigue una tendencia casi lineal, por lo que se considera que el modelo es escalable en el caso de aumentar el número de peticiones HTTP con las que entrenar al modelo.

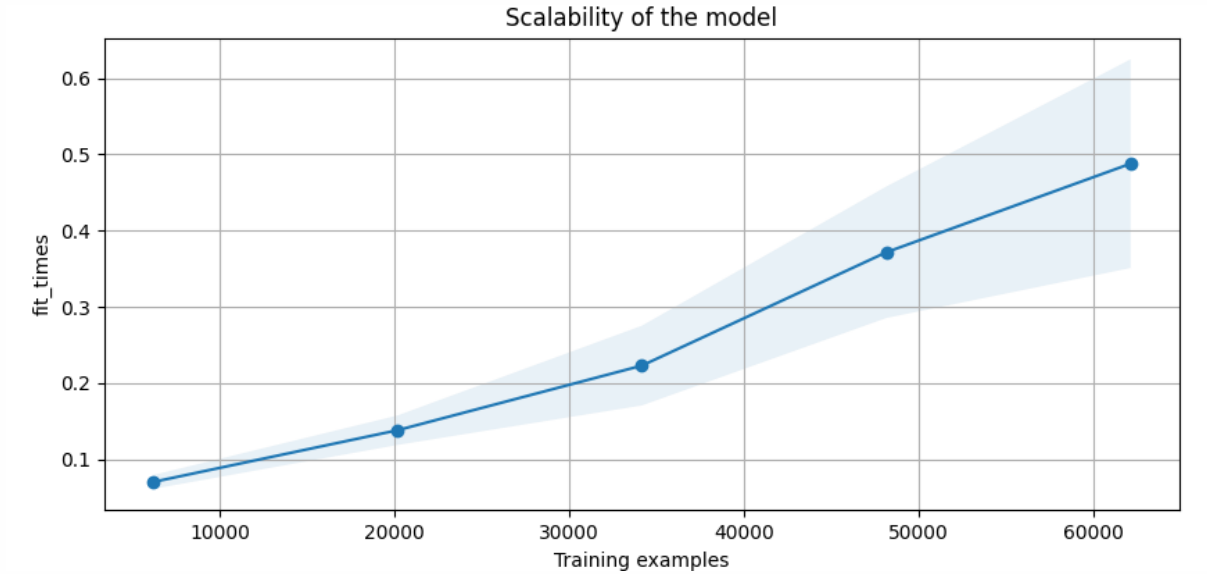

Figura 14. Escalabilidad del modelo

En cuanto al rendimiento del modelo, entendido como la exactitud (*accuracy)* que el modelo va consiguiendo conforme avanza el tiempo de aprendizaje, se muestra en la FIGURA 15. RENDIMIENTO DEL MODELO y se interpreta que a partir de 0.35 segundos, el modelo comienza a mejorar la exactitud.

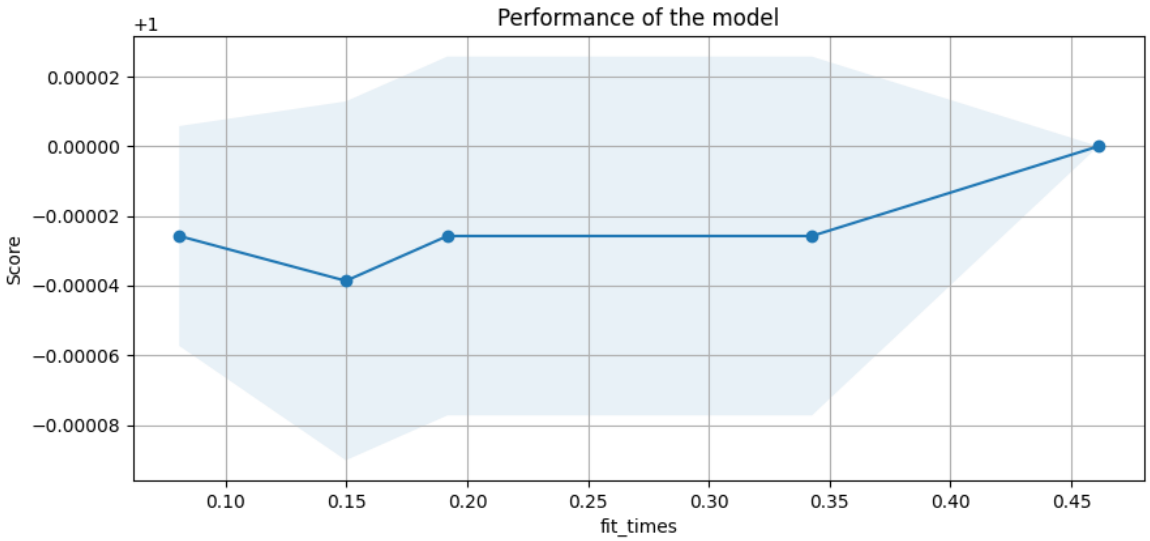

Figura 15. Rendimiento del modelo

Las curvas de aprendizaje [49] del árbol de decisión tanto para el *dataset* de entrenamiento (*training dataset*) como para el de pruebas (*test dataset*) se muestra en la FIGURA 16. CURVAS DE APRENDIZAJE. Se puede ver que coinciden tanto el training score como el *cross-validation score*, este último se refiere al *accuracy* obtenido tras validar el modelo contra el *test dataset*, siendo la exactitud del 95%.

Esto quiere decir que ambos casos, la introducción de más peticiones HTTP, no ha incrementado el *accuracy*, pero tampoco ha empeorado, se ha mantenido estable a un valor alto.

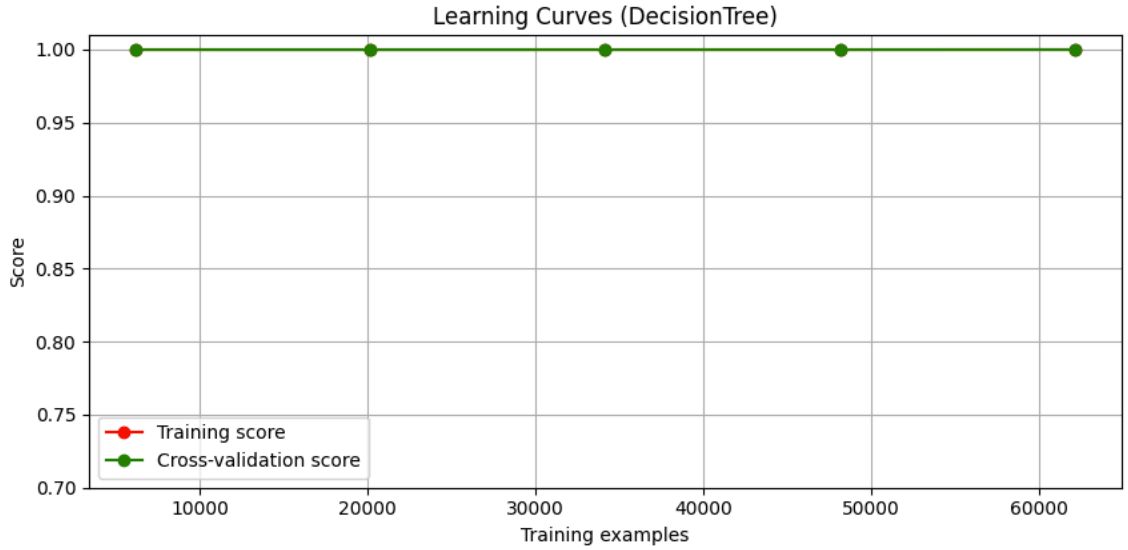

Figura 16. Curvas de aprendizaje

El conjunto de datos de pruebas (*testing dataset*) contiene un total de 9707 peticiones HTTP. El modelo ha sido capaz de clasificarlas todas correctamente.

Calculada la matriz de confusión [50], se observa que 7129 peticiones son verdaderos negativos, es decir, peticiones normales, lo que supone un 73,44% del total del conjunto de pruebas. El restante 26,56%, unas 2578 peticiones, son consideradas por el modelo como verdaderos positivos, es decir, anómalas.

Lo anterior se encuentra detallado en la salida que proporciona la ejecución del modelo:

```
    Total Instances: 9707
              Correctly classified:   9707 (100.000%)
              Incorrectly classified: 0 (0.000%)

    Confusion Matrix values:
              True Negatives  (TN):   7129 (73.442%)
              False Positives (FP):   0 (0.000%)
              False Negatives (FN):   0 (0.000%)
              True Positives  (TP):   2578 (26.558%)
              (Negative='Normal request', Positive='Anomalous request')
```

El informe de *scoring* de las *métricas precision, recall, f1-score* y *accuracy* que se han calculado con *scikit-learn* es el siguiente:

```
    Scoring report:

                precision    recall  f1-score   support

      normal       1.00      1.00      1.00      2578
   anomalous       1.00      1.00      1.00      7129

    accuracy                           1.00      9707
   macro avg       1.00      1.00      1.00      9707
weighted avg       1.00      1.00      1.00      9707
```

Se puede comprobar como el modelo propuesto da el mejor resultado para cada una de las métricas indicadas y explicadas en la TABLA 4. MÉTRICAS CALCULADAS.

Por último, se muestra en la FIGURA 17 el árbol de decisión obtenido con el modelo propuesto para el conjunto de datos del *test dataset*.

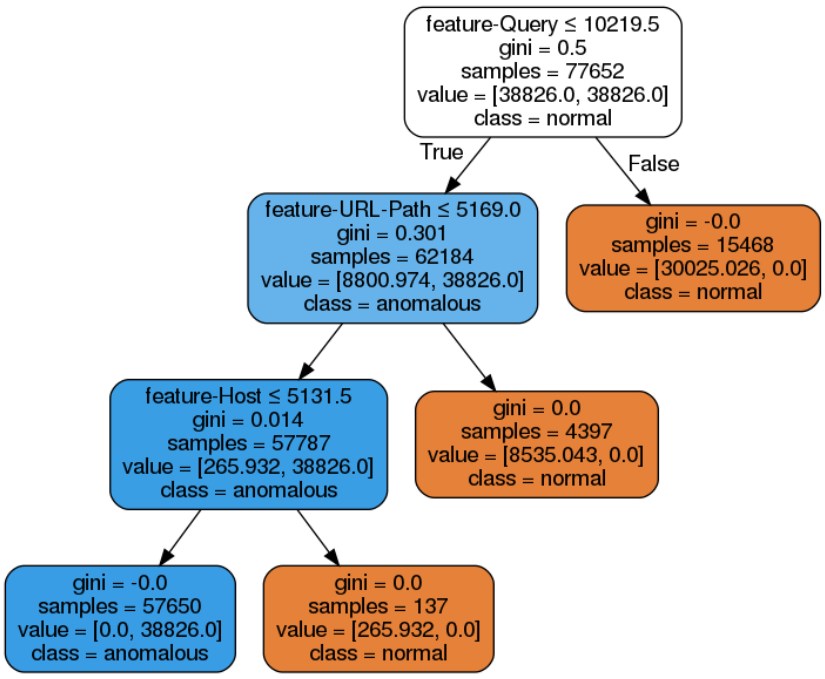

Figura 17. Árbol de decisión del modelo propuesto

5 Conclusiones y trabajo futuro

5.1 Conclusiones

Los objetivos marcados al comienzo de este TFM han sido alcanzados.

- Se ha desarrollado un modelo predictivo de *machine learning* con aprendizaje supervisado para la detección de amenazas web mediante clasificación basada en árboles de decisión que ha permitido clasificar un conjunto de peticiones HTTP en peticiones normales y anómalas, con un 100% de exactitud.
- Se ha adquirido conocimiento sobre las diferentes técnicas y algoritmos de *machine learning* utilizados actualmente para resolver problemas en el ámbito de la seguridad de las TIC.
- Se ha realizado un estudio de cada fase del desarrollo y ejecución de un modelo predictivo basado en árbol de decisión: selección de características, selección y particionado de *datasets*, selección de hiperparámetros y comprensión de las distintas métricas que produce la ejecución del modelo.

De los resultados obtenidos por el modelo propuesto, se puede aseverar que la elección del árbol de decisión ha sido buena, teniendo en cuanto las métricas de *scoring* obtenidas. También se ha observado como una selección automática de características realizado con algoritmos de correlación ha obtenido mejores resultados que con la selección manual atendiendo a criterios de experiencia en la seguridad TIC y sistemas web en particular. Esto nos hace reflexionar sobre que la intuición y experiencia puede ser un grado, pero el método científico, basado en este caso en la estadística, prevalece sobre lo anterior.

En conclusión, el TFM ha servido para conocer la disciplina del aprendizaje automático, lo aprendido ha servido para aplicar correctamente las diferentes técnicas y se han logrado buenos resultados con el modelo predictivo propuesto. Todo ello, sin desviaciones del plan de trabajo marcado al inicio.

5.2 Trabajo futuro

Algunas líneas de trabajo sobre las que se podrían seguir trabajando podrían ser:

- Comprobación del modelo sobre otros *datasets,* previo preprocesado, para hacerlos compatibles con el modelo propuesto. Sería interesante comparar las nuevas métricas obtenidas para así poder analizar la adaptabilidad del modelo propuesto a otros tipos de peticiones HTTP.
- Integración del modelo implementado en algún IDS conocido a modo de extensión. Por ejemplo. Suricata permite añadir reglas propias de detección de ataques (peticiones HTTP maliciosas) mediante *Lua scripting (https://suricata.readthedocs.io/en/suricata-5.0.2/rules/rule-lua-scripting.html#lua-scripting)*,.

6 Referencias

[1] Centro Criptológico Nacional, «CCN-CERT IA-13/19 Ciberamenazas y Tendencias. Resumen Ejecutivo 2019». jun. 10, 2019, [En línea]. Disponible en: https://www.ccn-cert.cni.es/informes/informes-ccn-cert-publicos/3767-ccn-cert-ia-13-19-ciberamenazas-y-tendencias-resumen-ejecutivo-2019/file.html.

[2] R. Sagar, R. Jhaveri, y C. Borrego, «Applications in Security and Evasions in Machine Learning: A Survey», *Electronics*, vol. 9, n.º 1, p. 97, ene. 2020, doi: 10.3390/electronics9010097.

[3] R. Rangadurai Karthick, V. P. Hattiwale, y B. Ravindran, «Adaptive network intrusion detection system using a hybrid approach», en *2012 Fourth International Conference on Communication Systems and Networks (COMSNETS 2012)*, ene. 2012, pp. 1-7, doi: 10.1109/COMSNETS.2012.6151345.

[4] E. K. Viegas, A. O. Santin, y L. S. Oliveira, «Toward a reliable anomaly-based intrusion detection in real-world environments», *Computer Networks*, vol. 127, pp. 200-216, nov. 2017, doi: 10.1016/j.comnet.2017.08.013.

[5] K. Ghanem, F. J. Aparicio-Navarro, K. G. Kyriakopoulos, S. Lambotharan, y J. A. Chambers, «Support Vector Machine for Network Intrusion and Cyber-Attack Detection», en *2017 Sensor Signal Processing for Defence Conference (SSPD)*, dic. 2017, pp. 1-5, doi: 10.1109/SSPD.2017.8233268.

[6] W. Laftah Al-Yaseen, Z. Ali Othman, y M. Z. Ahmad Nazri, «Hybrid Modified -Means with C4.5 for Intrusion Detection Systems in Multiagent Systems», *The Scientific World Journal*, 2015. https://www.hindawi.com/journals/tswj/2015/294761/ (accedido mar. 17, 2020).

[7] V. Timčenko y S. Gajin, «Ensemble classifiers for supervised anomaly based network intrusion detection», en *2017 13th IEEE International Conference on Intelligent Computer Communication and Processing (ICCP)*, sep. 2017, pp. 13-19, doi: 10.1109/ICCP.2017.8116977.

[8] «¿Qué es el Phishing? - Panda Security». / (accedido mar. 18, 2020).

[9] I. R. A. Hamid y J. H. Abawajy, «An approach for profiling phishing activities», *Computers & Security*, vol. 45, pp. 27-41, sep. 2014, doi: 10.1016/j.cose.2014.04.002.

[10] R. B. Basnet, A. H. Sung, y Q. Liu, «Feature Selection for Improved Phishing Detection», en *Advanced Research in Applied Artificial Intelligence*, vol. 7345, H. Jiang, W. Ding, M. Ali, y X. Wu, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 252-261.

[11] Y. Li, R. Xiao, J. Feng, y L. Zhao, «A semi-supervised learning approach for detection of phishing webpages», *Optik*, vol. 124, n.º 23, pp. 6027-6033, dic. 2013, doi: 10.1016/j.ijleo.2013.04.078.

[12] Q. Jia, L. Guo, Z. Jin, y Y. Fang, «Preserving Model Privacy for Machine Learning in Distributed Systems», *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, n.º 8, pp. 1808-1822, ago. 2018, doi: 10.1109/TPDS.2018.2809624.

[13] G. Jagannathan y R. N. Wright, «Privacy-preserving distributed k-means clustering over arbitrarily partitioned data», en *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, Chicago, Illinois, USA, ago. 2005, pp. 593–599, doi: 10.1145/1081870.1081942.

[14] N. Sutta, Z. Liu, y X. Zhang, «A Study of Machine Learning Algorithms on Email Spam Classification», en *EPiC Series in Computing*, mar. 2020, vol. 69, pp. 170-179, doi: 10.29007/qshd.

[15] C. Chen *et al.*, «A Performance Evaluation of Machine Learning-Based Streaming Spam Tweets Detection», *IEEE Transactions on Computational Social Systems*, vol. 2, n.º 3, pp. 65-76, sep. 2015, doi: 10.1109/TCSS.2016.2516039.

[16] M. Eminagaoglu, «A Qualitative Information Security Risk Assessment Model using Machine Learning Techniques», p. 7, 2012.

[17] S. Das, Y. Liu, W. Zhang, y M. Chandramohan, «Semantics-Based Online Malware Detection: Towards Efficient Real-Time Protection Against Malware», *IEEE Transactions on Information Forensics and Security*, vol. 11, n.º 2, pp. 289-302, feb. 2016, doi: 10.1109/TIFS.2015.2491300.

[18] G. Shu y D. Lee, «Testing Security Properties of Protocol Implementations - a Machine Learning Based Approach», en *27th International Conference on Distributed Computing Systems (ICDCS '07)*, jun. 2007, pp. 25-25, doi: 10.1109/ICDCS.2007.147.

[19] «Welcome to Python.org». https://www.python.org/ (accedido mar. 19, 2020).

[20] F. Pedregosa *et al.*, «Scikit-learn: Machine Learning in Python», *Journal of Machine Learning Research*, vol. 12, p. 2825–2830, oct. 2011.

[21] J. Brownlee, «A Tour of Machine Learning Algorithms», *Machine Learning Mastery*, ago. 11, 2019. https://machinelearningmastery.com/a-tour-of-machine-learning-algorithms/ (accedido mar. 19, 2020).

[22] «Algoritmos de Machine Learning | Conoce cuales son sus potencialidades», *GraphEverywhere*, sep. 26, 2019. https://www.grapheverywhere.com/algoritmos-de-machine-learning/ (accedido mar. 21, 2020).

[23] Z. A. Almaliki, «Do you know how to choose the right machine learning algorithm among 7 different types?», *Medium*, jul. 09, 2019. https://towardsdatascience.com/do-you-know-how-to-choose-the-right-machine-learning-algorithm-among-7-different-types-295d0b0c7f60 (accedido mar. 23, 2020).

[24] «Choosing the Right Machine Learning Algorithm». https://hackernoon.com/choosing-the-right-machine-learning-algorithm-68126944ce1f (accedido mar. 23, 2020).

[25] M. Almseidin, M. Alzubi, K. Szilveszter, y M. Al-kasassbeh, «Evaluation of Machine Learning Algorithms for Intrusion Detection System», ene. 2018.

[26] «C4.5», *Wikipedia, la enciclopedia libre*. feb. 08, 2020, Accedido: mar. 23, 2020. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=C4.5&oldid=123402893.

[27] X. Bellekens *et al.*, «A Taxonomy and Survey of Intrusion Detection System Design Techniques, Network Threats and Datasets», jun. 2018.

[28] I. Sharafaldin, A. Gharib, A. H. Lashkari, y A. A. Ghorbani, «Towards a Reliable Intrusion Detection Benchmark Dataset», *Software Networking*, vol. 2018, n.º 1, pp. 177-200, ene. 2018, doi: 10.13052/jsn2445-9739.2017.009.

[29] «Latest labeled databases available for intrusion detection?», *ResearchGate*. https://www.researchgate.net/post/latest_labeled_databases_available_for_intrusion_detection (accedido mar. 24, 2020).

[30] «Datasets | Research | Canadian Institute for Cybersecurity | UNB». https://www.unb.ca/cic/datasets/index.html (accedido mar. 24, 2020).

[31] «A Realistic Cyber Defense Dataset (CSE-CIC-IDS2018) - Registry of Open Data on AWS». https://registry.opendata.aws/cse-cic-ids2018/ (accedido mar. 24, 2020).

[32] «GSI / web-application-attacks-datasets», *GitLab*. https://gitlab.fing.edu.uy/gsi/web-application-attacks-datasets (accedido mar. 24, 2020).

[33] «Dataset Search». https://datasetsearch.research.google.com/ (accedido mar. 24, 2020).

[34] «Kaggle: Your Machine Learning and Data Science Community». https://www.kaggle.com/ (accedido mar. 24, 2020).

[35] «UCI Machine Learning Repository». http://mlr.cs.umass.edu/ml/ (accedido mar. 24, 2020).

[36] «HTTP DATASET CSIC 2010». https://www.isi.csic.es/dataset/ (accedido mar. 24, 2020).

[37] «Attack Challenge - ECML/PKDD Workshop». http://www.lirmm.fr/pkdd2007-challenge/index.html (accedido mar. 24, 2020).

[38] «PEP 8 -- Style Guide for Python Code», *Python.org*. https://www.python.org/dev/peps/pep-0008/ (accedido abr. 25, 2020).

[39] «HTTP DATASET CSIC 2010». https://www.isi.csic.es/dataset/ (accedido mar. 24, 2020).

[40] V. A. Khan, *vaseem-khan/URLcheck*. 2020.

[41] M. A. Hall, «Correlation-based feature selection for machine learning», 1999.

[42] «seaborn.heatmap — seaborn 0.10.0 documentation». https://seaborn.pydata.org/generated/seaborn.heatmap.html (accedido abr. 25, 2020).

[43] «sklearn.model_selection.GridSearchCV — scikit-learn 0.22.2 documentation». https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html (accedido abr. 25, 2020).

[44] «Cross-Validation in Machine Learning - Towards Data Science». https://towardsdatascience.com/cross-validation-in-machine-learning-72924a69872f (accedido abr. 25, 2020).

[45] M. Mithrakumar, «How to tune a Decision Tree?», *Medium*, nov. 12, 2019. https://towardsdatascience.com/how-to-tune-a-decision-tree-f03721801680 (accedido abr. 24, 2020).

[46] R. G. Mantovani, T. Horváth, R. Cerri, S. B. Junior, J. Vanschoren, y A. C. P. de L. F. de Carvalho, «An empirical study on hyperparameter tuning of decision trees», *arXiv:1812.02207 [cs, stat]*, feb. 2019, Accedido: abr. 24, 2020. [En línea]. Disponible en: http://arxiv.org/abs/1812.02207.

[47] «3.5. Validation curves: plotting scores to evaluate models — scikit-learn 0.22.2 documentation». https://scikit-learn.org/stable/modules/learning_curve.html#learning-curve (accedido abr. 25, 2020).

[48] R. Agarwal, «The 5 Classification Evaluation metrics every Data Scientist must know», *Medium*, nov. 26, 2019. https://towardsdatascience.com/the-5-classification-evaluation-metrics-you-must-know-aa97784ff226 (accedido abr. 25, 2020).

[49] «Plotting Learning Curves — scikit-learn 0.22.2 documentation». https://scikit-learn.org/stable/auto_examples/model_selection/plot_learning_curve.html (accedido abr. 25, 2020).

[50] «sklearn.metrics.confusion_matrix — scikit-learn 0.22.2 documentation». https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html (accedido abr. 25, 2020).

[51] CUNHA, I. da, 2016. El trabajo de fin de grado y de máster: redacción, defensa y publicación. Barcelona: Editorial UOC. ISBN 978-84-9064-391-4.

ANEXO I: Código Fuente

```
001: #!/usr/bin/env python
002: # -*- coding: utf-8 -*-
003:
004: import datetime
005: import os
006: import ssl
007: import sys
008: import csv
009: import glob
010: import pandas as pd
011: import matplotlib.pyplot as plt
012: import numpy as np
013: import seaborn as sns
014: import skfeature
015: import re
016: import pydotplus
017: from IPython.display import Image
018: from pathlib import Path
019: from pyunpack import Archive
020: from six.moves import urllib
021: from skfeature.function.statistical_based import CFS
022: from sklearn.utils import resample
023: from sklearn import metrics
024: from sklearn import preprocessing
025: from sklearn import svm
026: from sklearn.ensemble import RandomForestClassifier
027: from sklearn.ensemble import AdaBoostClassifier
028: from sklearn.feature_selection import SelectKBest
029: from sklearn.feature_selection import SelectFdr
030: from sklearn.feature_selection import chi2
031: from sklearn.feature_selection import f_classif
032: from sklearn.feature_selection import VarianceThreshold
033: from sklearn.naive_bayes import GaussianNB
034: from sklearn.model_selection import cross_validate
035: from sklearn.model_selection import GridSearchCV
036: from sklearn.model_selection import learning_curve
037: from sklearn.model_selection import validation_curve
038: from sklearn.model_selection import train_test_split
039: from sklearn.preprocessing import LabelEncoder
040: from sklearn.tree import export_graphviz
041: from sklearn.tree import DecisionTreeClassifier
042: from six import StringIO
043: from time import time
044: from urllib.parse import urlparse
045: from urllib import parse
046:
047: BASE_DIR = '/home/jose/uoc/tfm/'
048: DOWNLOAD_ROOT = "https://www.isi.csic.es/dataset/"
049: HTTPTRAFFIC_RAW_PATH = os.path.join("datasets/raw", "http-traffic/")
050: HTTPTRAFFIC_PROCESSED_PATH = os.path.join(
051:     "datasets/processed", "http-traffic/")
052: TRAINING_NORMAL_HTTPTRAFFIC_URL = DOWNLOAD_ROOT + "normalTrafficTraining.rar"
053: TEST_NORMAL_HTTPTRAFFIC_URL = DOWNLOAD_ROOT + "normalTrafficTest.rar"
054: TEST_ANOMALOUS_HTTPTRAFFIC_URL = DOWNLOAD_ROOT + "anomalousTrafficTest.rar"
055:
056: # hack for CSIC SSL certificate
057: ssl._create_default_https_context = ssl._create_unverified_context
058: # end of hack
059:
060:
061: def fetch_httpTraffic_data(httpTraffic_path=HTTPTRAFFIC_RAW_PATH,
062:                            trainingNormalTraffic_url=TRAINING_NORMAL_HTTPTRAFFIC_URL,
063:                            testNormalTraffic_url=TEST_NORMAL_HTTPTRAFFIC_URL,
064:                            testAnomalousTraffic_url=TEST_ANOMALOUS_HTTPTRAFFIC_URL):
065:     if not os.path.isdir(httpTraffic_path):
066:         os.makedirs(httpTraffic_path)
067:
068:     rar_trainingNormalTraffic_path = os.path.join(
069:         httpTraffic_path, "normalTrafficTraining.rar")
070:     rar_testNormalTraffic_path = os.path.join(
071:         httpTraffic_path, "normalTrafficTest.rar")
072:     rar_testAnomalousTraffic_path = os.path.join(
073:         httpTraffic_path, "anomalousTrafficTest.rar")
074:
075:     urllib.request.urlretrieve(
076:         trainingNormalTraffic_url, rar_trainingNormalTraffic_path)
077:     Archive(rar_trainingNormalTraffic_path).extractall(httpTraffic_path)
078:     os.remove(rar_trainingNormalTraffic_path)
079:
080:     urllib.request.urlretrieve(
081:         testNormalTraffic_url, rar_testNormalTraffic_path)
082:     Archive(rar_testNormalTraffic_path).extractall(httpTraffic_path)
```

```
    os.remove(rar_testNormalTraffic_path)

    urllib.request.urlretrieve(
        testAnomalousTraffic_url, rar_testAnomalousTraffic_path)
    Archive(rar_testAnomalousTraffic_path).extractall(httpTraffic_path)
    os.remove(rar_testAnomalousTraffic_path)


def parse_datafile(dataset_filepath, csv_filepath, label):
    data = []  # create an empty list to collect the data
    method_is_get = False
    method_is_post_or_put = False
    if not os.path.isdir(os.path.dirname(os.path.abspath(csv_filepath))):
        os.makedirs(os.path.dirname(os.path.abspath(csv_filepath)))

    # read all the txt file and save the dataset in the list data[]
    with open(dataset_filepath, 'r') as file_object:
        lines = filter(None, (line.rstrip()
                              for line in file_object))  # removes blank lines
        try:
            line = next(lines)
        except StopIteration as e:
            print(e)
        while line:
            row = []
            if line.startswith('GET'):
                method_is_get = True
                method_is_post = False
                method_is_put = False
            elif line.startswith('POST'):
                method_is_get = False
                method_is_post = True
                method_is_put = False
            elif line.startswith('PUT'):
                method_is_get = False
                method_is_post = False
                method_is_put = True
            row.append(line.split(' ')[0])  # method
            url = line.split(' ')[1]
            row.append(url)  # url
            row.append(line.split(' ')[2])  # protocol
            row.append(next(lines).split('User-Agent: ', 1)[1])
            row.append(next(lines).split('Pragma: ', 1)[1])
            row.append(next(lines).split('Cache-control: ', 1)[1])
            if method_is_put:
                row.append(next(lines).split('Accept:', 1)[1])
            else:
                row.append(next(lines).split('Accept: ', 1)[1])
            row.append(next(lines).split('Accept-Encoding: ', 1)[1])
            row.append(next(lines).split('Accept-Charset: ', 1)[1])
            row.append(next(lines).split('Accept-Language: ', 1)[1])
            row.append(next(lines).split('Host: ', 1)[1])
            row.append(next(lines).split('Cookie: ', 1)[1])
            if method_is_get:
                connection = next(lines).split('Connection: ', 1)[1]
                # content_type not applicable to GET method
                row.append("notpresent")
                # content_length not applicable to GET method
                row.append("notpresent")
                url_get_parameters = "notpresent" if not urlparse(
                    url).query else urlparse(url).query
                row.append(url_get_parameters)  # GET url_parameters
            elif method_is_post or method_is_put:
                content_type = next(lines).split('Content-Type: ', 1)[1]
                connection = next(lines).split('Connection: ', 1)[1]
                content_length = next(lines).split('Content-Length: ', 1)[1]
                url_parameters = next(lines)
                row.append(content_type)
                row.append(content_length)
                row.append(url_parameters)
            row.append(connection)
            row.append(label)
            data.append(row)
            try:
                line = next(lines)
            except StopIteration as e:
                print(e)
                break
    # save the data[] list into a csv file adding the columns names
    with open(csv_filepath, 'w', newline='') as csv_file:
        writer = csv.writer(csv_file, delimiter='|')
        # write headers
        writer.writerow(['feature-Method', 'feature-URL', 'feature-Protocol',
                         'feature-User-Agent', 'feature-Pragma', 'feature-Cache-Control',
                         'feature-Accept', 'feature-Accept-Encoding', 'feature-Accept-Charset',
                         'feature-Accept-Language', 'feature-Host', 'feature-Cookie',
                         'feature-Content-Type', 'feature-Content-Length', 'feature-Query',
                         'feature-Connection', 'Label'])
        # write data to the csv
```

```
172:         writer.writerows(data)
173:
174:
175: def correlation_matrix_heatmap(full_dataset, ouput_filename):
176:     data = pd.read_csv(full_dataset, sep='|')
177:     data['Label'], uniques = pd.factorize(
178:         data.iloc[:, -1])  # target column 'Label'
179:     # get correlations of each features in dataset
180:     corrmat = data.corr()
181:     top_corr_features = corrmat.index
182:     plt.figure(figsize=(20, 20))
183:     # plot heat map
184:     g = sns.heatmap(data[top_corr_features].corr(), annot=True, cmap="RdYlGn")
185:     plt.savefig(BASE_DIR + 'images/' + ouput_filename)
186:
187:
188: def feature_importances_graph(X, importances, output_filename):
189:     # Sort feature importances in descending order
190:     indices = np.argsort(importances)[::-1]
191:     # Rearrange feature names so they match the sorted feature importances
192:     names = [X.columns[i] for i in indices]
193:     # Create plot
194:     plt.figure(figsize=(20, 20))
195:     # Create plot title
196:     plt.title("Feature Importance")
197:     # Add bars
198:     plt.bar(range(X.shape[1]), importances[indices])
199:     # Add feature names as x-axis labels
200:     plt.xticks(range(X.shape[1]), names, rotation=45)
201:     # save image
202:     plt.savefig(BASE_DIR + 'images/' + output_filename)
203:
204:
205: def plot_learning_curve(estimator, title, output_filename, X, y, ylim=None, cv=None,
206:                         n_jobs=None, train_sizes=np.linspace(.1, 1.0, 5)):
207:     '''
208:     Generate 3 plots:
209:         (1) the test and training learning curve,
210:         (2) the training samples vs fit times curve, and
211:         (3) the fit times vs score curve.
212:     '''
213:     # Encoding of y
214:     labelencoder = LabelEncoder()
215:     y = labelencoder.fit_transform(y)
216:     fig, axes = plt.subplots(3, 1, figsize=(10, 15))
217:     if axes is None:
218:         _, axes = plt.subplots(1, 3, figsize=(20, 5))
219:     axes[0].set_title(title)
220:     if ylim is not None:
221:         axes[0].set_ylim(*ylim)
222:     axes[0].set_xlabel("Training examples")
223:     axes[0].set_ylabel("Score")
224:     train_sizes, train_scores, test_scores, fit_times, _ = learning_curve(
225:         estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes, return_times=True)
226:     train_scores_mean = np.mean(train_scores, axis=1)
227:     train_scores_std = np.std(train_scores, axis=1)
228:     test_scores_mean = np.mean(test_scores, axis=1)
229:     test_scores_std = np.std(test_scores, axis=1)
230:     fit_times_mean = np.mean(fit_times, axis=1)
231:     fit_times_std = np.std(fit_times, axis=1)
232:     # Plot learning curve
233:     axes[0].grid()
234:     axes[0].fill_between(train_sizes, train_scores_mean - train_scores_std,
235:                          train_scores_mean + train_scores_std, alpha=0.1,
236:                          color="r")
237:     axes[0].fill_between(train_sizes, test_scores_mean - test_scores_std,
238:                          test_scores_mean + test_scores_std, alpha=0.1,
239:                          color="g")
240:     axes[0].plot(train_sizes, train_scores_mean, 'o-', color="r",
241:                  label="Training score")
242:     axes[0].plot(train_sizes, test_scores_mean, 'o-', color="g",
243:                  label="Cross-validation score")
244:     axes[0].legend(loc="best")
245:     # Plot n_samples vs fit_times
246:     axes[1].grid()
247:     axes[1].plot(train_sizes, fit_times_mean, 'o-')
248:     axes[1].fill_between(train_sizes, fit_times_mean - fit_times_std,
249:                          fit_times_mean + fit_times_std, alpha=0.1)
250:     axes[1].set_xlabel("Training examples")
251:     axes[1].set_ylabel("fit_times")
252:     axes[1].set_title("Scalability of the model")
253:     # Plot fit_time vs score
254:     axes[2].grid()
255:     axes[2].plot(fit_times_mean, test_scores_mean, 'o-')
256:     axes[2].fill_between(fit_times_mean, test_scores_mean - test_scores_std,
257:                          test_scores_mean + test_scores_std, alpha=0.1)
258:     axes[2].set_xlabel("fit_times")
259:     axes[2].set_ylabel("Score")
260:     axes[2].set_title("Performance of the model")
```

```
261:     plt.savefig(BASE_DIR + 'images/' + output_filename)
262: 
263: 
264: def plot_validation_curve(estimator, title, output_filename, X, y, param_name, param_range):
265:     labelencoder = LabelEncoder()
266:     y = labelencoder.fit_transform(y)
267:     plt.figure(figsize=(10, 5))
268:     # Calculate accuracy on training and test set using range of parameter values
269:     train_scores, test_scores = validation_curve(
270:         estimator, X, y, param_name=param_name, param_range=param_range, cv=5, scoring="accuracy")
271:     # Calculate mean and standard deviation for training set scores
272:     train_mean = np.mean(train_scores, axis=1)
273:     train_std = np.std(train_scores, axis=1)
274:     # Calculate mean and standard deviation for test set scores
275:     test_mean = np.mean(test_scores, axis=1)
276:     test_std = np.std(test_scores, axis=1)
277:     # Plot mean accuracy scores for training and test sets
278:     plt.plot(param_range, train_mean,
279:              label="Training score", color="darkorange")
280:     plt.plot(param_range, test_mean,
281:              label="Cross-validation score", color="navy")
282:     # Plot accurancy bands for training and test sets
283:     plt.fill_between(param_range, train_mean - train_std,
284:                      train_mean + train_std, color="gray")
285:     plt.fill_between(param_range, test_mean - test_std,
286:                      test_mean + test_std, color="gainsboro")
287:     # Create plot
288:     plt.title(title)
289:     plt.xlabel(param_name)
290:     plt.ylabel("Accuracy")
291:     plt.tight_layout()
292:     plt.legend(loc="best")
293:     plt.savefig(BASE_DIR + 'images/' + output_filename)
294: 
295: 
296: def train_test_validation_split(full_dataset, train_size, test_size, validation_size, dir_name, verbose=True):
297:     proportion = 1 - validation_size
298:     # set the seed to fixed number in order to sample always the same datesets
299:     np.random.seed(0)
300:     total_rows_dataset = full_dataset.shape[0]
301:     if verbose:
302:         print("Splitting the dataset into train, test and validation sets...")
303:     train_set, validation_set, test_set = np.split(full_dataset.sample(frac=1), [int(
304:         train_size * len(full_dataset)), int(proportion * len(full_dataset))])
305:     train_set = train_set.sample(frac=1).reset_index(drop=True)
306:     test_set = test_set.sample(frac=1).reset_index(drop=True)
307:     validation_set = validation_set.sample(frac=1).reset_index(drop=True)
308:     X_train = train_set.loc[:, train_set.columns != 'Label']
309:     y_train = train_set['Label']
310:     X_test = test_set.loc[:, test_set.columns != 'Label']
311:     y_test = test_set['Label']
312:     X_validation = validation_set.loc[:, validation_set.columns != 'Label']
313:     y_validation = validation_set['Label']
314:     if verbose:
315:         print("    - train-set: " +
316:               str(int(train_set.shape[0] * 100 / total_rows_dataset)) + "%" + "  (" + str(train_set.shape[0]) + "
    samples)")
317:         print("    - test-set: " + str(int(test_set.shape[0] * 100 / total_rows_dataset)
318:                                        ) + "%" + "  (" + str(test_set.shape[0]) + " samples)")
319:         print("    - validation-set: " +
320:               str(int(validation_set.shape[0] * 100 / total_rows_dataset)) + "%" + "  (" +
    str(validation_set.shape[0]) + " samples)")
321:     train_set.to_csv(
322:         BASE_DIR + 'datasets/' + dir_name + 'train_set.csv', sep='|', index=False)
323:     test_set.to_csv(
324:         BASE_DIR + 'datasets/' + dir_name + 'test_set.csv', sep='|', index=False)
325:     validation_set.to_csv(
326:         BASE_DIR + 'datasets/' + dir_name + 'validation_set.csv', sep='|', index=False)
327:     return X_train, y_train, X_test, y_test, X_validation, y_validation
328: 
329: 
330: def path_from_URL(url):
331:     return urlparse(url).path
332: 
333: 
334: def count_params(txt):
335:     """Return number of parameters."""
336:     params = parse.parse_qs(txt)
337:     if not params:  # query has no any param
338:         return 'notpresent'
339:     else:
340:         return len(params)
341: 
342: 
343: def count_dots(txt):
344:     """Return the amount of "." in the text."""
345:     return txt.count(".")
346: 
347: 
```

```
348: def count_percentage(txt):
349:     """Return the amount of "%" in the text."""
350:     return txt.count("%")
351:
352:
353: def count_specialChars(txt):
354:     """Return the amount of special characters in the text."""
355:     return len(re.sub('[\w]+', '', txt))
356:
357:
358: def extract_label_values(lst):
359:     """Return a list with the last item of every list in lst"""
360:     return [item[-1] for item in lst]
361:
362:
363: def get_longest_param_value(query):
364:     dict = parse.parse_qs(query)
365:     all_values = dict.values()
366:     all_values_list = [item for elem in all_values for item in elem]
367:     if len(all_values_list):
368:         return len(max(all_values_list, key=len))
369:     else:
370:         return 0
371:
372:
373: def name_params(query):
374:     """ Return a string with the ordered concatenated list of parameters' names"""
375:     params_list = parse.parse_qs(query)
376:     if not params_list:  # query has no any param
377:         return 'notpresent'
378:     else:
379:         params_list_sorted = sorted(list(params_list))
380:         return '-'.join(params_list_sorted)
381:
382:
383: def get_resource_extension(URL_path):
384:     """Return the extension file of the URL resource or 0 if other case"""
385:     resource_extension = Path(URL_path).suffix
386:     if resource_extension:
387:         return resource_extension
388:     else:
389:         return 0
390:
391:
392: def get_resource_extension_length(URL_path):
393:     resource_extension = Path(URL_path).suffix
394:     if resource_extension:
395:         return len(resource_extension)
396:     else:
397:         return 0
398:
399:
400: def type_params(query):
401:     """Return a coded string describing the types of the params"""
402:     type_params_list = ''
403:     params = parse.parse_qs(query)
404:     if not params:  # query has no any param
405:         return 'notpresent'
406:     params_list = list(params.values())
407:     for p in params_list:
408:         if p[0].isdigit():
409:             type_params_list += str('[DIGIT]')
410:         else:
411:             type_params_list += str('[ALPHABET]')
412:     return type_params_list
413:
414:
415: def get_token_count(url):
416:     """Return the number of tokens. A token is a param and a param's value"""
417:     if url == '':
418:         return 0
419:     token_word = re.split('\W+', url)
420:     no_ele = 0
421:     for ele in token_word:
422:         l = len(ele)
423:         if l > 0:
424:             no_ele += 1
425:     return no_ele
426:
427:
428: def get_token_largest(url):
429:     """Return the length of the pargest token"""
430:     if url == '':
431:         return 0
432:     token_word = re.split('\W+', url)
433:     largest = 0
434:     for ele in token_word:
435:         l = len(ele)
436:         if largest < l:
```

```
437:             largest = l
438:     return largest
439:
440:
441: def get_token_avg(url):
442:     """Return the average lengths of all the token in the URL"""
443:     if url == '':
444:         return 0
445:     token_word = re.split('\W+', url)
446:     no_ele = sum_len = 0
447:     for ele in token_word:
448:         l = len(ele)
449:         sum_len += l
450:         if l > 0:  # for empty element exclusion in average length
451:             no_ele += 1
452:     try:
453:         return float(sum_len) / no_ele
454:     except:
455:         return 0
456:
457:
458: def humanize_seconds(seconds):
459:     min, sec = divmod(seconds, 60)
460:     hour, min = divmod(min, 60)
461:     return "%dh:%02dm:%02ds" % (hour, min, sec)
462:
463:
464: def print_stats(tn, fp, fn, tp, report, times):
465:     total_instances = tn + fp + fn + tp
466:     correcly_classified = (tn + tp) / total_instances * 100
467:     incorrectly_classified = (fn + fp) / total_instances * 100
468:     print('\n\n RESULTS for DecisionTree
     classifier:\n================================================================================')
469:     print('\n\tTime:')
470:     print('\t\tgathering datasets:         %.3fs' % (times['gathering']))
471:     print('\t\tprocessing datasets:        %.3fs' % (times['processing']))
472:     print('\t\tselecting features:         %.3fs' % (times['features']))
473:     print('\t\tselecting hyper-parameters: %s' %
474:           (humanize_seconds(times['auto_hyperparameter'])))
475:     print('\t\ttraining the model:         %.3fs' % (times['training']))
476:     print('\t\tpredicting the test set:    %.3fs' % (times['predicting']))
477:     print('\t\tplotting graphs:            %s' %
478:           (humanize_seconds(times['graphs'])))
479:     print('\n\tTotal Instances: %i' % (total_instances))
480:     print('\t\tCorrectly classified:   %i (%.3f%%)' %
481:           (tp + tn, correcly_classified))
482:     print('\t\tIncorrectly classified: %i (%.3f%%)' %
483:           (fp + fn, incorrectly_classified))
484:     print('\n\tConfusion Matrix values:')
485:     print('\t\tTrue Negatives  (TN):   %i (%.3f%%)' %
486:           (tn, tn / (tn + tp + fn + fp) * 100))
487:     print('\t\tFalse Positives (FP):   %i (%.3f%%)' %
488:           (fp, fp / (tn + tp + fn + fp) * 100))
489:     print('\t\tFalse Negatives (FN):   %i (%.3f%%)' %
490:           (fn, fn / (tn + tp + fn + fp) * 100))
491:     print('\t\tTrue Positives  (TP):   %i (%.3f%%)' %
492:           (tp, tp / (tn + tp + fn + fp) * 100))
493:     print('\t\t(Negative=\'Normal request\', Positive=\'Anomalous request\')')
494:     print('\n')
495:     print('\tScoring report:\n')
496:     print(report)
497:
498:
499: def auto_feature_selection(X_validation, y_validation):
500:     # to decrease the time elapsed by CFS, autopreselection of features by Variance and SelectKBest
501:     sel = VarianceThreshold(threshold=(.6 * (1 - .6)))
502:     sel.fit(X_validation)
503:     X_validation = X_validation[X_validation.columns[sel.get_support(
504:         indices=True)]]
505:     sel2 = SelectKBest(chi2, k=10).fit(X_validation, y_validation)
506:     X_validation = X_validation[X_validation.columns[sel2.get_support(
507:         indices=True)]]
508:     # get all the headers (features) in the dataframe
509:     features = list(X_validation.columns.values)
510:     X_validation_np = X_validation.to_numpy()
511:     y_validation_np = y_validation.to_numpy()
512:     n_samples = X_validation_np.shape[0]
513:     print("\n\nAuto selected features by Correlation Feature Selection (CFS) method:")
514:     # Aplication of CFS algorithm to select the best features
515:     features_selected_idx = CFS.cfs(X_validation_np, y_validation_np)
516:     selected_features = []
517:     for idx in features_selected_idx:
518:         # cfs returns a 6 elements list, if less than 6 features are selected, -1 are used to complete the list
519:         if idx >= 0:
520:             selected_features.append(features[idx])
521:             print('    - %s' % features[idx])
522:     return selected_features
523:
524:
```

```
525: def auto_hyperparameters_selection(X_validation, y_validation):
526:     ''' Tuning the hyper-parameters of the DecisionTree '''
527:     criterion = ['gini', 'entropy']
528:     splitter = ['best', 'random']
529:     max_depth = np.arange(2, 6)
530:     min_samples_split = [2, 6, 13, 20, 27, 34, 40]  # best 2 to 40
531:     min_samples_leaf = [1, 5, 9, 13, 17, 20]  # best: 1 to 20
532:     class_weight = ['balanced', None]
533:     param_grid = {'criterion': criterion,
534:                   'splitter': splitter,
535:                   'min_samples_split': min_samples_split,
536:                   'max_depth': max_depth,
537:                   'min_samples_leaf': min_samples_leaf,
538:                   'class_weight': class_weight}
539:     print("\n\nAuto selected hyperparameters (DecisionTreeClassifier) by Exhaustive Grid Search: ")
540:     clf = GridSearchCV(DecisionTreeClassifier(), param_grid, cv=5)
541:     clf.fit(X_validation, y_validation)
542:     for idx in clf.best_params_:
543:         print(f'    - {idx}: {clf.best_params_[idx]}')
544:     return clf.best_params_
545:
546:
547: def create_tree_image(classifier, features, labels, filename):
548:     dotfile_data = StringIO()
549:     export_graphviz(classifier, out_file=dotfile_data,
550:                     filled=True, rounded=True,
551:                     special_characters=True, feature_names=features,
552:                     class_names=labels)
553:     graph = pydotplus.graph_from_dot_data(dotfile_data.getvalue())
554:     graph.write_png(BASE_DIR + 'images/' + filename)
555:     Image(graph.create_png())
556:
557:
558: def test_other_models(X_train, y_train, X_test, y_test, auto_selected_features, manual_selected_features):
559:     print('\n\n RESULTS for other
    models:\n================================================================================')
560:
561:     # testing a DecisionTree but with K-Fold Cross-Validation method
562:     classifier = DecisionTreeClassifier(criterion='entropy', max_depth=5)
563:     cv_results = cross_validate(
564:         estimator=classifier, X=X_train[auto_selected_features], y=y_train, cv=10, return_estimator=True,
    return_train_score=True)
565:     # print(sorted(cv_results.keys()))
566:     best_test_score, idx = max((val, idx) for (
567:         idx, val) in enumerate(cv_results['test_score']))
568:     best_estimator = cv_results['estimator'][idx]
569:     print("\n\tAccuracy for DecisionTree model with k-fold cross-validation training: %.2f" %
570:           best_test_score)
571:     create_tree_image(classifier=best_estimator, features=auto_selected_features, labels=[
572:                       'normal', 'anomalous'], filename='http-traffic-tree-kfold.png')
573:
574:     # testing a DecisionTree but with manually selected features
575:     clf = DecisionTreeClassifier(criterion='entropy', max_depth=5)
576:     clf = clf.fit(X_train[manual_selected_features], y_train)
577:     predictor = clf.predict(X_test[manual_selected_features])
578:     print("\tAccuracy for DecisionTree model with manually selected features: %.2f"
579:           % metrics.accuracy_score(y_test, predictor))
580:     create_tree_image(classifier=clf, features=manual_selected_features, labels=[
581:                       'normal', 'anomalous'], filename='http-traffic-tree-manual-features.png')
582:
583:     # testing a RandomForest model
584:     clf = RandomForestClassifier(n_estimators=10)
585:     clf = clf.fit(X_train[auto_selected_features], y_train)
586:     predictor = clf.predict(X_test[auto_selected_features])
587:     print("\tAccuracy for RandomForest model: %.2f"
588:           % metrics.accuracy_score(y_test, predictor))
589:
590:     # testing an AdaBoost model
591:     clf = AdaBoostClassifier(n_estimators=100)
592:     clf = clf.fit(X_train[auto_selected_features], y_train)
593:     predictor = clf.predict(X_test[auto_selected_features])
594:     print("\tAccuracy for AdaBoost model: %.2f" %
595:           metrics.accuracy_score(y_test, predictor))
596:
597:     # testing an SVM model
598:     clf = svm.SVC()
599:     clf.fit(X_train[auto_selected_features], y_train)
600:     predictor = clf.predict(X_test[auto_selected_features])
601:     print("\tAccuracy for SVM model: %.2f" %
602:           metrics.accuracy_score(y_test, predictor))
603:
604:     # testing an Gaussian Naive Bayes model
605:     clf = GaussianNB()
606:     clf.fit(X_train[auto_selected_features], y_train)
607:     predictor = clf.predict(X_test[auto_selected_features])
608:     print("\tAccuracy for Gaussian Naive Bayes model: %.2f"
609:           % metrics.accuracy_score(y_test, predictor))
610:     print("\n\n")
611:
```

```
612:
613: def main():
614:     """ HTTP classifier """
615:     if not os.path.isdir('datasets'):
616:         sys.exit(
617:             "======\nERROR: The classifier must be run from the root project folder.\n======")
618:
619:     times = {
620:         "gathering":  0,
621:         "processing": 0,
622:         "features":   0,
623:         "training":   0,
624:         "predicting": 0
625:     }
626:
627:     '''
628:     ====== GATHERING & PARSING DATASETS ====================================
629:     '''
630:     t0 = time()
631:
632:     # dataset downloading
633:     f_normal_test_exists = os.path.exists(
634:         BASE_DIR + "datasets/raw/normalTrafficTest.txt")
635:     f_normal_train_exists = os.path.exists(
636:         BASE_DIR + "datasets/raw/normalTrafficTraining.txt")
637:     f_anom_test_exists = os.path.exists(
638:         BASE_DIR + "datasets/raw/anomalousTrafficTest.txt")
639:     if (f_normal_test_exists and f_normal_train_exists and f_anom_test_exists) == False:
640:         fetch_httpTraffic_data()
641:
642:     # parse the dataset file: normalTrafficTest
643:
644:     parse_datafile(BASE_DIR + "datasets/raw/normalTrafficTest.txt",
645:                    BASE_DIR + "datasets/processed/normalTrafficTest.csv",
646:                    "normal")
647:     # parse the dataset file: normalTrafficTraining
648:     parse_datafile(BASE_DIR + "datasets/raw/normalTrafficTraining.txt",
649:                    BASE_DIR + "datasets/processed/normalTrafficTraining.csv",
650:                    "normal")
651:     # parse the dataset file: anomalousTrafficTest
652:     parse_datafile(BASE_DIR + "datasets/raw/anomalousTrafficTest.txt",
653:                    BASE_DIR + "datasets/processed/anomalousTrafficTest.csv",
654:                    "anomalous")
655:     # combine all csv files in only one
656:     csv_files = [BASE_DIR + "datasets/processed/normalTrafficTest.csv",
657:                  BASE_DIR + "datasets/processed/normalTrafficTraining.csv",
658:                  BASE_DIR + "datasets/processed/anomalousTrafficTest.csv"]
659:     full_dataset_filename = BASE_DIR + "datasets/processed/full_dataset.csv"
660:     df_norm_test = pd.read_csv(csv_files[0], sep='|')
661:     df_norm_train = pd.read_csv(csv_files[1], sep='|')
662:     df_anom_test = pd.read_csv(csv_files[2], sep='|')
663:     df_full_sataset = pd.concat(
664:         [df_norm_test, df_norm_train, df_anom_test], axis=0, join='inner', ignore_index=True)
665:     df_full_sataset.reset_index()
666:     df_full_sataset.to_csv(full_dataset_filename, index=False, sep='|')
667:     for csv in csv_files:
668:         os.remove(csv)
669:
670:     times['gathering'] = time() - t0
671:
672:     '''
673:     ====== DATA PREPROCESSING ==============================================
674:     '''
675:     t0 = time()
676:     df = pd.read_csv(full_dataset_filename, sep='|')
677:     df['feature-Query-Length'] = np.where(df['feature-Query']
678:                                           == 'notpresent', 0, df['feature-Query'].str.len())
679:     df['feature-Query-Longest-Value'] = df['feature-Query'].apply(
680:         get_longest_param_value)
681:     df['feature-URL-Path'] = df['feature-URL'].apply(path_from_URL)
682:     df['feature-URL-Path-Length'] = df['feature-URL-Path'].apply(len)
683:     df['feature-URL-Path-ResourceExtension'] = df['feature-URL-Path'].apply(
684:         get_resource_extension)
685:     df['feature-URL-Path-ResourceExtension-Length'] = df['feature-URL-Path'].apply(
686:         get_resource_extension_length)
687:     df['feature-URL-Parameters-Num'] = df['feature-Query'].apply(count_params)
688:     df['feature-URL-Parameters-Names'] = df['feature-Query'].apply(name_params)
689:     df['feature-URL-Parameters-Type'] = df['feature-Query'].apply(type_params)
690:     df['feature-URL-Param-Value-Count-Dot'] = df['feature-Query'].apply(
691:         count_dots)
692:     df['feature-URL-Param-Value-Count-Percentage'] = df['feature-Query'].apply(
693:         count_percentage)
694:     df['feature-URL-Param-Value-Count-SpecialChars'] = df['feature-Query'].apply(
695:         count_specialChars)
696:     df['feature-Query-token_avg'] = df['feature-Query'].apply(get_token_avg)
697:     df['feature-Query-token_count'] = df['feature-Query'].apply(
698:         get_token_count)
699:     df['feature-Query-token_largest'] = df['feature-Query'].apply(
700:         get_token_largest)
```

```
701:     # reorder columns, setting the 'Label' column as the last one
702:     df = df[['feature-Method', 'feature-URL', 'feature-Host', 'feature-Cookie', 'feature-Content-Type',
703:              'feature-Content-Length', 'feature-Query', 'feature-Query-Length', 'feature-Query-Longest-Value',
    'feature-URL-Path',
704:              'feature-URL-Path-Length', 'feature-URL-Path-ResourceExtension', 'feature-URL-Path-ResourceExtension-
    Length', 'feature-URL-Parameters-Num', 'feature-URL-Parameters-Names', 'feature-URL-Parameters-Type', 'feature-
    URL-Param-Value-Count-Dot',
705:              'feature-URL-Param-Value-Count-Percentage', 'feature-URL-Param-Value-Count-SpecialChars',
706:              'feature-Query-token_avg', 'feature-Query-token_count', 'feature-Query-token_largest', 'Label']]
707:     # remove some columns because their variance is slow or biased (other more valuables features already
    extracted from them)
708:     df = df.drop(columns=['feature-Cookie', 'feature-URL'])
709:     df.to_csv(full_dataset_filename, sep='|', index=False)
710:     feature_columns = df.loc[:, df.columns != 'Label'].columns
711:     target_colum = df['Label']
712:     df_factorized = df  # copy dataframe to keep a copy of unfactorized df
713:     stacked = df_factorized[feature_columns].stack()
714:     df_factorized = pd.Series(
715:         stacked.factorize()[0], index=stacked.index, dtype='str').unstack()
716:     df_factorized['Label'] = target_colum
717:     df_factorized.to_csv(
718:         BASE_DIR + 'datasets/factorized/full_dataset.csv', sep='|', index=False)
719:     X_train, y_train, X_test, y_test, X_validation, y_validation = train_test_validation_split(
720:         df_factorized, 0.8, 0.1, 0.1, '/factorized/')
721:     X_train_unfactorized, y_train_unfactorized, X_test_unfactorized, y_test_unfactorized,
    X_validation_unfactorized, y_validation_unfactorized = train_test_validation_split(
722:         df, 0.8, 0.1, 0.1, '/processed/', False)
723:     times['processing'] = time() - t0
724:
725:     '''
726:     ====== FEATURES SELECTION  ===================================================
727:     '''
728:     t0 = time()
729:     # AUTO selection of features with Correlation-based Feature Selection (CFS) method
730:     auto_selected_features = auto_feature_selection(X_validation, y_validation)
731:
732:     # MANUAL selection of features (to be used in test_other_models)
733:     manual_selected_features = ['feature-Query-Length',
734:                                 'feature-Query-Longest-Value',
735:                                 'feature-URL-Parameters-Num',
736:                                 'feature-URL-Path-Length',
737:                                 'feature-URL-Param-Value-Count-SpecialChars']
738:
739:     times['features'] = time() - t0
740:
741:     '''
742:     ====== AUTOMATIC HYPERPARAMETERS TUNING ==================================
743:     '''
744:     t0 = time()
745:     best_hyperparameters = auto_hyperparameters_selection(
746:         X_validation[auto_selected_features], y_validation)
747:     times['auto_hyperparameter'] = time() - t0
748:
749:     '''
750:     ====== TRAINING AND TESTING THE MODEL ====================================
751:     '''
752:     t0 = time()
753:     classifier = DecisionTreeClassifier(**best_hyperparameters)
754:     # Train Decision Tree classifier
755:     classifier = classifier.fit(X_train[auto_selected_features], y_train)
756:     times['training'] = time() - t0
757:     # Predict the response for test dataset
758:     t0 = time()
759:     predictor = classifier.predict(X_test[auto_selected_features])
760:     total_time = "%0.3fs" % (time() - t0)
761:     times['predicting'] = time() - t0
762:
763:     '''
764:     ====== PLOTTING GRAPHS ===================================================
765:     '''
766:     t0 = time()
767:     labels = ['normal', 'anomalous']
768:     importances = classifier.feature_importances_
769:     print("\n\nPlotting graphs...")
770:     # create correlation matrix heatmap to further analysis
771:     correlation_matrix_heatmap(
772:         BASE_DIR + 'datasets/factorized/full_dataset.csv', 'correlation_heatmap.png')
773:     # create tree image
774:     create_tree_image(classifier=classifier, features=auto_selected_features,
775:                       labels=labels, filename='http-traffic-tree.png')
776:     # create graph with feature importances
777:     feature_importances_graph(
778:         X_train[auto_selected_features], importances, 'feature_importances.png')
779:     # create learning curves graphs
780:     plot_learning_curve(classifier, 'Learning Curves (DecisionTree)', 'learning-curves.png',
781:                         X_train[auto_selected_features], y_train, ylim=(-.1, 1.1), cv=None, n_jobs=4)
782:     # create graph for validation curve (max_depth)
783:     plot_validation_curve(classifier, 'Validation Curve for max_width (DecisionTree)', 'validation-curve-
    max_width.png',
```

```
                            X_train[auto_selected_features], y_train, 'max_depth', np.arange(1, 7, 1))
    # create graph for validation curve (min_samples_split)
    plot_validation_curve(classifier, 'Validation Curve for min_samples_split (DecisionTree)', 'validation-curve-
min_samples_split.png',
                            X_train[auto_selected_features], y_train, 'min_samples_split', [2, 6, 13, 20, 27, 34,
40])
    # create graph for validation curve (min_samples_leaf)
    plot_validation_curve(classifier, 'Validation Curve for min_samples_leaf (DecisionTree)', 'validation-curve-
min_samples_leaf.png',
                            X_train[auto_selected_features], y_train, 'min_samples_leaf', [1, 5, 9, 13, 17, 20])
    times['graphs'] = time() - t0

    '''
    ====== REPORTING ============================================================
    '''
    # compute the confusion matrix, the clasification report and print some stats
    c_matrix = metrics.confusion_matrix(
        y_test, predictor, labels=['normal', 'anomalous'])
    # true_negative, fase_positive, false_negative, true_positive
    tn, fp, fn, tp = c_matrix.ravel()
    report = metrics.classification_report(
        y_test, predictor, target_names=['normal', 'anomalous'])
    print_stats(tn, fp, fn, tp, report, times)

    '''
    ====== TESTING OTHER MODELS =================================================
    '''
    # testing the accuracy score for other model_selection
    test_other_models(X_train, y_train, X_test, y_test,
                      auto_selected_features, manual_selected_features)

    return 0


if __name__ == "__main__":
    main()
```