# Análisis de Tráfico en Dispositivos Móviles mediante Técnicas de Aprendizaje Profundo No Supervisado

# Traffic Analysis on Mobile Devices Using Unsupervised Deep Learning Techniques


**TRABAJO FIN DE GRADO**
**GRADO EN INGENIERÍA INFORMÁTICA**
**CURSO 2021–2022**

**Gema Blanco Núñez**
**Diego Atance Sanz**
**César Ureña Toledano**

*Directores*
**Luis Javier García Villalba**
**Sandra Pérez Arteaga**

Departamento de Ingeniería del Software e Inteligencia Artificial
Facultad de Informática
Universidad Complutense de Madrid

Madrid, Junio de 2022

# Agradecimientos

Nos gustaría agradecer a nuestros directores de proyecto, Luis Javier García Villalba y Sandra Pérez Artaga, por animarnos a formar parte de este gran proyecto.

A Ana L. Sandoval, por su apoyo y motivación constante y su paciencia.

A Luís Martinez, por su apoyo semanal y sus imprescindibles aportaciones y comentarios que han resuelto nuestras dudas.

A Daniel Povedano, por sus innumerables consejos.

A la Facultad de Informática, así cómo a todos aquellos profesores que, durante los 4 últimos años, y gracias a los conocimientos aportados, han hecho posible el desarrollo de este proyecto.

# Índice General

# Índice de Figuras

# Índice de Tablas

# Lista de Acrónimos

| | |
|---|---|
| AI | Artificial Intelligence |
| ARPANET | *Advanced Research Projects Agency Network* |
| CDN | *Content Delivery Network* |
| CNN | *Convolutional Neural Networks* |
| DM | *Data Mining* |
| FN | *False Negatives* |
| FP | *False Positives* |
| FTP | *File Transfer Protocol* |
| GCN | *Graph Convolutional Networks* |
| GNN | *Graph Neural Networks* |
| GPU | *Graphical Processing Unit* |
| HTTP | *HyperText Transfer Protocol* |
| HTTPS | *HyperText Transfer Protocol Secure* |
| IA | Inteligencia Artificial |

| | |
|---|---|
| IEEE | Institute of Electrical and Electronics Engineers |
| IoT | *Internet of Things* |
| IP | *Internet Protocol* |
| IPv4 | *Internet Protocol version 4* |
| IPv6 | *Internet Protocol version 6* |
| ISO | *International Organization for Standardization* |
| MitM | *Man in the Middle* |
| ML | *Machine Learning* |
| MLP | *MultiLayer Perceptron* |
| OSI | *Open Systems Interconnection* |
| PDU | *Protocol Data Unit* |
| POODLE | *Padding Oracle On Downgraded Legacy Encryption* |
| RBF | *Radial Basis Function* |
| RSA | *Rivest–Shamir–Adleman* |
| SVM | *Support vector machine* |
| TCP | *Transmission Control Protocol* |
| TCP/IP | *Internet protocol Suite* |
| TLS | *Transport Layer Security* |
| TN | *True Negatives* |
| TP | *True Positives* |

| | |
|---|---|
| UDP | *User Datagram Protocol* |
| XGBoost | *eXtreme Gradient Boosting* |

# Abstract

Today, due to the growth of mobile technologies is possible to do daily activities such as work from home, take online classes, talk with family or friends and playing remotely from a smartphone. Nevertheless, mixing this things of our lives, result attractive to a cybercriminal due to in many occasions, in one same mobile device there is the possibility to have applications of exclusive use for an enterprise such as email or digital repositories and at the same time, these interact with leisure applications potentially dangerous for the security of any organization. These actions can produce data filtration or confidential documents, until the device is compromised by a cybercriminal and has a way to the internal data of an enterprise. There exist different ways of minimizing risks, one of the most common ones in any organization is network monitoring , nevertheless, the use of data encryption algorithms of the applications is not enough due to the majority of the applications using robust encryption algorithms to hide the data so is difficult for the network analysts work. Nowadays, there exist solutions that allow to show the encrypted communication content by modules to weaken the encryption algorithms to vulnerable versions, that can take part in the integrity of network messages. Taking this into account this work presents a tool for being able of classify mobile applications traffic by the network behaviour using non supervised techniques.

**Keywords**: Data Science, Artificial Intelligence, Neural Networks

# Resumen

En la actualidad, debido al auge de las tecnologías móviles es posible realizar tareas cotidianas como trabajar, tomar clases en línea, conversar con familiares o amigos y jugar de manera remota desde un smatphone. Sin embargo, mezclar estos ámbitos de nuestra vida resulta atractivo a un ciberdelincuente debido a que en muchas ocasiones en un mismo equipo móvil se tienen aplicaciones de uso exclusivo de una empresa como correo electrónico o repositorios digitales y estas a su vez interactúan con aplicaciones de ocio o potencialmente peligrosas para la seguridad de cualquier organización. Estas acciones pueden propiciar desde una exfiltración de datos o documentos confidenciales, hasta que el equipo sea comprometido por un ciberdelincuente y tenga vía libre a los datos internos de la empresa. Existen diferentes formas de minimizar el riesgo, una de las comunes en cualquier organización es el monitoreo de la red, sin embargo, el uso de algoritmos de cifrado de los datos de las aplicaciones esta no es suficiente ya que la mayoría de las aplicaciones móviles utilizan robustos algoritmos de cifrado para ocultar sus datos lo cual dificulta la labor de los analistas de Red. En la actualidad existen soluciones que permiten visualizar el contenido de las comunicaciones cifradas mediante módulos para debilitar los algoritmos de cifrado a versiones vulnerables, lo cual podría repercutir en la integridad de los mensajes de red. Por lo anterior en el presente trabajo se muestra una herramienta capaz de clasificar tráfico de aplicaciones móviles mediate su comportamiento en la red utilizando técnicas no supervisadas de Deep Learning.

**Palabras clave**: Tráfico de Red, Inteligencia Artificial, Redes Neuronales

# Capítulo 1

# Introducción

## 1.1. Motivación

Desde el nacimiento de Internet en 1972, desarrollado por el Departamento de Defensa de los Estados Unidos y probado sobre la red de área extensa *Advanced Research Projects Agency Network* (ARPANET) [4], se han producido grandes avances tecnológicos que han permitido la masificación de este medio. Actualmente, el acceso a Internet es prácticamente imprescindible en el mundo occidental, siendo este una de las principales formas de comunicación, entretenimiento y trabajo entre las personas.

Debido a la gran importancia subyacente en este medio, diversas aplicaciones de *Android* son usadas en el día a día incluso como herramienta de trabajo. Por otra parte, el uso de estas aplicaciones ha propiciado la difusión información sensible que, en algunos casos, acaba siendo filtrada debido a posibles fallos en la seguridad de estas aplicaciones.

El aprendizaje automático o *machine learning* cuenta con numerosos campos de aplicación, entre ellos el análisis del tráfico de red. Mediante este análisis de datos puede extraerse información que, tratada de la forma adecuada y haciendo uso de algoritmos, permita clasificar el tráfico de las aplicaciones que circula por la red interna.

## 1.2. Objeto de la Investigación

En la actualidad, debido al auge de los dispositivos móviles, el acceso a internet se ha convertido en un medio imprescindible en la vida de las personas, tanto para el ámbito personal como el profesional. Por una parte, esto representa un gran peligro para la privacidad y seguridad de las personas ya que mediante el análisis del tráfico de red es posible extraer información acerca de los usuarios. Por otra parte, este análisis también ha derivado en aplicaciones prácticas beneficiosas para la sociedad como es la detección de ciberdelitos.

Los orígenes de la Inteligencia Artificial se remontan al año 1955 cuando se celebró la *Western Joint Computer Conference* en Los Ángeles y en la cual se organizó una sesión sobre *learning machines*. En este congreso se introdujo el término de redes neuronales y se expusieron trabajos relacionados con el reconocimiento de patrones [20].

Con el paso de los años gracias a la mejora del hardware, los avances en el campo de la Inteligencia Artificial y la gran cantidad de datos generados actualmente, han surgido

algoritmos capaces de identificar patrones en el tráfico de red a partir de los cuales llegar a unas conclusiones sobre los datos subyacentes. Así es el caso de un algoritmo diseñado para detectar los dispositivos *Internet of Things* (IoT) conectados a una red a partir del tráfico de red [24].

El propósito de este trabajo es el análisis de tráfico en dispositivos móviles basado en técnicas de aprendizaje profundo no supeevisado para tener la mayor precisión posible a partir de tráfico de red.

## 1.3. Plan de Trabajo

El desarrollo de este trabajo se ha realizado a lo largo del período que comprende el curso 2021-2022 y se ha dividido en tres fases:

1. **Investigación**: los primeros meses, desde septiembre hasta diciembre, los miembros del equipo se centraron en adquirir conocimientos acerca del contexto y en investigar otros trabajos relacionados con el tema del proyecto. Durante esta fase todos los alumnos involucrados en el trabajo compartieron conocimientos y dudas a través de videollamadas celebradas peródicamente cada semana a través de *Google Meet*. Además, estas reuniones también tuvieron el objeto de sincronizar a todos los miembros acerca de los avances producidos y compartir herramientas que facilitaron la investigación y el trabajo como es *Google Scholar*, gracias al cual se ha podido encontrar numerosos artículos cientícos y técnicos.

   Una vez se definieron los elementos bases del proyecto, se comenzó a realizar capturas de tráfico de red con *PCAPdroid* [13], una aplicación de Android que permite guardar los paquetes de red generados por una aplicación concreta. Estas capturas son las que recogen datos de aplicaciones Android específicas y que forman el dataset necesario para entrenar el algoritmo.

2. **Desarrollo**: tras asentar las bases y objetivos del proyecto y con los conocimientos adquiridos en la fase anterior se comenzó a definir los siguientes pasos que permitirían desarrollar un algoritmo capaz de detectar aplicaciones a partir de tráfico de red. El lenguaje de programación elegido para el desarrollo fue Python debido a las altas prestaciones en Inteligencia Artificial. A pesar de que esta fase se centra en el desarrollo, la investigación no cesó y se mantuvo enfocada a la investigación sobre conceptos de *Python* y librerías como *Keras* [5], *Tensor Flow* [1] y *Scapy*, la cual permitió la manipulación de los paquetes de red. Además, también continuó activa la captura de tráfico para la generación del dataset.

3. **Experimentación**: una vez finalizada la implementación del modelo se procesó el dataset de capturas de tráfico y se entrenó el modelo con distintos parámetros con el objetivo de obtener los mejores resultados.

En la Figura 1.1 se muestra un diagrama de Gantt con los períodos que comprenden cada una de las principales tareas llevadas a cabo durante el desarrollo del proyecto.


Figura 1.1: Diagrama de Gantt del Proyecto

## 1.4. Estructura del Trabajo

Se compone de 6 capítulos y 4 anexos que siguen la estructura descrita a continuación:

El Capítulo 1 es una introducción en la que se expone la motivación, el contexto, el objeto de la investigación, el plan y estructura de trabajo. En él se ofrece una visión general de temas que son tratados en profundidad a lo largo de los próximos capítulos.

El Capítulo 2 trata conceptos básicos para la comprensión del análisis del tráfico de red. Entre los temas tratados se encuentra una introducción al protocolo *Internet protocol Suite* (TCP/IP) y al modelo *Open Systems Interconnection* (OSI) utilizado como modelo de referencia para la transmisión de información en Internet; el funcionamiento del encaminamiento o *routing* y el protocolo *Transport Layer Security* (TLS).

El Capítulo 3 introduce términos relacionados con la fase de entrenamiento del proyecto. En este capítulo se introduce el concepto de Inteligencia Artificial (IA), se muestran los tipos de aprendizaje automático que existen y algoritmos relacionados con cada uno de ellos, así como los grafos como método para la representación de datos y se explica el funcionamiento de las redes neuronales convolucionales. Asimismo, también se explican las principales fases que constituyen proyectos de análisis de tráfico de red.

El Capítulo 4 expone el estado del arte del proyecto, es decir, las investigaciones realizadas sobre el análisis de tráfico de red y la detección de aplicaciones móviles mediante IA, así como otros trabajos relacionados. Además, se abordarán de manera general algunas de las herramientas analizadas por los autores del presente proyecto.

El Capítulo 5 es el de contribuciones y experimentos, en él se describe el modelo propuesto para la detección de aplicaciones móviles mediante el análisis de tráfico de red cifrado así como los experimentos realizados y los resultados obtenidos.

El Capítulo 6 hace referencia a las conclusiones.

El Capítulo 7 presenta las contribuciones de cada miembro al Proyecto.

Los Capítulos 8 y 9 corresponden a la traducción al inglés de la Introducción y las Conclusiones.

# Capítulo 2

# Conceptos básicos sobre redes

En este capítulo se explican conceptos teóricos involucrados en el análisis del tráfico de red que permitirán al lector tener una mayor comprensión de términos a los que se refiere en capítulos posteriores. En la sección 2.1 se hace referencia a los principios sobre los que se genera el tráfico de red, ofreciendo aspectos relevantes en cuanto al modelo OSI, el protocolo TCP/IP, las redes móviles y las características del tráfico generado por dispositivos móviles.

## 2.1. Tráfico de red

La comunicación entre dispositivos tecnológicos a través de Internet se basa en el intercambio de información encapsulada en lo que se conoce como paquetes de red o paquetes de datos, los cuales siguen una estructura y reglas definidas en el modelo de referencia OSI [7] (Ver Sección 2.1.1) y el protocolo estándar TCP/IP[31] (Ver Sección 2.1.2) utilizado en redes.

Una de las tecnologías más extendidas actualmente que da soporte a la conectividad de dispositivos a Internet es WiFi. Mediante esta tecnología los paquetes viajan de un origen, donde el emisor genera la información, a un destino, donde el receptor recibe la información, atravesando puntos intermedios conocidos como nodos o routers.

### 2.1.1. Modelo OSI

El modelo OSI es un modelo de referencia definido por *International Organization for Standardization* (ISO) para actividades de red que utiliza siete capas estructuradas. A partir de este modelo otros protocolos de red, como el protocolo TCP/IP (Ver Sección 2.1.2), definen sus capas.

Los datos se transfieren de una capa a otra y cada una de las capas en el modelo OSI tiene asociados unos protocolos determinados. A continuación se describen las capas de este modelo desde la inferior (capa 1) hasta la superior (capa 7) [26]:

1. **Capa Física:** define las características físicas del hardware, es decir, el medio físico de transmisión (entre ellos cable coaxial, fibra óptica...), las señales eléctricas y la transmisión del flujo de bits...

2. **Capa de Enlace:** este nivel administra la transferencia de datos en el medio de red, es decir, permite que los elementos físicos se comuniquen de forma correcta. En esta capa destacan los siguientes estándares del Institute of Electrical and Electronics Engineers (IEEE):
   - IEEE 802 para las conexiones LAN
   - IEEE 802.11 para las conexiones WiFi
3. **Capa de Red:** administra las direcciones de datos y la transferencia entre redes, es decir, se encarga del encaminamiento de los datos.
4. **Capa de Transporte:** administra la transferencia de datos y garantiza que la entrega sea correcta.
5. **Capa de Sesión:** administra las conexiones y terminaciones entre los sistemas que cooperan para que los enlaces entre las máquinas permanezcan activos mientras se transfiere la información.
6. **Capa de Presentación:** esta capa se encarga de transferir al sistema receptor la información de un modo comprensible para el sistema.
7. **Capa de Aplicación:** el último nivel se compone de los servicios y aplicaciones de comunicación estándar y que permite a los usuarios ejecutar acciones y comandos.

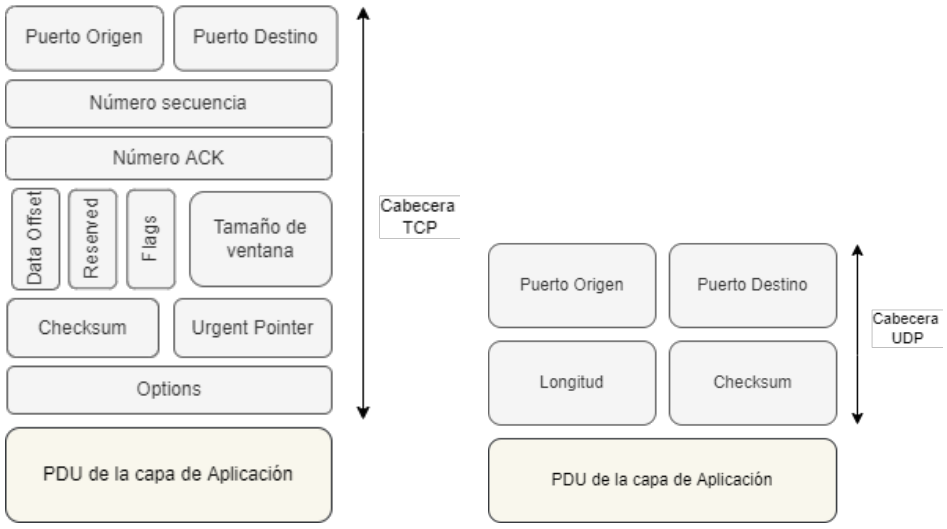
### 2.1.2. Protocolo TCP/IP

La comunicación en Internet es facilitada por un conjunto de protocolos conocidos como TCP/IP.

El protocolo TCP/IP fue desarrollado en 1972 por la Agencia de investigación de Proyectos Avanzados de Defensa de los Estados Unidos y probado sobre ARPANET, una red de área extensa propiedad del departamento. Desde entonces surgieron varias versiones estabilizándose con la cuarta versión de TCP/IP (IPv4) en 1983, que es mantenida hasta la actualidad por su robustez y escalabilidad [4].

Este protocolo se compone de cuatro capas [31]:

- **Capa de aplicación:** define los protocolos de bajo nivel (por ejemplo, *HyperText Transfer Protocol* (HTTP) para el acceso a páginas web o *File Transfer Protocol* (FTP) para la transferencia de archivos) usados por la aplicaciones para intercambiar y transmitir datos.
- **Capa de transporte:** recibe los datos de la capa de aplicación, los divide en fragmentos más pequeños denominados paquetes y añade los datos necesarios para enviar la información a la capa de red. Dentro de esta capa destacan dos protocolos: *Transmission Control Protocol* (TCP) y *User Datagram Protocol* (UDP). La principal diferencia entre ambos es que TCP ofrece servicios orientados a conexión, es decir, fiables (comprobación de paquetes perdidos, detección de errores, control del flujo...) mientras que UDP no ofrece servicios orientados a conexión (no fiable). En las siguientes imágenes puede observarse la estructura de los datagramas UDP y TCP mostrados en la Figura 2.1.




(a) Datagrama TCP (b) Datagrama UDP

Figura 2.1: Formatos de datagrama TCP y UDP

- **Capa de red:** añade los *Protocol Data Unit* (PDU) de la capa de transporte a la cabecera *Internet Protocol* (IP), lo que se conoce con el nombre de datagramas IP y los transmite a la capa de interfaz de red. Este datagrama contiene toda la información necesaria para que el mensaje llegue a la dirección destino final de la forma más óptima. Esta dirección destino puede ser una dirección *Internet Protocol version 6* (IPv6) o *Internet Protocol version 4* (IPv4), dependiendo de la cual los campos y la estructura de la cabecera IP diferirán.
- **Capa de interfaz de red:** encapsula los datagramas IP en tramas IP y los transmite a la red.

En la Figura 2.2 se muestra la organización en capas del protocolo TCP/IP descritas anteriormente. De esta forma, las cabeceras se van añadiendo desde la capa de aplicación hasta la capa de interfaz de red cuando el emisor envía un mensaje, mientras que se van quitando desde la capa de interfaz de red hasta llegar a la capa de aplicación cuando el receptor recibe el mensaje.

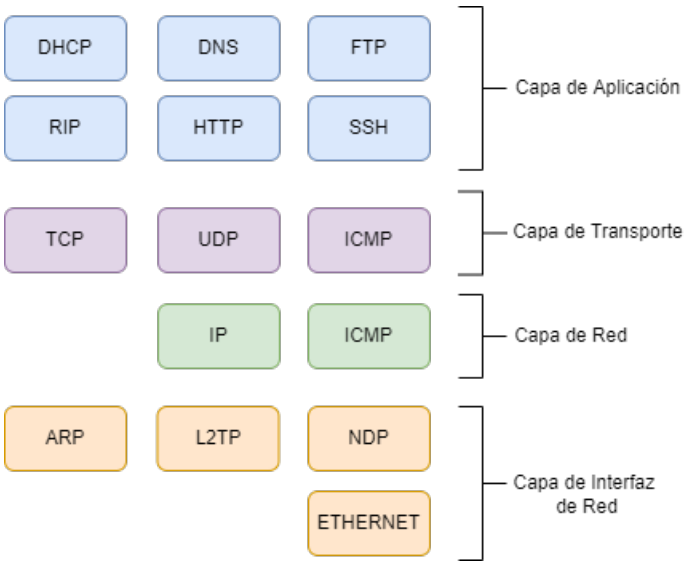



Figura 2.2: Familia de protocolos TCP/IP

### 2.1.3. Protocolo TLS

El protocolo TLS[15] da seguridad en las comunicaciones para evitar una intrusión de un atacante y con esto descifrar o modificar el mensaje de la comunicación.

Se suele usar conjuntamente en la capa de aplicación con el protocolo HTTP para formar el protocolo *HyperText Transfer Protocol Secure* (HTTPS) que da seguridad en el nivel de aplicación para páginas web y aplicaciones.

Hay muchas versiones que se pueden aplicar de este protocolo. Las ventajas de este protocolo de seguridad están disponibles si se usa de la manera correcta. Un error en el desarrollo de estos puede hacer que la comunicación no sea segura.

A continuación se muestran prácticas impropias al aplicar estos protocolos que pueden hacer que este no sea seguro.

#### 2.1.3.1. Practicas Impropias

A continuación se muestran algunas de las prácticas que pueden comprometer la seguridad de las comunicaciones[15]:

- **La falta de un certificado de validación:** hace que el protocolo sea vulnerable a ataques *Man in the Middle* (MitM). Estos ataques pueden tomar información o desencriptar información de la comunicación.
- **Ataque MitM en una conexión TLS:** cuando un atacante quiere meterse en medio de una comunicación, por ejemplo en un café con un wifi abierto, el atacante puede redirigir las comunicaciones entre un usuario y el router hacia otro dispositivo de su elección y conseguir todas las comunicaciones que van hacia él.

  En una conexión abierta el atacante puede ver y modificar el tráfico. Al estar cifrado, el destinatario no tiene ningún indicativo cuando le llega el mensaje. Sin embargo cuando se modifica el tráfico TLS, el certificado cuando el mensaje llega

al destinatario, no es el mismo. Para que el destinatario no se de cuenta, tiene que descifrar, obtener la información y volver a cifrar el mensaje. Esto solo se puede dar si hay alguna vulnerabilidad en la aplicación del protocolo, como por ejemplo no pedir un certificado de validación.

El certificado se aplica de la siguiente manera: el servidor le manda un certificado al cliente para que este pueda verificar que se trata del servidor esperado. Si el certificado de validación no pertenece a este servidor la conexión no es segura y la comunicación no se realiza. Después de esto se generan las claves de cifrado y se puede iniciar la transmisión.

- **Mal uso de los redireccionamientos en el protocolo HTTP:** sirven para redirigir al usuario de una página a otra. Se realiza una petición HTTP a un host y este redirige al usuario a otra dirección HTTP. Los redireccionamientos en HTTP a HTTPS pueden ser inseguros. Solo una petición de entrada a HTTP puede desvelar credenciales de usuario. Cuando el servidor quiere redirigir al usuario una página HTTPS segura, la conexión aun no lo es con lo que puede entrar un tercero y modificar el paquete de información para redirigir al usuario a un sitio idéntico al que quiere ir pero que no sea seguro ya que está en su dominio y de esta manera puede obtener sus credenciales fácilmente. La mejor práctica para evitar esto es intentar nunca entrar en páginas HTTP por parte del cliente.

- **Cifrados débiles y protocolos obsoletos:** ha pasado con el TLS y muchos más. Cuando un desarrollador no tiene en cuenta que atacantes pueden descubrir puntos débiles en la seguridad de los protocolos, se quedan obsoletos y comprometen la seguridad. Los cifrados y protocolos van mejorando a partir de ataques que pueden comprometer la seguridad.

  Los cifrados cada vez van siendo menos seguros si no evolucionan ya que la capacidad de cómputo cada vez es mayor y hace que el tiempo necesario para descubrir las claves sea cada vez menor para los atacantes. Depende de la longitud y fuerza de las claves.

  - ***Padding Oracle On Downgraded Legacy Encryption* (POODLE):** ataque que exploró la posibilidad de convertir TLS, que permitía al atacante descifrar el tráfico ya que este protocolo estaba obsoleto.
  - **HEARTBLEED:** bug que permitía descifrar contenido de TLS a través de la librería de código abierto OpenSSL que permitía a los atacantes leer la memoria de un servidor o cliente para obtener claves SSL privadas.
  - **ROBOT:** uso inseguro del método de encriptación *Rivest–Shamir–Adleman* (RSA). Este llevaba presente 19 años desde su implementación y permitía mediante un ataque MitM, realizar un descifrado del RSA utilizando las claves privadas de servidores TLS con vulnerabilidades.
  - **Usando una clave que ya no es privada:** el atacante podría ver la información de los usuarios que estén utilizando esa clave y los que la han usado pero ya no, también pueden ver su información comprometida.

  Si hay un agujero en la seguridad, taparlo lo antes posible es lo más seguro para los usuarios.

#### 2.1.3.2. Protocolo TLS v1.3

La versión 1.3 del protocolo TLS introduce mejoras en rendimiento y seguridad. Entre estas mejoras se encuentran[15]:

- Mejora su seguridad frente a ataques como el ROBOT.
- Mejora la velocidad de realizar el TLS handshake, que acelera la conexión.

Además, se han eliminado antiguos protocolos simétricos de encriptación. Todos los intercambios de clave pública ahora tienen *forward secrecy* (no compromete la seguridad de las claves usadas anteriormente). Ahora todos los mensajes están cifrados. Esto hace que el dominio y el certificado también lo estén.

#### 2.1.3.3. Perspectiva de la industria

Usos impropios de TLS[15] se esperan de desarrolladores independientes pero no de grandes compañías. Un estudio del proyecto CivilSphere muestra lo contrario. Se evaluaron varias aplicaciones móviles y el 81 % tuvo problemas con estas prácticas impropias. Se notificaron estos problemas a los desarrolladores y muchos de ellos nunca los revisaron. Son problemas comunes y normalmente no se hace nada por solucionarlos. Se realizan redireccionamientos HTTP con anunciantes y vendedores dentro de aplicaciones, muchos no utilizan certificado de validación o utilizan protocolos TLS desactualizados.

Esto concluye que las prácticas adecuadas para obtener las ventajas que nos proporciona el TLS son:

- Certificado de validación.
- Evitar redireccionamientos HTTP forzando el uso de HTTPS mediante el servidor.
- Claves privadas suficientemente grandes.
- Usar las últimas versiones de TLS.
- Si se detecta algún agujero de seguridad, mitigarlo cuanto antes utilizando una opción más segura.
- Revisar que el certificado de validación es válido y firmado por alguna autoridad certificada.

# Capítulo 3

# Aprendizaje Automático

## 3.1. Introducción

A lo largo de este Capítulo se introducen conceptos básicos sobre la Inteligencia Artificial que serán necesarios para la comprensión de los Capítulos posteriores. En la Sección 3.1.1 se comienza introduciendo el concepto de IA y en la Sección 3.1.2 se explican los principales modelos de la IA y una de sus posibles clasificaciones.

### 3.1.1. Inteligencia Artificial

La IA tiene varias definiciones de diferentes autores, entre ellas se define como:

- «El estudio de cómo lograr que las computadoras realicen tareas que, por el momento, los humanos hacen mejor» [12]
- «La IA es el arte de crear máquinas con capacidad de realizar funciones que realizadas por personas requieren de inteligencia» [18]

A pesar de contar con numerosas definiciones, todas tienen en común que el objetivo de la IA es lograr que una máquina tenga una inteligencia de tipo *general* similar a la humana [21].

Dependiendo del tipo de inteligencia que emulen, distinguimos dos tipos de IA [32]:

- **IA débil:** corresponde a la IA que tiene comportamiento inteligente en un entorno muy especializado.
- **IA fuerte:** corresponde a la IA que es capaz de *pensar* como un ser humano.

Según Searle, la IA fuerte es imposible de conseguir, pero cabe destacar que la IA general y la IA fuerte no son el mismo concepto. Existe IA general que emula la inteligencia general huamana sin llegar a ser IA fuerte.

### 3.1.2. Principales modelos de la Inteligencia Artificial

Dentro de la IA destacan cuatro modelos que se describen a continuación[21]:

1. **Modelo simbólico:** se trata de un modelo *top-down*, es decir, se parte de lo más general a lo más específico, que se basa en el razonamiento lógico y la *búsqueda heurística*.
2. **Modelo conexionista:** se trata de un modelo *bottom-up*, es decir, se parte de lo más específico a lo más general, que se basa en la hipótesis de que la inteligencia surge de la actividad dividida entre varias unidades interconectadas que procesan la información de forma paralela. Estas unidades son aproximaciones de las neuronas biológicas.
3. **Modelo evolutivo:** intenta emular el efecto de la evolución biológica dotando a los algoritmos de operaciones de mutación y cruce de *cromosomas*, de forma que se crearán nuevas generaciones de programas modificados cuyas soluciones son mejores que las de generaciones pasadas. Este modelo es principalmente usado en problemas de optimización.
4. **Modelo corpóreo:** surge a partir de la necesidad de eliminar una limitación de los modelos vistos anteriormente: la falta de capacidad sensorial y motora. De esta forma, se cree que el modelo corpóreo es el que permitirá avanzar hacia inteligencias de tipo general.

A su vez, estos modelos se pueden clasificar en dos grandes enfoques de la IA[10]:

- **IA simbólica:** se trata de la IA cognitiva basada en conocimiento con un enfoque deductivo *top-down*. Se corresponde con el modelo simbólico, la visión más clásica de la IA.
- **IA subsimbólica o conexionista:** se trata de la IA basada en datos con un enfoque inductivo *bottom-up*. Se corresponde con el modelo conexionesta. Dentro de este enfoque se encuentra el aprendizaje automático o *machine learning*, una disciplina de la IA desarrollada más en profundidad en la Sección 3.2.

Existen varias formas de clasificar los tipos de IA, en la Figura 3.1 se muestra un esquema general de cómo se clasifican las disciplinas de la IA que se explican en este Capítulo. El aprendizaje automático o *machine learning* es una rama de la IA y el aprendizaje profundo o *deep learning* es un subconjunto del aprendizaje automático.

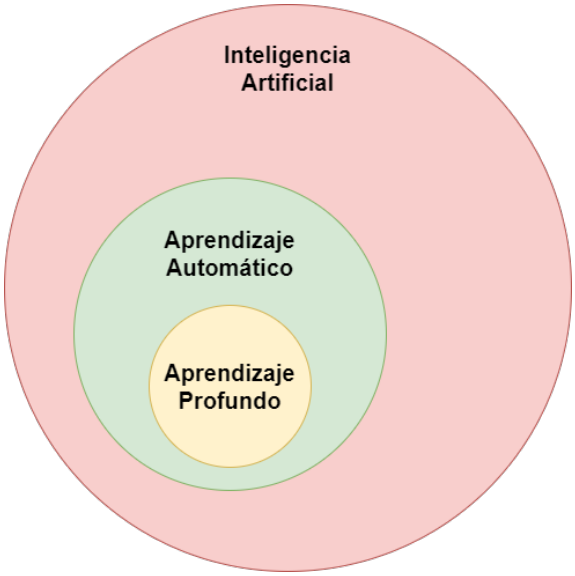



Figura 3.1: Clasificación de las disciplinas de la Inteligencia Artificial

## 3.2. Aprendizaje Automático

Existen 2 aproximaciones básicas para el *machine learning* o aprendizaje automático: **aprendizaje supervisado** (Sección 3.2.1) y **aprendizaje no supervisado** (Sección 3.2.2).

La principal diferencia entre ellos reside en que el aprendizaje supervisado utiliza conjuntos de datos o *datasets* etiquetados para el entrenamiento del modelo, es decir, se proporcionan las respuestas, etiquetas o variable objetivo, mientras que en el no supervisado solo se proporcionan las variables explicativas (independientes o de entrada) y es el algoritmo el que debe predecir las respuestas.

Lógicamente, esta diferencia básica de aproximación hace que el aprendizaje no supervisado sea superior para ciertos usos de *Machine Learning* (ML), pero en contraposición el tiempo que se emplea en la obtención de datos y la depuración es mayor.

### 3.2.1. Aprendizaje supervisado

El aprendizaje supervisado se caracteriza por usar *datasets* etiquetados. Dichos *datasets* se utilizan para entrenar los algoritmos de clasificación de forma precisa, pero se deben conocer los resultados correctos de la clasificación de forma previa, por lo que su uso no es siempre viable.

El aprendizaje supervisado puede utilizarse principalmente en dos tipos de problemas[11]:

1. **Clasificación:** En este tipo de problemas se usa un algoritmo para asignar categorías a los datos. Como ejemplos de algoritmos de clasificación encontramos clasificadores lineares, arboles de decisión o *random forest* (Sección 3.2.1.1).

2. **Regresion:** En este caso el algoritmo se basa en tratar de entender la relación entre

variables dependientes e independientes. Los modelos de regresión son muy útiles para tratar de predecir datos numéricos basándose en datos previos. Ejemplos de algoritmos de regresión son: modelo de regresión linear, logística y polinomial.

#### 3.2.1.1. Algoritmos de clasificación supervisados

##### 3.2.1.1.1. Random Forest y árboles de decisión

Random Forest [37] es un algoritmo de clasificación supervisado. La idea de este es combinar varios árboles de decisión individuales para conseguir un "bosque": que de un resultado más preciso y estable.

Los árboles de decisión son los componentes que contruyen el *Random Forest*. Estos, como cualquier árbol binario, estan formados por nodos y ramas. Los nodos serían los elementos del dataset y las ramas las decisiones que el árbol toma sobre los nodos. En la Figura 3.2 se puede ver un ejemplo de estos. En este caso se esta comprobando uno de los elementos del dataset (definido como un conjunto de ceros y unos, azules o rojos y que pueden estar subrayados). El objetivo del árbol de decisión es decidir qué componentes del dataset son rojos y cuales no y si estan subrayados o no. Finalmente, el dataset queda dividido según los criterios de decisión establecidos.

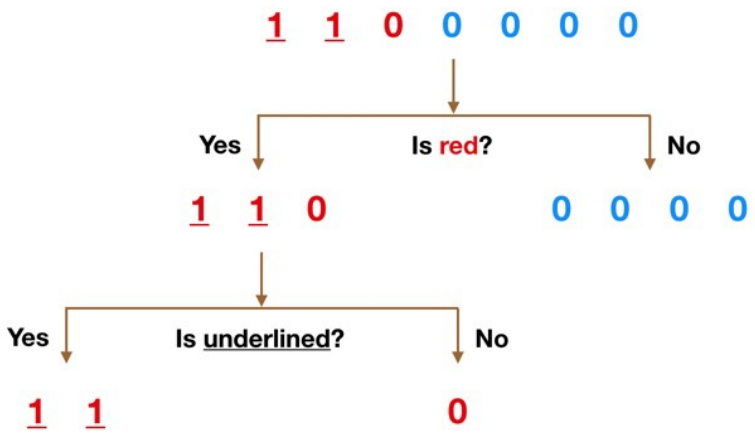



Figura 3.2: Ejemplo árbol de decisión

El clasificador Random Forest, utiliza un gran número de estos árboles que operan en conjunto. Cada árbol en el *bosque* produce una predicción de clase de los datos de entrada donde la clasificación final es la predicción que más común sea entre todos los árboles individuales, es decir, la predicción de clase más votada es la clasificación final.

##### 3.2.1.1.2. Support Vector Machines

*Support vector machine* (SVM) es un algoritmo de aprendizaje supervisado que se puede utlilizar para problemas de clasificación y regresión y para detectar valores atípicos. Podríamos decir que las SVM «son un raro ejemplo de una metodología donde la intuición geometrica, matemáticas elegantes, garantía teórica y algoritmos prácticos se encuentran» [3]. En la Figura 3.3 se observa un ejemplo de clasificación de clases binarias y multiclases usando SVM.

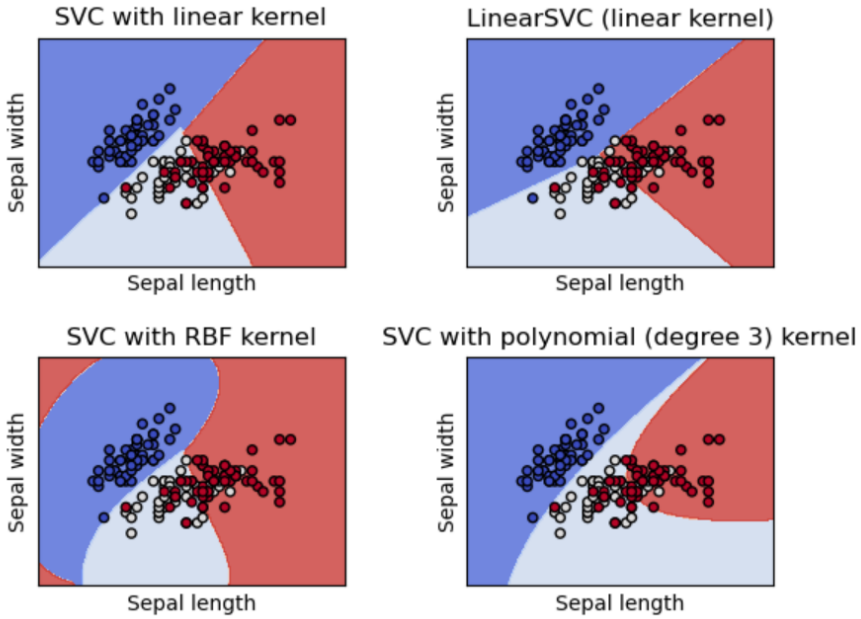



Figura 3.3: Ejemplo de problema de clasificación con SVM [28]

El objetivo de las SVM es encontrar un hiperplano que separe dos clases que más diferencia tenga entre si dentro de un conjunto de datos. En la separación lo ideal es ampliar el llamado margen funcional, definido como la anchura de la región paralela al hiperplano que no tiene puntos de datos interiores, ya que en general cuanto mayor sea el margen menor será el error de generalización del clasificador[28]. En la Figura 3.4 se muestra la función de decisión para un problema lineal con tres elementos en el margen funcional que reciben el nombre de *vectores de soporte*.

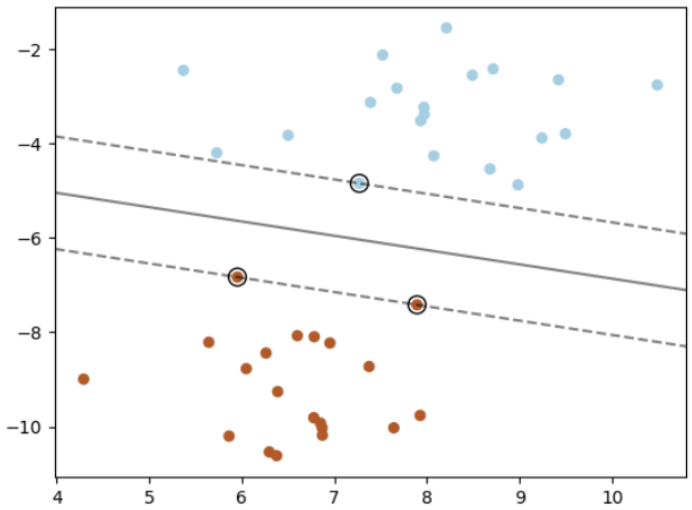



Figura 3.4: Función de decisión en un problema lineal donde se observa el *margen funcional* [28]

Una de las ventajas de esta metodología es su versatilidad, ya que existen diferentes funciones *Kernel* que se pueden usar como función de decisión. Las funciones Kernel que se encuentran disponibles en la librería scikit-learn[27] son las siguientes[28]:

1. **Lineal:** $(x, x')$
2. **Polinomial:** $(\gamma(x, x') + r)^d$
3. ***Radial Basis Function* (RBF):** $-\gamma||x - x'||^2$
4. **sigmoid:** $tanh(\gamma(x, x') + r)$

El parámetro $\gamma$ define cuánta influencia tiene un solo ejemplo de entrenamiento, de forma que cuanto mayor sea gamma, más parecidos deben ser otros ejemplos para verse afectados. Por otro lado, existe un parámetro $c$ común a todos los SVM *kernels*, que se encarga de compensar la clasificación errónea de ejemplos de entrenamiento, de forma que una $c$ alta apunta a clasificar correctamente todos los ejemplos de entrenamiento mientras que una $c$ baja suaviza la clasificación. La elección de estos dos parámetros $\gamma$ y $c$ es vital en el funcionamiento del modelo.

#### 3.2.1.1.3. XGBoost

(eXtreme Gradient Boosting) es un algoritmo de aprendizaje supervisado que puede ser utilizado para clasificación y regresión. El XGBoost se basa en:

**Boosting**, es un proceso en el que múltiples clasificadores simples, se combinan para formar uno mucho más potente.

**Gradient boosting** se basa en una aproximación gradiente descendente para mejorar la velocidad y asi mejorar el rendimiento del boosting simple. ***eXtreme Gradient Boosting* (XGBoost)** es una versión optimizada del gradient boosting.

Los principales hiperparámetros del XGBoost[9] son:

- *learning_rate:* es la reducción del tamaño de cada paso utilizado para evitar el sobreaprendizaje. Después de cada paso del *boosting* se toman los pesos de las nuevas características y el *learning_rate* reduce los pesos de estas características para que sea un *boosting* más conservador.
- *n_estimators:* es el número de estimadores o clasificadores utilizados en el proceso del *boosting* que forman el bosque de estimadores.
- *max_depth:* es la profundidad máxima de cada árbol o estimador. Usar un número alto de estimadores implica hacer que el modelo sea más complejo y sea más susceptible al sobreaprendizaje.
- *min_child_weight:* es el número mínimo que necesita la suma de los pesos de un hijo. Cuanto más grande sea el número más conservador será el algoritmo.
- *reg_alpha:* es la regularización L1 en término de pesos. Un valor más grande hará al modelo más conservativo.

### 3.2.2. Aprendizaje no supervisado

Dentro del aprendizaje no supervisado[8] se encuentran algoritmos de aprendizaje automático capaces de clasificar datos y hallar patrones en ellos sin la necesidad de intervención humana ya que no se aporta ningún tipo de etiquetas asociadas a los datos de entrada, de donde reciben ese nombre.

Se centran principalmente en las siguientes tareas:

1. **Clustering:** Es una técnica de *data mining*(Ver Sección 4.1.3) para agrupar datos sin etiquetar basándose en sus similitudes o diferencias. Muy usada en segmentación de mercado, compresión de imágenes... Un algoritmo de *clustering* es el *K-Means* que asigna los datos similares en diferentes grupos dependiendo de su grado de similitud.

2. **Asociación:** Técnica para encontrar relaciones entre variables de un dataset. Es muy utilizado en análisis de mercado para motores de recomendación, usando en la línea de: "clientes que compraron este producto, también compraron...".

3. **Reducción dimensional:** Técnica usada cuando las dimensiones de un dataset son demasiadas. Reduce el número de dimensiones, preservando al máximo la integridad de los datos. Muy usada en la etapa de preprocesamiento. Un algoritmo de reducción dimensional es el PSA.

### 3.2.3. Principales diferencias entre el aprendizaje supervisado y no supervisado

Lo que diferencia principalmente a estas dos técnicas[8] es el uso de datasets etiquetados. Los métodos de aprendizaje supervisado requieren de datos de entrada etiquetados manualmente de forma previa, mientras que los no supervisados, no lo necesitan.

En los algoritmos supervisados, el modelo aprende del dataset etiquetado, haciendo predicciones iterativamente y ajustándose hasta obtener respuestas correctas. Suelen obtener resultados más precisos que su contraparte no supervisada, pero requieren de trabajo humano para obtener datos etiquetados correctamente, lo que puede suponer un problema.

Los modelos no supervisados trabajan independientemente, descubriendo las relaciones e interacciones de los datos para cumplir su propósito. Pese a todo, suele ser necesario validar los resultados para detectar posibles recomendaciones extrañas.

Otros aspectos en los que difieren estos dos tipos de aprendizaje son:

- **Objetivos:** en el aprendizaje supervisado el objetivo suele ser predecir ciertas salidas sobre nuevos conjuntos de datos aportados al modelo, mientras que, en el no supervisado, se trata agrupar muestras que tengan características similares entre ellas entre grandes volúmenes de datos. El propio modelo es el encargado de encontrar esos patrones relevantes en los datos.

- **Aplicaciones:** los modelos supervisados son ideales para detección de spam, análisis del clima, predicciones de precios... En cambio, los no supervisados destacan en detección de anomalías, motores de recomendaciones, aplicaciones médicas...

- **Complejidad:** los modelos no supervisados requieren de muchísimo más poder computacional que los no supervisados, debido a los ingentes datasets necesarios para su correcto entrenamiento.
- **Desventajas:** por una parte, los métodos supervisados pueden consumir mucho tiempo para ser entrenados y para la etiquetación de los datasets, por otra parte, los métodos no supervisados pueden tener resultados altamente imprecisos, si no son validados posteriormente.

### 3.2.4. Elección del modelo aplicado

Para tomar una decisión acertada sobre qué modelo usar[8], se deben tener en cuenta ciertas cosas:

- **Evaluar el dataset existente:** Debe considerarse si el dataset a utilizar se encuentra etiquetado o no, y en caso de no estarlo, si se dispone de un grupo de expertos para etiquetarlo.
- **Definir los objetivos:** ¿Se trata de un problema bien definido y conocido? ¿o se está tratando de predecir nuevos datos a partir de los conocidos?
- **Revisar las opciones conocidas de modelos:** Se debe revisar si existen algoritmos con la dimensionalidad necesaria y estudiar si son capaces de soportar el volumen de datos a tratar.

## 3.3. Redes Neuronales

Las redes neuronales [25] son modelos simples del funcionamiento del sistema nervioso donde las unidades básicas son las neuronas, que suelen ser organizadas en capas. Estos modelos emula la manera que tiene el cerebro de procesar los datos. Funcionan normalmente con un número elevado de neuronas en diferentes capas.

Las redes neuronales se organizan en capas. Una capa de entrada, una o varias capas ocultas y una capa de salida. Estas se conectan con diferentes pesos. Los datos de entrada se introducen en la primera capa y se van propagando desde la neurona inicial a través de todas las neuronas de las siguientes capas, terminando en la capa de salida. Estas capas pueden tener un gran número de configuraciones, que suelen ser elegidas por el programador teniendo en cuenta los datos de entrada. Para transmitir información de una capa a otra, a cada neurona llega información de las neuronas de la capa anterior. A esta información se le aplica lo que se llama función de activación. Dependiendo de los datos de entrada a cada neurona, esta función define los datos de salida de esta a partir de cierto umbral, si estos datos estan por encima del umbral la neurona se activa y los datos salen de la neurona.

Estas redes aprenden revisando todos los registros individuales, produciendo una predicción para cada uno de estos y realizando ajustes en los pesos cuando la predicción es incorrecta. Este proceso se repite muchas veces mejorando sus predicciones hasta alcanzar uno o varios criterios de parada y es conocido como *backpropagation*.

La red aprende a través del entrenamiento. Una vez se ha entrenado la red, se puede aplicar a casos futuros de los que se desconoce el resultado.

### 3.3.1. Funciones de activación

Como se ha visto en el apartado anterior, cuando los datos de entrada llegan a una neurona, si estos estan por encima de un determinado umbral, determinado por la función de activación, los datos son pasados a la siguiente capa de neuronas. Hay diferentes tipos. Entre ellas estan[14]:

- Relu:

$$f(x) = max(0, x) = \begin{cases} 0 & for \quad x < 0 \\ x & for \quad x \geq 0 \end{cases} \tag{3.1}$$

- Sigmoide:

$$f(x) = \frac{1}{1 - e^{-x}} \tag{3.2}$$

- Tangente Hiperbólica:

$$f(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{3.3}$$

### 3.3.2. Redes Neuronales Convolucionales basadas en Grafos

*Graph Convolutional Networks* (GCN) (o en español Redes neuronales convolucionales basadas en grafos) [23], son un tipo de redes neuronales que utilizan el método tradicional de estas con un input basado en grafos y con el método de convolución aplicado a las diferentes capas en el que se utiliza el método de aprendizaje con o sin supervisión.

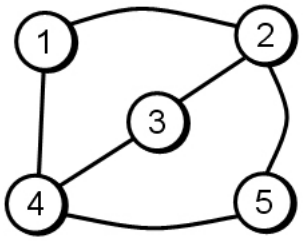
#### 3.3.2.1. Grafos

Muchos de los problemas que se ven en el mundo real pueden ser representados como grafos. Por ejemplo, las moléculas formadas por átomos, redes sociales, simulación en patrones de tráfico en redes de carreteras...

Aplicar esta representación al modelo de las redes neuronales puede ser más preciso y representativo que el modelo tradicional para este tipo de problemas.

Los grafos contienen nodos (o vértices) y aristas. Los nodos en los ejemplos anteriores pueden ser los átomos que forman las moléculas y las aristas, los enlaces que unen los átomos. O bien en el caso de una red social, los nodos representarían las personas y las aristas, las relaciones entre todas ellas.

La representación más común de los grafos utilizada en las GCN es la matriz de adyacencia. Donde cada nodo representa una fila y una columna y las relaciones entre nodos se indica en la fila y columna correspondiente a ambos nodos (Ver Figura 3.5).




| M | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 1 | 0 |
| 2 | 1 | 0 | 1 | 0 | 1 |
| 3 | 0 | 1 | 0 | 1 | 0 |
| 4 | 1 | 0 | 1 | 0 | 1 |
| 5 | 0 | 1 | 0 | 1 | 0 |

Figura 3.5: Matriz de adyacencia de un grafo

#### 3.3.2.1.1. Posibles aplicaciones en una Red Neuronal

Utilizando grafos con el modelo de las redes neuronales nos puede ayudar a predecir:

- Clasificación de nodos: Predecir el tipo de un nodo.
- Predicción de relaciones: Predecir si dos nodos estan conectados.
- Detección de clusters: Identificar grupos de nodos interconectados.
- Similaridad de dos redes: Como de similares son dos grafos.

### 3.3.2.2. Convoluciones

La convolución es el procesado que diferencia las Redes Neuronales básicas de las *Convolutional Neural Networks* (CNN). Consiste en aplicar un kernel (producto escalar contra una matriz) a un grupo de píxeles cercanos. Esta operación se realiza en todas las neuronas de entrada y genera una nueva matriz de salida (Ver Figura 3.6).

El tamaño y los valores de cada elemento del kernel cambian dependiendo del problema a resolver.

Las CNN han demostrado ser muy eficientes a la hora de extraer características de los datos, y las capas de convolución, hoy en día, representan la estructura de muchos modelos de deep learning. Lo que las hace tan eficientes es la capacidad que tienen de aprender secuencias de filtros para extraer patrones cada vez más complejos.

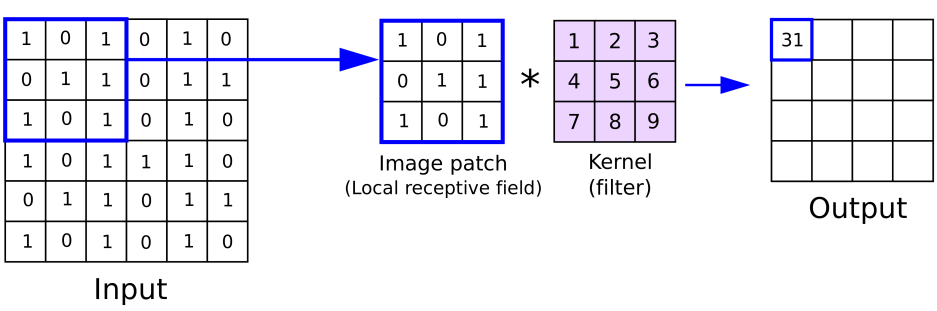



Figura 3.6: Ejemplo de kernel

#### 3.3.2.3. Filtros

En la práctica no solo se aplica un kernel a cada matriz. Se aplica lo que se llaman filtros, que es un conjunto de kernels. Para $n$ filtros tenemos n matrices de salida. Para los kernels de cada filtro se utilizan los mismos pesos, es decir, los valores de la matriz por la que se hace el producto escalar son los mismos.

En este caso con x matrices de entrada que representan los grafos se les pueden aplicar $n$ filtros que van a producir $n$ matrices de salida que resaltan ciertas características del grafo original, diferentes dependiendo del filtro utilizado.

#### 3.3.2.4. Graph Convolutional Networks

Es un tipo de red neuronal convolucional que trabaja directamente con grafos y se aprovecha de las propiedades estructurales de estos. Los datos de los grafos son no estructurados, es decir, no siguen un orden como en una imagen donde se puede atribuir unas dimensiones lineales para x e y. Así que construir modelos de aprendizaje automático para datos representados como grafos, no es trivial.

##### 3.3.2.4.1. Convoluciones en GNN

Con lo visto anteriormente sobre las convoluciones, la misma idea se puede aplicar a los grafos (Ver Figura 3.8). La arquitectura es la siguiente (Ver Figura 3.7):

- Las características se extraen pasando el input por una serie de filtros de convolución y capas de agrupación.
- Los canales de las características resultantes son mapeados a un vector de tamaño fijo (por ejemplo, usando una capa de agrupación global)
- Finalmente unas pocas capas conectadas producen la salida final.

En el caso de las GCN la entrada está constituida por:

- Un array que contiene para cada nodo del grafo en $N$, $C$ características.
- La matriz de adyacencia.

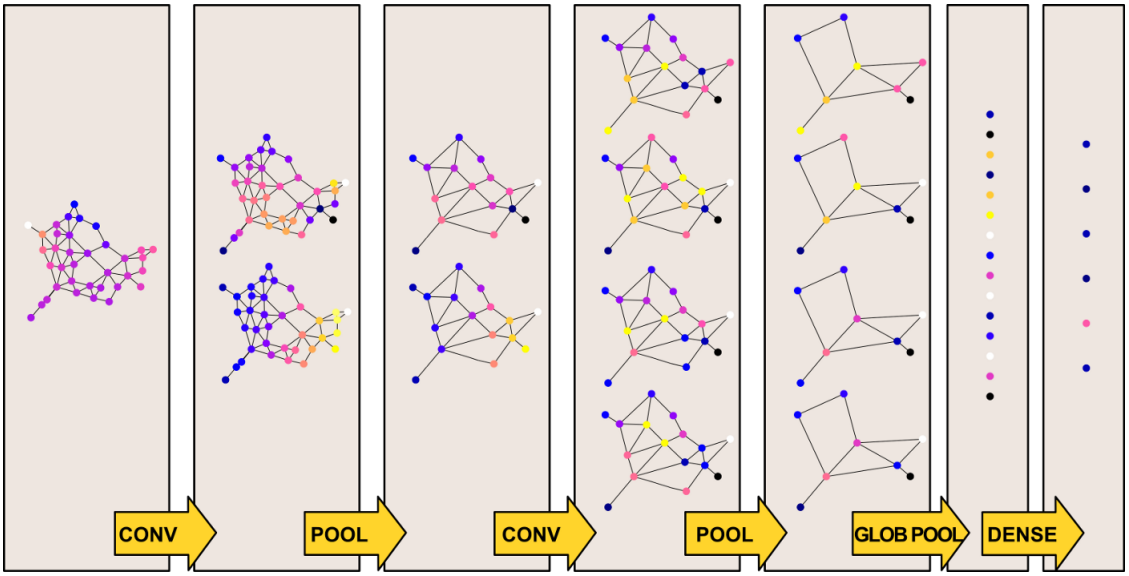



Figura 3.7: Arquitectura de GCN

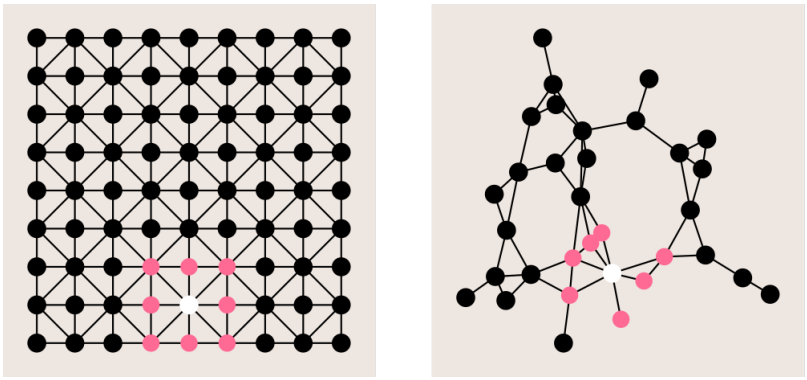


Figura 3.8: Convoluciones en matriz y grafo

#### 3.3.2.5. Diferencias entre CNNs y GNNs

La principal diferencia entre las CNN y *Graph Neural Networks* (GNN) es que las CNN estan diseñadas para operar en datos estructurados de manera regular y las GNN son la versión generalizada de estas donde el número de conexiones entre nodos varía mientras los nodos no cambian (Ver Figura 3.9).

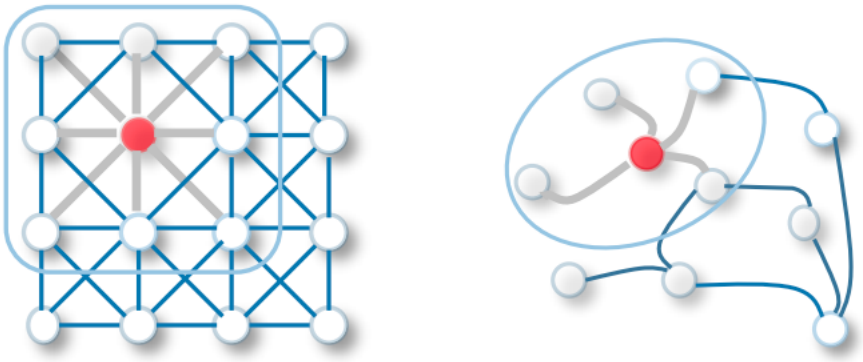


Figura 3.9: Diferencia entre CNNs y GNNs

# Capítulo 4

# Estado del Arte

A lo largo de este capítulo se exponen algunas investigaciones realizadas en el ámbito de la detección de aplicaciones móviles a partir tráfico de red encriptado. Principalmente, se ha investigado sobre tres herramientas desarrolladas por sus respectivos autores: AppScanner [33], FlowPrint [35] y MAppGraph [29]. En la Sección 4.1 se introducen las principales metodologías seguidas en trabajos relacionados con el análisis de tráfico de red. En las secciones 4.2, 4.3 y 4.4 se habla más en detalle de las principales características de estas tres herramientas. Posteriormente, en la Sección 4.5 se presentan otros trabajos relacionados.

## 4.1. Metodología de análisis del tráfico de red mediante aprendizaje automático

El tráfico de red puede ser analizado en al menos dos niveles distintos:

1. **Nivel de paquetes:** los paquetes de red son las tramas IP que son transmitidas a la red en la Capa de Red 2.1.2 y son la unidad básica y de más bajo nivel de transmisión en la red.

2. **Nivel de flujos:** un flujo es una secuencia de paquetes que tienen la misma quíntupla (source IP, destination IP, source port, destination port, protocol) y cuyo tiempo de llegada entre dos paquetes consecutivos es menor que un threshold establecido [34].

**El proceso básico de análisis del tráfico de red cuenta con tres fases:** una primera fase en la que los datos son preprocesados con diferentes técnicas 4.1.2 seguida de una fase de análisis de los datos (minería de datos) 4.1.3 y evaluación de los resultados 4.1.4 con el objetivo de extraer patrones o conclusiones [17].

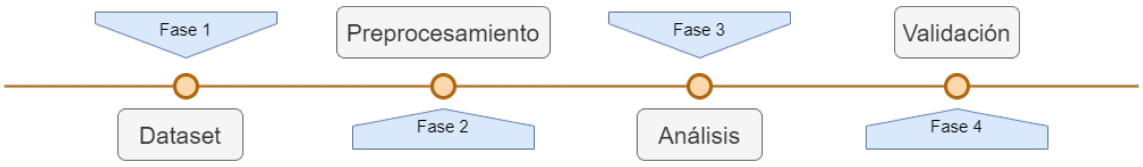



Figura 4.1: Principales etapas del *data mining*

### 4.1.1. Dataset

El primer paso para llevar a cabo un análisis del tráfico de red es contar con un **conjunto de datos o *dataset*** con todas las características a evaluar. Contar con un *dataset* robusto, heterogéneo (o representativo) y balanceado es de vital importancia para el entrenamiento y prueba de un modelo en fases posteriores.

### 4.1.2. Preprocesamiento

El preprocesamiento consiste en transformar los datos extraídos en crudo (como por ejemplo los datos de las capturas de tráfico de red) en datos que sean de utilidad para el objetivo propuesto y en el tratamiento de los mismos con el fin de obtener una mayor precisión y eficiencia en la minería de datos 4.1.3

Existen diferentes técnicas para llevar a cabo este preprocesamiento, entre ellas destacamos las que se han utilizado en este proyecto:

- **Selección de características:** consiste en eliminar características redundantes o irrelevantes del conjunto de datos, generando así un nuevo *dataset* con menor ruido o datos sin valor añadido para el modelo.
- **Limpieza de datos o *Data cleaning*:** es el proceso por el cual se identifican y corrigen datos que son incorrectos, incompletos, duplicados, irrelevantes o presentan cualquier problema para el objetivo del *dataset*.
- **Escalado y normalización de los datos:** este proceso consiste en cambiar los valores del *dataset* con el fin de usar una escala común, sin distorsionar las diferencias en los intervalos de valores ni perder información y es un proceso que tiene una gran importancia para el correcto funcionamiento de muchos algoritmos, especialmente aquellos que usan redes neuronales.

### 4.1.3. Minería de datos

La minería de datos o *Data Mining* (DM) es el proceso de análisis del conjunto de datos con el fin de encontrar anomalías, patrones y correlaciones. Existen numerosas técnicas de minería de datos, entre ellas se encuentran:

1. **Algoritmos de clustering:** consisten en agrupar elementos de conjunto de datos en subconjuntos distintos de forma que los elementos más similares pertenezcan a una misma clase. Entre ellos destaca el algoritmo *K-means*.
2. **Algoritmos de clasificación:** el objetivo de estos alogitmos es asignar una clase a cada elemento del conjunto de datos. Entre ellos destacan:
    - **Redes Neuronales:** son un mecanismo de predicción que se basa en encontrar la relación existente entre los elementos de un conjunto de datos. En el Capítulo 3 se profundiza en el funcionamiento de las redes neuronales.
    - **Árboles de decisión:** son algoritmos iterativos que consisten en dividir los datos de entrada en regiones basadas en intervalos de variables independientes (o features). Se pueden visualizar como un árbol en el que cada nodo interno

representa una pregunta de clasificación y cada rama una respuesta, de forma que las hojas del árbol se encuentran las clasificaciones a la que pertenecen los elementos del conjunto de datos. En la Figura 4.2 se muestra un ejemplo sencillo de un árbol de decisión.

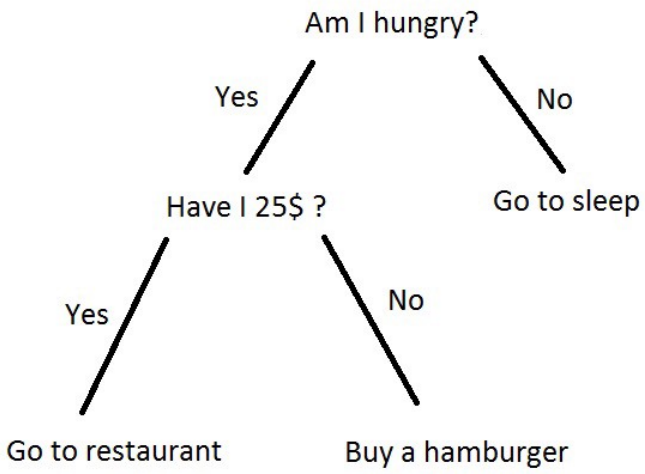



Figura 4.2: Ejemplo un de árbol de decisión

### 4.1.4. Evaluación

En la fase de evaluación de los resultados se establece una métrica a evaluar que permita la comparación y el refinamiento del modelo.

A continuación se muestran las principales métricas utilizadas y sus significados:

1. **Matriz de confusión:** tabla que permite observar los resultados del modelo supervisado sobre un conjunto de datos. Está formada por los siguientes campos:

   *a*) **Verdaderos Negativos *True Negatives* (TN):** número total de elementos identificados como 0 o Falso y pronosticados por el modelo como 0 o Falso.

   *b*) **Verdaderos Positivos *True Positives* (TP):** número total de elementos identificados como 1 o Verdadero y pronosticados por el modelo como 1 o Verdadero.

   *c*) **Falsos Negativos *False Negatives* (FN):** número total de elementos identificados como 1 o Verdadero y pronosticados por el modelo como 0 o Falso.

   *d*) **Falsos Positivos *False Positives* (FP):** número total de elementos identificados como 0 o Falsos y pronosticados por el modelo como 1 o Verdadero.

   El modelo ideal es aquel que no tiene ningún FP ni FN ya que implicaría que el modelo ha realizado todas las predicciones de forma correcta. En la Figura 4.3 se muestra el formato de la matriz de confusión.

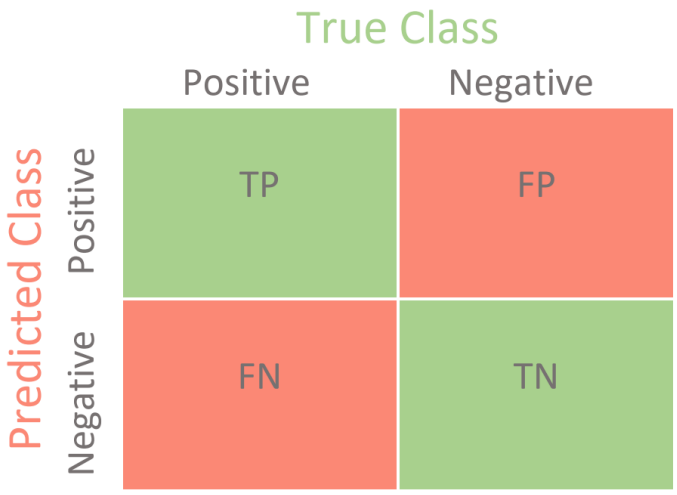



Figura 4.3: Matriz de confusión

2. **Exactitud o *Accuracy*:** es el porcentaje total de elementos clasificados correctamente y se calcula como:

$$\frac{TP + TN}{FP + FN} \tag{4.1}$$

   El resultado es un valor entre 0 y 1 siendo el modelo un mejor predictor cuanto mayor es este valor.

3. ***Recall*:** es el número de elementos correctamente identificados como positivo del total de positivos verdaderos y se calcula como:

$$\frac{TP}{TP + FN} \tag{4.2}$$

4. ***Precision*:** es el número de elementos correctamente identificados como positivo del total de elementos identificados como positivos y se calcula como:

$$\frac{TP}{TP + FP} \tag{4.3}$$

5. **Especifidad o *Specificity*:** es el número de elementos identificados correctamente como negativos del total de negativos verdaderos y se calcula como:

$$\frac{TN}{TN + FP} \tag{4.4}$$

6. ***Sensitivity*:** es el número de elementos identificados correctamente como positivos entre el número de elementos negativos verdaderos:

$$\frac{TP}{FP} \tag{4.5}$$

7. ***F1 Score*:** es una medida que combina la Precision y el Recall y se calcula como:

$$\frac{2 * Precision * Recall}{Precision + Recall} \tag{4.6}$$

## 4.2. AppScanner

AppScanner [33] presenta un enfoque supervisado para extraer e identificar aplicaciones de Android en base a su tráfico cifrado o no cifrado generado.

La contribución de este trabajo propone el uso de ciertas características estadísticas y no estadísticas extraídas del tráfico extraído que se utilizan para entrenar modelos supervisados de aprendizaje automático.

Los algoritmos de aprendizaje automático comparados en AppScanner son SVM y *Random Forest*, ambos utilizados de seis maneras diferentes en el agrupamiento de los datos:

- **Per Flow SVC:** utiliza el algoritmo SVM con características extraídas de los flujos.
- **Per Flow Random Forest:** utiliza el algoritmo *Random Forest* con características extraídas de los flujos.
- **Single Large SVC:** utiliza el algoritmo SVM con características estadísticas de todas las aplicaciones.
- **Single Large Random Forest:** utiliza el algoritmo *Random Forest* con características estadísticas de todas las aplicaciones.
- **Per App SVC:** utiliza el algoritmo SVM por cada aplicación con características estadísticas.
- **Per App Random Forest:** utiliza el algoritmo *Random Forest* por cada aplicación con características estadísticas.

De entre los seis modelos, el *Random Forest* es el más eficiente de los algoritmos utilizados. En la implementación de la herramienta, se utiliza el clasificador *Random Forest* de la librería de Pyhton sklearn.

### 4.2.1. Resultados

La Tabla 4.1 muestra los resultados obtenidos para distintos modelos de clasificación, obtienendo los mejores resultados para el *Per App Random Forest*.

| Aproximación | Precision | Recall | Accuracy |
|---|---|---|---|
| Per Flow SVC | 77.1 % | 71.9 % | 71.5 % |
| Per Flow Random Forest | 84.4 % | 83.1 % | 82.1 % |
| Single Large SVC | 51.3 % | 60.2 % | 42.4 % |
| Single Large Random Forest | 89.5 % | 85.9 % | 86.9 % |
| Per App SVC | 96.1 % | 64.8 % | 99.7 % |
| Per App Random Forest | 96.0 % | 82.5 % | 99.8 % |

Tabla 4.1: Comparación de los resultados con diferentes parámetros en AppScanner [33]

Otro de los resultados destacables es el impacto en términos de precisión del clasificador al aumentar el número de aplicaciones usadas para el entrenamiento. En estos resultados se aprecia cómo la precisión disminuye al aumentar el número de aplicaciones.

## 4.3. FlowPrint

FlowPrint[35] presenta un enfoque semisupervisado para extraer fingerprints de las aplicaciones móviles del tráfico de red. Se encuentran automáticamente correlaciones temporales entre las características relacionadas con el destino del tráfico de red y se usan estas correlaciones para generar fingerprints de aplicaciones, que pueden ser reutilizadas más adelante para reconocer aplicaciones conocidas o para detectar aplicaciones no vistas anteriormente. La principal contribución de este trabajo es crear fingerprints de red sin conocimiento previo de las aplicaciones que se ejecutan en la red.

### 4.3.1. Resultados

Todos los resultados que se comentan en esta Sección pertenecen a los autores de FlowPrint [35].

Los resultados de FlowPrint se evaluan con la medida *F1 Score* a optimizar (Ver Sección 4.1.4). En este modelo se consideran los siguientes parámetros configurables[35]:

1. $T_{batch}$**:** establece la duración de cada captura de tráfico para procesar en cada ejecución.
2. $T_{window}$**:** especifica la ventana de tiempo para que los clústeres de destino se consideren activos simultáneamente.
3. $T_{correlation}$**:** describe la correlación mínima entre cada par de clústeres de destino para que exista una arista en el grafo.
4. $T_{similarity}$**:** indica la similitud Jaccard mínima requerida entre las huellas para ser tratadas como equivalentes.

En la Tabla 4.2 se muestran los resultados obtenidos en términos de *F1 Score* para cinco conjuntos de parámetros distintos.

| $T_{batch}$ | $T_{window}$ | $T_{correlation}$ | $T_{similarity}$ | F1-Score |
|---|---|---|---|---|
| 3600 | 5 | 0.3 | 0.5 | 0.8164 |
| 300 | 5 | 0.3 | 0.5 | 0.8294 |
| 300 | 30 | 0.3 | 0.5 | 0.8367 |
| 300 | 30 | 0.1 | 0.5 | 0.8543 |
| 300 | 30 | 0.1 | 0.9 | 0.9190 |

Tabla 4.2: Comparación del F1-Score de Flowprint con diferentes parámetros [35]

Otro de los resultados es el obtenido al emplear diferentes *datasets* con FlowPrint frente a AppScanner (versión con un clasificador *Random Forest*, la cual alcanzó el mayor rendimiento en la evaluación de AppScanner). Cabe mencionar que en términos de *Recall* FlowPrint consigue clasificar mejor todo tipo de tráfico, mientras que AppScanner solo proporciona un cierto nivel suficiente de reconocimiento para una fracción de las aplicaciones.

En la evaluación de FlowPrint se llega a clasificar y reconocer aplicaciones de iOS y Android con un *accuracy* del 89,2 %, superando los resultados de AppScanner. Además, es capaz de detectar aplicaciones no vistas anteriormente con una precisión del 93.5 %.

## 4.4. MAppGraph

MAppGraph[29] es una herramienta que pretende abordar la clasificación de aplicaciones móviles a partir de tráfico de datos cifrado, tal y como hicieron AppScanner 4.2 y FlowPrint 4.3, pero consiguiendo un modelo con una exactitud mayor a las herramientas mencionadas anteriormente.

AppScanner[33] extrae características *side-channel* a partir de flujos de tráfico de datos para entrenar un modelo de aprendizaje automático que sea capaz de clasificar aplicaciones móviles. La principal problemática es que las aplicaciones móviles usan *Content Delivery Network* (CDN) o servicios compartidos y, por tanto, pueden existir flujos con características similares pertenecientes a aplicaciones distintas que confundan al modelo de clasificación.

Para solucionar esta problemática, FlowPrint[35] propuso construir a partir de la huella o *fingerprint* de una aplicación un grafo de comunicación entre un dispositivo móvil, los destinos y las características asociadas. De este modo, las aplicaciones son inferidas comparando sus huellas con las huellas almacenadas en el modelo. Debido a la complejidad a la hora de construir los grafos, el modelo fue entrenado con datos recopilados en un breve periodo de comunicación, lo cual no asegura reconocer todos los comportamientos del usuario al usar una determinada aplicación.

Con el objetivo de abordar las anteriores limitaciones, los autores de MAppGraph recopilaron tráfico de datos de 101 aplicaciones de Google Play (se centraron en dispositvos Android conectados a una misma red) con más de 30 horas de tráfico por aplicación.

De esta forma, con cada tramo de tráfico se forma un grafo de comunicación donde los nodos disponen de unas características asociadas y están conectados mediante aristas con un peso asignado en función de la correlación entre dichos nodos.

El último paso en el desarrollo de MAppGraph es la clasificación, cuyos resultados se muestran en la Sección 4.4.1.

### 4.4.1. Resultados

Todos los resultados que se comentan en esta Sección pertenecen a los autores de MAppGraph [29].

Los resultados experimentales muestran que MAppGraph supera las técnicas de referencia (AppScanner y FlowPrint) con una mejora del rendimiento de hasta un 20 % en términos de *Precision*, *Recall*, *F1-Score* y *Accuracy* (Ver Sección 4.1.4), situándose en

torno a un **93.5 %** en todas las métricas, lo que hace que clasifique de manera precisa las aplicaciones.

En la Tabla 4.3 se muestra una comparativa con los resultados de MAppGraph frente a *MultiLayer Perceptron* (MLP) y las versiones que dieron mejores resultados de AppScanner y Flowprint. Los MLP son redes neuronales (Ver Sección 3.3) de tipo *feed-forward*, con neuronas de tipo perceptrón. La función de agregación es una suma ponderada, y la función de activación, un sigmoide, lo que permite un aprendizaje por retropropagación[22].

| Técnica | Precision | Recall | F1-Score | Accuracy |
|---|---|---|---|---|
| MLP | 0.9081 | 0.9075 | 0.9074 | 0.9075 |
| AppScanner mejorado | 0.8634 | 0.7938 | 0.7828 | 0.7938 |
| FlowPrint mejorado | 0.8759 | 0.8341 | 0.8275 | 0.8341 |
| MAppGraph | 0.9364 | 0.9346 | 0.9347 | 0.9346 |

Tabla 4.3: Comparación del rendimiento global de MAppGraph frente a otros modelos [29]

Cabe mencionar que en comparación con FlowPrint mejorado, el cual también considera la correlación cruzada entre los servicios de las aplicaciones y servicios de terceros representando las comunicaciones mediante grafos, MAppGraph mejora los resultados hasta un 7 %.

En otra comparativa del modelo DGCNN (MAppGrah) frente a MLP con cuatro valores distintos para el número de nodos $N$ los resultados indicaron que a medida que desciende el número de nodos, el rendimiento obtenido es peor. El rendimiento se incrementa hasta un valor óptimo antes de volver a descender con un número mayor de nodos.

Otro resultado interesante es el obtenido al comparar MAppGraph, Flowprint mejorado y MLP variando el parámetro del tamaño de la ventana, es decir, la duración de los trozos de tráfico. Los resultados apuntan que a medida que se decrementa la duración de los trozos de tráfico, el rendimiento disminuye. En MAppGraph el rendimiento se reduce un 7 % al reducir la duración del tráfico de 5 a 1 minuto.

Respecto al *slice duration*, el parámetro utilizado para calcular la correlación cruzada entre dos nodos, los resultados obtenidos en la comparación de MAppGraph y Flowprint mejorado indican que cuanto menor es la duración del *slice* menor número de aristas hay en el grafo, mientras que al aumentar el valor del *slice duration* los grafos se vuelven más poblados de aristas.

## 4.5. Otros trabajos relacionados

### 4.5.1. Identificando eventos y dispositivos IoT basándose en el tamaño de paquetes del tráfico cifrado

Es este trabajo [30] tratan de identificar eventos y dispositivos IoT a través del trafico cifrado.

Para esto, se crea un dataset a partir de archivos .pcap de tráfico cifrado de estos eventos y dispositivos. Al dataset de las longitudes de los paquetes le aplican la moda, media y mediana y a patir de estas tres variables de cada uno de los paquetes se aplican varios clasificadores para distinguir si es un dispositivo IoT o no y si lo es, intentar especificar que dispositivo es(esquema en figura 4.4).

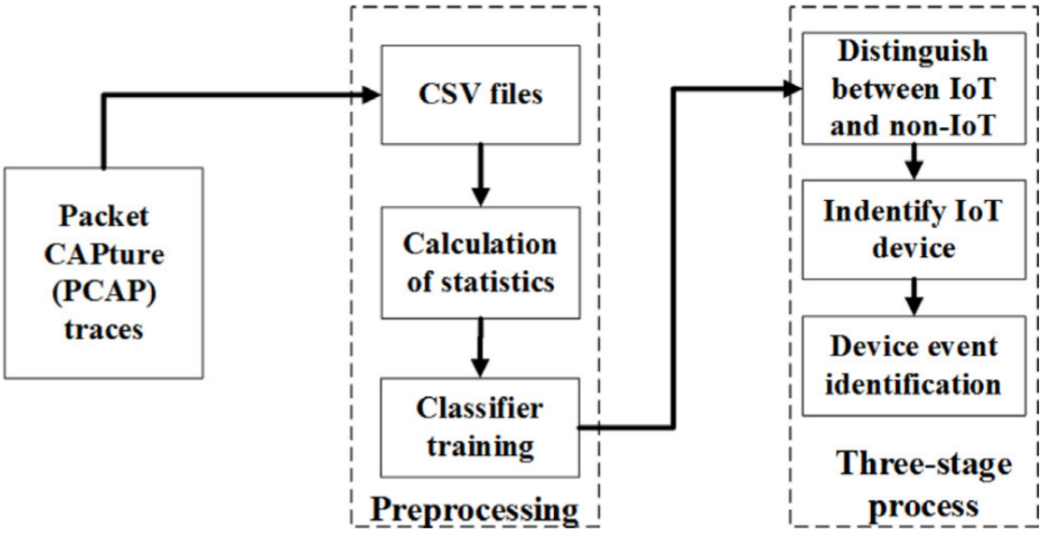



Figura 4.4: Esquema del funcionamiento del algoritmo

Los clasificadores analizados son: árboles de decisión, *Random Forest* y SVM.

El resultado es de una precisión mínima de 96 % utilizando ordenación aleatoria y 93 % en ordenación cronológica para identificación de si es un dispositivo IoT o no. Por encima del 94 % en identificar el dispositivo IoT y 99 % de precisión utilizando ordenación cronológica y aleatoria en la identificación de los eventos del dispositivo IoT.

Para determinar cuál de los clasificadores es mejor se realizan test de hipótesis basados en *Friedman*, *Nemenyi* y *Wilcoxon Signed-Rank*. Estos dan como resultado que el ***Random Forest*** es el mejor clasificador para identificar dispositivos de IoT en todos los escenarios evaluados. Es el tercero más rápido, siendo el primero los árboles de decisión.

### 4.5.2. Análisis de tráfico malicioso cifrado

En el trabajo [2] se analiza tráfico malicioso de capturas .pcap con varios clasificadores incluyendo *Random Forest*. A continuación, un pequeño resumen.

#### 4.5.2.1. Datasets

Se utilizan dos datasets de capturas publicados de tráfico malicioso y benigno:

- Dataset CTU-13 proviene de un trabajo de investigación de Czech Technical University. Este dataset tiene 13 capturas de tráfico maligno y benigno elegido y ejecutado en una maquina virtual, es decir no es de tráfico capturado de la red real. Disponibles como archivos .pcap.

- Dataset de Malware Capture Facility Project de Czech Technical University ATG Group, este de tráfico malware y benigno real. Una característica interesante es que se deja ejecutando el malware durante un periodo de tiempo muy largo, en algunos casos hasta algunos meses. Disponible también como archivos .pcap.

El dataset combinado tiene un total de 72 capturas, 59 de malware y 13 benignas.

Cada captura esta contenida en un archivo .pcap e inlcuye una lista de host infectados y benignos, así como Bro IDS logs generados de los archivos .pcap.

Bro es una herramienta de codigo abierto para inspección de tráfico, grabación de logs y detección de ataques. Usan Bro para generar logs de tráfico que incluyen información de los flujos de red y otros metadatos. Luego esta información se utiliza para obtener características del flujo del tráfico.

#### 4.5.2.2. Métodos

Se experimentan con tres tipos de algoritmos de machine learning que son los siguientes: SVM, *Random Forest*, XGBoost.

Se utiliza la validación cruzada en todos los experimentos para ser lo más parcial posible maximizando el número de tests independientes con los datos disponibles.

La métrica de evaluación para los experimentos es el *Accuracy*, explicado en la sección 4.1.4

#### 4.5.2.3. Experimentos

- SVM: con este método, usando un kernel lineal, se consigue una precisión un poco por encima del 92 %, lo que muestra que con este clasificador, es posible separar el tráfico malicioso del que no lo es. Se utilizan otros tipos de kernel como Radial Basis Funcion o Polynomial y los resultados son similares a el linear o un poco por debajo con lo que en las conclusiones solo se tiene en cuenta el lineal.

- *Random Forest*: el parámetro clave para este clasificador es el numero de estimadores. La precisión varía dependiendo del numero de estimadores elegido. Para este problema, eligiendo el numero mínimo de estimadores que es uno, el clasificador tiene una precisión de casi el 97 %. Con 50 estimadores, el clasificador tiene una precisión de casi el 99 %, que no aumenta hasta 1000 estimadores con lo que es su mejor precisión.

  En todos los experimentos realizados posteriormente, se elige un número de estimadores más elevado, 500, para asegurarse de conseguir los resultados óptimos ya que presenta un coste adicional muy pequeño.

  Utilizando un número de características extraídas muy pequeño, en concreto 6 de las más de 35, el clasificador ya se consigue una precisión superior al 98,5 %. Esto es significativamente mejor que el SVM.

  Esto muestra la importancia de dejar al modelo elegir las características y no elegir las más intuitivas que es lo que hace el SVM.

- XGBoost: al igual que el *Random Forest*, necesita un número de estimadores que suele ser mayor que en *Random Forest*. En este caso, para obtener una precisión

cercana a la óptima, se necesita un número grande de estimadores. En todos los experimentos realizados con este clasificador, se utilizan 1000 estimadores, el doble que con el *Random Forest*.

La curva de precisión de este clasificador respecto al número de estimadores, es más lenta y con menos precision inicial. En este caso teniendo un estimador, la precisión es cercana al 94 % mientras en el *Random Forest* este ya estaba en casi un 97 %. Aun así, la precisión es un poco mayor que la del *Random Forest*, con un poco más del 99 %.

Al igual que con el *Random Forest* con un número bajo de la características analizadas aplicado al entrenamiento, el algoritmo es capaz de rendir con una precision muy buena.

#### 4.5.2.4. Conclusiones

El rendimiento del XGBoost(99,15 %) y *Random Forest*(98,78 %) son muy similares pero un poco mas preciso el XGBoost, mientras que el SVM rinde de manera más pobre con una precisión del 92 %. Con un número de características muy bajo y estas no siendo las más intuitivas, estos clasificadores funcionan de manera eficiente.

# Capítulo 5

# Modelo basado en Aprendizaje Profundo para el Análisis de Tráfico en Dispositivos Móviles

En este Capítulo se presenta el modelo propuesto para llevar a cabo la detección de aplicaciones a partir de tráfico de red cifrado. Todo el código se ha desarrollado en el lenguaje de programación Python por las ventajas que aporta en el campo de la IA. En la Sección 5.1 se describe el conjunto de datos utilizados para llevar a cabo el entrenamiento del modelo, así como sus características. Seguidamente, la Sección 5.2 contiene cuestiones relevantes en cuanto a la metodología llevada a cabo en el preprocesamiento de los datos previa al entrenamiento. Posteriormente en la Sección 5.3 se describe el modelo creado que lleva a cabo el entrenamiento y la Sección 5.4 contiene los parámetros utilizados para realizar dichos experimentos. Por último, en la Sección 5.5 se muestran los resultados obtenidos.

En la Figura 5.1 se muestra las distintas fases que componen la metodología propuesta.

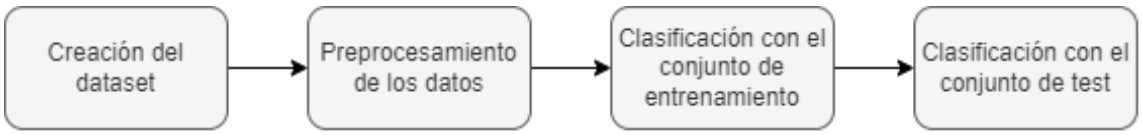



Figura 5.1: Diagrama de la Metodología

## 5.1. Dataset

El conjunto de datos recopilados consta de datos procedentes de 12 aplicaciones.

Dichas capturas de datos se realizaron en dispositivos con distintas versiones de Android mediante la herramienta PCAPdroid [13], la cual permite guardar en formato .pcap los paquetes de datos de una red pertenecientes a una determinada aplicación.

Con el objetivo de facilitar la identificación de las capturas de datos se establecieron unas normas de nomenclatura:

**App3_12_1541025917_304_gbn**

Donde:

- **App3:** hace referencia al nombre de la aplicación a la que pertenecen los paquetes de datos.
- **12:** es la versión de Android del dispositivo.
- **1541025917:** es la versión de la aplicación móvil.
- **304:** es la fecha (D/M) en la que se realizó la captura de datos.
- **gbn:** iniciales del nombre del autor de la captura de datos.

Además de estos campos también se ha recopilado información adicional acerca de las capturas, como si se han realizado llamadas de audio o si se han enviado imágenes. Toda esta información está almacenada en una hoja de Excel colaborativa en la que los usuarios han registrado sus capturas de datos.

## 5.2. Preprocesamiento

Los datos que forman el Dataset se sometieron a una fase de preprocesamiento antes de utilizarlos en la fase de entrenamiento con el fin de acondicionarlos a los requerimientos del modelo. Este preprocesamiento tuvo lugar en dos fases: una primera fase de extracción de las características a partir de las capturas de datos 5.2.1 y una segunda fase en la que se realizó la construcción de los grafos que representan los datos 5.2.2.

### 5.2.1. Extracción de características

Se realiza una extracción de características de los datos. La siguiente parte del preprocesamiento es la construcción de los grafos.

#### 5.2.1.1. Diagrama del proceso

En la Figura 5.2 se muestra un diagrama de proceso de la extracción de características descrita anteriormente.

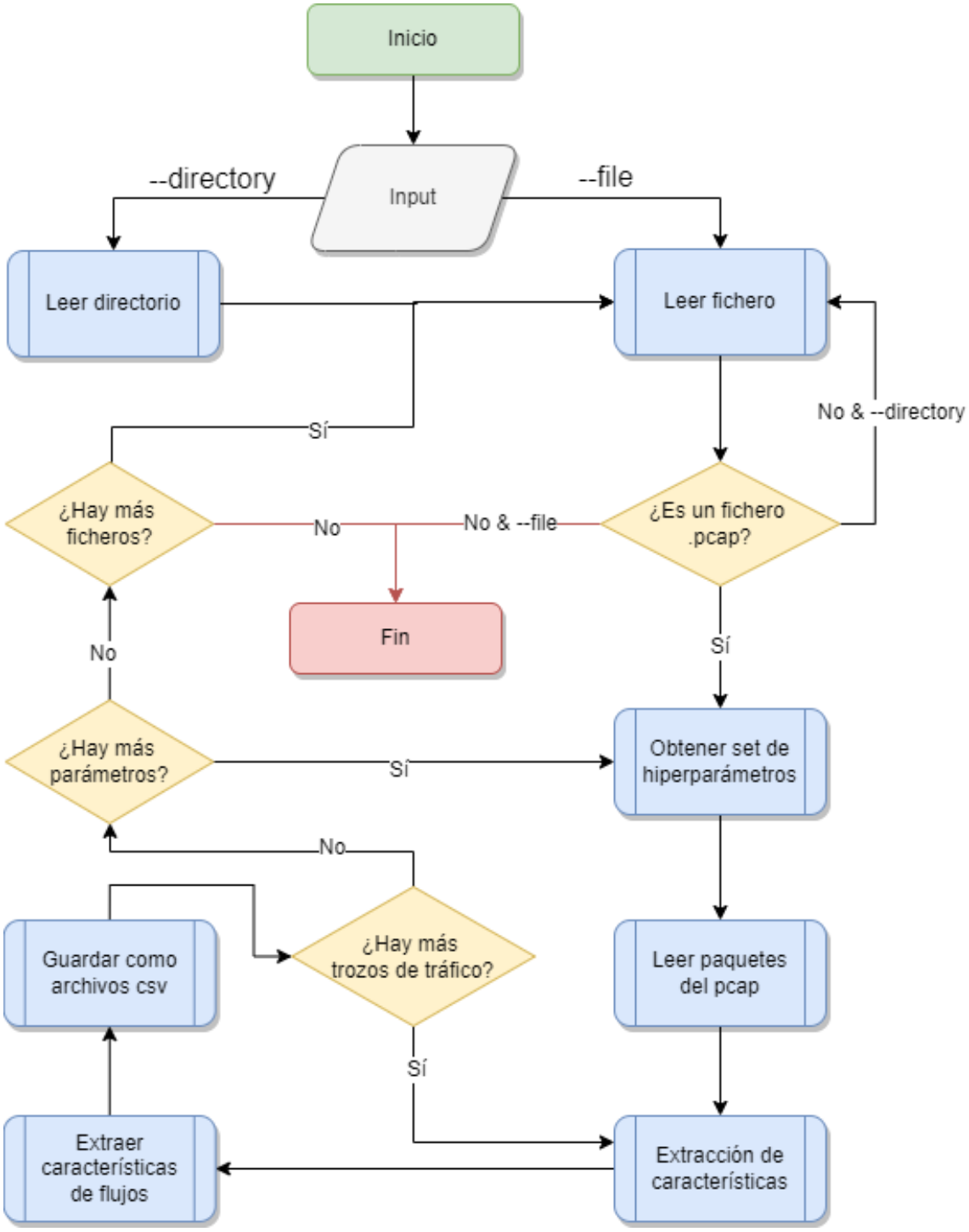



Figura 5.2: Diagrama de la extracción de características

### 5.2.2. Construcción de grafos

En la última fase del preprocesamiento se lleva a cabo la construcción de grafos que representarán las conexiones establecidas en una captura de datos y sus características extraídas. De esta forma, el modelo es entrenado posteriormente con GCN, a las cuales se hace referencia en el Capítulo 3.

#### 5.2.2.1. División del conjunto de datos

Durante la fase de construcción de grafos se realiza la división del *dataset* en dos subconjuntos: entrenamiento y test. El primer subconjunto de datos será utilizado para entrenar el modelo, mientras que el segundo subconjunto servirá para validar la precisión y fiabilidad del modelo a la hora de identificar aplicaciones. La proporción de entrenamiento-test del conjunto de datos escogida es 80-20 y el mecanismo de selección de muestras es aleatorio.

En la Figura 5.3 se muestra mediante un diagrama el proceso de división del conjunto de datos.

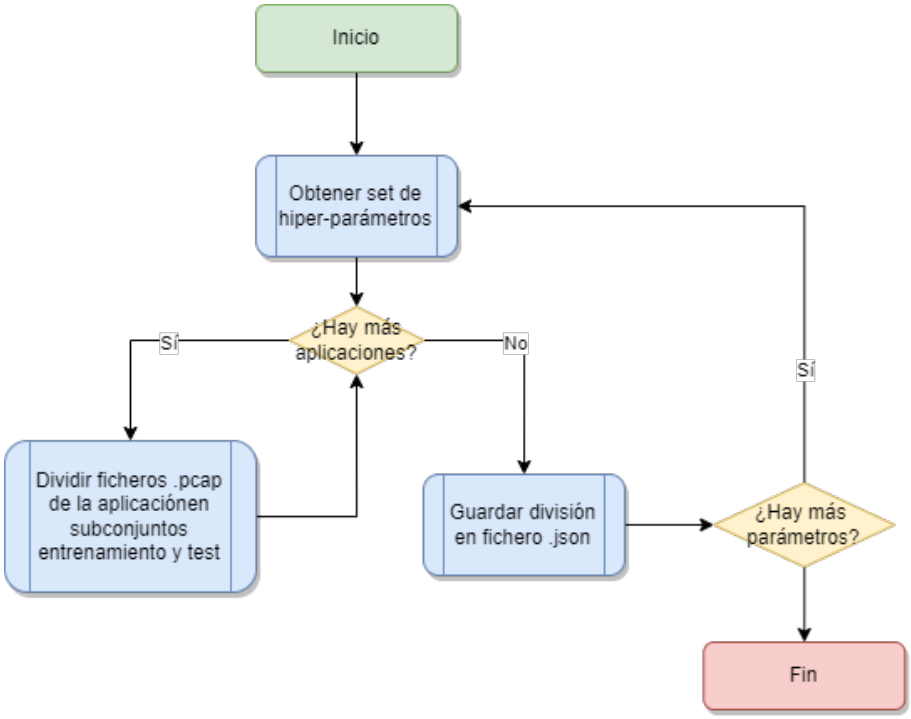



Figura 5.3: Diagrama de división del conjunto de datos en entrenamiento y *test*

La información de la división en entrenamiento y test se guarda por cada configuración de parámetros en formato JSON siguiendo la estructura:

**app name**: [[**información entrenamiento**][**información test**]]

Posteriormente en la construcción de grafos se hará una distinción entre grafos de entrenamiento y grafos de test guardándolos en carpetas diferentes.

#### 5.2.2.2. Parámetros utilizados

En la fase de construcción de grafos surgen dos nuevos parámetros:

- $N$**:** es el número de nodos máximo en el grafo. Con este parámetro se puede ajustar el tamaño final de los grafos.
- $T_{slice}$ **:** parámetro usado para calcular el peso entre los nodos.

#### 5.2.2.3. Estructura de los grafos

Dado una configuración determinada de parámetros, se construyen los grafos de las aplicaciones almacenándolos en un diccionario donde la clave es el nombre de la aplicación y el valor es una tupla de dataframes con las características y pesos del grafo.

**app → (features, weights)**

El grafo resultante de cada aplicación se construye a partir de múltiples grafos generados por cada uno de los ficheros .csv que contienen las características extraídas de los ficheros .pcap 5.2.1. Cada grafo cuenta con un identificador propio, de forma que en el grafo resultante se puede reconocer a qué grafos pertenecen las características y los pesos. En la Figura 5.4 se muestra el proceso de construcción de un grafo para una aplicación determinada.

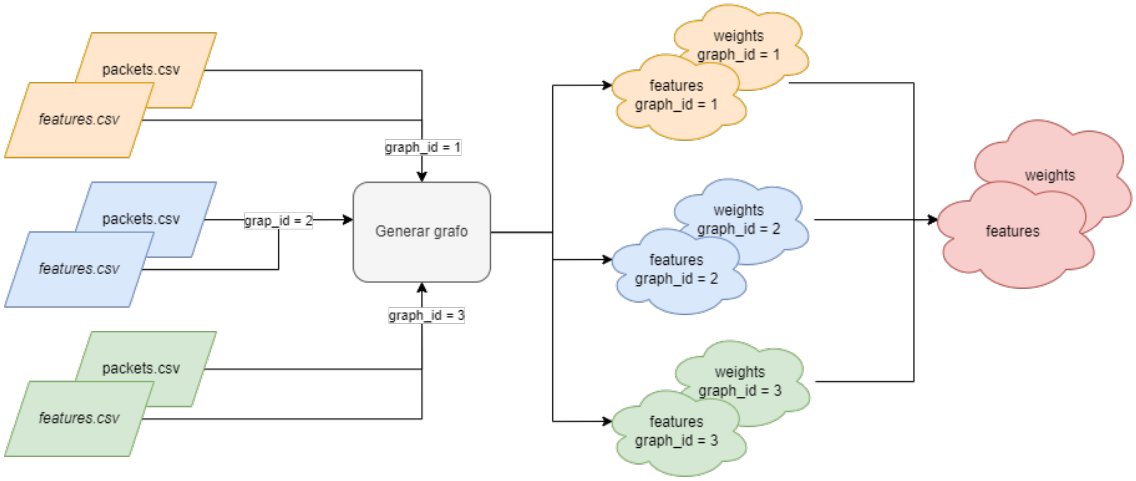



Figura 5.4: Diagrama de construcción del grafo de una aplicación

Un grafo está formado por una serie de nodos y aristas, donde cada nodo se identifica con una dirección IP y puerto destino y un conjunto de características, y las aristas representan las conexiones entre los nodos con un peso determinado. En la Figura 5.5 se puede observar una aproximación de la estructura de un grafo.

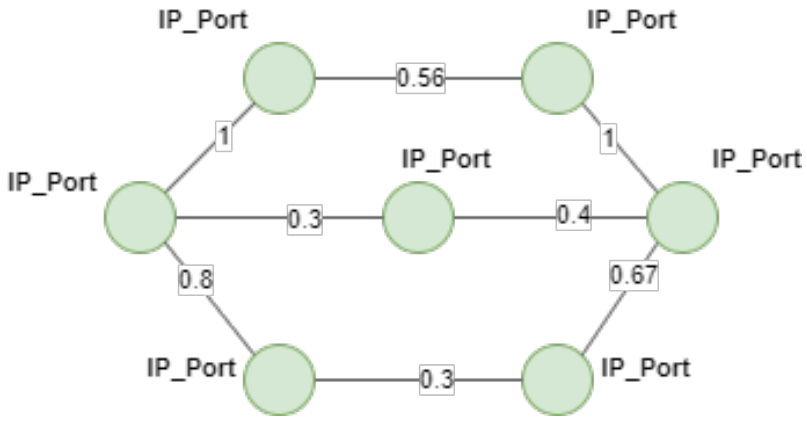



Figura 5.5: Ejemplo de un grafo de una aplicación

Los nodos se corresponden directamente con cada una de las ráfagas del fichero con las características extraídas (*features.csv*). El peso entre dos nodos se calcula como una aproximación de la correlación cruzada [19] con la longitud de la intersección entre los *timestamps* o marcas temporales de los paquetes pertenecientes a las ráfagas que representan dichos nodos.

```
1 def compute_weight(self,window_indx1, window_indx2):
2     intersection = window_indx1.intersection(window_indx2)
3     return len(intersection)
```

#### 5.2.2.4. Diagrama del proceso

En la Figura 5.6 se muestra un diagrama del proceso de construcción de los grafos descritos anteriormente.

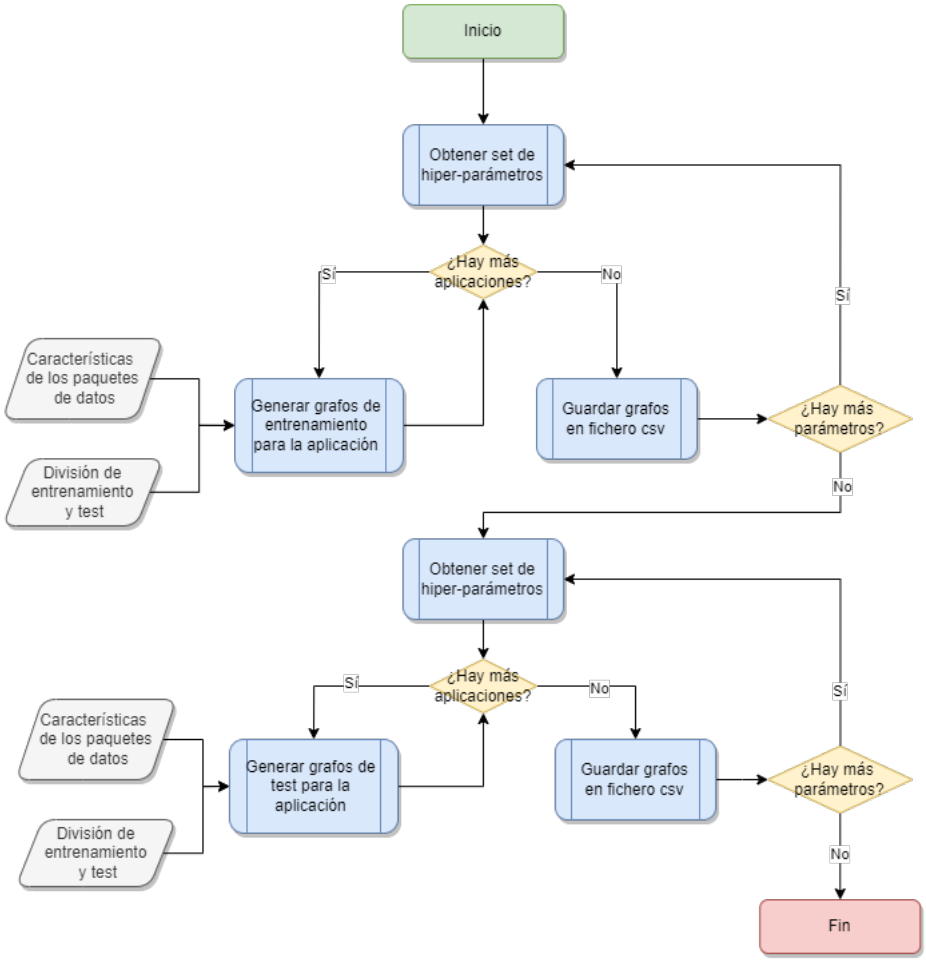



Figura 5.6: Diagrama del Proceso de Construcción de Grafos

## 5.3. Descripción del modelo

En esta Sección se describe detalladamente los modelos utilizados en la fase de entrenamiento.

### 5.3.1. Métricas a optimizar

Como ya se ha comentado en el Capítulo 3 Sección 4.1.4 es necesario establecer una métrica que permita evaluar el modelo. Con el objetivo de obtener el modelo más óptimo se establecieron unas métricas que permitieron comparar los modelos genererados e ir ajustando los parámetros para refinar el modelo.

Las métricas a optimizar son las siguientes:

- ***Recall* o Exhaustividad:** informa sobre la cantidad de elementos que se pueden identificar de forma correcta y se calcula como:

$$\frac{TP}{TP + FN} \tag{5.1}$$

- ***Precision* o Precisión:** mide la calidad del modelo según la cantidad de elementos identificados como "$X$" correctamente del total de elementos identificados como "$X$" y se calcula como:

$$\frac{TP}{TP + FP} \tag{5.2}$$

- ***F1 Score*:** permite evaluar el rendimiento entre la precisión y la exhaustividad del modelo y se calcula como:

$$\frac{2 * Precision * Recall}{Precision + Recall} \tag{5.3}$$

- **Exactitud o *Accuracy*:** es el porcentaje total de elementos clasificados correctamente y se calcula como:

$$\frac{TP + TN}{FP + FN} \tag{5.4}$$

  El resultado es un valor entre 0 y 1 siendo el modelo un mejor predictor cuanto mayor es este valor.

### 5.3.2. Modelo No Supervisado

Una vez construidos los grafos de entrenamiento y test a partir de las capturas del tráfico de las diferentes aplicaciones se pasan al entrenamiento del modelo con la previa extracción de los datos necesaria.

#### 5.3.2.1. Extracción de los grafos

Para poder realizar el entrenamiento del modelo, es necesario extraer los datos de los grafos generados para que puedan ser introducidos en el modelo de aprendizaje.

Para ello, es necesario elegir uno de los diferentes conjuntos de datos extraídos según los hiperparámetros comentados en la sección **??**. Esta extracción es idéntica para los diferentes conjuntos aunque significativa para el posterior entrenamiento y test ya que estos pueden dar distintos resultados.

Una vez elegido el conjunto de datos, dados los archivos con las características y pesos de cada aplicación, utilizando la librería de Python StellarGraph[6] se crean los grafos contenidos en los archivos para cada aplicación como instancias de grafos de StellarGraph, que serán más tarde introducidos en un modelo de entrenamiento de grafos de esta misma librería. Se extraen los nodos y los pesos de cada uno agrupando todas las entradas del dataframe que pertenezcan al mismo nodo y los pesos pertenecientes a estos mismos nodos.

Para más tarde poder hacer el test, a cada grafo se le asigna su etiqueta que es el nombre de la aplicación a la que pertenece. Cuando se han extraido todos los datos se obtinene una lista con todas las etiquetas de cada grafo que tiene el mismo tamaño que el número de grafos del dataset.

#### 5.3.2.2. Entrenamiento

El entrenamiento del modelo no supervisado se hace utilizando la librería de Python StellarGraph[6], la cual ofrece algoritmos de aprendizaje automático sobre grafos. Se ha utilizado un modelo de red neuronal convolucional de grafos en el que las capas, las funciones de activación y el número de épocas (las veces que se vuelve hacía atrás para hacer correcciones en la pérdida de la red, cuanto menor sea la pérdida mejor) han sido elegidas haciendo múltiples pruebas con diferentes configuraciones que se verán más en detalle a continuación en la Sección 5.4.

El entrenamiento se hace sobre la similitud de un número de parejas de grafos con la distancia o similitud entre ellos. El número de parejas de grafos establecidos es igual al número de grafos. Se ha establecido de esta manera según los resultados dados en la Sección 5.5. Para calcular la similitud entre grafos se han considerado dos métricas diferentes:

- Distancia entre los autovectores de la matriz Laplaciana de los grafos. Matriz Laplaciana tomada de la librería NetworkX[16].
- Distancia Jaccard entre dos grafos. Obtenida de la librería ScyPy[36].

A partir del entrenamiento, se calculan vectores embedidos para realizar la clasificación supervisada con algoritmos de aprendizaje automático a partir del entrenamiento no supervisado.

#### 5.3.2.3. Clasificación

Para la clasificación se utilizaron los algoritmos de aprendizaje automático vistos en el Capítulo 3 Sección 3.2 a partir de los datos calculados en el vector embebido a partir de los datos del entrenamiento no supervisado. Para todos ellos se ha hecho uso de las funciones *RandomizedSearchCV* y *GridSearchCV* de la librería *SKLearn* de *Python*. Estas se encargan de buscar los mejores parámetros para los clasificadores con los datos que se le proporcionan a este.

En el caso de *RandomizedSearchCV* no se prueban todos los parámetros posibles para los clasificadores. Se hace una distribución a partir de el número de iteraciones deseado, reduciendo asi el tiempo de ejecución comparado con *GridSearchCV*. Esto se ha utilizado para el XGBoost ya que su tiempo de ejecución es más elevado que SVM y *Random Forest*.

La funcion *GridSearchCV* se ejecuta para todos los parámetros seleccionados, dando asi la configuración óptima para los algoritmos, con la desventaja de tener un tiempo de ejecución más elevado. Esto se ha utilizado para *Random Forest* y SVM ya que el tiempo de ejecución para los parámetros seleccionados es menor que para XGBoost y se ejecutó en un tiempo razonable.

Se ha utilizado una división de los datos de la siguiente manera: 80 % de los datos para entrenamiento y el 20 % restante para test.

## 5.4. Parámetros

El modelo de aprendizaje no supervisado y los algoritmos de clasificación utilizados, tienen una serie de parámetros de los que depende el resultado en la clasificación del dataset elegido. En esta sección se muestran los parámetros utilizados en la búsqueda de la mejor predicción posible.

### 5.4.1. GCN no supervisada

La GCN no supervisada utilizada como se ha visto en la Sección 5.3.2.2 se ha tomado de la librería de Python StellarGraph[6]. De este modelo se ha probado a cambiar la configuración de las capas, diferentes funciones de activación, diferentes tamaños de las muestras y épocas. Estas pruebas se ha realizando ejecuciones para cada uno de los diferentes parámetros.

Al ser un modelo no supervisado, para saber como de buena es la predicción, es necesario hacer uso de un clasificador supervisado de la forma que se ha visto en las Secciónes 5.3.2.2 y 5.3.2.3. Teniendo esto en cuenta, la ejecución para buscar los mejores parámetros se ha realizado dos veces: una con un clasificador *Random Forest* por defecto, es decir, sin haber buscado los mejores parámetros para este y la segunda vez, con todos los clasificadores ya en su configuración óptima encontrada, para de esta manera, validar los resultados obtenidos antes de la búsqueda de la configuración óptima.

Los parámetros utlilzados fueron:

- **Configuración de las capas:** Para todas las pruebas se han utilizado dos capas ocultas. Para cada capa se indica el número de neuronas. Se han utilizado las siguientes configuraciones: *([64,32], [128,64], [256,128], [512,256], [1024,512], [2048,1024])* para la primera ejecución y *([64,32], [128,64], [256,128], [512,256])* para la segunda. Para esta segunda ejecución se eliminaron las configuraciones que tenían peores resultados.

- **Funciones de activación:** se utilizaron las vistas en la Sección 3.3.1, la tangente hiperbólica (como *tanh* en el código), sigmoide (como *sigmoid* en el código) y Relu (como *relu* en el código). Se utilizaron las siguientes configuraciones: *(['tanh','tanh'])* únicamente en la primera ejecución dado el alto número de configuraciones de capas

y *(['sigmoid','sigmoid'],['sigmoid','relu'],['sigmoid','tanh'],['tanh','sigmoid'])* en la segunda ejecución.

- **Tamaños de las muestras:** El tamaño de las muestras indica el tamaño de las muestras que se va a pasar a la ejecución. Si por ejemplo se tienen 1000 grafos y el tamaño de la muestra esta establecido en 20, se pasarán los 20 primeros grafos sobre la configuración de capas y sobre el resultado de esta ejecución se pasarán los 20 siguientes grafos y asi sucesivamente hasta haber ejecutado los 1000 grafos. Los diferentes tamaños de las muestras han sido: *(10, 20, 40, 60, 80, 100)* para la primera ejecución y *(20)* únicamente para la segunda ejecución dados los resultados anteriores.
- **Épocas:** Las épocas son el número de veces que se pasa el conjunto de datos por la configuración de la red pero con los datos de cada época guardados en el modelo, lo que hace que en cada época se aprenda sobre los datos de la época anterior. El número de épocas utilizado fue: *(10, 50, 100)* en la primera ejecución y *(50,70,100)* en la segunda ejecución.
- **Similitud entre grafos:** Como se ha visto en la Sección 5.3.2.2 para el entrenamiento se han utilizado dos medidas de similitud entre grafos: Distancia entre los autovectores de la matriz Laplaciana de los grafos y la distancia Jaccard entre los grafos.

### 5.4.2. *Random Forest*

Para buscar los mejores parámetros para el *Random Forest* como se ha comentado en la Sección 5.3.2.3 se ha utilizado la función *GridSearchCV* que realiza una búsqueda de los mejores parámetros elegidos, ejecutando todas las combinaciones posibles. Los parámetros a optimizar fueron:

- *max_depth*: Las máximas profundidades fueron: *(14, 20, 30, 100)*.
- *min_samples_leaf*: Las muestras utilizadas fueron: *(2, 3, 4, 5)*.
- *min_samples_split*: Los números de muestras utlilzadas fueron: *(1, 2, 3, 4)*.
- *n_estimators*: Los números de estimadores utilizados fueron: *(100, 200, 400, 600, 800, 1000)*.

### 5.4.3. XGBoost

Para buscar los mejores parámetros para el XGBoost como se ha comentado en la Sección 5.3.2.3 se ha utilizado la función *RandomSearchCV* que realiza una búsqueda de los mejores parámetros elegidos, algunas de las combinaciones posibles. Los parámetros a optimizar fueron:

- *learning_rate*: Las velocidades utilizadas fueron: *(0.1, 0.01, 0.001)*
- *n_estimators*:Las muestras utilizadas fueron: *(50, 100, 300, 500)*.
- *max_depth*: Las profundidades utilizadas fueron: *(2, 3, 5, 7)*.

- *min_child_weight*: Las muestras utilizadas fueron: *(2, 3, 5, 7)*.
- *reg_alpha*: Las regularizaciones utilizadas fueron: *(0.1, 1, 10, 100)*.

### 5.4.4. SVM

Para buscar los mejores parámetros para el SVM como se ha comentado en la Sección 5.3.2.3 se ha utilizado la función *GridSearchCV* que realiza una búsqueda de los mejores parámetros elegidos, ejecutando todas las combinaciones posibles. Los parámetros a optimizar fueron:

- *gamma*: Los coeficientes utilizados fueron: *(0.1, 1, 10)*
- *C*: Los parámetros utilizados fueron: *(0.1, 1, 10, 100)*.
- *degree*: Los grados utilizados fueron: *(0, 1, 2, 3, 4)*.

## 5.5. Experimentos

En esta sección se presentan los experimentos de clasificación realizados con el modelo no supervisado descrito anteriormente en la Sección 5.3.2 con los parámetros descritos en la Sección 5.4. Los experimentos realizados con el modelo no supervisado y los algoritmos de clasificación se han ejecutado en el entorno de ejecución de *Google Colab* ya que permite obtener *Graphical Processing Unit* (GPU)s en la red para ejecutar los programas mucho más rápidamente ya que estos requieren procesar una gran cantidad de datos. Para mantener las mismas condiciones en los tiempos de ejecución de los algoritmos, se tuvo en cuenta la GPU asignada para tener igualdad entre todos ellos.

Antes de realizar ningun experimento, se tuvo que decidir las métricas a optimizar. En base al tipo de datos se consideró que al ser un problema de clasificación multiclase en el que todas las clases tienen el mismo valor y un error en la clasificación de estos no es significativo, la métrica que se buscó optimizar fue el *Accuracy* aunque también se tuvo en cuenta la media del *recall* y precisión para comparar todas las aplicaciónes junto con la matriz de confusion de todas ellas.

Como se ha visto en las Secciónes **??** y 5.2.2.2, el tráfico de nuestro dataset esta dividido en diferentes configuraciones. Para todas las configuraciones se han realizado experimentos aunque la única configuración de la que se van a mostrar los experimentos es la vista en la Tabla 5.1 ya que el rendimiento de esta ha sido la mejor de todas las configuraciones de parámetros probadas en el modelo no supervisado y los algoritmos de clasificación supervisada para este. En todas las otras configuraciones, los resultados fueron similares entre los algoritmos pero en todas con un rendimiento inferior o similar al obtenido en la configuración para la que se van a mostrar los experimentos.

En las secciones a continuación, se va a mostrar los resultados obtenidos con los tres algoritmos de clasificación supervisados, teniendo en cuenta que para todos ellos se ha utilizado el aprendizaje no supervisado con GCN como entrenamiento visto en la Sección 5.3.2.2 y con los parámetros comentados en la Sección 5.4.1. Algunas configuraciones de los parámetros para la GCN no han sido ejecutadas para todos los algoritmos ya que primero se buscaron los mejores resultados para *Random Forest* para probar más fácilmente las

mejores configuraciones de la GCN y después se extendió a los otros dos algoritmos con lo que se tienen más pruebas con diferentes parámetros en *Random Forest*. Para los otros dos algoritmos se probó alguna de las configuraciones utilizadas incialmente para *Random Forest* y se vió que los mejores parámetros en la GCN tienen buen rendimiento para todos los algoritmos para los que se han realizado pruebas.

Las pruebas se han realizado con el dataset comentado en la Sección 5.1. De este se consiguen sacar en total 1940 nodos de las aplicaciones analizadas.

| $T_{window}$ | Overlap | N | $T_{slice}$ | Similitud |
|---|---|---|---|---|
| 5 | 3 | 7 | 10 | *Jaccard* |

Tabla 5.1: Configuración de los grafos más utilizada

### 5.5.1. *Random Forest*

Inicialmente se realizaron pruebas con *Random Forest* para buscar los mejores parámetros a utilizar en la GCN. Se eligieron unos cuantos valores para los parámetros a optimizar vistos en la Sección 5.4.1. La métrica a optimizar para estas pruebas fué el *accuracy*. Pruebas con tamaño de muestra 20 en Tabla 5.2. Pruebas con tamaño de muestra 60 en Tabla 5.3. Pruebas con tamaño de muestra 100 en Tabla 5.5.

| Épocas | Capas | *Accuracy* |
|---|---|---|
| 10 | [64,32] | 0.858 |
| | [128,64] | 0.806 |
| | [256,128] | 0.795 |
| | [512,256] | 0.817 |
| | [1024,512] | 0.836 |
| | [2048,1024] | 0.833 |
| 50 | [64,32] | 0.869 |
| | [128,64] | 0.866 |
| | [256,128] | 0.899 |
| | [512,256] | 0.855 |
| | [1024,512] | 0.88 |
| | [2048,1024] | 0.863 |
| 100 | [64,32] | 0.825 |
| | [128,64] | 0.858 |
| | [256,128] | 0.893 |
| | [512,256] | 0.888 |
| | [1024,512] | 0.91 |
| | [2048,1024] | 0.852 |

Tabla 5.2: Prueba de parámetros en GCN con 20 en el tamaño de las muestras

| Épocas | Capas | *Accuracy* |
|---|---|---|
| 10 | [64,32] | 0.82 |
| | [128,64] | 0.839 |
| | [256,128] | 0.817 |
| | [512,256] | 0.817 |
| | [1024,512] | 0.792 |
| | [2048,1024] | 0.801 |
| 50 | [64,32] | 0.855 |
| | [128,64] | 0.825 |
| | [256,128] | 0.842 |
| | [512,256] | 0.814 |
| | [1024,512] | 0.776 |
| | [2048,1024] | 0.839 |
| 100 | [64,32] | 0.822 |
| | [128,64] | 0.833 |
| | [256,128] | 0.797 |
| | [512,256] | 0.831 |
| | [1024,512] | 0.855 |
| | [2048,1024] | 0.809 |

Tabla 5.3: Prueba de parámetros en GCN con 60 en el tamaño de las muestras

| Épocas | Capas | *Accuracy* |
|---|---|---|
| 10 | [64,32] | 0.792 |
| | [128,64] | 0.861 |
| | [256,128] | 0.82 |
| | [512,256] | 0.79 |
| | [1024,512] | 0.727 |
| | [2048,1024] | 0.817 |
| 50 | [64,32] | 0.833 |
| | [128,64] | 0.85 |
| | [256,128] | 0.787 |
| | [512,256] | 0.817 |
| | [1024,512] | 0.82 |
| | [2048,1024] | 0.861 |
| 100 | [64,32] | 0.842 |
| | [128,64] | 0.798 |
| | [256,128] | 0.825 |
| | [512,256] | 0.79 |
| | [1024,512] | 0.773 |
| | [2048,1024] | 0.825 |

Tabla 5.4: Prueba de parámetros en GCN con 100 en el tamaño de las muestras

Como se puede observar en las tres tablas, los mejores parámetros obtenidos fueron con el tamaño de la muestra en 20, 50 épocas y [256,128] como configuración de capas, junto con el tamaño de la muestra 20, 100 épocas y [1024,512] como configuración de capas, donde la primera configuración obtiene prácticamente un **90 %** de *accuracy* y la segunda un **91 %**. Con estos resultados, se eligió la primera configuración ya que necesita la mitad de épocas y muchas menos neuronas para obtener prácticamente el mismo resultado necesitando de esta manera menos tiempo de ejecución. Para esta misma configuración y con el dóble de épocas, el *accuracy* se queda practicamente en el mismo. Con todo esto analizado se procedió a utilizar esta configuración de 50 épocas de la GCN para buscar la mejor configuración del *Random Forest*.

Se pasó a ejecutar la función *GridSearchCV* para buscar los mejores parámetros del *Random Forest*, donde se eligieron los parámetros a probar, comentados en la Sección 5.4.2. Los mejores parámetros fueron:

- *max_depth*: 100 como máxima profundidad de cada árbol del bosque.
- *min_samples_leaf*: 1 como mínimo de muestras requeridas en en un nodo hoja.
- *min_samples_split*: 3 como mínimo de muestras requeridas para separar un nodo.
- *n_estimators*: 600 como número de estimadores o árboles.

Se procedió a realizar pruebas con estos parámetros y utilizando diferentes épocas para contrastar los anteriores resultados en la búsqueda de los mejores parámetros de la GCN y ver si había alguna diferencia. También se tomó en cuenta la precisión y el *recall* a partir del peso de cada aplicación, es decir, a partir del soporte que tuviese cada aplicación.

| Épocas | *Accuracy* | Precisión | *Recall* |
|---|---|---|---|
| 10 | 0.82 | 0.84 | 0.82 |
| 20 | 0.88 | 0.88 | 0.88 |
| 30 | 0.87 | 0.89 | 0.87 |
| 40 | 0.87 | 0.87 | 0.87 |
| 50 | 0.92 | 0.93 | 0.92 |
| 60 | 0.90 | 0.90 | 0.90 |
| 70 | 0.88 | 0.88 | 0.88 |
| 80 | 0.89 | 0.89 | 0.89 |
| 90 | 0.89 | 0.90 | 0.89 |
| 100 | 0.88 | 0.88 | 0.88 |

Tabla 5.5: Prueba de mejores parámetros en *Random Forest* por epocas

Viendo los resultados se consideró que los resultados de las épocas con los realizados previamente eran acordes y se demostró que la mejor configuración de la GCN hasta el momento con la mejor configuración del *Random Forest* daba los mejores resultados, en torno a un **92 %** de *accuracy*. Los valores de precisión, *recall* y *f1-score* para cada aplicación con los mejores parámetros probados, fueron los mostrados en la Tabla 5.6.

| Aplicación | Precisión | *Recall* | *F1-score* | Soporte |
|---|---|---|---|---|
| *App1* | 1.00 | 0.90 | 0.95 | 21 |
| *App2* | 1.00 | 0.50 | 0.67 | 8 |
| *App3* | 1.00 | 0.75 | 0.86 | 16 |
| *App4* | 0.89 | 0.94 | 0.91 | 34 |
| *App5* | 0.95 | 1.00 | 0.98 | 40 |
| *App6* | 1.00 | 0.20 | 0.33 | 5 |
| *App7* | 0.97 | 0.88 | 0.92 | 33 |
| *App8* | 0.96 | 0.96 | 0.96 | 46 |
| *App9* | 1.00 | 0.87 | 0.93 | 15 |
| *App10* | 0.80 | 0.96 | 0.87 | 49 |
| *App11* | 0.91 | 0.99 | 0.95 | 80 |
| *App12* | 0.97 | 0.91 | 0.94 | 34 |

Tabla 5.6: Prueba de mejores parámetros en *Random Forest*

| Predicted | App1 | App2 | App3 | App4 | App5 | App6 | App7 | App8 | App9 | App10 | App11 | App12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **App1** | 19 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| **App2** | 0 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 2 | 0 |
| **App3** | 0 | 0 | 12 | 1 | 0 | 0 | 0 | 0 | 0 | 3 | 0 | 0 |
| **App4** | 0 | 0 | 0 | 32 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 |
| **App5** | 0 | 0 | 0 | 0 | 40 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **App6** | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 3 | 0 |
| **App7** | 0 | 0 | 0 | 0 | 1 | 0 | 29 | 1 | 0 | 1 | 0 | 1 |
| **App8** | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 44 | 0 | 0 | 1 | 0 |
| **App9** | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 13 | 1 | 0 | 0 |
| **App10** | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 47 | 1 | 0 |
| **App11** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 79 | 0 |
| **App12** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 31 |

Tabla 5.7: Matriz de confusión de clasificación aplicaciones con *Random Forest*

Se puede ver que con los valores seleccionados en la GCN y en el algoritmo de clasificación se obtiene un **92 %** de *accuracy*. Esto se puede considerar como una clasificación general de las aplicaciones precisa. Se puede ver en la Tabla 5.7 la matriz de confusión para todas las aplicaciones probadas donde se puede ver para cada muestra de test que aplicación se le asigna en la clasificación. Casi todas las aplicaciones son clasificadas de manera muy precisa como por ejemplo *App4* que con 40 muestras de test no falla en ninguna y se le asignan solo dos que no son de otras aplicaciones. Algunas como *App6* o *App2* no están tan bien clasificadas. Esto se puede deber a la falta de datos de estas aplicaciones, ya que se puede ver que tienen poco soporte y son las aplicaciones que menos datos tienen en el dataset utilizado.

Después de estas primeras pruebas, se pasó a realizar otra prueba de configuraciones en la GCN, esta vez con algunas de las configuraciones de capas anteriormente probadas y nuevas funciones de activación para estas. Para la segunda ejecución las configuraciones de los diferentes parámetros son las comentadas en la Sección 5.4.1. Esta vez se utilizó la configuración del *Random Forest* que mejores resultados dió. De esta manera, se buscaba encontrar una configuración de funciones de activación que mejorase los resultados obtenidos hasta el momento. Esta vez también se tomó el tiempo de ejecución de los algoritmos de clasificacion supervisados. Los resultados son los de la tabla 5.8.

| Épocas | Capas | Activación | Accuracy | Tiempo de Ejecucion |
|---|---|---|---|---|
| 50 | [64,32] | *['sigmoid', 'sigmoid']* | 0.87 | 9.18 segundos |
| | | *['sigmoid', 'relu']* | 0.88 | 6.83 segundos |
| | | *['sigmoid', 'tanh']* | 0.87 | 9.13 segundos |
| | | *['tanh', 'sigmoid']* | 0.85 | 10.45 segundos |
| | [128,64] | *['sigmoid', 'sigmoid']* | 0.88 | 12.97 segundos |
| | | *['sigmoid', 'relu']* | 0.90 | 9.57 segundos |
| | | *['sigmoid', 'tanh']* | 0.89 | 13.51 segundos |
| | | *['tanh', 'sigmoid']* | 0.91 | 13.89 segundos |
| | [256,128] | *['sigmoid', 'sigmoid']* | 0.89 | 18.57 segundos |
| | | *['sigmoid', 'relu']* | 0.87 | 13.12 segundos |
| | | *['sigmoid', 'tanh']* | 0.89 | 14.86 segundos |
| | | *['tanh', 'sigmoid']* | 0.90 | 18.67 segundos |
| | [512,256] | *['sigmoid', 'sigmoid']* | 0.89 | 21.21 segundos |
| | | *['sigmoid', 'relu']* | 0.87 | 17.78 segundos |
| | | *['sigmoid', 'tanh']* | 0.90 | 17.87 segundos |
| | | *['tanh', 'sigmoid']* | 0.93 | 24.34 segundos |
| 70 | [64,32] | *['sigmoid', 'sigmoid']* | 0.87 | 9.34 segundos |
| | | *['sigmoid', 'relu']* | 0.90 | 6.84 segundos |
| | | *['sigmoid', 'tanh']* | 0.92 | 9.36 segundos |
| | | *['tanh', 'sigmoid']* | 0.90 | 9.24 segundos |
| | [128,64] | *['sigmoid', 'sigmoid']* | 0.89 | 12.99 segundos |
| | | *['sigmoid', 'relu']* | 0.91 | 9.23 segundos |
| | | *['sigmoid', 'tanh']* | 0.91 | 13.52 segundos |
| | | *['tanh', 'sigmoid']* | 0.87 | 12.47 segundos |
| | [256,128] | *['sigmoid', 'sigmoid']* | 0.90 | 17.71 segundos |
| | | *['sigmoid', 'relu']* | 0.91 | 13.32 segundos |
| | | *['sigmoid', 'tanh']* | 0.86 | 14.59 segundos |
| | | *['tanh', 'sigmoid']* | 0.90 | 16.88 segundos |
| | [512,256] | *['sigmoid', 'sigmoid']* | 0.92 | 22.63 segundos |
| | | *['sigmoid', 'relu']* | 0.88 | 19.79 segundos |
| | | *['sigmoid', 'tanh']* | 0.90 | 18.13 segundos |
| | | *['tanh', 'sigmoid']* | 0.88 | 23.35 segundos |
| 100 | [64,32] | *['sigmoid', 'sigmoid']* | 0.89 | 8.85 segundos |
| | | *['sigmoid', 'relu']* | 0.88 | 6.89 segundos |
| | | *['sigmoid', 'tanh']* | 0.90 | 8.91 segundos |
| | | *['tanh', 'sigmoid']* | 0.87 | 7.72 segundos |
| | [128,64] | *['sigmoid', 'sigmoid']* | 0.90 | 12.67 segundos |
| | | *['sigmoid', 'relu']* | 0.91 | 9.85 segundos |
| | | *['sigmoid', 'tanh']* | 0.88 | 12.38 segundos |
| | | *['tanh', 'sigmoid']* | 0.87 | 9.93 segundos |
| | [256,128] | *['sigmoid', 'sigmoid']* | 0.91 | 18.10 segundos |
| | | *['sigmoid', 'relu']* | 0.89 | 13.50 segundos |
| | | *['sigmoid', 'tanh']* | 0.91 | 14.20 segundos |
| | | *['tanh', 'sigmoid']* | 0.88 | 12.66 segundos |
| | [512,256] | *['sigmoid', 'sigmoid']* | 0.91 | 21.17 segundos |
| | | *['sigmoid', 'relu']* | 0.92 | 18.22 segundos |
| | | *['sigmoid', 'tanh']* | 0.90 | 18.18 segundos |
| | | *['tanh', 'sigmoid']* | 0.89 | 18.96 segundos |

Tabla 5.8: Segunda prueba de capas en GCN con funciones de activación utilizando *Random Forest*

Como se puede ver, todas las pruebas tienen un resultado más o menos bueno para el conjunto de datos utilizado, siendo el mejor resultado un *accuracy* de un **93 %**, su configuración es [512,256] como capas, *['tanh', 'sigmoid']* como funciones de activación y 50 épocas, ligeramente mejor que con los mejores parámetros anteriores. Teniendo en cuenta que su tiempo de ejecución es el más elevado, 24.34 segundos, se ha tenido que evaluar el resto de parámetros para considerar el mejor de ellos. Después de analizar todos los resultados, se puede ver que los parámetros que han hecho al algoritmo más eficiente han sido 70 épocas, [64,32] como capas y *['sigmoid', 'tanh']* como capas de activación con un **92 %** de *accuracy* y un tiempo de ejecución de 9.36 segundos. Se ha comparado el tiempo de ejecución de estos nuevos mejores parámetros y los anteriores ya que el *accuracy* es el mismo, donde estos últimos parámetros tienen un tiempo de ejecución menor.

Comparando la Tabla 5.8 y la Tabla 5.5 parece que a partir de 70 épocas, el algoritmo se estanca y no es capaz de mejorar los anteriores resultados con ménos épocas, por lo que el rendimiento óptimo de este algoritmo parece estar entre las 40 y 70 épocas, dependiendo de la configuración utilizada en la GCN.

La matriz de confusión y los resultados de cada aplicación son práticamente idénticos que para los anteriores parámetros por lo que no han sido incluidos.

### 5.5.2. XGBoost

Una vez hecha la primera prueba con *Random Forest* para buscar los mejores parámetros de la GCN, como hemos visto en la Sección 5.4.3 con los parámetros y valores elegidos, se pasó a buscar los mejores parámetros del XGBoost con la función *RandomizedSearchCV*. Los mejores parámetros fueron:

- *learning_rate*: 0.1 como velocidad de aprendizaje.
- *n_estimators*: 500 como número de estimadores o árboles.
- *max_depth*: 5 como máxima profundidad de cada árbol del bosque.
- *min_child_weight*: 5 como suma mínima del peso necesario para un hijo.
- *reg_alpha*: 0.1 como regularizacion L1 en término de pesos.

Con estos parámetros obtenidos se paso a probar con diferentes épocas para comprobar resultados con los otros algoritmos y con varios números de épocas para ver la evolución de los resultados con estos parámetros. Los parámetros de la GCN utilizados fueron los mejores obtenidos en la primera ejecución de búsqueda de los mejores parámetros para esta comentado al inicio de la Sección 5.5.1. Los resultados en esta ejecución se muestran en la Tabla 5.9

| Épocas | *Accuracy* | Precisión | *Recall* |
|---|---|---|---|
| 10 | 0.84 | 0.84 | 0.84 |
| 20 | 0.87 | 0.88 | 0.87 |
| 30 | 0.86 | 0.87 | 0.86 |
| 40 | 0.88 | 0.88 | 0.88 |
| 50 | 0.89 | 0.89 | 0.89 |
| 60 | 0.90 | 0.90 | 0.90 |
| 70 | 0.87 | 0.87 | 0.87 |
| 80 | 0.89 | 0.89 | 0.89 |
| 90 | 0.90 | 0.90 | 0.90 |
| 100 | 0.85 | 0.85 | 0.85 |

Tabla 5.9: Prueba con mejores parámetros en *XGBoost*

Como se puede ver los mejores resultados fueron con 60 y 90 épocas, con un *accuracy*, precisión y *recall* del **90 %**. Con la misma configuración de la GCN y el mismo conjunto de datos utilizado el rendimiento del XGBoost es algo inferior al del Random Forest no solo en *accuracy*, también en tiempo de ejecución que aunque en esta etapa no se tomaron datos sobre esto, si se notó esa diferencia de un tiempo de ejecución bastante mayor. Los valores de precisión, *recall* y *f1-score* de cada aplicación para los mejores parámetros y épocas probados fueron los mostrados en la Tabla 5.10. Se puede ver que los resultados generales son precisos. Hay alguna clasificación de aplicaciones específicas que destacan por encima de otras con un rendimiento más pobre, como puede ser el caso de *App11* que su clasificación esta por encima de un **95 %**. Esto se puede deber a la cantidad de datos en el dataset utilizado. En la Tabla 5.11 podemos ver la matriz de confusión de la clasificación de aplicaciones con los parámetros utilizados.

| Aplicación | Precisión | *Recall* | *F1-score* | Soporte |
|---|---|---|---|---|
| *App1* | 0.89 | 0.76 | 0.82 | 21 |
| *App2* | 0.67 | 0.25 | 0.36 | 8 |
| *App3* | 0.89 | 0.50 | 0.64 | 16 |
| *App4* | 0.91 | 0.88 | 0.90 | 34 |
| *App5* | 0.95 | 0.97 | 0.96 | 40 |
| *App6* | 0.80 | 0.80 | 0.80 | 5 |
| *App7* | 0.86 | 0.94 | 0.90 | 33 |
| *App8* | 0.90 | 0.96 | 0.93 | 46 |
| *App9* | 0.93 | 0.93 | 0.93 | 15 |
| *App10* | 0.78 | 0.96 | 0.86 | 49 |
| *App11* | 0.95 | 0.96 | 0.96 | 80 |
| *App12* | 0.97 | 0.88 | 0.92 | 34 |

Tabla 5.10: Prueba de mejores parámetros con 60 épocas en *XGBoost*

| Predicted | App1 | App2 | App3 | App4 | App5 | App6 | App7 | App8 | App9 | App10 | App11 | App12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **App1** | 16 | 1 | 0 | 0 | 1 | 0 | 2 | 0 | 0 | 1 | 0 | 0 |
| **App2** | 0 | 2 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| **App3** | 0 | 0 | 8 | 0 | 0 | 0 | 2 | 1 | 0 | 4 | 1 | 0 |
| **App4** | 0 | 0 | 0 | 30 | 0 | 0 | 1 | 0 | 0 | 2 | 1 | 0 |
| **App5** | 0 | 0 | 0 | 1 | 39 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **App6** | 0 | 0 | 0 | 1 | 0 | 4 | 0 | 0 | 0 | 0 | 3 | 0 |
| **App7** | 0 | 0 | 0 | 0 | 0 | 0 | 31 | 1 | 0 | 1 | 0 | 0 |
| **App8** | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 44 | 1 | 0 | 0 | 0 |
| **App9** | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 14 | 0 | 0 | 0 |
| **App10** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 47 | 0 | 0 |
| **App11** | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 77 | 1 |
| **App12** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 1 | 30 |

Tabla 5.11: Matriz de confusión de clasificación aplicaciones con *XGBoost*

Dada la tardanza en la ejecución comparada con los otros algoritmos, se decidió bajar el número de estimadores al XGBoost para obtener un tiempo similar a los otros dos algoritmos. Esto se tomó en cuenta con la segunda ejecución de búsqueda de mejores parámetros para la GCN donde se utilizaron los tres algoritmos de clasificación con sus mejores parámetros y donde se tomó en cuenta el tiempo de ejecución. Los resultados de esta ejecución para el *XGBoost* están en la Tabla 5.12.

Como se puede ver para prácticamente todas las ejecuciones de 100 épocas, el algoritmo se estanca y no gana rendimiento comparado con 50 o 70 épocas. El funcionamiento de este algoritmo con las diferentes configuraciones probadas para la GCN parece que es óptimo con un número de épocas entre 50 y 100 épocas, dependiendo de la configuración utilizada.

Con el numero de estimadores reducido a un 10 % de los mejores parámetros seleccionados por la función *RandomizedSearchCV*, el tiempo de ejecución para los casos con menos neuronas son similares a los obtenidos con *Random Forest* pero su rendimiento esta un poco por debajo. Teniendo en cuenta todos los casos el que mejor *accuracy* tiene es la configuracion de capas [512,256] teniendo como función de activación *['sigmoid', 'sigmoid']* y habiendo ejecutado 70 épocas, con un **91 %** y un tiempo de ejecución de casi 50 segundos. Este es el doble de lento que el mejor de *Random Forest* y 10 veces mas lento que el seleccionado óptimo con un *accuracy* ligeramente inferior. Aun así, el tiempo de ejecución y el rendimiento son mejores que para los mejores parámetros probados anteriormente, lo que hace que estos parámetros de la GCN sean los mejores para este algoritmo con el dataset utilizado.

| Épocas | Capas | Activación | Accuracy | Tiempo de Ejecucion |
|---|---|---|---|---|
| 50 | [64,32] | *['sigmoid', 'sigmoid']* | 0.86 | 7.19 segundos |
| | | *['sigmoid', 'relu']* | 0.83 | 5.66 segundos |
| | | *['sigmoid', 'tanh']* | 0.83 | 7.19 segundos |
| | | *['tanh', 'sigmoid']* | 0.83 | 7.74 segundos |
| | [128,64] | *['sigmoid', 'sigmoid']* | 0.85 | 14.51 segundos |
| | | *['sigmoid', 'relu']* | 0.87 | 11.10 segundos |
| | | *['sigmoid', 'tanh']* | 0.84 | 14.30 segundos |
| | | *['tanh', 'sigmoid']* | 0.86 | 15.00 segundos |
| | [256,128] | *['sigmoid', 'sigmoid']* | 0.87 | 28.53 segundos |
| | | *['sigmoid', 'relu']* | 0.85 | 22.17 segundos |
| | | *['sigmoid', 'tanh']* | 0.86 | 23.62 segundos |
| | | *['tanh', 'sigmoid']* | 0.87 | 27.06 segundos |
| | [512,256] | *['sigmoid', 'sigmoid']* | 0.85 | 52.13 segundos |
| | | *['sigmoid', 'relu']* | 0.86 | 41.72 segundos |
| | | *['sigmoid', 'tanh']* | 0.86 | 42.17 segundos |
| | | *['tanh', 'sigmoid']* | 0.88 | 55.51 segundos |
| 70 | [64,32] | *['sigmoid', 'sigmoid']* | 0.83 | 6.99 segundos |
| | | *['sigmoid', 'relu']* | 0.86 | 5.79 segundos |
| | | *['sigmoid', 'tanh']* | 0.86 | 7.32 segundos |
| | | *['tanh', 'sigmoid']* | 0.86 | 7.63 segundos |
| | [128,64] | *['sigmoid', 'sigmoid']* | 0.87 | 14.05 segundos |
| | | *['sigmoid', 'relu']* | 0.88 | 11.42 segundos |
| | | *['sigmoid', 'tanh']* | 0.85 | 14.58 segundos |
| | | *['tanh', 'sigmoid']* | 0.82 | 14.40 segundos |
| | [256,128] | *['sigmoid', 'sigmoid']* | 0.88 | 29.22 segundos |
| | | *['sigmoid', 'relu']* | 0.89 | 21.78 segundos |
| | | *['sigmoid', 'tanh']* | 0.85 | 22.95 segundos |
| | | *['tanh', 'sigmoid']* | 0.88 | 27.05 segundos |
| | [512,256] | *['sigmoid', 'sigmoid']* | 0.91 | 49.82 segundos |
| | | *['sigmoid', 'relu']* | 0.87 | 40.65 segundos |
| | | *['sigmoid', 'tanh']* | 0.89 | 41.62 segundos |
| | | *['tanh', 'sigmoid']* | 0.87 | 54.79 segundos |
| 100 | [64,32] | *['sigmoid', 'sigmoid']* | 0.87 | 6.86 segundos |
| | | *['sigmoid', 'relu']* | 0.84 | 5.76 segundos |
| | | *['sigmoid', 'tanh']* | 0.83 | 7.05 segundos |
| | | *['tanh', 'sigmoid']* | 0.82 | 6.78 segundos |
| | [128,64] | *['sigmoid', 'sigmoid']* | 0.88 | 13.94 segundos |
| | | *['sigmoid', 'relu']* | 0.87 | 12.08 segundos |
| | | *['sigmoid', 'tanh']* | 0.86 | 14.23 segundos |
| | | *['tanh', 'sigmoid']* | 0.82 | 12.33 segundos |
| | [256,128] | *['sigmoid', 'sigmoid']* | 0.87 | 30.71 segundos |
| | | *['sigmoid', 'relu']* | 0.89 | 21.69 segundos |
| | | *['sigmoid', 'tanh']* | 0.89 | 22.76 segundos |
| | | *['tanh', 'sigmoid']* | 0.84 | 22.46 segundos |
| | [512,256] | *['sigmoid', 'sigmoid']* | 0.87 | 52.05 segundos |
| | | *['sigmoid', 'relu']* | 0.86 | 41.19 segundos |
| | | *['sigmoid', 'tanh']* | 0.88 | 41.10 segundos |
| | | *['tanh', 'sigmoid']* | 0.87 | 49.55 segundos |

Tabla 5.12: Segunda prueba de capas en GCN con funciones de activación utilizando *XGBoost*

### 5.5.3. SVM

Después de haber realizado la primera prueba con *Random Forest* para buscar los mejores parámetros de la GCN, como se ha visto en la Sección 5.4.3 con los diferentes parámetros y valores elegidos, se pasó a buscar los mejores parámetros del SVM con la función *GridSearchCV* para los parámetros de la GCN obtenidos. Los mejores parámetros resultantes de esta ejecución fueron:

- *gamma:* 1 como coeficiente del kernel.
- *C:* 10 como parámetro de regularización.
- *degree:* 0 como grado del polinomio de la función del kernel.

Con estos parámetros obtenidos se pasó a probarlos con diferentes números de épocas para ver la evolución de los resultados con estos parámetros y la configuración de la GCN utilizada previamente, comentados en el inicio de la Sección 5.5.1. Los resultados de estas ejecuciones se muestran en la Tabla 5.13.

| Épocas | *Accuracy* | Precisión | *Recall* |
|---|---|---|---|
| 10 | 0.92 | 0.92 | 0.92 |
| 20 | 0.92 | 0.91 | 0.92 |
| 30 | 0.94 | 0.95 | 0.94 |
| 40 | 0.94 | 0.94 | 0.94 |
| 50 | 0.95 | 0.95 | 0.95 |
| 60 | 0.93 | 0.94 | 0.93 |
| 70 | 0.84 | 0.84 | 0.84 |
| 80 | 0.79 | 0.79 | 0.79 |
| 90 | 0.84 | 0.84 | 0.84 |
| 100 | 0.79 | 0.79 | 0.79 |

Tabla 5.13: Prueba con mejores parámetros en *SVM*

Como se puede ver, este algoritmo empieza dando muy buenos resultados desde etapas muy tempranas en la ejecución. Con 10 épocas, ya esta en un *accuracy* del 92 % y con 50 épocas ya esta en torno a **95 %** que es su mejor resultado. Este algoritmo con los mejores parámetros elegidos y con 50 épocas es capaz de clasificar las aplicaciones de una manera muy precisa. Con la misma configuración de parámetros en la GCN y el mismo conjunto de datos utilizado, el SVM parece separar algo mejor las diferentes aplicaciones que *Random Forest* y XGBoost. Además en el tiempo de ejecución es muy inferior al de estos, como se verá en la segunda ejecución de la busqueda de parámetros para la GCN. En la Tabla 5.14 podemos ver el rendimiento de cada aplicación con los mejores parámetros y un *accuracy* general del **95 %**.

| Aplicación | Precisión | *Recall* | *F1-score* | Soporte |
|---|---|---|---|---|
| *App1* | 1.00 | 0.86 | 0.92 | 21 |
| *App2* | 0.80 | 0.50 | 0.62 | 8 |
| *App3* | 0.82 | 0.88 | 0.85 | 16 |
| *App4* | 0.94 | 0.94 | 0.94 | 34 |
| *App5* | 0.95 | 1.00 | 0.98 | 40 |
| *App6* | 1.00 | 1.00 | 1.00 | 5 |
| *App7* | 0.92 | 1.00 | 0.96 | 33 |
| *App8* | 0.98 | 0.96 | 0.97 | 46 |
| *App9* | 0.93 | 0.93 | 0.93 | 15 |
| *App10* | 0.94 | 0.96 | 0.95 | 49 |
| *App11* | 0.99 | 0.99 | 0.99 | 80 |
| *App12* | 1.00 | 1.00 | 1.00 | 34 |

Tabla 5.14: Prueba de mejores parámetros con 50 épocas en *SVM*

En la Tabla 5.15 podemos ver la matriz de confusión de la clasificación de las aplicaciones. Se puede ver que en lo general las aplicaciones que menos datos tienen en el dataset son las que bajan un poco el rendimiento del clasificador. Aplicaciones que tienen bastantes datos en el dataset como puede ser *App11* tienen un rendimiento muy preciso del **99 %**. Las que tienen un rendimiento más pobre en general puede deberse a tener menos datos.

| Predicted | App1 | App2 | App3 | App4 | App5 | App6 | App7 | App8 | App9 | App10 | App11 | App12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **App1** | 18 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| **App2** | 0 | 4 | 0 | 1 | 2 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| **App3** | 0 | 0 | 14 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 |
| **App4** | 0 | 0 | 1 | 32 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| **App5** | 0 | 0 | 0 | 0 | 40 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **App6** | 0 | 0 | 0 | 0 | 0 | 5 | 0 | 0 | 0 | 0 | 0 | 0 |
| **App7** | 0 | 0 | 0 | 0 | 0 | 0 | 33 | 0 | 0 | 0 | 0 | 0 |
| **App8** | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 44 | 0 | 0 | 0 | 0 |
| **App9** | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 14 | 0 | 0 | 0 |
| **App10** | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 47 | 0 | 0 |
| **App11** | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 79 | 0 |
| **App12** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 34 |

Tabla 5.15: Matriz de confusión de clasificación aplicaciones con *XGBoost*

Una vez realizadas las primeras pruebas con todos los algoritmos y sus mejores parámetros obtenidos se pasó a la segunda ejecución para buscar los mejores parámetros de la GCN donde además como hemos visto en las secciones anteriores se tuvo en cuenta el tiempo de ejecución. Los resultados para el SVM están en la Tabla 5.16. Se puede ver que el rendimiento general de este algoritmo es muy bueno y su tiempo de ejecución es muy inferior a los otros dos algoritmos, con ninguna ejecución por encima de 1 segundo. El mejor resultado en esta segunda ejecución es con [512,256] como configuración de capas, 50 épocas y *['sigmoid', 'tanh']* o *['tanh', 'sigmoid']* como capas de activación, con un **95 %** de *accuracy* y un tiempo de ejecución de medio segundo.

También, como se puede ver para prácticamente todas las ejecuciones de 100 épocas, el algoritmo empieza a sobreaprender y pierde bastante rendimiento comparado con la mitad de épocas. El funcionamiento de este algoritmo con las diferentes configuraciones probadas para la GCN parece que es óptimo con un número de épocas no muy alto, sin pasar de 70 épocas. Comparando los resultados obtenidos en la Tabla 5.16 con la Tabla 5.13, el mejor funcionamiento de este parece estar de las 30 hasta las 50 épocas.

Este resultado es el mismo que el obtenido con la primera configuración de la GCN con lo que ambas configuraciones presentan un resultado similar. No obstante la configuración de la primera ejecución tiene menos capas con lo que se ha elegido como la más óptima. No se ha mostrado la matriz de confusión y los resultados de cada aplicación ya que eran practicamente iguales a los de la anterior ejecución.

| Épocas | Capas | Activación | Accuracy | Tiempo de Ejecucion |
|---|---|---|---|---|
| 50 | [64,32] | *['sigmoid', 'sigmoid']* | 0.92 | 0.22 segundos |
| | | *['sigmoid', 'relu']* | 0.93 | 0.20 segundos |
| | | *['sigmoid', 'tanh']* | 0.91 | 0.20 segundos |
| | | *['tanh', 'sigmoid']* | 0.86 | 0.19 segundos |
| | [128,64] | *['sigmoid', 'sigmoid']* | 0.92 | 0.26 segundos |
| | | *['sigmoid', 'relu']* | 0.92 | 0.27 segundos |
| | | *['sigmoid', 'tanh']* | 0.92 | 0.27 segundos |
| | | *['tanh', 'sigmoid']* | 0.91 | 0.26 segundos |
| | [256,128] | *['sigmoid', 'sigmoid']* | 0.92 | 0.36 segundos |
| | | *['sigmoid', 'relu']* | 0.92 | 0.34 segundos |
| | | *['sigmoid', 'tanh']* | 0.92 | 0.37 segundos |
| | | *['tanh', 'sigmoid']* | 0.87 | 0.32 segundos |
| | [512,256] | *['sigmoid', 'sigmoid']* | 0.94 | 0.54 segundos |
| | | *['sigmoid', 'relu']* | 0.92 | 0.53 segundos |
| | | *['sigmoid', 'tanh']* | 0.95 | 0.54 segundos |
| | | *['tanh', 'sigmoid']* | 0.95 | 0.50 segundos |
| 70 | [64,32] | *['sigmoid', 'sigmoid']* | 0.88 | 0.21 segundos |
| | | *['sigmoid', 'relu']* | 0.90 | 0.21 segundos |
| | | *['sigmoid', 'tanh']* | 0.92 | 0.21 segundos |
| | | *['tanh', 'sigmoid']* | 0.81 | 0.21 segundos |
| | [128,64] | *['sigmoid', 'sigmoid']* | 0.89 | 0.26 segundos |
| | | *['sigmoid', 'relu']* | 0.92 | 0.29 segundos |
| | | *['sigmoid', 'tanh']* | 0.91 | 0.27 segundos |
| | | *['tanh', 'sigmoid']* | 0.81 | 0.26 segundos |
| | [256,128] | *['sigmoid', 'sigmoid']* | 0.90 | 0.37 segundos |
| | | *['sigmoid', 'relu']* | 0.92 | 0.37 segundos |
| | | *['sigmoid', 'tanh']* | 0.88 | 0.37 segundos |
| | | *['tanh', 'sigmoid']* | 0.82 | 0.36 segundos |
| | [512,256] | *['sigmoid', 'sigmoid']* | 0.92 | 0.57 segundos |
| | | *['sigmoid', 'relu']* | 0.90 | 0.56 segundos |
| | | *['sigmoid', 'tanh']* | 0.89 | 0.58 segundos |
| | | *['tanh', 'sigmoid']* | 0.76 | 0.56 segundos |
| 100 | [64,32] | *['sigmoid', 'sigmoid']* | 0.89 | 0.22 segundos |
| | | *['sigmoid', 'relu']* | 0.87 | 0.23 segundos |
| | | *['sigmoid', 'tanh']* | 0.87 | 0.23 segundos |
| | | *['tanh', 'sigmoid']* | 0.78 | 0.24 segundos |
| | [128,64] | *['sigmoid', 'sigmoid']* | 0.89 | 0.27 segundos |
| | | *['sigmoid', 'relu']* | 0.88 | 0.30 segundos |
| | | *['sigmoid', 'tanh']* | 0.88 | 0.29 segundos |
| | | *['tanh', 'sigmoid']* | 0.79 | 0.29 segundos |
| | [256,128] | *['sigmoid', 'sigmoid']* | 0.88 | 0.39 segundos |
| | | *['sigmoid', 'relu']* | 0.89 | 0.40 segundos |
| | | *['sigmoid', 'tanh']* | 0.90 | 0.41 segundos |
| | | *['tanh', 'sigmoid']* | 0.77 | 0.39 segundos |
| | [512,256] | *['sigmoid', 'sigmoid']* | 0.87 | 0.57 segundos |
| | | *['sigmoid', 'relu']* | 0.91 | 0.60 segundos |
| | | *['sigmoid', 'tanh']* | 0.87 | 0.61 segundos |
| | | *['tanh', 'sigmoid']* | 0.72 | 0.58 segundos |

Tabla 5.16: Segunda prueba de capas en GCN con funciones de activación utilizando *SVM*

### 5.5.4. Comparación de Resultados

Habiendo realizado una serie de experimentos para los tres algoritmos propuestos, se pasó a comparar los resultados de estos. El *Random Forest* consiguió un *accuracy*, precisión, *recall* y *F1-Score* de alrededor del **92 %** para sus mejores parámetros y varias configuraciones de la GCN, comentadas en la Sección 5.5.1, dando en general, unos resultados precisos y tiempos de ejecución variados donde el más eficiente en cuanto a tiempo de ejecución y rendimiento está en 9.36 segundos con un accuracy del 92 %.

El XGBoost tiene unas métricas similares, con alrededor de un **91 %** de precisión para sus mejores parámetros y los parámetros de la GCN comentados en la Sección 5.5.2, con un tiempo de ejecución, en general, bastante superior al del *Random Forest* y con muchos menos estimadores. Las pruebas iniciales con un numero de estimadores similar no mostraron una mejoría significativa de estos resultados y además mostró un tiempo de ejecución muy lento, por lo que se decidió bajar el número de estimadores para que tuviese un tiempo de ejecución similar al del *Random Forest*. Los mejores resultados del XGBoost se consiguieron con un tiempo de ejecución 10 veces mayor a los mejores resultados seleccionados para *Random Forest*, con 50 segundos, por lo que el *Random Forest* parece mejor clasificador que el XGBoost para el dataset y configuración de la GCN utilizados.

Por último, el SVM ha mostrado unas métricas superiores al *Random Forest* y XGBoost, donde utilizando sus mejores parámetros y los parámetros de la GCN seleccionados, comentados en la Sección 5.5.3, se consiguió un *accuracy*, precisión, *recall* y *F1-Score* de alrededor de un **95 %** donde además su tiempo de ejecución es muy inferior a ambos. Además, empieza a dar mejores resultados desde muy pronto en el entrenamiento de la GCN. Únicamente con 10 épocas ejecutadas en el entrenamiento de esta son necesarias para igualar el rendimiento del *Random Forest*.

Para los mejores resultados de cada algoritmo ver la Tabla 5.17.

| Algoritmo | *Accuracy* | Capas de la GCN | Función de Activación de la GCN | Épocas en la GCN |
|---|---|---|---|---|
| *Random Forest* | 0.91 | *[64,32]* | *['sigmoid','tanh']* | 70 |
| XGBoost | 0.92 | *[512,256]* | *['sigmoid','sigmoid']* | 70 |
| SVM | 0.95 | *[256,128], [512,256]* | *['tanh','tanh'],['sigmoid','tanh']* | 50 |

Tabla 5.17: Mejores resultados para los tres algoritmos utilizados

Respecto a la clasificación de las diferentes aplicaciones, hay algunas que presentan un *accuracy* mayor que otras. En la Tabla 5.18 se puede ver la media de las métricas de las aplicaciones para los tres algoritmos utilizados. Se puede ver que en general las que menos *F1-Score* tienen son las que menos soporte tienen. Esto se debe a que para estas aplicaciones, no se han conseguido recabar tantos datos para ser incluidos en el dataset y en los experimentos, con lo que aumentar el dataset de ciertas aplicaciones, podría generar un rendimiento aún mayor.

| Aplicación | Precisión | *Recall* | *F1-score* | Soporte |
|---|---|---|---|---|
| *App1* | 0.96 | 0.84 | 0.90 | 21 |
| *App2* | 0.82 | 0.42 | 0.55 | 8 |
| *App3* | 0.90 | 0.71 | 0.78 | 16 |
| *App4* | 0.91 | 0.92 | 0.92 | 34 |
| *App5* | 0.95 | 0.99 | 0.97 | 40 |
| *App6* | 0.93 | 0.67 | 0.71 | 5 |
| *App7* | 0.92 | 0.94 | 0.93 | 33 |
| *App8* | 0.95 | 0.96 | 0.95 | 46 |
| *App9* | 0.95 | 0.91 | 0.93 | 15 |
| *App10* | 0.84 | 0.96 | 0.89 | 49 |
| *App11* | 0.95 | 0.98 | 0.97 | 80 |
| *App12* | 0.98 | 0.93 | 0.95 | 34 |

Tabla 5.18: Media de todos los algoritmos de las métricas de las aplicaciones

Comparando con los trabajos vistos en el Capítulo 4, como por ejemplo *FlowPrint* o *MAppGraph*, donde se obtiene un *accuracy* de alrededor de un **90 %** y **94 %** respectivamente en clasificaciones similares, donde se utilizan enfoques semi-supervisados y supervisados, el modelo no supervisado propuesto también puede clasificar de manera precisa las diferentes aplicaciones seleccionadas con alrededor de un **95 %** de *accuracy* utilizando el algoritmo SVM, considerado el mejor de los tres probados en cuanto a rendimiento y eficiencia. En la Tabla 5.19 se muestran los resultados obtenidos en otros trabajos de referencia frente al obtenido con GCN y SVM. Para comparar los resultados se han tomado los resultados de FlowPrint obtenidos con el dataset *ReCon extended* [35], y la aproximación *Per App Random Forest* para los resultados de AppScanner. Para AppScanner[33], su *accuracy* está en un 99,8 % pero según el trabajo de *MAppGraph*[29], es de un 88,6 % por lo que esto no deja muy claro su *accuracy* real. Aún así, en la tabla se han mostrado los mejores resultados hipotéticos de este.

| Aproximación | Accuracy | Precision | Recall |
|---|---|---|---|
| AppScanner | 99.8 | 96.0 | 82.5 |
| FlowPrint | 0.8922 | 0.8984 | 0.8922 |
| MAppGraph | 0.9346 | 0.9364 | 0.9346 |
| GCN-SVM | 0.95 | 0.95 | 0.95 |

Tabla 5.19: Comparativa de los resultados obtenidos en trabajos de refencia

# Capítulo 6

# Conclusiones y Trabajo Futuro

En este capítulo se van a detallar las conclusiones (Ver Sección 6.1) a las que se ha llegado con los experimentos y resultados descritos en el Capítulo 5 y los posibles trabajos futuros (Ver Sección 6.2) que podrían mejorar los resultados obtenidos.

## 6.1. Conclusiones

El objetivo de este Proyecto ha sido analizar el tráfico de red usando algoritmos de *machine learning* que permitieran clasificar a qué aplicaciones de Android pertenece dicho tráfico. Ante la inexistencia de un *dataset* que contuviera las características necesarias para entrenar el algoritmo, los miembros del equipo elaboraron un *dataset* compuesto de capturas de tráfico de red de las aplicaciones Android requeridas.

Tras el preprocesamiento del *dataset* con la metodología contada en la Sección 5.2 se procedió a analizar el tráfico de red mediante algoritmos de aprendizaje no supervisado usando GCN. El entrenamiento no supervisado se realizó teniendo en cuenta la similitud entre los grafos con dos tipos de métricas (Ver Sección 5.3.2.2). Posteriormente, se pasó el resultado del entrenamiento en forma de vector embedido a los algoritmos de clasificación supervisada usados: *Random Forest*, XGBoost y SVM.

Los mejores resultados obtenidos en los experimentos de la Sección 5.5 han sido realizados con una misma configuración de hiper-parámetros. Las comparaciones de los distintos resultados se han hecho en base a las medidas descritas en la Sección 5.3.1.

En los experimentos realizados con el clasificador supervisado *Random Forest* primeramente se obtuvo la mejor configuración para la GCN probando distintos parámetros vistos en la Sección 5.4.1, pero manteniendo la misma función de activación. Con esta primera aproximación se obtuvo un *accuracy* del 92 % para el algoritmo de clasificación *Random Forest* con los mejores parámetros encontrados (Ver Sección 5.5.1). Posteriormente, se ajustó el entrenamiento utilizando los mejores parámetros obtenidos en la primera prueba, pero probando diferentes funciones de activación. Finalmente, con la mejor configuración obtenida para GCN usando *Random Forest* (Ver Tabla 5.17) se obtuvo un *accuracy* del 92 %. Aunque no se mejoró el *accuracy* ajustando la función de activación, sí se logró reducir el tiempo de ejecución a 9.36 segundos.

El resultado obtenido con el algoritmo XGBoost usando los parámetros de la mejor configuración de *Random Forest* es de un *accuracy* del 0.87 %. Comparado con el algoritmo *Random Forest*, XGBoost ofrece un menor rendimiento tanto en términos de *accuracy* como en tiempo de ejecución. Los experimentos mostraron que con la mejor configuración (Ver Tabla 5.17) se consigue un *accuracy* del 91 % y un tiempo de ejecución de 49.82 segundos. En términos de eficiencia, el algoritmo *XGBoost* no logró mejorar el *accuracy* ni el tiempo de ejecución con respecto al algoritmo *Random Forest*.

El resultado obtenido con el algoritmo SVM usando los parámetros de la mejor configuración de *Random Forest* es de un *accuracy* del 0.84 %, bastante menor al obtenido con los algoritmos *XGBoost* (0.87 %) y *Random Forest* (92 %). Sin embargo, con la misma configuración y con 50 épocas se consigue el mejor resultado hasta el momento, con un *accuracy* del 95 %. Los experimentos mostraron que la mejor configuración para este algoritmo (Ver Tabla 5.17) consigue un *accuracy* del 95 % y un tiempo de ejecución de 0.50 segundos.

Finalmente, podemos concluir que el algoritmo SVM es el que mejor resultados ha dado en comparación con los algoritmos *Random Forest* y XGBoost tanto en términos de *accuracy* como en tiempo de ejecución.

Respecto a los resultados obtenidos en otros trabajos de referencia comentados en el Capítulo 4, se han logrado mejorar los resultados en términos de *accuracy*, *precision* y *recall* mediante GCN (aprendizaje no supervisado) frente a las técnicas de aprendizaje semi-supervisado y supervisado usadas en AppScanner, FlowPrint y MAppGraph. En la Tabla 5.19 se muestran los mejores resultados obtenidos para cada uno de los trabajos frente al mejor resultado obtenido en este proyecto.

## 6.2. Trabajo Futuro

Como posibles trabajos futuros pueden señalarse los siguientes:

- Realizar la clasificación de aplicaciones mediante el análisis del tráfico de red en tiempo real.
- Probar otros algoritmos de clasificación supervisada que puedan mejorar los resultados.
- Probar diferentes tipos de redes neuronales para ver si pueden mejorar los resultados.
- Ampliación del dataset para poder clasificar más aplicaciones y tratar de mejorar los resultados para las aplicaciones con menos datos.

# Capítulo 7

# Contribuciones

Este Trabajo de Fin de Grado (TFG) ha sido realizado por un equipo formado por tres miembros, lo cual ha requerido que se mantuviera una buena comunicación entre los que lo componían. Esta comunicación se ha mantenido de forma periódica con los miembros del Grupo GASS y el resto de compañeros mediante reuniones celebradas todos los lunes a través de *Google Meet* desde el inicio del Proyecto, aproximadamente mediados de septiembre del año 2021, hasta la finalización del mismo en junio de 2022. Asimismo, también se creó un canal de comunicación más fluido a través de aplicaciones de mensajería móvil que permitieron resolver dudas y hacer comunicados de una forma más rápida y directa. A lo largo de este Capítulo se describen las contribuciones hechas por cada miembro que componen este Trabajo de Fin de Grado.

### 7.0.1. Gema Blanco Núñez

En mi trayectoria académica del grado en Ingeniería Informática cursado en la Universidad Complutense de Madrid me he interesado en el itinerario de Tecnología Específica de Computación, especialmente por las asignaturas impartidas sobre Inteligencia Artificial. Una vez empezado mi último curso académico (curso 2021-2022) comencé a buscar un Trabajo de Fin de Grado en el que participar, fue entonces cuando me puse en contacto con el profesor de la Facultad de Informática Luis Javier García Villalba. Tras conocer más detalles acerca del Proyecto y ser invitada a participar en él acepté la oferta, ya que el uso de la Inteligencia Artificial cuadraba con mis intereses, más aún tratándose de un tema tan desconocido en un principio como es el análisis de tráfico de red. Además, el hecho de participar en un TFG con el grupo de investigación GASS y con varios compañeros me ha permitido conocer un poco mejor el funcionamiento de los grupos de investigación desde dentro y seguir mejorando mis habilidades en trabajos grupales.

Durante la primera fase del Proyecto, es decir, los meses desde mediados de septiembre hasta finales de diciembre, contribuí realizando investigaciones sobre el estado del arte utilizando la herramienta *Google Scholar*, recomendada por el Grupo GASS. Concretamente, durante esta fase presenté al grupo en una de nuestras numerosas reuniones un trabajo realizado sobre clasificación de aplicaciones mediante el análisis del tráfico de red: MAppGraph. Desde entonces, me centré en estudiar esta herramienta para entender sus características y detalles de implementación, conocimientos que fui compartiendo con el resto de mis compañeros a través de resúmenes y presentaciones

compartidas en las reuniones celebradas mediante *Google Meet*. También participé en la elaboración del *dataset* requerido en la fase de entrenamiento del Proyecto realizando capturas de tráfico de red con mi propio móvil con sistema operativo Android usando la aplicación *PCAPdroid*.

Una vez finalizado lo sustancial de la fase de investigación, aproximadamente a principios del mes de enero, comenzó la fase de desarrollo, la cual se extendió hasta principios del mes de abril. En esta fase el código se desarrolló sobre Python, así que comencé recordando algunos aspectos sobre Inteligencia Artificial que había cursado anteriormente y sobre código de Python y librerías que usaríamos como *Keras*, *Tensor Flow*, *scikit-learn* y *Scapy*. En lo que respecta al código, desarrollé la parte del preprocesamiento de los datos que se basa en la construcción de los grafos. También contribuí, en menor medida, en la implementación de algún pequeño módulo en la fase de preprocesamiento que se basa en la extracción de características de los datos. Además, durante los meses que componen esta fase continué realizando capturas de tráfico de datos que formarían parte del *dataset*.

Finalizada la fase de desarrollo, mis compañeros y yo comenzamos a elaborar la memoria del Trabajo de Fin de Grado. En dicha redacción realicé:

- Introducción del TFG (Capítulo 1).
- Descripción del modelo OSI y el Protocolo TCP/IP en el Capítulo 2.
- Introducción en el capítulo sobre Aprendizaje Automático y contribuí a la sección sobre el algoritmo SVM (Capítulo 3).
- Metodología de análisis del tráfico de red mediante aprendizaje automático, descripción de MAppGraph y de los resultados obtenidos en AppScanner, FlowPrint y MAppGraph en el Capítulo sobre el estado del arte (Capítulo 4).
- Introducción, dataset, y construcción de grafos en el Capítulo 5.
- Capítulo de Contribuciones y trabajo futuro (6).

Cabe mencionar que durante la redacción de la memoria todos los miembros del grupo hemos participado en la corrección de erratas o errores que iban surgiendo.

### 7.0.2. Diego Atance Sanz

Tras ponerme en contacto con el profesor Luis Javier García Villalba a finales de Julio para pedirle información sobre los diversos Trabajos de Fin de Grado que ofertaba, no dudé en aceptar su propuesta para formar parte en éste proyecto, ya que combina la ciberseguridad y el análisis de datos, dos de los campos de la informática que más me han llamado al atención durante mis estudios, y además se encuentra enmarcado en un entorno real, permitiendo conocer cómo se realiza el trabajo en un grupo de investigación real.

Tras confirmar mi participación en el proyecto a principios de Agosto conocí al resto de profesores y compañeros involucrados y fuí informado del plan de ación, las diversas etapas en las que se dividiría el proyecto y sus respectivos plazos.

Durante la primera etapa, la etapa de investigación, busqué numerosos artículos para informarme sobre el estado del arte actual respecto al análisis de tráfico, las herramientas utilizadas con ése propósito, y las alternativas existentes para realizar nuestro proyecto.

Para buscar la información mencionada se usó *Google Scholar*, herramienta que fué de gran ayuda durante toda esta etapa del proyecto.

Durante esta etapa de investigación, también realicé un análisis del código de alguna de las aplicaciones más utilizadas para análisis de tráfico, así como una pequeña ingeniería inversa de la misma, generando una serie de diagramas de flujo y de componentes para analizar su funcionamiento, junto con toda la documentación necesaria para ponerlas en funcionamiento.

A lo largo de esta fase del proyecto, también traduje y resumí varios documentos académicos relacionados con el propósito de la investigación, adaptándolos para facilitar su lectura al resto de componentes del equipo.

Fué sobre estas fechas cuando comenzó la creación del *dataset*, tomando muestras de tráfico de nuestros propios dispositivos mediante la application para Android *PCapDroid*, que permite aislar el tráfico de una única aplicación y obtener una captura del tráfico generado por la misma en formato .PCAP. Esta tarea de creación del dataset, junto a su correcto etiquetado, se ha continuado realizando a lo largo de todos los meses que se ha dedicado al proyecto.

Una vez finalizada la fase de investigación y presentado un documento con todos los avances realizados durante la misma, comenzó la fase de desarrollo. Dicha fase comprendió los meses de enero a abril, aproximadamente. Durante dicha fase, y mientras mis compañeros desarrollaban el código en sí de la aplicación, me dediqué a la realización de capturas de tráfico, así cómo a refrescar mis conocimientos de LaTeX, con el objetivo de hacer mas llevadera la fase de elaboración de la memoria. También realicé tareas relacionadas con la tranformación del dataset para su correcto funcionamiento en la aplicación.

Una vez finalizado el desarrollo, comenzó la fase de redacción de la memoria del Trabajo de Fin de Grado, donde además de colaborar en diversas secciones, me encargué principalmente de la traducción al inglés de aquello necesario, así como a la corrección de errores de compilación de LaTeXy el correcto referenciado y citación de elementos.

Durante la totalidad del proyecto, todos los miembros del equipo hemos mantenido reuniones a través de *Google Meet* para actualizar a los coordinadores de nuestros avances. De la misma manera y con el propósito de mantener una comunicación fluida a nivel interno del equipo, se ha utilizado un grupo de *Whatsapp*.

### 7.0.3. César Ureña Toledano

Antes de comenzar mi cuarto año en el grado de Ingeniería Informática con las asignaturas específicas del itinerario de Computación y con la matrícula del Trabajo de Fin de Grado realizada, me puse a realizar una búsqueda de las posibles opciones que se adecuasen con mis intereses. Estos eran principalmente los que implicasen Inteligencia Artificial. Después de una búsqueda exhaustiva y varias opciones posibles, al realizar una reunión con el profesor Luis Javier García Villalba, me decidí por realizar este proyecto ya que los objetivos del proyecto eran los que más cuadraban con mis intereses. Una vez dentro del proyecto, me puse al día de lo que investigar con el grupo GASS y aquí empezaba la

primera fase del proyecto.

La primera fase del proyecto duró desde mi inicio de la participación con el grupo, a finales de agosto, hasta finales de diciembre. En esta fase, empecé aprendiendo conceptos sobre redes que en las asignaturas de grado correspondientes al trafico de red no llegué a conocer. Cuando ya tenía los conocimientos necesarios, realicé investigaciones que más tarde iban a pertenecer al estado del arte, utilizando la herramienta *Google Scholar*. Estas investigaciones se centraron en investigar a cerca del TLS y sus certificados, trabajos en los que se utilizaba el algoritmo de aprendizaje automático *Random Forest* y la herramienta AppScanner. Todos los conocimientos adquiridos acerca de estos temas y herramientas, fueron compartidos con el resto de los compañeros a través de reuniones semanales celebradas a través de *Google Meet*. Los resúmenes los empecé a realizar en LaTex por recomendación de algunos miembros del grupo lo que me ayudó a tener más conocimientos a la hora de redactar la memoria final. Durante esta fase también participé en la creación de un dataset de capturas de tráfico de red, que posteriormente íbamos a necesitar en las pruebas y experimentos para la herramienta que en ese momento no se había comenzado a desarrollar.

Finalizada la primera fase, a principios de enero se comenzó la segunda fase, perteneciente al desarrollo de la herramienta y la continuación de la creación del dataset que terminó a principios del mes de abril cuando la herramienta ya estaba prácticamente desarrollada en su totalidad. El desarrollo se iba a realizar en *Python* por lo que tuve que recordar algunos conceptos sobre diferentes librerías, funciones y complementos del lenguaje que me serían útiles en el desarrollo de la herramienta. Además aprendí nuevos conceptos y utilidades de diferentes librerías como *scikit-learn* y *Scapy*. Una vez recordados y aprendidos todos los conceptos, se comenzó el desarrollo de la herramienta. Utilizamos el entorno de desarrollo *PyCharm*. Yo me encargué de realizar el preprocesamiento inicial de las capturas de tráfico y la extracción de características de estas, lo que es la primera fase del preprocesamiento antes de pasar a la creación de los grafos. Una vez terminado el desarrollo del preprocesamiento, desarrollamos el modelo a entrenar. Para ello desarrollé la creación de los grafos usando la librería de *StellarGraph* que ademas tomamos de referencia para la creacion de la GCN tomando ejemplos de modelos utilizando GCNs. Este tipo de redes la utilizamos de manera no supervisada en el desarrollo por lo que tuvimos que ver de alguna manera cómo de bueno o malo era este entrenamiento. Para ello utilizamos algoritmos de aprendizaje supervisado para clasificar las aplicaciones android que teniamos en el dataset realizado. Desarrollé además funciones para probar los diferentes parámetros y buscar los óptimos para la configuración de la GCN y los algoritmos de aprendizaje supervisado.

Finalizada la parte de desarrollo y aún con muchas de las pruebas a realizar, mis compañeros del grupo y yo comenzamos a redactar la memoria del Trabajo de Fin de Grado. En dicha redacción contribui con todas las partes a continuación:

- Descripción del protocolo TLS y sus subsecciones (Captítulo 2).
- Random Forest y árboles de decisión, XGBoost y todo lo relacionado con las redes neuronales (Capítulo 3).
- Contribuí en AppScanner y la sección otros trabajos relacionados (Capítulo 4).
- Realicé la extracción de características, su diagrama en la sección del preprocesamiento, la descripción del modelo no supervisado y la descripción

de los parámetros utilizados para GCN, todos los algoritmos de aprendizaje supervisado para la clasificación y los experimentos usando todos los parámetros descritos(Capítulo 5).

Durante todo el proceso la comunicación y división del trabajo entre los miembros la hemos hecho mediante aplicaciones de mensajería instantánea y si era necesario nos reuníamos en *Google Meet* aparte de las reuniones semanales con todo el grupo.

# Capítulo 8

# Introduction

## 8.1. Motivation

Since the birth of the Internet in 1972, developed by the Department of Defense of the United States and tested on the wide area network ARPANET [4], there have been great technological advances that have allowed the massification of this means, medium. Currently, Internet access is practically essential in the Western world, being this one of the main forms of communication, entertainment and work between people.

Due to the great underlying importance of this medium, various *Android* applications are used on a daily basis, even as a work tool. On the other hand, the use of these applications has led to the sharing of sensitive information that, in some cases, ends up being filtered due to possible security flaws in these applications.

Machine learning or *machine learning* has numerous fields of application, including data traffic analysis. Through this data analysis, information can be extracted that, treated in the appropriate way and using algorithms, allows classifying the traffic of the applications that circulates through the internal network.

## 8.2. Object of the Investigation

Nowadays, due to the rise of mobile the Internet access has become an essential aspect in people's lives, both personally and professionally. On the one hand, this represents a great danger for people's privacy and security because of the information that can be extracted about the users through the analysis of data traffic. On the other hand, this analysis has been used for practical applications that has benefits for society as the detection of cybercrimes.

The origins of Artificial Intelligence go back to the year 1955 when the *Western Joint Computer Conference* was held in Los Angeles and in which a session on *learning machines* was organized. In this congress the term neural networks was introduced and works related to pattern recognition [20] were presented.

Over the years, thanks to the improvement of hardware, advances in the field of Artificial Intelligence and the large amount of data currently generated, algorithms have emerged capable of identifying patterns in data traffic from which to arrive at some conclusions about the underlying data. This is the case of an algorithm designed to detect

IoT devices connected to a network from data traffic [24].

The purpose of this work is the detection of typical *Android* mobile applications with the highest possible accuracy from data traffic by previously training a *deep learning* algorithm.

## 8.3. Workplan

The development of this work has been carried out throughout the period that comprises the 2021-2022 academic year and has been divided into three phases:

1. **Research**: For the first few months, September through December, team members focused on gaining background knowledge and researching other work related to the project topic. During this phase, all the students involved in the work shared knowledge and doubts through video calls held periodically every week through *Google Meet*. In addition, these meetings also had the purpose of synchronizing all the members about the advances produced and sharing tools that facilitated research and work such as *Google Scholar*, thanks to which it has been possible to find numerous scientific and technical articles .

   Once the basic elements of the project were defined, data traffic capture began with *PCAPdroid*[13], an Android application that allows you to save the network packets generated by a specific application. These captures are the ones that collect data from specific Android applications and that form the dataset necessary to train the algorithm.

2. **Development**: After establishing the bases and objectives of the project and with the knowledge acquired in the previous phase, the following steps were defined that would allow the development of an algorithm capable of detecting applications from data traffic. The programming language chosen for development was Python due to its high performance in Artificial Intelligence. Although this phase is focused on development, research did not stop and remained focused on research on *Python* concepts and libraries such as *Keras* [5], *Tensor Flow* [1] and *Scapy*, which allowed manipulation of network packets. In addition, the capture of traffic for the generation of the dataset also continued to be active.

3. **Experimentation**: Once the implementation of the model was complete, the traffic capture dataset was processed and the model was trained with different parameters in order to obtain the best results.

Figure 8.1 shows a Gantt chart with the periods that comprise each of the main tasks carried out during the development of the project.

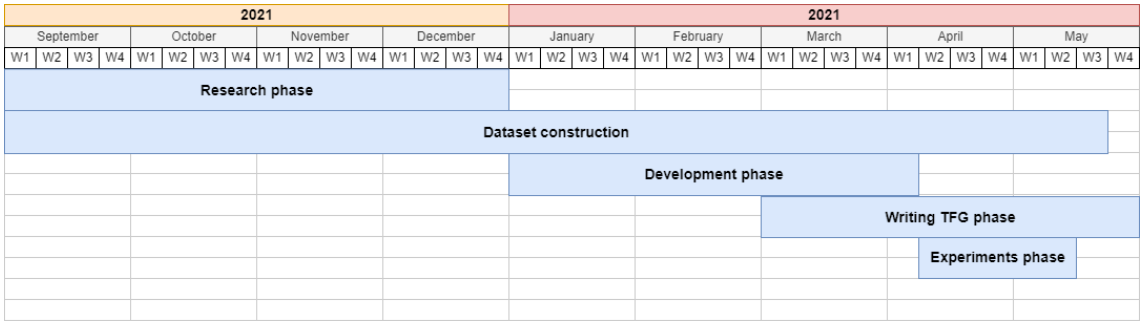



Figura 8.1: Gantt Diagram of the Project

## 8.4. Struture of the Work

The work consists of 6 chapters and 4 annexes that follow the structure described below:

The Chapter 1 is an introduction in which the motivation, the context, the object of the investigation and the work plan and structure are exposed. It offers a general and superficial vision of topics that are treated in depth throughout the next chapters.

Chapter 2 deals with basic concepts for understanding data traffic analysis. Topics covered include an introduction to the TCP/IP protocol and the OSI model used as a reference model for transmitting information on the Internet; the operation of routing or *routing* and the TLS protocol.

Chapter 3 introduces terms related to the training phase of the project. This chapter introduces the concept of Artificial Intelligence (AI), shows the types of machine learning that exist and algorithms related to each of them, as well as graphs as a method for data representation, and explains the operation of convolutional networks. Likewise, this Chapter also explains the main phases that constitute data traffic analysis projects.

Chapter 4 exposes the state of the art of the project, that is, the research carried out on the analysis of network traffic and the detection of mobile applications through AI, as well as other related works. In addition, some of the tools analyzed by the authors of this project will be addressed in a general way.

Chapter 5 is the one on contributions and experiments, it describes the model proposed for the detection of mobile applications through the analysis of encrypted data traffic as well as the experiments carried out and the results obtained.

Chapter 6 refers to the conclusions about the Project.

Chapter 7 presents the contributions of each member to the Project.

Chapters 8 and 9 correspond to the English translation of the Introduction and Conclusions.

# Capítulo 9

# Conclusions and Future Work

This chapter will detail the conclusions (See Section 9.1) that have been reached with the experiments and results described in Chapter **??** and possible future work (See Section 6.2) that could improve the results obtained.

## 9.1. Conclusions

The objective of this Project has been to analyze network traffic using *machine learning* algorithms that would allow classifying which Android applications said traffic belongs to. Given the lack of a *dataset* containing the necessary characteristics to train the algorithm, the team members created a *dataset* composed of network traffic captures of the required Android applications.

After preprocessing the *dataset* with the methodology described in Section 5.2, network traffic was analyzed using unsupervised learning algorithms using GCN. The unsupervised training was performed taking into account the similarity between the graphs with two types of metrics (See Section 5.3.2.2). Subsequently, the training result was passed as an embedded vector to the supervised classification algorithms used: *Random Forest*, XGBoost and SVM.

The best results obtained in the experiments of Section 5.5 have been carried out with the same hyper-parameter configuration. The comparisons of the different results have been made based on the measurements described in Section 5.3.1.

In the experiments carried out with the supervised classifier *Random Forest*, the best configuration for the GCN was first obtained by testing different parameters seen in Section 5.4.1, but maintaining the same function of activation. With this first approximation, an *accuracy* of 92 % was obtained for the classification algorithm *Random Forest* with the best parameters found (See Section 5.5.1). Subsequently, the training was adjusted using the best parameters obtained in the first test, but testing different activation functions. Finally, the best configuration obtained for GCN using *Random Forest* (View Table 5.17) has an *accuracy* of 92 %. Although the *accuracy* was not improved by adjusting the activation function, it did reduce the execution time to 9.36 seconds.

The result obtained with the XGBoost algorithm using the same parameters as in the best configuration of the *Random Forest* algorithm is an *accuracy* of 0.87 %. Compared to the *Random Forest* algorithm, XGBoost offers lower performance both in terms of *accuracy*

and execution time. The experiments showed that the best configuration for this algorithm (View Table 5.17) achieve an *accuracy* of 91 % and an execution time of 49.82 seconds. In terms of efficiency, the *XGBoost* algorithm failed to improve *accuracy* and execution time over the *Random Forest* algorithm.

The result obtained with the SVM algorithm using the same parameters as in the best configuration of the *Random Forest* algorithm is an *accuracy* of 0.84 %, much lower than that obtained with the *XGBoost* algorithms ( 0.87 %) and *Random Forest* (92 %). However, with the same configuration and 50 epochs, the best result so far is achieved, with an *accuracy* of 95 %. The experiments showed that the best configuration for this algorithm (View Table 5.17) achieve an *accuracy* of 95 % and an execution time of 0.50 seconds.

Finally, we can conclude that the SVM algorithm is the one that has given the best results compared to the *Random Forest* and XGBoost algorithms both in terms of *accuracy* and execution time .

Regarding the results obtained in other reference works discussed in Chapter **??**, the results have been improved in terms of *accuracy*, *precision* and *recall* by means of (unsupervised learning) versus semi-supervised and supervised learning techniques used in AppScanner, FlowPrint, and MAppGraph. Table 5.17 shows the best results obtained for each of the works compared to the best result obtained in this project.

## 9.2. Future work

As possible future works the following can be pointed out:

- Perform application classification by analyzing network traffic in real time.
- Test other supervised classification algorithms that may improve results.
- Test different types of neural networks to see if they can improve the results.
- Expansion of the dataset to be able to classify more applications and try to improve the results for applications with less data.

# Bibliografía


[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Mark Stamp Anish Singh Shekhawat, Fabio Di Troia. Feature analysis of encrypted malicious traffic. *Expert Systems with Applications*, 125:130–141, 2019.

[3] Kristin P. Bennett and Colin Campbell. Support vector machines: Hype or hallelujah? *SIGKDD Explor. Newsl.*, 2(2), 2000.

[4] B. Carne. *Professional's Guide to Data Communication in a TCP/IP World*. Artech House, 2004.

[5] François Chollet et al. Keras. https://github.com/fchollet/keras, 2015.

[6] CSIRO's Data61. Stellargraph machine learning library. https://github.com/stellargraph/stellargraph, 2018.

[7] J.D. Day and H. Zimmermann. The osi reference model. *Proceedings of the IEEE*, 71(12):1334–1340, 1983.

[8] Julianna Delua. Supervised vs. Unsupervised Learning: What's the Difference? https://www.ibm.com/cloud/blog/supervised-vs-unsupervised-learning, March 2021.

[9] XGBoost Developers. Xgboost parameters. https://xgboost.readthedocs.io/en/stable/parameter.html.

[10] Belén Díaz Agudo. Sistemas basados en conocimiento, 2020. Apuntes de la asignatura de Inteligencia Artificial I.

[11] IBM Cloud Education. What is supervised learning? https://www.ibm.com/cloud/learn/supervised-learning#:~:text=Supervised%20learning%2C%20also%20known%20as,data%20or%20predict%20outcomes%20accurately., August 2020.

[12] Kevin Knight Elaine Rich. *Artificial intelligence*. McGraw-Hill, 1991.

[13] Emanuele Faranda. PCAPdroid: User Guide. https://github.com/emanuele-f/PCAPdroid,https://emanuele-f.github.io/PCAPdroid/quick_start.html, Jan 2020.

[14] IBM SPSS - Redes neuronales 28. https://www.ibm.com/docs/en/SSLVMB_28.0.0/pdf/es/IBM_SPSS_Neural_Network.pdf.

[15] European Union Agency for Cybersecurity. ENISA Report: Encrypted Traffic Analysis. https://www.enisa.europa.eu/publications/encrypted-traffic-analysis, April 2020.

[16] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA, 2008.

[17] Manish Joshi and Theyazn Hassn Hadi. A review of network traffic analysis and prediction techniques, 2015.

[18] Ray Kurzweil. *The Age of Intelligent Machines*. MIT Press, 1990.

[19] Bernard Gold Lawrence R. Rabiner. *Theory and application of digital signal processing*. Prentice-Hall, 1975.

[20] R. López de Mántaras and P. Meseguer. *Inteligencia Artificial*, chapter 1, pages 19–22. CSIC Consejo Superior de Investigaciones Científicas, 2017.

[21] R. López de Mántaras and P. Meseguer. *Inteligencia Artificial*, chapter 0, pages 7–17. CSIC Consejo Superior de Investigaciones Científicas, 2017.

[22] Virginie Mathivet. *Inteligencia Artificial para desarrolladores Conceptos e implementación en Java*, chapter 7. Eni, 2019.

[23] Inneke Mayachita. Understanding Graph Convolutional Networks for Node Classification. https://towardsdatascience.com/understanding-graph-convolutional-networks-for-node-classification-a2bfdb7aba7b, June 2020.

[24] Yair Meidan, Michael Bohadana, Asaf Shabtai, Juan David Guarnizo, Martín Ochoa, Nils Ole Tippenhauer, and Yuval Elovici. Profiliot: a machine learning approach for iot device identification based on network traffic analysis. *Proceedings of the Symposium on Applied Computing*, 2017.

[25] El modelo de redes neuronales. https://www.ibm.com/docs/es/spss-modeler/SaaS?topic=networks-neural-model, August 2021.

[26] Modelo de referencia osi. https://docs.oracle.com/cd/E19957-01/820-2981/ipov-8/index.html.

[27] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[28] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Support vector machines. https://scikit-learn.org/stable/modules/svm.html, 2011.

[29] Thai-Dien Pham, Thien-Lac Ho, Tram Truong-Huu, Tien-Dung Cao, and Hong-Linh Truong. Mappgraph: Mobile-app classification on encrypted network traffic using deep graph convolution neural networks. In *Annual Computer Security Applications Conference*, page 1025–1038, 2021.

[30] Antônio J. Pinheiro, Jeandro de M. Bezerra, Caio A.P. Burgardt, and Divanilson R. Campelo. Identifying iot devices and events based on packet length from encrypted traffic. *Computer Communications*, 144:8–17, 2019.

[31] Internet Protocol. RFC 791, September 1981.

[32] John R. Searle. Minds, brains, and programs. *Behavioral and Brain Sciences*, 3(3), 1980.

[33] Vincent F. Taylor, Riccardo Spolaor, Mauro Conti, and Ivan Martinovic. Appscanner: Automatic fingerprinting of smartphone apps from encrypted network traffic. In *2016 IEEE European Symposium on Security and Privacy (EuroS P)*, pages 439–454, 2016.

[34] Tram Truong-Huu, Nidhya Dheenadhayalan, Partha Kundu, Vasudha Ramnath, Jingyi Liao, Sin Teo, and Sai Praveen Kadiyala. An empirical study on unsupervised network anomaly detection using generative adversarial networks. 10 2020.

[35] Thijs van Ede, Riccardo Bortolameotti, Andrea Continella, Jingjing Ren, Daniel J Dubois, Martina Lindorfer, David Choffnes, Maarten van Steen, and Andreas Peter. Flowprint: Semi-supervised mobile-app fingerprinting on encrypted network traffic. In *Network and Distributed System Security Symposium (NDSS)*, 2020.

[36] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[37] Tony Yiu. Understanding Random Forest. https://towardsdatascience.com/understanding-random-forest-58381e0602d2, June 2019.