Universidad Oberta de Cataluña

Análisis de la viabilidad de la aplicación de técnicas de Auto Machine Learning para la detección de intrusiones

Trabajo de fin de máster

Autor: Marcos Barcina Blanco
30-5-2022
Director: Enric Hernández Jiménez
Máster Universitario de Ciberseguridad y Privacidad

Resumen

El propósito de este trabajo es evaluar la idoneidad del uso de técnicas y herramientas de Auto Machine Learning en el ámbito de la ciberseguridad, específicamente en la detección de intrusiones mediante el análisis del tráfico de red. Para ello, se ha implementado un sistema que entrena varios modelos de predicción sobre el dataset introducido mediante técnicas tanto de Machine Learning tradicional, como técnicas de AutoML y realiza una comparación con los resultados obtenidos.

Este sistema se ha implementado en el lenguaje de programación Python y hace uso de librerías como Scikit-learn y TPOT, ambas muy presentes en el desarrollo de tecnologías de ML y son de código abierto.

Finalmente, se han escogido datasets previamente utilizados en otros experimentos de aplicación de técnicas de Machine Learning similares. Esta elección permite comparar los resultados obtenidos con aquellos realizados en el estado del arte.

Descriptores

AutoML, Machine Learning, algoritmos de clasificación, hiperparámetro

Abstract

The purpose of this work is to assess the suitability of applying Auto Machine Learning techniques and tools to the domain of cyber security, specially to network traffic analysis. In order to achieve this, a system that trains several prediction models on a dataset has been developed. The predictions models are trained by using conventional Machine Learning techniques or newer AutoML ones and then the results are compared.

The programming language of choice is Python and libraries such as Scikit-learn and TPOT have been imported to implement some classification algorithms. These libraries are open source and their use is very extended for solving Machine Learning problems.

The chosen datasets have been previously used in other similar experiments of Machine Learning applied to cyber security. This choice allows for a more in depth comparison between the performance of the current Machine Learning techniques and the actual viability of AutoML.

Keywords

AutoML, Machine Learning, algoritmos de clasificación, hyperparameter

Índice

Índice

Índice de figuras

Índice de tablas

1. INTRODUCCIÓN

Este documento constituye la memoria del proyecto “Análisis de la viabilidad de la aplicación de técnicas de Auto Machine Learning para la detección de intrusiones”.

La estructura de capítulos y apartados de este documento se ha realizado siguiendo los criterios y recomendaciones que establece el documento “El trabajo de fin de grado y de máster: Redacción, defensa y publicación” [1].

El proyecto consiste en el desarrollo de un programa que entrene diferentes modelos de predicción de Machine Learning y AutoML [2] sobre un dataset, realice las predicciones y compare los resultados.

Concretando se pretende:

- Implementar diferentes algoritmos de clasificación tradicionales que sean capaz de entrenarse y realizar predicciones sobre un dataset.
- Implementar un sistema de AutoML que realice varias predicciones mediante diferentes algoritmos sobre un dataset y sea capaz de elegir el algoritmo específico y los hiperparámetros óptimos para cada problema.
- Ofrecer métodos que puedan interpretar las predicciones realizadas y comparar los resultados extraídos de ambos enfoques.

Este proyecto tiene como objetivo comprobar la verdadera viabilidad de la aplicación de técnicas de automatización del proceso de Machine Learning al análisis del tráfico de red en la ciberseguridad. Esto se debe a que el AutoML soluciona algunos problemas intrínsecos del ML tradicional como la necesidad de un experto que entrene los modelos, pero su uso aún no está muy extendido y puede presentar nuevos problemas.

En los siguientes capítulos y apartados explicaremos a fondo los detalles del proyecto, Entre los capítulos más destacados se encuentran:

- **Objetivos, estado del arte y alcance.** Este capítulo define y justifica el proyecto. Para ello, se define el objetivo del proyecto. También se explica la situación actual del estado del arte relacionado con el proyecto. Finalmente se define el alcance de este y se especifica claramente qué incluye y qué no.

- **Desarrollo del proyecto.** Este capítulo recoge el proceso de desarrollo del proyecto. Comienza con los requisitos funcionales y no funcionales del proyecto. Continúa con la especificación técnica del diseño de la arquitectura implementada. Además, se recogen las tecnologías utilizadas para cada parte, así como una breve justificación de su elección.

- **Implementación del sistema.** En este capítulo se recogen los detalles de la implementación del código del sistema desarrollado.

- **Planificación técnica de recursos.** En este capítulo se gestionan los plazos para el desarrollo del proyecto mediante la especificación de las tareas y fases durante el desarrollo del proyecto.

- **Resultados obtenidos.** En este capítulo se recogen los resultados obtenidos de cada experimento de AutoML y ML y también se exponen, analizan y comparan con los resultados obtenidos por otros experimentos similares externos realizados.

- **Conclusiones y trabajo futuro.** Este capítulo recoge las conclusiones y lecciones aprendidas durante la realización del proyecto. También incluye aspectos a mejorar en un futuro.

2. OBJETIVOS, ESTADO DEL ARTE Y ALCANCE

En este capítulo se recoge el objeto, el estado del arte actual y el alcance del proyecto.

2.1 OBJETIVO

El proyecto tiene como objetivo principal ofrecer una posible solución para el uso de Automated Machine Learning para la detección de intrusiones. Esto se pretende conseguir mediante:

- Implementación de varios modelo de predicción tradicionales de Machine Learning que pueda clasificar conexiones como “normales” o “irregulares” a partir de los datos de un dataset de intrusiones.
- Implementación de técnicas de Machine Learning automatizado que puedan elegir automáticamente el modelo de predicción y los hiperparámetros asociados óptimos del modelo escogido para obtener las mejores predicciones.
- Uso de métricas diseñadas para la evaluación del rendimiento de cada modelo entrenado y su comparación con otros modelos.

Todos los objetivos buscan mejorar los sistemas de detección de intrusiones actuales frente a las amenazas cibernéticas de Internet haciendo uso de las técnicas más recientes de AutoML y evaluar su rendimiento.

2.2 ANTECEDENTES

Las técnicas de Machine Learning actuales han jugado un papel importante en la creación de modelos de predicción en todo tipo de ámbitos, incluido el de seguridad. Sin embargo, debido a la constante evolución del ecosistema, los modelos de predicción actuales se enfrentan a nuevos problemas.

Uno de estos problemas es el proceso de “entrenamiento” que se debe realizar antes de que el modelo sea plenamente funcional. Normalmente estos modelos son entrenados de manera manual, esto quiere decir que un profesional se encarga de crear el modelo, limpiar las fuentes de datos, buscar los algoritmos de predicción óptimos y modificar de manera iterativa los parámetros de este algoritmo hasta dar con la mejor solución (que ofrezca más precisión, más eficiencia, velocidad, …). Esto resulta en modelos de predicción estáticos que no actúan bien cuando la naturaleza de los datos comienza a cambiar (ya que han sido entrenados y optimizados para un grupo de datos en concreto).

Los modelos de ML de ciberseguridad sufren en especial, ya que el ecosistema de amenazas y ataques cambia constantemente y a un ritmo muy rápido por lo que el modelo puede quedarse “desfasado” rápidamente si no se le presta atención.

En la actualidad, realizar este proceso de entrenamiento por un humano ya no es eficiente, por lo que ha surgido el Automated Machine Learning, que soluciona esta necesidad a través de la elección y entrenamiento automático de los modelos.

2.3 ESTADO DEL ARTE

El “boom” en el desarrollo y el abaratamiento de los dispositivos y las tecnologías de la información ha supuesto un aumento extraordinario en la cantidad de datos que pueden ser extraídos y almacenados. El acceso a esta cantidad ingente de datos ha propiciado el desarrollo de un subcampo de la Inteligencia Artificial, conocido como Machine Learning. El Machine Learning se entiende como el estudio de algoritmos y modelos estadísticos que aprenden automáticamente a través del uso de datos. Sobre estos algoritmos se construye un modelo de predicción y es entrenado con datos de prueba con el objetivo de hacer predicciones sobre otros datos del mismo tipo.

Desde la aparición de este campo han surgido numerosos algoritmos que buscan resolver este problema. Estos algoritmos han sido tradicionalmente clasificados en grupos principales según el método de aprendizaje que emplean [3]. En el artículo se recogen varios tipos de aprendizaje; supervisado, no supervisado, semi-supervisado y *Reinforcement Learning*.

El aprendizaje supervisado hace referencia al entrenamiento de modelos mediante datos etiquetados que otorgan pares de valores de entrada y valores de salida deseados. El objetivo es que el algoritmo encuentre reglas que le permitan predecir el valor de salida deseado dado uno o varios valores de entrada. Dentro de este campo existe la clasificación, que pretende asignar el valor de salida a una categoría mediante algoritmos como los Árboles de decisión o Naive Bayes, y la regresión, que, en vez de clasificarlo dentro de una categoría, se predice un valor de una variable continua como puede ser la edad o el precio. Dentro de esta última se utilizan algoritmos como la Regresión Lineal.

Por otra parte, los modelos no supervisados trabajan con datos que no han sido etiquetados de ninguna manera, esto quiere decir que el algoritmo, basándose únicamente en los datos de entrada, debe ser capaz de encontrar la estructura en estos datos. Aquí también se hace una división de categorías con el *Clustering* y la Reducción Dimensional. En el *Clustering*, el algoritmo extrae características de los datos de entrada y los organiza en grupos que comparten las mismas características, obteniendo así varios *clusters* y pudiendo reconocer las características de un nuevo conjunto de datos y asignarlo al grupo adecuado. Un ejemplo de esta última es el algoritmo *K-means clustering*. En la reducción dimensional, los algoritmos eliminan progresivamente las características redundantes de los datos para poder trabajar con un volumen de datos menor, pero igual de representativo como por ejemplo los algoritmos de *Feature Selection*.

Finalmente, en el aprendizaje semi-supervisado se combinan los 2 tipos anteriores de aprendizaje. Normalmente implica el uso de una pequeña cantidad de datos etiquetados para mejorar el rendimiento del modelo sobre una cantidad mayor de datos no etiquetados. Este tipo

de aprendizaje suele ser más efectivo cuando es necesario entrenar al modelo con datos etiquetados, pero supone un coste demasiado grande realizar este proceso.

El *Reinforcement Learning* se basa en la creación de un modelo que toma acciones en un entorno para maximizar la recompensa en base a las recompensas y penalizaciones definidas. Un ejemplo de la aplicación de este tipo de aprendizaje resulta en las redes neuronales, que buscan maximizar el resultado de una función de coste con las predicciones calculadas y los valores reales.

En el ámbito de la ciberseguridad, las técnicas de Inteligencia Artificial han sido de gran ayuda a la hora de hacer frente al creciente número de amenazas y ataques durante la última década. Tanto los modelos tradicionales de Machine Learning como los más novedosos de Deep Learning (DL) han tenido un mejor rendimiento que los modelos más tradicionales de ciberseguridad como los antivirus, cortafuegos, IDS, … en la detección y en dar respuesta a las amenazas [4]. Estas técnicas ya han sido implementadas en la mayoría de campos de la ciberseguridad como el “Filtrado de spam” o la “Detección de Malware”. Para la resolución de problemas de filtrado de spam, se han aplicado una gran cantidad de algoritmos. Muchas de estas aplicaciones se recogen en un estudio realizado por TS Guzella [5]. Este artículo engloba varios métodos de aprendizaje como Naive Bayes, SVM, Regresión Logística, Redes neuronales artificiales, Boosting, …, los resume y explican sus características y aportaciones más efectivas al ser aplicados en la resolución de problemas de clasificación de spam.

El segundo campo también tiene varios estudios realizados como la detección de malware en sistemas Android a través de 1C-SVM [6], donde los autores muestran cómo extrayendo datos del sistema características como los permisos o los grafos de flujo de ejecución es posible entrenar a un modelo para detectar comportamientos anómalos.

Otro campo en el que la implantación de estas técnicas ha tenido éxito ha sido en la “Detección de intrusiones”. Tradicionalmente los IDS implementan una serie de reglas que se aplican secuencialmente para detectar anomalías. Este enfoque requiere invertir bastante tiempo y recursos del sistema, por lo que es normal que ya se hayan aplicado técnicas de ML a este problema [7]. En el artículo se detalla el problema de un sistema de detección tradicional y se proponen maneras de mejorar el proceso de detección de un IDS conocido (Snort) a través del uso de árboles de decisión. En otro experimento más centrado en probar las capacidades de los algoritmos de ML supervisado, NB. Amor [8] compara en su estudio el rendimiento de aplicar “Árboles de decisión” y “Naive Bayes” a un dataset etiquetado de conexiones TCP/IP (KDD’99) en el que cada una se clasifica como normal o con el nombre del tipo de ataque asociado, ambos modelos de predicción consiguiendo un PCC (*Percent of correct classification*) del 91%, pero con Naive Bayes ofreciendo un mejor rendimiento. Un artículo parecido [9] también realiza una comparación entre los algoritmos de clasificación “Árboles de decisión” y “SVM” (*Support vector machines*) donde consiguen precisiones altas, pero un mejor rendimiento para los árboles y señalan el problema de que “SVM” solo puede realizar clasificaciones binarias. Incluso existen comparaciones entre la efectividad de Redes Neuronales y Árboles de decisión [10]. En el artículo los autores señalan que las redes neuronales, si bien son capaces de predecir la mayoría de las conexiones, no se adaptan correctamente a la detección de los ataques con pocas

muestras para el entrenamiento. A pesar de esto, la efectividad de las técnicas más avanzadas de ML como el Deep Learning también ha sido probada en entornos más prácticos. E. Hodo [11] muestra que un modelo de predicción basado en una red neuronal puede ser muy efectiva a la hora de detectar ataques tipo DDOS en una red IoT consiguiendo un 99% de precisión sobre las muestras extraídas de una simulación. De manera similar, J. Li propone la implementación de un framework para la detección de intrusiones [12] a través del uso de Redes Neuronales para la clasificación de anomalías y comprueba el rendimiento al simular ataques de *UDP Flooding*.

Sin embargo, el uso de estos modelos ha generado su propia serie de nuevos desafíos [13]. Uno de ellos es consecuencia del esquema tradicional de desarrollo de modelos de predicción. Este esquema incluye pasos como la preparación y procesado de datos, la selección de características, la generación de modelos, la optimización de hiperparámetros y la comparación de modelos que suponen un gran coste de desarrollo ya que todos estos pasos deben ser realizados manualmente por una persona o un grupo de profesionales. Además, pasos cómo escoger y optimizar parámetros del modelo suelen realizarse a través de prueba y error.

Las técnicas de Automated Machine Learning (AutoML) tienen como objetivo aportar soluciones a este problema. Los sistemas de AutoML construidos permiten automatizar los pasos intermedios del proceso de desarrollo de un modelo de predicción [14]. No solo supone eliminar una gran carga de trabajo repetitivo de los desarrolladores, sino que también permite a personas con poco conocimiento sobre Inteligencia Artificial crear e implementar modelos que permiten realizar predicciones de calidad. Para realizar estos ajustes, existen técnicas desde búsqueda por fuerza bruta o aleatoria para el ajuste de hiperparámetros hasta NAS (*Neural architecture search*) para la construcción de la arquitectura del modelo.

La viabilidad de estos modelos ya ha sido demostrada ya que diferentes empresas como “Cloud AutoML” por Google [15] o “Azure AutoML” [16] por Microsoft han puesto sus servicios de cara al público, permitiendo a usuarios no especializados crear sus propios modelos que se ajusten a sus necesidades. Debido a su alta necesidad de computación, incluso se están desarrollando técnicas para que los modelos puedan funcionar en dispositivos con menos recursos como teléfonos móviles [17].

En este contexto, cabe esperar que el próximo paso de la Inteligencia Artificial en el campo de ciberseguridad también pase por la implementación de estas técnicas. En el campo, ya han surgido algunos experimentos como la detección de ataques de *phising* [18] en el que se comparan los resultados obtenidos por modelos de AutoML generados por diferentes *frameworks* con modelos creados por algoritmos tradicionales. Los autores encontraron que, si bien entrenar los modelos manualmente podía llegar a ser más rápido, los modelos de AutoML tenían un mejor índice de precisión.

De la misma manera, un estudio realizado sobre el filtrado del SMS spam [19] arroja resultados parecidos sobre el rendimiento de los modelos de AutoML en el ámbito de la ciberseguridad. En este caso solo se comparaba el rendimiento de tres frameworks diferentes para la creación de modelos de AutoML, pero se obtenía un resultado satisfactorio de todos ellos que justificaba su

viabilidad. Cabe destacar que algunos de los frameworks utilizados para desarrollar los modelos de predicción coincidían en ambos experimentos (“TPOT” [20] y “H2O AutoML”).

El lenguaje más popular para el desarrollo de estos modelos de predicción es Python ya que contiene librerías como “Sci-kit learn” [21], “PyTorch”, “Keras” y “TensorFlow” que ofrecen una gran cantidad de funcionalidades para el desarrollo de modelos de Machine Learning y Deep Learning y otras de procesamiento de datos como “Pandas”o “Numpy”. Además, en los experimentos descritos anteriormente, todos los frameworks de desarrollo de modelos AutoML se han desarrollado sobre el lenguaje Python (“autogluon”, “auto-sklearn”, “GAMA”, “H2O AutoML”, “TPOT”, “hyperopt-sklearn”) y han presentado buenos resultados.

Otros lenguajes como Java o Julia ofrecen también soporte para el desarrollo de modelos de predicción, pero no tienen la misma cantidad de herramientas y contenido en línea de las que Python dispone.

2.4 ALCANCE

A continuación, se detalla el alcance del proyecto, donde se define todo lo que está incluido en el mismo:

- Diseño y desarrollo de un programa que implemente varios modelos que realicen sus predicciones sobre los datasets introducidos.
- Implementación de 6 algoritmos de clasificación para el entrenamiento de los modelos de predicción: SVC Lineal, Regresión Logística, Árbol de decisión, KNN, Naive Bayes y MLP Classifier.
- Implementación de un algoritmo de AutoML de clasificación que busque la tubería y los hiperparámetros óptimos para el dataset analizado.
- Desarrollo de métodos que realicen las operaciones de lectura y limpieza en el dataset analizado.
- Desarrollo de métodos que realicen las operaciones de procesado en el dataset analizado.
- Desarrollo de métodos que evalúen el rendimiento del modelo y muestren los resultados obtenidos de manera comprensible.

Queda fuera del alcance del proyecto lo siguiente:

- Integración del programa desarrollado con un IDS.
- Análisis en tiempo real de los paquetes de una red.

3. DESARROLLO DEL PROYECTO

Este capítulo se documentan los aspectos técnicos del proyecto y se recoge el proceso de desarrollo de la solución. Para ello se han recogido los requisitos funcionales y no funcionales del sistema y la especificación del diseño de la arquitectura. También se especifican las tecnologías seleccionadas y las fuentes de datos escogidas.

3.1 REQUISITOS INICIALES

Una vez el proyecto haya sido desarrollado es necesario que la solución cumpla con los requisitos funcionales y no funcionales definidos en este apartado.

3.1.1 Requisitos funcionales

Aquí se describe las funciones del sistema y el comportamiento que debe tener.

- RF1. El sistema debe realizar predicciones a partir de los datos de los paquetes de red definidos en el dataset.
- RF2. El sistema debe clasificar un paquete, ósea una fila del dataset, como limpio o sospechoso, en base a la predicción realizada.
- RF3. El sistema leerá los datos extraídos de un formato CSV.
- RF4. El sistema debe poder realizar operaciones básicas de limpieza sobre los datos de extraídos del dataset.
- RF5. El sistema debe poder modificar los datos extraídos del dataset con el fin de hacer su formato o valores más apropiados para el proceso de entrenamiento.
- RF6. El sistema debe entrenarse a partir de los datos extraídos de un dataset.
- RF7.El sistema debe implementar múltiples algoritmos de predicción de clasificación para realizar sus operaciones.
- RF8. El sistema debe ser escalable y estar abierto a cambiar algoritmos existentes o introducir nuevos.
- RF9. El sistema debe ser capaz de buscar el algoritmo que mejor se ajuste para cada problema de manera automática.
- RF10. El sistema debe ser capaz de ajustar los hiperparámetros de los algoritmos para mejorar sus predicciones de manera automática.
- RF11. El sistema debe implementar múltiples indicadores para evaluar las predicciones realizadas.
- RF12. El sistema debe ser capaz de comparar las predicciones realizadas por cada algoritmo.
- RF13. El sistema debe ser capaz de determinar el algoritmo más adecuado en base a las predicciones realizadas.
- RF14. El sistema debe poder visualizar las predicciones finales realizadas.
- RF15. El sistema debe implementar métodos para que los usuarios puedan interpretar los resultados

3.1.2 Requisitos no funcionales

Aquí se describen los atributos de calidad del sistema y funcionalidades no comprendidas en los requisitos funcionales del sistema.

- RNF1. El sistema debe realizar sus predicciones en un tiempo óptimo.
- RNF2. Las tecnologías utilizadas para el desarrollo del sistema deberán tener el menor coste posible.
- RNF3. Los resultados deben ser comprensibles e interpretables incluso para usuarios no experimentados en Machine Learning.
- RNF4. El código desarrollado para crear el sistema deberá ser limpio y comprensible.

En este apartado se recoge el diseño de la arquitectura de la solución, así como las fuentes de datos escogidas y utilizadas.

3.2 DISEÑO DE LA ARQUITECTURA

En este apartado se recoge el diseño de la arquitectura de la solución. Otorga una visión global del sistema que sea desarrollado.

3.2.1 Arquitectura de la web

La arquitectura del modelo es aquella propia de los modelos de predicción, pero adaptada a las necesidades del proyecto.

Para el correcto funcionamiento de la solución, la aplicación Flask se conecta con varios tipos de bases de datos de los cuáles consulta, gestiona y actualiza información. En la siguiente figura podemos ver una imagen de la arquitectura de alto nivel de la aplicación.

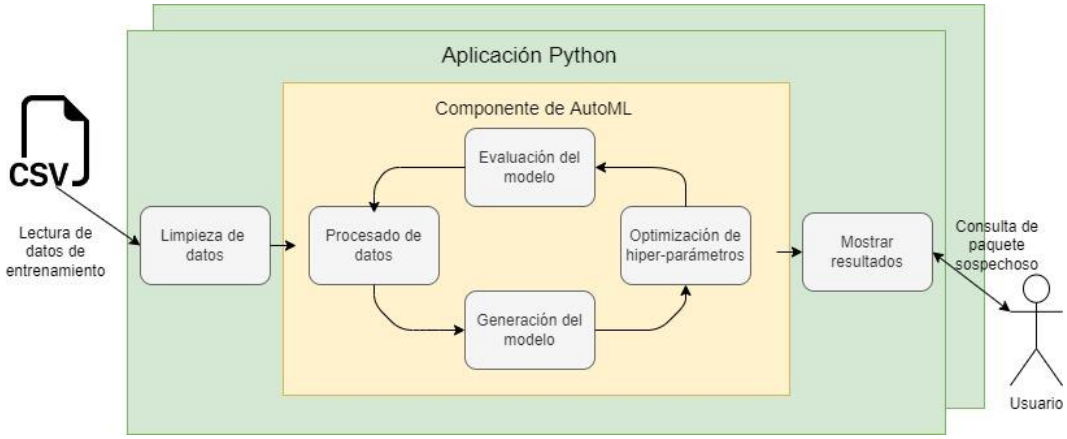

Ilustración 1. Arquitectura general de la solución

Como observamos en esta imagen, dentro de la aplicación existen varios módulos. Estos son los siguientes:

- **Limpieza de datos.** Se encarga de la lectura inicial del dataset que contiene los datos de prueba. También se encarga de implementar los métodos con las operaciones necesarias para realizar las operaciones de limpieza del dataset que suelen incluir el cambio del tipo de dato de una columna, la eliminación de filas vacías o con valores no válidos, …
- **Componente de AutoML.** Contiene los módulos que se encargan tanto de las operaciones del entrenamiento de los modelos tradiciones de ML, como de las operaciones necesarias para la implementación de un modelo de AutoML. Se ejecutan de manera iterativa hasta generar el mejor modelo de predicción y con los mejores parámetros encontrados que se ajusten al dataset recibido. También pueden ejecutarse por separado cada modelo para extraer sus resultados y facilitar una futura comparación.
    - Procesado de datos. Utiliza técnicas para transformar los datos de entrenamiento como la estandarización, normalización, SMOTE, … que pueden ayudar con el rendimiento.
    - Generación del modelo. Implementa varios modelos de predicción. El problema que se debe solucionar es de clasificación y existen muchos algoritmos de clasificación que pueden ajustarse mejor o peor al problema.
    - Optimización de hiperparámetros. A la hora de generar el modelo, se pueden ajustar los parámetros de creación con la intención de mejorar su rendimiento.
    - Evaluación del modelo. Calcular las métricas del modelo y realizar su evaluación respecto a ellas. Dependiendo de las necesidades, se puede evaluar un modelo respecto a su precisión, tasa de falsos positivos, tasa de falsos negativos, … (Ej. Si valoramos mucho la seguridad, igual preferimos un modelo que tienda a evaluar paquetes de manera más regular como falsos positivos que falsos negativos).
- **Mostrar resultados.** Una vez escogido el modelo que mejor se ajusta al problema, el sistema puede exportar el tipo de modelo escogido y sus parámetros óptimos. El sistema puede utilizar los parámetros exportados para recrear el modelo y así realizar predicciones y conocer los resultados del modelo con un mayor grado de profundidad. También puede visualizar los resultados de las predicciones realizadas de manera gráfica para comprender mejor el rendimiento del modelo escogido (mediante indicadores adicionales o gráficos como una matriz de confusión).

El desarrollo de la aplicación se va a realizar a través de Visual Studio, una herramienta que facilita la creación de proyectos en lenguajes como Python ya que facilitan la creación de un proyecto, así como su navegación. A continuación, podemos ver la estructura de carpetas que se puede esperar del proyecto.

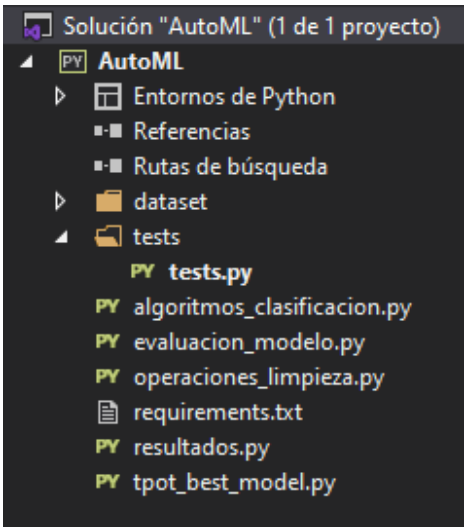

Ilustración 2. Estructura de carpetas

Se puede comprobar que todos los módulos tienen una clase asignada excepto el de la optimización de hiperparámetros y el procesado de datos. En el primer caso, esto se debe a que es más sencillo y ordenado realizar la definición de los hiperparámetros de cada modelo en el momento de la creación del propio modelo. Así que las tareas de este módulo se realizan en el propio módulo de generación del modelo.

Respecto al procesado de datos ocurre una situación parecida, cada algoritmo de clasificación funciona mejor si se aplican a los datos ciertas modificaciones específicas, por lo que no suele valer la pena aplicar estas operaciones de antemano al dataset y después probar cada algoritmo. Al ser un proceso estrechamente conjunto con la fase de división de los datos para el entrenamiento, se ha decidido también incluir las funcionalidades en el módulo de generación del modelo.

De manera más específica:

- <<`algoritmos_clasificacion`>>, es donde se definen todos los algoritmos de clasificación implementados, los métodos para el procesado de los datos como la estandarización o normalización y los hiperparámetros para cada modelo generado. También implementa el método encargado de aplicar la técnicas de AutoML.
- <<`evaluacion_modelo`>>, que se encarga de las tareas relacionadas con el módulo de evaluación del modelo. Implementa métodos que calculan indicadores de rendimiento sobre los resultados obtenidos.
- <<`operaciones_limpieza`>>, cuyas funciones se centran en la lectura del dataset introducido y funciones básicas de limpieza como eliminar columnas y filas vacías o la creación de “dummies” para columnas con valores que no pueden ser simplemente introducidas en un modelo de ML.
- <<`resultados`>>, que simplemente implementa métodos para visualizar gráficamente los resultados obtenidos de las predicciones de un modelo.

- <<`tpot_best_model`>>, este archivo contiene los datos del modelo y los hiperparámetros generados por el método de Auto Machine Learning implementado.

Finalmente tenemos las carpetas `tests` y `datasets`. La primera contiene el archivo <<`tests`>> donde se definen las pruebas que se realizan sobre los algoritmos implementados para extraer resultados y visualizarlos. La segunda contiene los datasets descargados y utilizados para poner en práctica los modelos.

Los archivos restantes son simples archivos necesarios para el correcto funcionamiento del proyecto en Visual Studio (como el entorno de Python).

3.2.2 Fuentes de datos escogidas

Las fuentes de datos han sido seleccionadas han tenido en cuenta el objetivo del proyecto y que también hayan sido utilizados en otros experimentos similares para poder realizar una mejor comparación.

Se han elegido los siguientes datasets.

- **KDD Cup 99:** Dataset con varias conexiones a cada cual se le etiqueta un tipo de ataque o la falta de cualquier tipo de ataque (tráfico normal). Las columnas que describen la información sobre la conexión son 41, definiendo características como la duración, protocolo, servicio, flag, … La columna restante define si la conexión es normal o si es parte de algún tipo de ataque. Está pensado para resolver un problema de detección de intrusiones y se pueden comparar los resultados obtenidos con los mostrados en los artículos referenciados en el estado del arte.
  El dataset y toda su información puede ser accedido en el archivo de UCI KDD [22]. En este trabajo se ha decido utilizar la versión que solo contiene un 10% del dataset original debido principalmente a las limitaciones de hardware con las que se ha realizado.

- **CICIDS2017**. Se compone de una colección de varios datasets con los datos del tráfico extraído de la monitorización de una red como la duración, el número total de paquetes, longitud total de los paquetes enviados o recibidos, ... De todos los datasets, se ha cogido el primero que clasifica cada fila (conexión) como “Benign” o “DDoS”. Estos datasets también están pensados para resolver el problema de detección de intrusiones y en el capítulo de resultados obtenidos se explica la elección de este dataset.
  Todos los datasets pueden ser accedidos y descargado de Kaggle [23], el dataset elegido ha sido “Friday-WorkingHours-Afternoon-DDos.pcap_ISCX.csv”.

El sistema también puede hacer uso de cualquier otro dataset, solo es necesario definir un nuevo proceso de lectura y algunas operaciones de limpiado propias del nuevo dataset escogido. Se han contemplado otros datasets, pero solo se han realizado las pruebas sobre los anteriormente descritos.

3.3 TECNOLOGÍAS UTILIZADAS

En este subapartado se va a nombrar y explicar todas las tecnologías que, al combinarse, forman el producto final.

De cada tecnología se expondrá cuál es su función principal, así como qué alternativas existen y cuál es la razón detrás de su elección.

3.3.1 Python

Python es un lenguaje de programación de alto nivel de propósito general. Permite diseñar proyectos de baja o alta escala y facilita escribir código limpio y claro. Es un lenguaje dinámico y da soporte a múltiples paradigmas de programación como estructural, orientado a objetos y funcional.

Las razones por las que se ha elegido este lenguaje son porque es el lenguaje más utilizado en problemas de Inteligencia Artificial, contiene muchas librerías que ayudan con el proceso de creación de un modelo ML y es Open Source. Además, Python se ha utilizado para realizar los experimentos de los modelos de predicción de AutoML descritos en el apartado anterior.

Se han considerado otros lenguajes alternativos como Java, pero al final no se ha optado por ellos ya que no ofrecían un abanico de posibilidades tan amplio y era algo más complicado encontrar material en internet respecto a su uso.

3.3.2 TPOT

TPOT (Tree-Based Pipeline Optimization Tool) es una librería de Python que se centra en la automatización de la construcción de “tuberías de ML” (manera de automatizar los pasos de desarrollo de un modelo de ML). El proceso normal para la creación y entrenamiento de un modelo tradicional de ML implica una tarea repetitiva de prueba y error en la selección de características, modelo , parámetros, … En la siguiente imagen, extraída de la propia documentación oficial, podemos ver exactamente la parte que TPOT ayuda a automatizar.

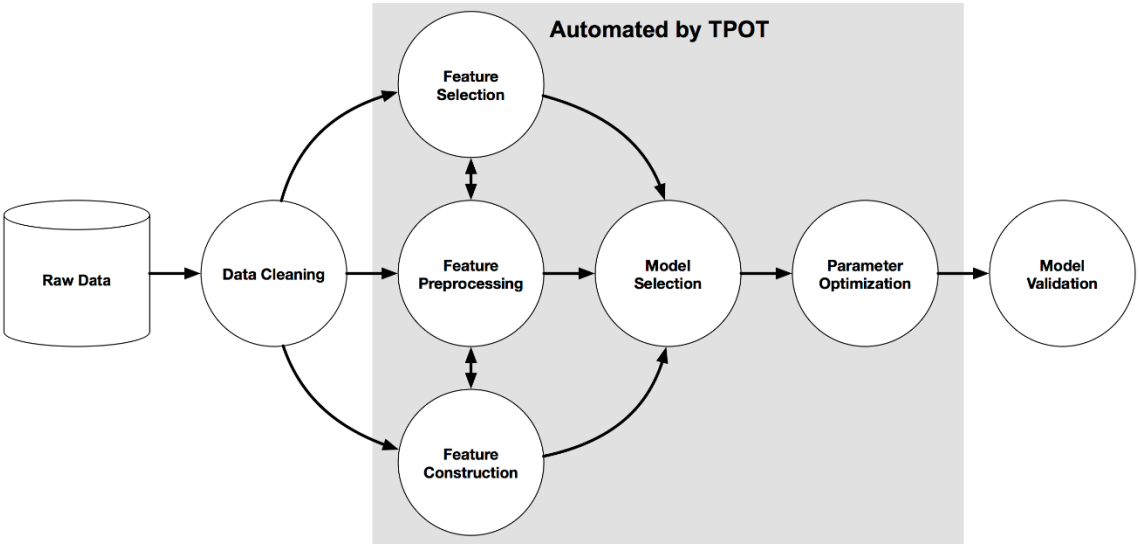

Ilustración 3. Automatización de TPOT. Imagen extraída de http://epistasislab.github.io/tpot/

Las operaciones de limpiado de datos y validación del modelo siguen recayendo en manos del desarrollador, pero todo lo relacionado con la selección de características modelo y parámetros está completamente automatizado por la librería, lo que facilita y aligera en gran medida el trabajo necesario para conseguir un modelo de calidad.

Al aplicar TPOT a un conjunto de datos se inicia un ciclo iterativo en el que TPOT prueba miles de tuberías y evalúa sus resultados hasta encontrar una que se ajuste de manera óptima al problema. Una vez TPOT termina de ejecutarse, devuelve código de Python donde define la tubería y el desarrollador puede, si lo desea, modificar ese código sin ningún problema.

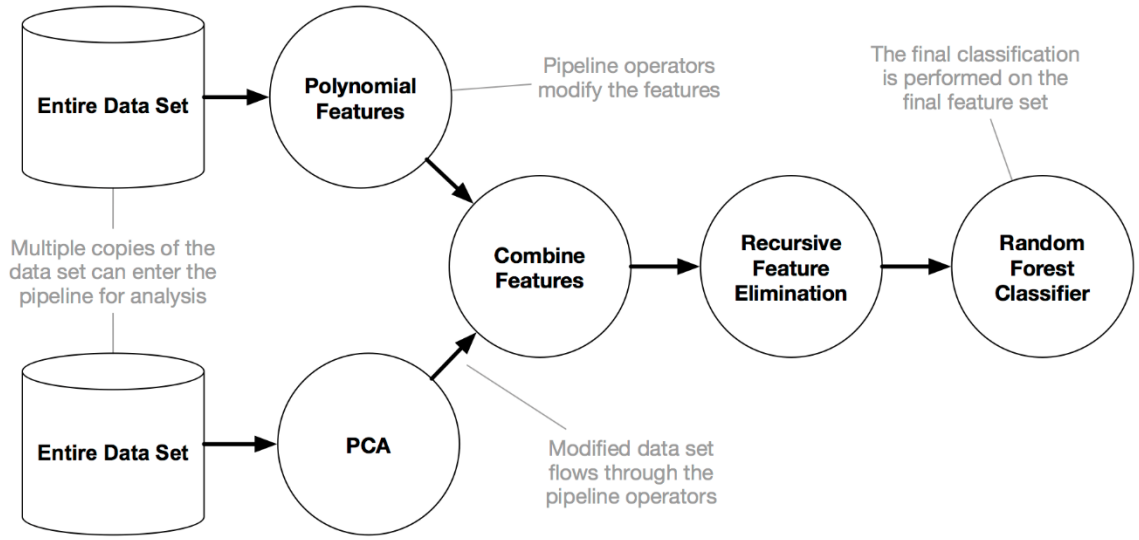

Ilustración 4. Ejemplo de una tubería creada por TPOT.

Imagen extraída de http://epistasislab.github.io/tpot/

Por ejemplo, una tubería suele implicar que se aplica algún tipo de operación a los datos como la eliminación recursiva de características (para solucionar problemas de imbalanceo de datos) y después es cuando se aplica el algoritmo de clasificación, en el caso del ejemplo es un Random Forest Classifier.

TPOT hace uso de la librería "scikit-learn", que es otra librería popular de ML en Python, para llevar a cabo sus funciones. Las operaciones y algoritmos que puede aplicar son extraídos de esta librería principalmente. Actualmente TPOT aún continua en desarrollo y se espera que sus autores sigan añadiendo nuevas funcionalidades y optimicen la librería en un futuro, lo cual garantiza que, al menos durante un tiempo, se siga pudiendo desarrollar programas alrededor de ella.

Como se ha visto en el apartado anterior, existen varias alternativas que permiten el trabajo con modelos de AutoML. Las razones principales por las que se ha elegido TPOT es que existe bastante documentación oficial al respecto, es compatible con el sistema operativo Windows y es Open Source.

Otras alternativas pueden ser "autogluon", "auto-sklearn", "GAMA", "H2O AutoML", "hyperopt-sklearn" y "azure machine learning". Si bien estas alternativas no se han seleccionado, vale la pena mencionarlas ya que pueden cubrir funcionalidades que no posee TPOT actualmente o tener un mayor rendimiento.

3.3.3 Scikit-learn

Scikit-learn es una librería para Python centrada en el desarrollo de modelos de Machine Learning. Contiene una amplia gama de algoritmos tanto de aprendizaje supervisado como no supervisado. Además, ofrece funcionalidades no solo para la creación y entrenamiento de modelos, sino para su evaluación y comparación con otros (creación de gráficos como una curva ROC o una matriz de confusión).

Scikit-learn requiere de un conocimiento básico de técnicas básicas de Machine Learning como el entrenamiento de modelos, realizar predicciones,"cross-validation", … La librería da soporte a todas las operaciones del proceso de Machine Learning y es bastante sencilla de implementar para cualquier usuario familiarizado con Python y ML.

Por ejemplo, evaluar el rendimiento de un modelo basado en un algoritmo específico sobre un dataset es tan sencillo como el siguiente.

```
y = df['remainder__outcome']
X = df.drop('remainder__outcome', axis=1)
lr = LinearRegression()
resultado = cross_validate(lr, X, y)
```

Ilustración 5. Ejemplo de código para implementar cross-validation.

Solo es necesario definir el conjunto de datos, es decir, dividir el dataset en 2 variables. “y” es la variable a predecir (normalmente es una columna de un dataset) y “X” es un array con el resto de las variables que influyen en la variable a predecir (normalmente el resto de las columnas del dataset). Ambas variables son la base de realizar cualquier operación de ML en la librería, ahora tan solo es necesario introducirlas en la función deseada. En este caso se evalúa un modelo de Regresión Lineal mediante “cross-validation” y, después, se calcula el resultado. El proceso se puede realizar en tres líneas de código, lo cual agiliza mucho el entrenamiento de modelos.

Esta secuencia será muy común a la hora de implementar los algoritmos tradicionales de ML en el programa ya que las diferencias de implementación se reducen al uso de otra función y a la modificación de algunos parámetros.

Existen alternativas como Keras o PyTorch que ofrecen funcionalidades similares. Principalmente se ha optado por Scikit-learn debido a que es un requisito para el uso de TPOT y algunas otras librerías de AutoML discutidas anteriormente. La librería permite la creación y entrenamiento de modelos de una manera más tradicional y manual.

3.3.4 Pandas

Pandas es una librería escrita en Python que permite leer, analizar y manipular datos. Pandas puede extraer información de diversos tipos de datasets y convertirlos en “dataframes”, que es el objeto principal de la clase. A través de los dataframes se pueden realizar operaciones de manipulación de datos (transformar el tipo de una columna, aplicar una expresión genérica, separar por grupos, …) y/o analizar los datos (calcular datos estadísticos básicos como la media o la desviación típica). Si bien no añade una funcionalidad completamente nueva, facilita la implementación de estas operaciones y es compatible con la mayoría de las librerías de Inteligencia Artificial de Python.

Se ha elegido Pandas para realizar las operaciones relacionadas con la lectura y el procesado de datos del sistema, que es el único paso del proceso de entrenamiento de un modelo ML que las otras librerías no cubren. Como se ha mencionado, esta librería se ha utilizado principalmente para las operaciones de lectura y limpieza de datos y la creación de “dataframes” que a su vez sirven para crear las variables “X” e “y” de las que se hablado antes y que son la base de cualquier operación de ML.

La razón principal de su elección por encima de otras librerías es que su uso está muy extendido en la comunidad y es extremadamente flexible. Al igual que las anteriores, la herramienta es completamente Open Source.

Entre las alternativas existentes se encuentran Spark y Polars, que ofrecen una funcionalidad similar a Pandas al permitir al usuario trabajar con “dataframes”. Sin embargo, estas librerías están dirigidas al análisis de grupos de datos pequeños y extremadamente grandes respectivamente, por lo que se ha decidido utilizar Pandas.

3.3.5 Matplotlib

Matplotlib es una librería de Python diseñada para la creación de gráficos estáticos, interactivos y animados. Permite generar gráficos de barras, de puntos, esquemas, figuras de manera sencilla y rápida. Normalmente cualquier gráfico se puede definir mediante los ejes X e Y y la elección del tipo de gráfico a representar.

En este proyecto su uso se ha limitado a la elaboración de gráficos para la visualización gráfica previa de información del dataset como, por ejemplo, comprobar si la variable a predecir puede tener un problema de balanceo ya que hay demasiadas muestras de un tipo y no suficiente de otro. También se ha utilizado para la generación de matrices de confusión como medidas para entender mejor los resultados de las predicciones de un modelo.

No existe una razón específica para su elección, simplemente es una librería que cubre la funcionalidad buscada y existe una documentación oficial que explica correctamente su funcionamiento.

Entre otras alternativas se incluyen librerías como Plotline, Plotly, Bokeh y pygal. Todas las librerías de este estilo ofrecen una funcionalidad similar y también están correctamente documentadas. Como se ha mencionado, sería posible sustituir Matplotlib por cualquiera de estas alternativas y no supondría cambios apreciables.

3.3.6 Seaborn

Seaborn es una librería para la visualización de datos basada en Matplotlib. Está orientada a la visualización de datasets y permite la creación de gráficos más complejos. Al igual que Matplotlib, se ha utilizado para la visualización del dataset y los resultados de los modelos de predicción.

Se ha escogido porque se puede combinar bien con Matplotlib y permite la creación de gráficos más avanzados como un mapa de calor (“heatmap”) que facilita la visualización de gráficos como la matriz de confusión.

Las alternativas son las mismas que en el apartado anterior y es probable que también contengan funcionalidades que permitan replicar lo realizado en el proyecto.

3.3.7 Pickle

Pickle es una librería de Python orientada a la serialización y deserialización de estructuras de objetos. Este proceso de serialización convierte el objeto en memoria en una secuencia de bytes que puede ser almacenada en disco o enviada a otro sistema. Más tarde, esta secuencia puede ser usada para recuperar ese objeto en el mismo estado en el que se serializó. Este proceso solo funciona en el lenguaje Python, pero es extremadamente sencillo de realizar ya que la serialización genera un archivo “.pkl” que después puede ser abierto por Pickle.

En proyectos de Machine Learning Pickle aporta una gran utilidad, ya que puede servir para almacenar un dataset (normalmente de gran tamaño) al que se le han aplicado varias operaciones de limpieza y así eliminar la necesidad de volver a realizar las operaciones de lectura o limpieza que suelen consumir mucho tiempo cuando la cantidad de datos es extremadamente alta.

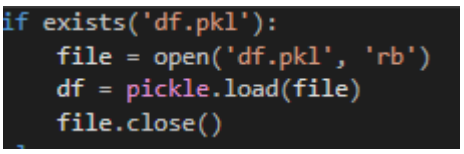

Ilustración 6. Código de deserialización de Pickle

Por otra parte, este mismo proceso de serialización también puede utilizarse para el almacenamiento de modelos de predicción ya entrenados. La parte de entrenar un modelo, dependiendo del algoritmo, suele consumir una gran cantidad de tiempo y no es viable realizar este proceso de entrenamiento cada vez que se necesite que el modelo realice una predicción. Es por ello por lo que almacenarlos una vez han sido entrenados facilita en gran medida la realización de estudios de comparación o, simplemente, el uso y almacenamiento de diferentes modelos.

Respecto a las alternativas, existen otras que permiten serializar como JSON. Esto tendría ventajas como que es posible que otro lenguaje de programación interprete el objeto serializado pero es más lento y, debido al uso exclusivo para agilizar las tareas de lectura y entrenamiento de modelos, Pickle resulta ser de mayor utilidad.

3.3.8 Diagrams.net

Diagrams.net es una aplicación web gratuita de creación de diagramas que está estrechamente integrada con Google Drive.

Mediante esta herramienta se pueden editar una gran cantidad de diagramas como pueden ser diagramas de flujo, diagramas entidad-relación, …Se ha utilizado para la creación del diagrama de la arquitectura de alto nivel del programa.

Existen otras alternativas que permiten realizar diagramas como StarUML o Visual Paradigm que, si bien ofrecen capacidades superiores a las de Diagramas.net para el modelado de estos diagramas, no se han escogido ya que son herramientas de pago y no poseen tanta flexibilidad.

3.3.9 Trello y Team Gantt

Trello es un software de administración de proyectos con interfaz web. Trello permite la creación de listas en las que se introducen tarjetas que representan una tarea. Con esto y su posibilidad de uso de “Power ups” (mejoras) es muy sencillo organizar las tareas de un proyecto y a quién se le asignan esas tareas, así como su estado.

Una mejora muy útil ha sido la llamada “Team Gantt” ya que ha facilitado la creación de un cronograma estilo Gantt del proyecto a partir de las tareas definidas en las tarjetas de Trello.

Existen muchas alternativas a estas herramientas para la organización de proyectos y creación de cronogramas como Kanban, Microsoft Planner o Excel.

Se ha optado por Trello debido a que el alumno ya estaba familiarizado con su uso y es una herramienta gratuita.

4. IMPLEMENTACIÓN DEL SISTEMA

Este capítulo recoge la implementación en el código de los métodos de Machine Learning y AutoML seleccionados, así como el de los módulos de operaciones de limpieza, evaluación y muestra de resultados.

4.1 OPERACIONES DE LIMPIEZA

Estas operaciones incluyen la lectura y la preparación de los datos que van a ser utilizados para el entrenamiento de los modelos de predicción y se han implementado en la clase evaluacion_modelo.py. En los datasets escogidos, no ha sido necesario aplicar demasiadas correcciones ya que estaban preparados para ser usado en este tipo de problemas de ML, pero si otros datasets requiriesen operaciones adicionales de limpieza, serían implementadas en esta clase.

```
def read_dataset(dir, features):
    """Lee el dataset introducido y devueleve un dataframe"""
    df = pd.read_csv(dir, names = features, header=None)

    df = df.dropna() #Eliminamos los valores Nan del dataset

    df.drop_duplicates(subset=features, keep='first', inplace = True)
    return df
```

Ilustración 7. Método de lectura del dataset

Cabe destacar que, a la hora de realizar las operaciones iniciales de lectura y limpieza básica del dataset, se puede realizar o la operación de eliminación de duplicados. A pesar de que suele ser una operación normal en cualquier dataset, puede resultar negativo para el rendimiento del modelo. Al hablar de ML, eliminar los duplicados tiene sus ventajas e inconvenientes. El mayor cambio es que, al eliminarlos, es probable que el modelo generalice mejor ya que se reduce el riesgo de *overfitting*.

A pesar de esto, los duplicados existentes en el dataset son representativos de lo que se puede esperar de un conjunto de conexiones de manera natural (y es probable que futuros conjuntos de datos sigan una distribución parecida), por lo que eliminar duplicados puede afectar en gran medida a los valores de peso.

Otra operación que no es exactamente limpieza pero que se ha incluido aquí es la creación de *dummies* ya que siempre es necesario. Cuando una variable es categórica, es recomendable crear *dummies* de ella, para que el proceso de ML sea correcto ya que los algoritmos no pueden procesar *strings* de texto como una entrada válida.

```python
def create_dummies(df, names):
    """Crea dummies de las columnas categoricas"""
    transformer = make_column_transformer((preprocessing.OneHotEncoder(), names), remainder='passthrough')
    transformed = transformer.fit_transform(df)
    transformed_df = pd.DataFrame(transformed, columns=transformer.get_feature_names_out())


    transformed_df.to_pickle('df_no_duplicates.pkl')

    return transformed_df
```

Ilustración 8. Método para la creación de dummies

Es posible simplemente cambiar la categoría por un valor numérico, pero, a diferencia de la creación de *dummies*, es más probable que cree problemas durante el entrenamiento.

4.2 OPERACIONES DE PROCESADO

Estas operaciones también tienen como objetivo preparar los datos para los modelos de predicción, pero, en este caso, se trata de aplicar técnicas que pueden mejorar el proceso de ML en vez de centrarse en el propio formato de los datos.

Un ejemplo es la aplicación de estandarización o normalización a los datos para mejorar el rendimiento de los modelos de ML. Estas solo se suelen aplicar cuando los valores de los datos oscilan mucho entre sí.

```python
def apply_standarization(X_train, X_test):
    """Aplica estandarizacion (reescalar los valores para que tengan media de 0 y varianza de 1).
       Utilizar cuando el rango de los valores no sea demasiado alto."""
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)             #Fit calcula la mediana y la varianza. Tra
    X_test = scaler.transform(X_test)                   #Por eso no se hace fit en test, ya están

    return X_train, X_test
```

Ilustración 9. Función de estandarización

Otras técnicas incluyen el Undersampling o SMOTE cuando la representación de las clases está muy desbalanceada y afecta de manera negativa a la capacidad de predicción del modelo. A pesar de que se han implementado varias técnicas, no se va a mostrar el código de todas ya que siguen un esquema extremadamente similar.

4.3 ALGORITMOS DE CLASIFICACIÓN

Con el objetivo de comprobar la utilidad y la capacidad de las técnicas de AutoML, se procede a la implementación de varios modelos de predicción de Machine Learning tradicionales supervisados. Estos modelos se utilizan para resolver el problema y comparar los resultados obtenidos con aquellos obtenidos a través de las técnicas de AutoML.

4.3.1 SVC Lineal

El primero es SVC lineal, que funciona bien para realizar clasificación con un alto número de datos. La implementación de algoritmo está realizada con la librería anteriormente mencionada “Scikit-learn” que facilita en gran medida la realización del proceso.

```
def SVC_lineal(X_train, X_test, y_train):

    lsvc = LinearSVC()
    lsvc.fit(X_train, y_train)

    y_pred = lsvc.predict(X_test)
    y_pred_proba = 0

    return y_pred, y_pred_proba
```

Ilustración 10. Implementación de SVC Lineal

Todas las implementaciones de los algoritmos siguen un esquema similar en el sentido que siempre reciben como entrada los parámetros X_train, X_test y y_train. Estos representan el set de datos con los que entrenar al modelo y un set de datos de prueba para que el modelo realice predicciones sobre estos.

También se declara el algoritmo de clasificación que se va a utilizar y, finalmente, se realizan las predicciones y se devuelven los resultados en forma de y_pred y y_pred_proba (este segundo siendo una probabilidad necesaria para calcular la curva ROC). Las propias predicciones siempre servirán para calcular estadísticas como la precisión, mientras que la otra variable solo la usaremos en casos específicos como la creación de la curva ROC (que solo es posible en problemas de clasificación binaria).

4.3.2 Regresión Logística

La regresión logística es un modelo estadístico que pretende calcular la probabilidad de que una de 2 opciones ocurra (la función tiende a 0 o 1) a partir de varias variables independientes. La regresión logística está pensada para resolver problemas de clasificación binaria, donde solo existen 2 clases.

```
def regresion_logistica(X_train, X_test, y_train):

    regressor = LogisticRegression(multi_class='multinomial')

    regressor.fit(X_train, y_train)
    y_pred = regressor.predict(X_test)
    y_pred_proba = regressor.predict_proba(X_test)[::,1]

    return y_pred, y_pred_proba
```

Ilustración 11. Implementación de Regresión Logística

Afortunadamente, también se ofrece la posibilidad de utilizar Regresión logística multinomial, que es una versión adaptada para soportar la existencia de más de 2 clases en la variable a predecir. Existe la posibilidad de modificar este parámetro en función de las características que tenga la variable "y" del dataset analizado como en el caso de K99 Cup, que incluye un valor diferente para cada tipo de ataque.

4.3.3 Árbol de decisión

Los árboles de decisión son un método de clasificación de ML no lineal que permite clasificar a un valor en diferentes categorías a través de un sistema de ramas y decisiones.

En su implementación solo se han definido algunas cosas básicas como el criterio "Gini" y la estrategia de *split*. Otros hiperparámetros como *max_depth* y *min_sample_split* no los modificamos ya que queremos comparar el uso de estos métodos tradicionales con uno de AutoML.

```
def decision_tree(X_train, X_test, y_train):

    clf = DecisionTreeClassifier(criterion="gini", splitter='best', class_weight='balanced')
    clf = clf.fit(X_train,y_train)

    y_pred = clf.predict(X_test)

    y_pred_proba = clf.predict_proba(X_test)[::,1]

    return y_pred, y_pred_proba
```

Ilustración 12. Implementación de un Decision Tree Classifier

La librería ofrece versiones avanzadas de los árboles de decisión como los bosques. El funcionamiento de estos es parecido y se puede describir como la combinación de varios árboles (el número de árboles de decisión que se entrenan es un hiperparámetro) de decisión para mejorar la precisión y reducir el riesgo de *over-fitting*.

4.3.4 K-nearest neighbors (KNN)

KNN es un algoritmo simple que sirve para realizar tanto tareas de clasificación como de regresión. El algoritmo encuentra la distancia entre un punto y el resto de los puntos más cercanos a él y lo relaciona con la clase más frecuente. En otras palabras, el algoritmo utiliza la proximidad para agrupar puntos individuales similares.

```
def k_nearest_neighbor(X_train, X_test, y_train):

    knn = KNeighborsClassifier(n_neighbors=5)

    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)

    y_pred_proba = knn.predict_proba(X_test)[::,1]

    return y_pred, y_pred_proba
```

Ilustración 13. Implementación de KNN Classifier

Tiene el problema de que no escala bien cuando el número de muestras es demasiado alto, lo que compromete el rendimiento del modelo. El híper parámetro de número de vecinos viene por defecto en 5, pero puede ser modificado para buscar un mejor rendimiento.

4.3.5 Naive Bayes

Un algoritmo de clasificación que asume que las características son independientes entre sí a la hora de realizar su predicción. Es una de las implementaciones más simples de un algoritmo de Redes Bayesianas, pero suele presentar un rendimiento bastante bueno a pesar de su sencillez. Debido a estas características, el algoritmo suele ser implementado en aplicaciones que se encargan de del filtrado de emails de spam.

```
def naive_bayes(X_train, X_test, y_train):

    gnb = GaussianNB()

    gnb.fit(X_train, y_train)
    y_pred = gnb.predict(X_test)

    y_pred_proba = gnb.predict_proba(X_test)[::,1]

    return y_pred, y_pred_proba
```

Ilustración 14. Implementación de Naive Bayes

Este algoritmo devuelve la probabilidad de que un punto pertenezca a una categoría en vez de la propia categoría. Como se ha mencionado al principio, este algoritmo asume que las características son completamente independientes entre sí, algo que no es realista en cualquier

conjunto de datos de la vida real. A pesar de esto, el algoritmo presenta un buen rendimiento ya que las relaciones no suelen influir lo suficiente en el resultado final de las predicciones por lo que se puede aplicar en la mayoría de los problemas de Machine Learning que requieran clasificación.

4.3.6 Red neuronal

Finalmente, se ha optado por la implementación de un modelo de predicción basado en redes neuronales, concretamente un *Multi-Layer Perceptron Classifier*. Las redes neuronales funcionan emulando las neuronas en un cerebro humano mediante la creación de varias capas, una de entrada, una o varias capas ocultas y una de salida. Las neuronas de cada capa se conectan con aquellas de la capa siguiente y realizan operaciones.

Una vez se llega a la capa final, el rendimiento del modelo se calcula mediante una función de coste que calcula la diferencia entre los resultados obtenidos y los esperados. A partir de esta función, cada vez que se reajustan los parámetros de las capas, se busca disminuir lo máximo posible su valor.

```
def neural_network(X_train, X_test, y_train):

    mlp = MLPClassifier()
    mlp.fit(X_train, y_train)
    y_pred = mlp.predict(X_test)

    y_pred_proba = mlp.predict_proba(X_test)[::,1]

    return y_pred, y_pred_proba
```

Ilustración 15. Implementación de MLP Classifier

A la hora de optimizar los hiperparámetros, normalmente se modifica el número de capas intermedias (mediante el hiperparámetro *hidden_layer_sizes*). Otros parámetros modifican el ritmo de aprendizaje de la red neuronal o el propio *solver* (encargado de la optimización de pesos de cada neurona en las capas) que utiliza el modelo.

Al igual que con las implementaciones previas, no se modifican manualmente por cuestión de comparar estos métodos tradicionales con los métodos de AutoML ya que se incluye en las tareas de optimización de hiperparámetros.

4.4 MODELO DE PREDICCIÓN CON AUTOML

Como se ha especificado en el apartado de tecnologías, para la implementación un modelo de AutoML se ha utilizado la librería TPOT. El método principal ejecuta una operación en la que busca la mejor tubería para los datos introducidos, que se compone de operaciones de procesado de datos, el algoritmo seleccionado y las hiperparámetros óptimos a la hora de definir las propiedades del algoritmo.

Su implementación es también relativamente sencilla y puede realizarse en unas pocas líneas de código.

```
def autoML(X_train, X_test, y_train, generations, population_size):

    model = tpot.TPOTClassifier(generations=generations, population_size=population_size, verbosity=2)
    model.fit(X_train, y_train)

    return model
```

Ilustración 16. Implementación del método de AutoML

Los parámetros más importantes que modificar son *generations*, que define el número de iteraciones que se van a realizar de optimización de pipeline (también se puede definir un tiempo máximo) y *population_size* que son el número de "individuos" que mantienen la programación genética cada generación. Cuanto mayores sean estos argumentos, mayor será la probabilidad de que se encuentre una tubería más efectiva, pero a costa de requerir más tiempo de procesado.

Otros argumentos incluyen *offspring_size* (número de hijos de cada generación, el estándar es el mismo que *population_size*), *mutation_rate* y *crossover_rate*, que sirven para modificar el comportamiento del algoritmo genético que usa el método. El parámetro *verbosity* definido en la imagen sirve simplemente para que el programa devuelva feedback visual del progreso actual que lleva TPOT.

Finalmente, existe el argumento *scoring* para la evaluación de la calidad de las tuberías generadas. La medida por defecto es la precisión (*accuracy*), pero se puede modificar a muchas otras como *average_precision*, *f1*, *balanced_accuracy*, …

El proceso de búsqueda puede durar varias horas, incluso días dependiendo de los parámetros introducidos. El proceso de selección de la tubería óptima es progresivo, esto quiere decir que durante la ejecución del método siempre se almacena la última tubería que ha producido el mejor resultado. Una vez encontrada la mejor tubería o la mejor tubería en ese momento de la ejecución, es posible exportarla en un archivo de código de Python.

```
def export_model(model):
    model.export('tpot_best_model.py')
```

Ilustración 17. Exportar una tubería

El archivo creado está prácticamente listo para ser utilizado por el usuario ya que solo requiere que se introduzca la ruta para la lectura del dataset. Aun así, también se puede copiar simplemente las operaciones y el modelo que utiliza y aplicarlas en cualquier otro sitio, ya que todos los métodos que utiliza son de la librería scikit-learn.

4.5 EVALUACIÓN DEL MODELO

Para la evaluación del modelo se han implementado varios mecanismos. La mayoría de ellos se centran en la evaluación de las predicciones una vez el modelo ha sido entrenado, el resto son métodos más “gráficos” de evaluación previo al entrenamiento del modelo para tener una idea general de las propiedades del dataset.

Respecto a los métodos de evaluación antes del procesamiento, se ha implementado uno que permite representar la frecuencia de las clases de la variable a predecir en un gráfico. Puede ayudar a ver situaciones en las que una falta o un exceso de repetición de una variable está alterando los resultados del modelo de predicción.

Por otra parte, los métodos de evaluación de los resultados son los más importantes. Se han implementado 3 características principales sobre las que puntuar el modelo: *Accuracy*, *Precision* y *Recall*. Otras características extra para su evaluación pueden ser TPR (*True Positive Rate*) y FPR (*False Positive Rate*).

```
def show_scores(y_test, y_pred):
    print("######## Puntuaciones ######## ")
    print("\n")
    print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
    print("Precision:",metrics.precision_score(y_test, y_pred, average='weighted'))
    print("Recall:",metrics.recall_score(y_test, y_pred, average='weighted'))
```

Ilustración 18. Indicadores de evaluación del rendimiento

También se han implementado métodos para mostrar la matriz de confusión o la curva ROC con el objetivo de facilitar la interpretación de los resultados de las predicciones de cada modelo y encontrar detalles que medidas como la precisión no pueden mostrar.

```
def matriz_confusion(y_test, y_pred):
    c_f_m = metrics.confusion_matrix(y_test, y_pred)
    cm_df = pd.DataFrame(c_f_m)
    labels = ['back', 'butter_overflow', 'loadmodule', 'guess_passwd', 'imap', 'ipsweep',
        'multihop', 'neptune', 'nmap', 'normal', 'phf', 'perl', 'pod', 'portsweep', 'ftp_w
    plt.figure(figsize=(15,10))
    sns.set(font_scale=1.4)
    sns.heatmap(cm_df, annot=True, annot_kws={"size":12}, fmt='g',xticklabels=labels, ytic
    plt.ylabel('Clase real')
    plt.xlabel('Clase predecida')
    plt.show()
```

Ilustración 19. Implementación de la matriz de confusión

Los indicadores implementados sirven para la evaluación del modelo de manera manual, es decir es necesario calcular estas métricas varias veces y entrenar varios modelos para obtener resultados que representen realmente el rendimiento de cada algoritmo. En este caso, scikit-learn también ofrece métodos que permiten evaluar el rendimiento de un modelo.

Realizar la operación de *cross-validation* sirve para comprobar cómo de bien un modelo de clasificación generaliza, es decir, cómo actúa sobre datos que no son pertenecientes al conjunto inicial de entrenamiento. Esto ayuda a reducir el riesgo de *overfitting* y es algo muy importante a tener en cuenta cuando el modelo va a ser desplegado en un entorno real porque es probable que, eventualmente, reciba datos de diferentes fuentes.

Esto normalmente se obtiene mediante la implementación de *K-folds*, que sirven para dividir los datos de entrenamiento en diferentes grupos (número de grupos es igual a k, normalmente se utiliza un valor de k=10) y entrenar una iteración del modelo con un grupo como datos de test y el resto de los grupos como datos de entrenamiento. Tras esto, se realizan predicciones con el modelo y se guardan las métricas de evaluación extraídas.

5. PLANIFICACIÓN TÉCNICA DE RECURSOS

Este capítulo tiene como objetivo recoger información respecto a la planificación temporal y de recursos del proyecto como puede ser un cronograma inicial de las tareas. También se recogen aquellos aspectos relacionados con la planificación de los recursos humanos y la asignación de tareas.

5.1 RECURSOS HUMANOS

En este apartado se definen los roles que han participado en la realización del proyecto, así como sus principales funciones.

El equipo para el desarrollo del proyecto ha contado con la participación de dos personas, un director de proyecto y un desarrollador.

- **Director del proyecto**. Encargado de realizar el control y el seguimiento sobre el proyecto. Ayuda al alumno con el desarrollo del proyecto a través de sugerencias y ayuda con ciertos problemas.
- **Desarrollador**. Este perfil engloba los roles de investigación, desarrollo y diseño dentro del proyecto. Se encarga de la realización de todas las tareas relacionadas con esas áreas del proyecto, así como de la metodología de trabajo y la redacción de la memoria.

La siguiente figura muestra la estructura interna del proyecto en forma de organigrama.

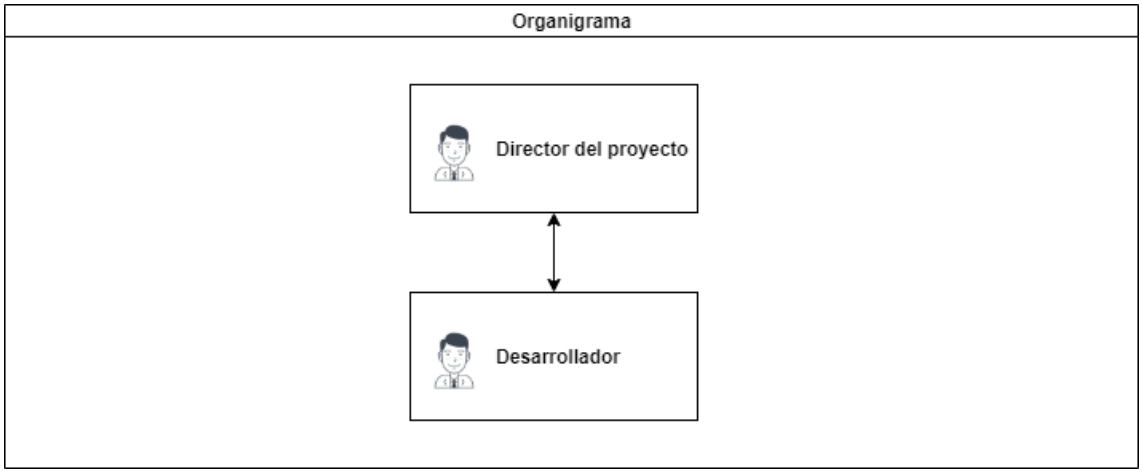

Ilustración 20. Organigrama del proyecto

5.2 PLANIFICACIÓN TEMPORAL

5.2.1 Metodología de desarrollo

La metodología seguida para el desarrollo de las tareas relacionadas con el proyecto es una llamada “Kanban”. Esta metodología pertenece a la familia de las metodologías ágiles, que son muy populares entre los proyectos de software.

Las razones por las que se ha decidido utilizar este tipo de metodología han sido las siguientes:

- **Ideal para grupos pequeños.** Como se ha explicado en el anterior apartado, el grupo de trabajo del proyecto es muy pequeño. Tampoco existen roles claros asignados a los miembros del grupo ni asociaciones de quién debe realizar cierta tarea.
- **Evolución continua y capacidad de cambio.** Las tareas se organizan en 3 tipos, el *backlog,* en curso y completadas. Esto permite una mayor flexibilidad a la hora de calcular cuánto tiempo puede llevar una tarea y , en caso de que surja algún problema, disponer de más tiempo para solucionarlo. No solo permite flexibilidad en relación con el tiempo, sino también a las tareas que se planean realizar ya que estas pueden variar durante el desarrollo del proyecto.

Si bien esta metodología no se ha aplicado al pie de la letra, se ha adaptado a las necesidades del proyecto y del alumno.

5.2.2 Tareas

En línea con la metodología “Kanban”, se han definido una serie de tareas a realizar para el desarrollo completo del proyecto. Estas tareas se han divido en relación con el tipo de actividad que conlleva.

A continuación, se muestran todas las tareas descritas, así como una pequeña explicación de lo que incluyen.

Diseño:

- **T1-Definición de requisitos.** Esta tarea incluye la definición de todos los requisitos que el proyecto debe cumplir.
- **T2-Revisión del estado del arte y elecciones tecnológicas.** Esta tarea implica la lectura y aprendizaje de materia actual en torno al tema elegido. En este caso, el tema estudiado es aquel relacionado con la aplicación de técnicas de Machine Learning (preferiblemente automático) a un dominio de ciberseguridad como la detección de intrusiones. Este estudio se lleva a cabo a través de artículos académicos, experimentos, … que hayan sido publicados sobre la materia en cuestión.

  Durante esta fase, y en base a la información encontrada, se eligen las tecnologías sobre las que se va a desarrollar el proyecto (lenguaje de programación, librerías, fuentes de información, …) y su justificación.

- **T3-Diseño de la arquitectura.** Con la finalidad de dar una visión general del modelo desarrollado se detalla la arquitectura de la propuesta a través de un diagrama que muestre las partes y como se conectan, así como una explicación de cada uno de los componentes.

Desarrollo:

- **T1-Generación de modelos.** Esta tarea implica el desarrollo y la implementación del propio proyecto. Esta implementación se realizará con las tecnologías escogidas.
- **T2-Pruebas y conclusiones.** Sobre el modelo se realizarán varias predicciones para conocer sus resultados y compararlos con otros resultados existentes para evaluar su calidad. Tras esto, se justificarán los resultados obtenidos en base a la implementación realizada.

Memoria:

- **T1-Redacción de la memoria.** Esta tarea implica la redacción de la memoria en la que se detallan los aspectos importantes sobre la realización del proyecto como los objetivos, requisitos, implementación, …
- **T2-Elaboración de la defensa.** Esta tarea supone la realización del vídeo para la defensa del trabajo de fin de máster realizado.

Con el objetivo de tener una representación visual de estas tareas, se ha utilizado la aplicación Trello para representar estas tareas mediante listas y tarjetas.

Al no seguir el método tradicional de clasificación de clasificación de las tareas en función de su estado, se ha optado por el uso de una mejora de Trello llamada “Team Gantt” que permite añadir un porcentaje de completado a cada tarea y la creación de un cronograma con estas.

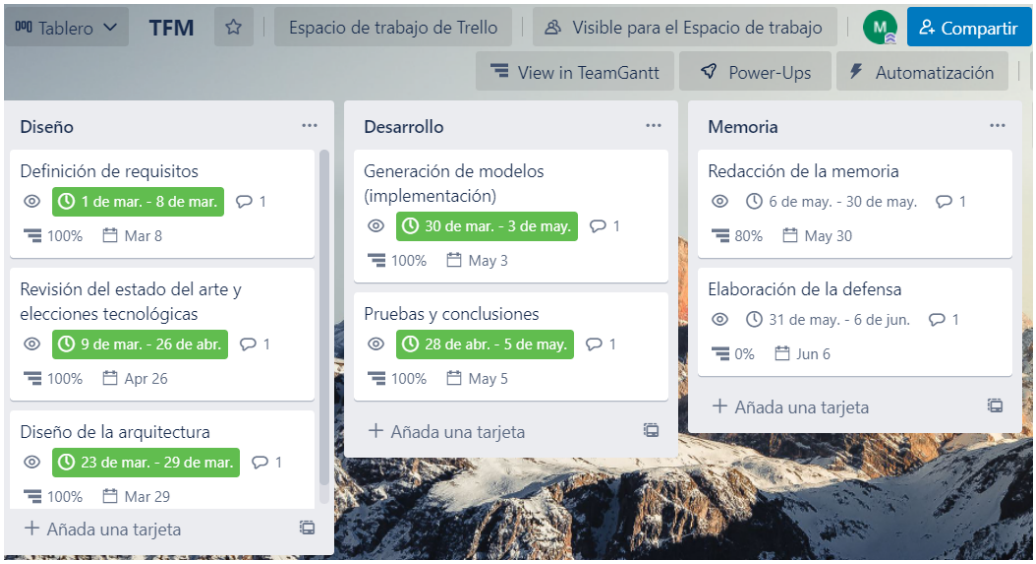

Ilustración 21. Tareas representadas en Trello

5.2.3 Cronograma de tareas

A pesar de que la metodología “Kanban” no requiere de un cronograma debido a su naturaleza, para este proyecto se ha optado por realizar uno ya que ayuda con la organización y estimación del tiempo pueden requerir las tareas (con el objetivo de evitar retrasos y problemas a la hora de completar el proyecto).

El cronograma se basa en las tareas descritas en el apartado donde se explican las tareas y se les asigna un tiempo estimado en el que pueden ser desarrolladas, acorde con la dificultad percibida. Es necesario resaltar que las tareas no tienen un número de horas asignado a ellas, sino un plazo.

También cabe destacar que, aunque en el cronograma se muestre que cierta tarea acaba en una fecha exacta, esto no significa que esa tarea se pueda revisitar si surgen necesidades o funcionalidades extra que se deben añadir necesarias para el desarrollo de otras tareas.

El cronograma se ha generado a través de la mejora “Team Gantt”, que permite añadir las tareas descritas en las tarjetas a un cronograma y la asignación de plazos a cada una de estas tareas. Se ofrece la opción de visualizar el cronograma en días, semanas o meses.

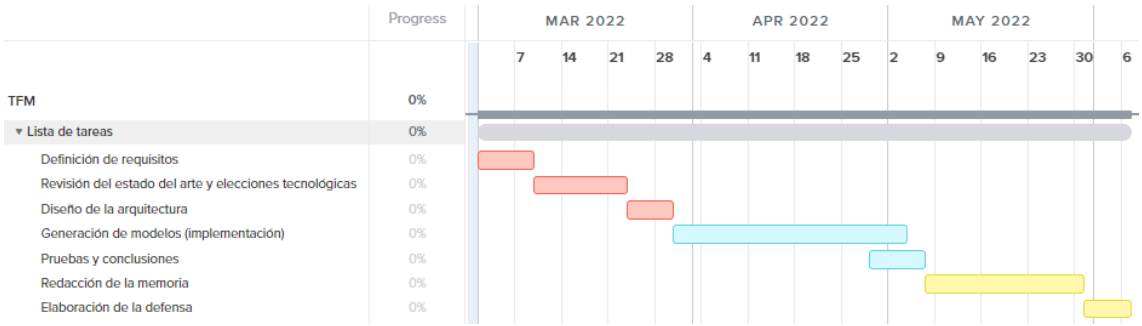

Ilustración 22. Cronograma inicial de tareas

En la anterior figura, se muestra un cronograma con la distribución inicial de las tareas a realizar del proyecto. Las tareas son prácticamente secuenciales y los plazos están calculados para que sean acordes a las entregas de seguimiento planteadas por la asignatura del TFM.

No ha habido ninguna desviación del cronograma inicial alta y el desarrollo del proyecto se ha realizado correctamente.

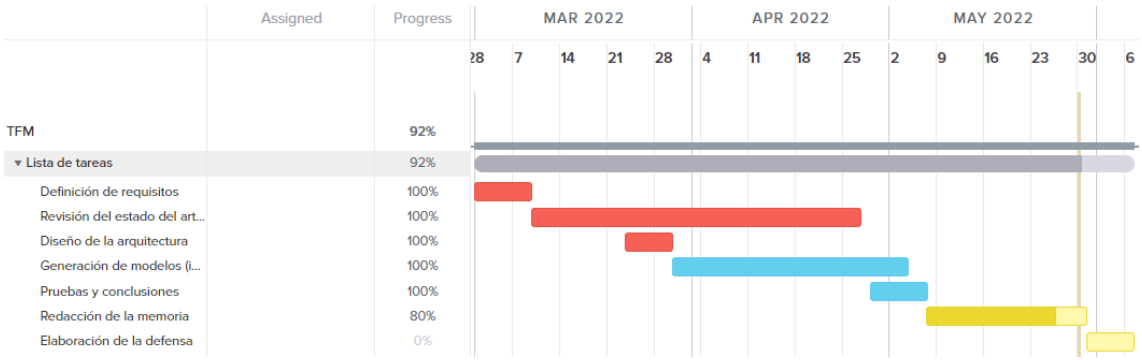

Ilustración 23. Cronograma final de tareas

El único cambio apreciable respecto al cronograma inicial ha sido el plazo de finalización de la tarea “Revisión del estado del arte y justificaciones tecnológicas”, que ha sido alargado hasta el 26 de abril.

Este cambio se debe a que el director del proyecto, tras la revisión del estado del arte durante la segunda entrega de seguimiento, concluyó que era un estudio incompleto y debía ser más exhaustivo, por lo que el alumno lo realizó de nuevo y lo entregó otra vez en la tercera entrega de seguimiento. Otras modificaciones del proyecto debido al feedback del director se pudieron implementar en los plazos establecidos inicialmente (como la adición de el método de evaluación de modelos mediante *cross-validation*).

6. RESULTADOS OBTENIDOS

Este capítulo tiene como objetivo mostrar los experimentos realizados, así como los resultados obtenidos. También se incluyen las conclusiones extraídas de ellos.

6.1 KDDCUP 99

En primer lugar, se han realizado las pruebas sobre el dataset KDDCup 99. Este dataset contiene la información de varias conexiones, a cada cual se le asigna una amenaza en particular o ninguna. El único tratamiento que se ha dado al dataset ha sido la creación de variables dummy para las columnas “protocol_type”, “service” y “flag” ya que contienen variables categóricas, además de la eliminación de valores nulos y duplicados.

Se ha calculado principalmente la precisión de cada algoritmo ya que, en los experimentos referenciados en el estado del arte, también se ha usado la precisión como la métrica principal.

6.1.1 Prueba inicial

En una primera iteración del proyecto, los *splits* de los datos se habían realizado a “mano” con los métodos de *train_test_split* de scikit-learn y después se habían entrenado los modelos de predicción para extraer la precisión de él. Este proceso se repetía 5 veces y se calculaba la media de las precisiones extraídas.

Por otra parte, el proceso de AutoML no pudo completarse del todo y hubo que extraer una tubería prematura debido a las restricciones de tiempo para la tercera entrega de seguimiento.

Tabla 1. Precisión media KDDCup 99 prueba inicial

Algoritmo	Precisión media
SVC lineal	0.961178119075748
Regresión logística	0.936472786218644
Árbol de decisión	0.999606496850355
KNN	0.994741324834464
Naive Bayes	0.945905462082813
MLP Classifier	0.981300656647524
Árbol de decisión + RBFSampler (Pipeline de AutoML)	0.996984210526342

Mediante este método, se llegó a la conclusión de que el mejor modelo de predicción era mediante árboles de decisión y TPOT, a pesar de su larga duración, había conseguido identificar esto de manera automática al ofrecer un modelo similar.

6.1.2 Prueba completa

El experimento se repitió siguiendo las recomendaciones del director del proyecto utilizando el método de evaluación de modelo de *cross-validation* con un valor k de 5 (que divide el dataset en 5 grupos y después escoge uno como test y los otros 4 como entrenamiento). Aunque el experimento ya se ha realizado de esta manera (pero se han divido de manera manual), vale la pena intentarlo con un método específicamente preparado para la realización de este tipo de operaciones.

Por otra parte, se otorgó el tiempo necesario al método de TPOT para que pudiese terminar el proceso de AutoML y ofrecer la tubería óptima. Esta búsqueda se realizó con los siguientes parámetros:

- *Generations = 5*. Número de iteraciones que se realiza el proceso de optimización de la tubería.
- *Population_size = 50*. Número de "individuos" que mantienen la programación genética cada generación.
- Cv = 5. Definimos que el entrenamiento de modelos debe hacerse mediante *cross-validation* con una valor k de 5 (como en los algoritmos tradicionales).
- *Scoring = "Accuracy"*. La métrica con la que evaluar los modelos, elegimos la precisión al igual que con los algoritmos tradicionales.

Cabe destacar que la duración total del proceso de búsqueda fue de 11 horas y 43 minutos. En la siguiente tabla se muestra la mejor tubería encontrada, además de los resultados del resto de algoritmos aplicando *cross-validation*.

Tabla 2. Precisión media KDDCup 99 prueba completa

Algoritmo (Cross-validation K=5)	Precisión media
SVC lineal	0.971082363530828
Regresión logística	0.936422544010613
Árbol de decisión	0.998894123679964
KNN	0.994793460225979
Naive Bayes	0.829626515426512

MLP Classifier	0.988048291437741
Random Forest Classifier (Pipeline de AutoML)	0.9988551972940343

Para los algoritmos tradicionales, excepto en el caso de Naive Bayes, se han obtenido precisiones medias marginalmente diferentes que en el experimento inicial. Esto se debe simplemente a pequeñas variaciones estadísticas a la hora de entrenar el modelo. Aun así, los resultados obtenidos siguen la misma tendencia respecto a la precisión y a la velocidad y sirven para dotar de mayor seguridad a los resultados previos.

Con el proceso terminado, TPOT eligió una tubería diferente que la obtenida en un primer momento. Si bien no es exactamente un árbol de decisión*, RandomForestClassifier* es una evolución de este tipo de modelos que simplemente combina varios árboles de decisión. Esta es una decisión lógica, ya que este modelo actúa parecido a un árbol de decisión, pero ayuda a reducir problemas como el overfitting.

De manera más específica, los hiperparámetros principales elegidos han sido *n_estimators=100* (número de árboles de decisión en el bosque) y *max_features=0.15* (número máximo de características a tener en cuenta a la hora de realizar el mejor *split*).

Sorprendentemente, la tubería exportada no contiene ningún método previo para el procesado de los datos, solo el algoritmo de clasificación de bosques aleatorios.

6.1.2.1 Resultados de la tubería de TPOT

Con el objetivo de comprobar mejor el rendimiento del modelo seleccionado por TPOT, se ha procedido a realizar una serie de predicciones y visualizarlas en una matriz de confusión. En primer lugar, se han obtenido las métricas siguientes:

- *Accuracy = 0.998901008324862*
- *Precision = 0.998679958978441*
- *Recall = 0.998901008324862*

Estas concuerdan con la media obtenida en la tabla anterior y muestran un rendimiento bastante bueno. También generamos la matriz de confusión en la que las categorías predichas se muestran en el eje horizontal y las reales en el eje vertical.

A pesar de que en la diagonal existen muchos aciertos, el modelo tiene algunos problemas como la confusión de ataques del tipo *ipsweep* por ataques de *nmap* o la detección de falsos positivos identificando paquetes normales como un ataque de *warezclient*. Aún así es un modelo con un buen rendmiento.

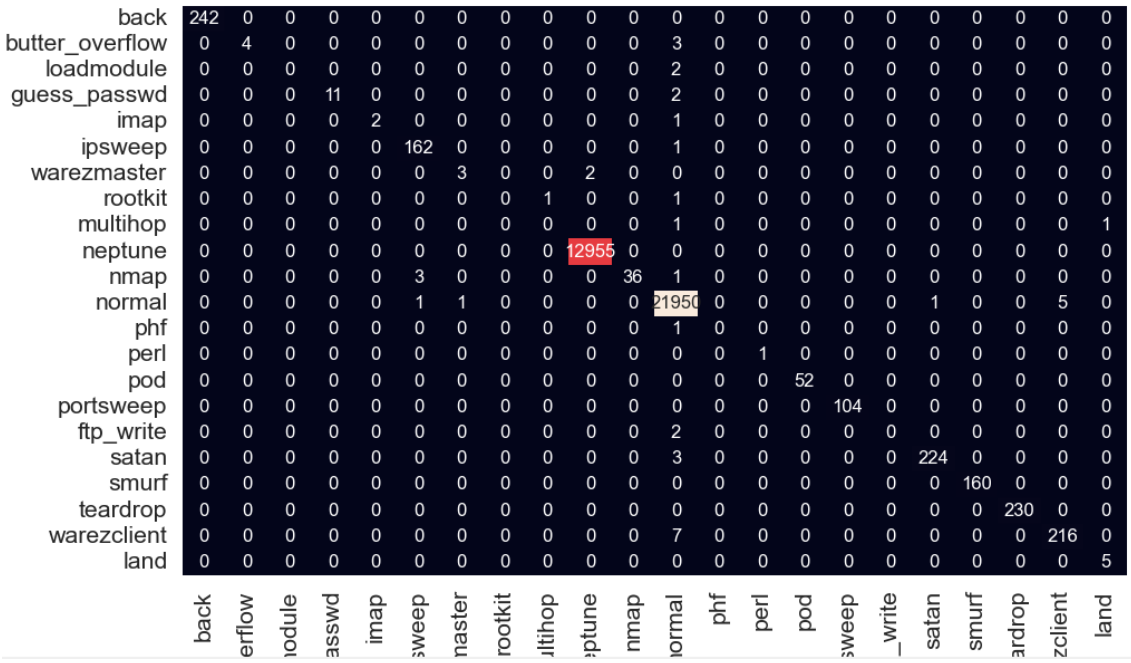

Ilustración 24. Matriz de confusión de RandomForestClassifier para KDDCUP 99

6.1.3 Conclusiones

Con las pruebas realizadas en el dataset, se han conseguido niveles de precisión comparables a los definidos en los artículos que han trabajado con este dataset. Los modelos lineales han tenido más problemas, sobre todo en la velocidad, a la hora de entrenar el modelo y realizar sus predicciones.

El mejor algoritmo para resolver el problema ha resultado ser *DecisionTreeClassifier*, lo cual concuerda con los experimentos de comparación de árboles de decisión con SVM (SVC lineal es parecido, pero se adapta mejor a datasets de gran tamaño) y con Redes Neuronales. En ambos experimentos se concluye que los árboles de decisión ofrecen un mejor rendimiento en general y se obtienen precisiones que rondan el 99%, al igual que en los resultados obtenidos en estos experimentos. La única excepción es el primero donde compara los árboles de decisión con Naive Bayes y obtiene precisiones parecidas, pero con un mejor rendimiento de Naive Bayes, lo cual ha ocurrido al contrario en las pruebas realizadas.

Respecto al uso de AutoML, en un primer momento, tenía elegido *DecisionTreeClassifier* como el algoritmo óptimo para la resolución del problema. Esto concuerda con los resultados obtenidos de ejecutar el algoritmo a mano. Una vez acabado, el algoritmo elegido ha sido *RandomForestClassifier* que es prácticamente el mismo concepto y también ha mostrado una precisión muy alta.

Se concluye que, si se dispone de los recursos de computación adecuados, el uso de técnicas de AutoML a través de TPOT puede tener resultados muy positivos a la hora de realizar predicciones y no requiere de demasiados conocimientos técnicos profundos para su implementación, eliminando así la parte humana de prueba y error de optimización que estos problemas normalmente requieren.

6.2 CICIDS2017

El segundo experimento se ha realizado sobre el primero de todos los datasets CICIDS2017. Este dataset contiene la información sobre conexiones como el primero, pero solo clasifica las conexiones como normales o como DDoS. Este es un problema de clasificación binaria. El dataset solo ha necesitado operaciones de limpieza como la eliminación de valores nulos, infinitos y duplicados.

Aunque no se haya mencionado en el estado del arte, el experimento con el que se van a comparar los resultados es uno bastante similar realizado en 2019 sobre el mismo dataset [24]. En este experimentos se calculan las métricas *accuracy*, *precisión*, *recall* y *f-score* sobre todos los datasets de CICIDS2017 de árboles de decisión, bosques aleatorios, naive bayes y SVM.

Por motivos de tiempo y computación, se ha decido calcular las métricas sobre el primer dataset. A pesar de esto, no se pierde mucha información ya que los algoritmos tienden a rendir de manera parecida en el resto de datasets. Los resultados del primero son representativos.

6.2.1 Prueba

Al igual que en el experimento anterior, se han realizado las pruebas mediante el uso de cross-validation con un valor de k de 5 y se han calculado las métricas correspondientes sobre los resultados.

El proceso para encontrar la tubería óptima de ML se ha realizado con los mismos parámetros que en el experimento anterior y se ha esperado hasta su finalización completa. Una vez creada la tubería, se ha copiado el modelo y los hiper parámetros y se le ha aplicado *cross-validation* como a los algoritmos tradicionales.

Los resultados obtenidos se recogen en la siguiente tabla.

Tabla 3. Resultados de CICIDS2017

Algoritmo	Accuracy	Precision	Recall	F-score
SVC lineal	0.942145	0.946418	0.942125	0.942059
Regresión logística	0.932823	0.937349	0.932818	0.932027

Árbol de decisión	0.999860	0.999796	0.999146	0.999860
KNN	0.978437	0.975226	0.978297	0.971595
Naive Bayes	0.814670	0.859008	0.814670	0.801959
MLP Classifier	0.931207	0.941818	0.931203	0.928977
Random Forest Classifier (Pipeline de AutoML)	0.999903	0.999903	0.999889	0.999811

Podemos observar en los resultados que existe una distribución similar a las del primer experimento. Son los árboles de decisión y los bosques aleatorios los algoritmos que mejor rendimiento muestran mientras que los modelos lineales tienen problemas de velocidad, aunque realicen unas predicciones buenas.

La tubería elegida por TPOT, de nuevo, es un bosque aleatorio con los hiper parámetros ligeramente modificados y ofrece casi los mejores resultados de todos los algoritmos. La ligera diferencia entre este y el árbol de decisión puede deberse a simple margen estadístico y es probable que en otro set de iteraciones hubiese superado al árbol de decisión.

Los hiperparámetros principales elegidos han sido *n_estimators=100* (número de árboles de decisión en el bosque), *max_features=0.15002* (número máximo de características a tener en cuenta a la hora de realizar el mejor *split*) y *min_samples_leaf=3* (el mínimo número de muestras requerido para ser un nodo *leaf*.

Al igual que en el anterior experimento, la tubería tampoco cuenta con un método previo de procesado de datos. Esto puede deberse a que los propios datos del dataset no están balanceados y los valores no difieren mucho entre sí.

6.2.1.1 Resultado de la tubería TPOT

Esta vez ya tenemos los resultados generados por la tubería TPOT, aun así, podemos probar a visualizar las predicciones del modelo. Como el problema de clasificación es binario, se puede realizar la curva ROC. Aun así, debido a la cantidad minúscula de errores, ya se puede intuir que la curva ROC será prácticamente perfecta

Ilustración 26. Curva ROC de RFC para CCIDS2017

También podemos extraer la matriz de confusión de un conjunto de predicciones para comprobar en qué falla el modelo exactamente. En la matriz se comprueba que el modelo solo falla en la predicción de algunas conexiones como falsos negativos y asigna la categoría de “Benign” a conexiones que en realidad son DDoS. Por el contrario, el modelo no muestra ningún caso que sea un falso positivo.

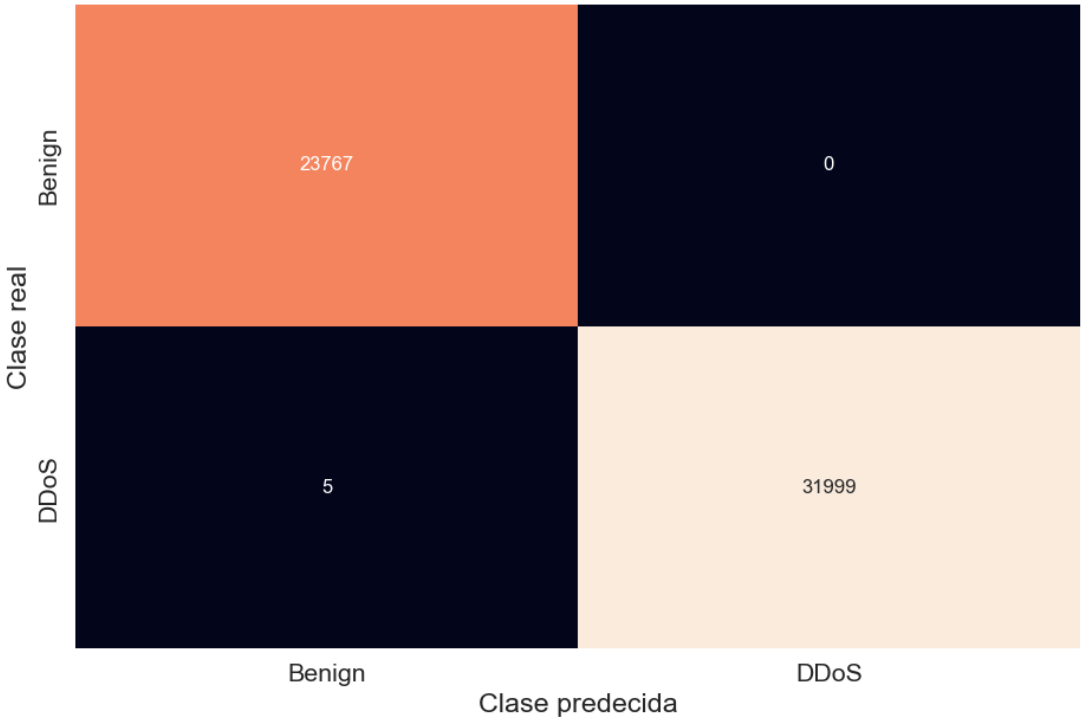

Ilustración 25. Matriz de confusión de RFC para CICIDS2017

6.2.2 Conclusiones

Los resultados extraídos en el experimento coinciden con los mostrados en el artículo del dataset número uno. Los modelos que ofrecen los mejores resultados en el artículo son los basados en árboles de decisión y en los bosques aleatorios mientras que naive bayes ofrece el peor rendimiento en general. La precisión de los árboles y el bosque ronda el 99% como en el artículo y, si bien no se ha utilizado exactamente SVM, SVC lineal tiene un funcionamiento similar así que sus resultados son comparables (solo hay una diferencia de 2%).

Como se ha mencionado antes, estos resultados son trasladables al resto de los datasets ya que en el propio artículo se puede ver que los árboles de decisión y el bosque aleatorio siempre demuestran tener las mejores predicciones, así que es esperable que suceda lo mismo si se analizase el resto.

También se puede solidificar la opinión de que las técnicas de AutoML pueden llegar prácticamente al mejor modelo de predicción introduciendo únicamente el conjunto de datos de entrenamiento sin necesidad de ninguna ayuda o indicación por parte del usuario más allá de definir los parámetros. El único problema es el tiempo requerido para encontrar la tubería ya que tarda bastantes horas en completar la búsqueda.

7. CONCLUSIONES Y RESULTADOS

Este capítulo recoge las conclusiones, lecciones aprendidas y posibles mejoras que el proyecto ha supuesto para el alumno.

7.1 CONCLUSIONES

El proyecto realizado y descrito en la memoria ha supuesto un gran reto para el alumno ya que no solo suponía el uso de un amplio abanico de nuevas tecnologías, sino que también ha sido necesaria una mayor disciplina a la hora de desarrollar un proyecto de mayores dimensiones.

En el caso de uso de nuevas tecnologías, muchas de las herramientas para el aprendizaje y la aplicación de Machine Learning han supuesto algo novedoso para el alumno. A pesar de esto, existían muchos recursos de aprendizaje en Internet que se han aprovechado. En contraste, a la hora de buscar información sobre las técnicas de AutoML, los recursos disponibles eran más escasos. Esto ha supuesto una dificultad para el alumno ya que tenía que sacar las respuestas que buscaba directamente de la documentación oficial en la mayoría de los casos o, incluso, al mirar el código fuente de las librerías que ha utilizado lo que ha fomentado la autonomía y la capacidad de búsqueda aprendida durante el máster

En lo referente al propio desarrollo del proyecto, al ser un proyecto que demandaba una mayor cantidad de trabajo y de manera más autónoma, el alumno ha seguido una metodología de trabajo para el desarrollo. Si bien no ha sido una demasiado estricta, ha servido para seguir una agenda de trabajo más equilibrada durante el semestre.

Como reflexión final, se puede afirmar que se ha conseguido el objetivo principal del proyecto de analizar la viabilidad real de las tecnologías actuales de AutoML y si pueden ser aplicadas al dominio de la detección de intrusiones por red. En los experimentos se ha demostrado que estas técnicas son perfectamente capaces de obtener un modelo de predicción que tenga un rendimiento bueno sobre el dataset escogido sin necesidad de interacción del usuario. También se ha mostrado que la principal desventaja de estas técnicas son el tiempo necesario para escoger la tubería adecuada.

7.2 TRABAJO FUTURO

Existen varias vías por las que profundizar en este análisis.

Respecto a las fuentes de datos escogidas:

- Realizar las pruebas sobre un número mayor de datasets y expandir la funcionalidad a otros dominios de la ciberseguridad (como el análisis de logs para la detección de anomalías).

Respecto a los algoritmos utilizados:

- Realizar las pruebas con los métodos que ofrezcan otras librerías de AutoML de las mencionadas como alternativas a TPOT.
- Introducir un número mayor de algoritmos sobre los que realizar las pruebas.
- Buscar técnicas más avanzadas de AutoML que permita escoger los modelos de manera dinámica para adaptarlo a las nuevas necesidades de Machine Learning como cuando se dispone de un flujo de datos constante (*data streams*) y no de un dataset estático.

8. BIBLIOGRAFÍA

[1]

«El trabajo de fin de grado y de máster», *Editorial UOC*. https://www.editorialuoc.com/el-trabajo-de-fin-de-grado-y-de-master.

[2]

«Automated machine learning», *Wikipedia*. 6 de mayo de 2022. [En línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=Automated_machine_learning&oldid=1086484644

[3]

B. Mahesh, *Machine Learning Algorithms -A Review*. 2019. doi: 10.21275/ART20203995.

[4]

K. Shaukat, S. Luo, V. Varadharajan, I. A. Hameed, y M. Xu, «A Survey on Machine Learning Techniques for Cyber Security in the Last Decade», *IEEE Access*, vol. 8, pp. 222310-222354, 2020, doi: 10.1109/ACCESS.2020.3041951.

[5]

«A review of machine learning approaches to Spam filtering - ScienceDirect». https://www.sciencedirect.com/science/article/pii/S095741740900181X.

[6]

J. Sahs y L. Khan, «A Machine Learning Approach to Android Malware Detection», en *2012 European Intelligence and Security Informatics Conference*, ago. 2012, pp. 141-147. doi: 10.1109/EISIC.2012.34.

[7]

C. Kruegel y T. Toth, «Using Decision Trees to Improve Signature-Based Intrusion Detection», en *Recent Advances in Intrusion Detection*, Berlin, Heidelberg, 2003, pp. 173-191. doi: 10.1007/978-3-540-45248-5_10.

[8]

N. B. Amor, S. Benferhat, y Z. Elouedi, «Naive Bayes vs decision trees in intrusion detection systems», en *Proceedings of the 2004 ACM symposium on Applied computing*, New York, NY, USA, mar. 2004, pp. 420-424. doi: 10.1145/967900.967989.

[9]

S. Peddabachigari, A. Abraham, y J. Thomas, «Intrusion detection systems using decision trees and support vector machines», *International Journal of Applied Science and Computations*, vol. 11, ene. 2004.

[10]

Y. Bouzida y F. Cuppens, «Neural networks vs. decision trees for intrusion detection», ene. 2006.

[11]

E. Hodo *et al.*, «Threat analysis of IoT networks using artificial neural network intrusion detection system», en *2016 International Symposium on Networks, Computers and Communications (ISNCC)*, may 2016, pp. 1-6. doi: 10.1109/ISNCC.2016.7746067.

[12]

J. Li, C. Manikopoulos, J. Jorgenson, y J. Ucles, «HIDE: a Hierarchical Network Intrusion Detection System Using Statistical Preprocessing and Neural Network Classification», ene. 2001.

[13]

M. Bahri, F. Salutari, A. Putina, y M. Sozio, «AutoML: state of the art with a focus on anomaly detection, challenges, and research directions», *Int J Data Sci Anal*, feb. 2022, doi: 10.1007/s41060-022-00309-0.

[14]

X. He, K. Zhao, y X. Chu, «AutoML: A survey of the state-of-the-art», *Knowledge-Based Systems*, vol. 212, p. 106622, ene. 2021, doi: 10.1016/j.knosys.2020.106622.

[15]

«Cloud AutoML: modelos personalizados de aprendizaje automático», *Google Cloud*. https://cloud.google.com/automl?hl=es.

[16]

«Automated Machine Learning | Microsoft Azure». https://azure.microsoft.com/en-us/services/machine-learning/automatedml/.

[17]

Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, y S. Han, «AMC: AutoML for Model Compression and Acceleration on Mobile Devices», 2018, pp. 784-800. [En línea]. Disponible en: https://openaccess.thecvf.com/content_ECCV_2018/html/Yihui_He_AMC_Automated_Model_ECCV_2018_paper.html

[18]

R. Purwanto, A. Pal, A. Blair, y S. Jha, «Man versus Machine: AutoML and Human Experts' Role in Phishing Detection», *arXiv:2108.12193 [cs]*, ago. 2021. [En línea]. Disponible en: http://arxiv.org/abs/2108.12193

[19]

W. Saeed, «Comparison of Automated Machine Learning Tools for SMS Spam Message Filtering», *arXiv:2106.08671 [cs]*, jun. 2021. [En línea]. Disponible en: http://arxiv.org/abs/2106.08671

[20]

«TPOT», *AutoML*. http://automl.info/tpot/

[21]

«scikit-learn: machine learning in Python — scikit-learn 1.0.2 documentation». https://scikit-learn.org/stable/.

[22]

«KDD Cup 1999 Data». http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

[23]

«CICIDS2017». https://www.kaggle.com/cicdataset/cicids2017.

[24]

A. Alsaeedi y M. Zubair, «Performance Analysis of Network Intrusion Detection System using Machine Learning», *IJACSA*, vol. 10, n.º 12, 2019, doi: 10.14569/IJACSA.2019.0101286.