Análisis de tráfico de Dispositivos Móviles mediante Técnicas de Aprendizaje Autómatico Supervisado

– –

Movile Device Traffic analysis with Supervised Machine Learning Techniques

TRABAJO FIN DE GRADO
GRADO EN INGENIERÍA
CURSO 2021–2022

Marcos Gálvez Santamaria
Patricia Martín-Salas Gesteira
Carlos Ramírez Martínez

Directores

Luis Javier García Villalba
Luis Alberto Martínez Hernández

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

Madrid, Junio de 2022

Agradecimientos

Le queremos dar las gracias a todas las personas que nos han apoyado durante el proyecto por ayudarnos en los momentos difíciles de la investigación y darnos ánimos para continuar. Además, queremos agradecerle a nuestros directores Luis Javier García Villalba y Luis Alberto Martínez Hernández por darnos la oportunidad de emprender este proyecto así por su confianza en nosotros, y a todo el equipo GASS que nos han apoyado y ayudado durante todo el proceso. Gracias a todos.

Índice General

Índice de Figuras

Índice de Tablas

Índice de Algoritmos

Lista de Acrónimos

IP *Internet Protocol*

ML *Machine Learning*

Abstract

This project, is intended to develop a tool which implements machine learning and deep learning to analyze the network´s traffic. This means, contribute with an application that investigates the cyber attacks. Due to the advantages of Python, its have been selected to be the language in which the project has been develop, using the different libraries it contains such as *Keras* and *Tensor Flow*.

Keywords: Forensics Analysis,identification, Mobile Device,Deep learning, neural networks , artificial inteligence.

Resumen

Este proyecto, esta destinado a desarrollar una herramienta que implemente el Machine Learning y el Deep learning para el análisis de tráfico. Es decir, aportar una aplicación de investigación al análisis forense con el fin de perseguir los ciberataques. Dado a las ventajas que aporta Python, se ha optado por ser el lenguaje de desarrollo de este proyecto haciendose uso de las diferentes libreria que contiene como *Keras* y *Tensor Flow*

Palabras clave: Análisis Forense, identificación, Aplicaciones, Deep learning,Redes neuronales, Inteligencia Artificial.

Capítulo 1

Introducción

1.1. Motivación

Nuestra experiencia, incentivada por el contacto con la sociedad y la visualización de noticias, nos ha dejado ver que a día de hoy hay multitud de individuos que han sido atacados por algún tipo de ciberdelicuencia, ya sea fraude financiero, robo de identidad, falsificación de información, robo de propiedad intelectual y ciberacoso.
A día de hoy, hay muchas formas de atacar a un usuario mediante el uso de la red informática ya sea suplantando la identidad con técnicas como *Man In The Middle* o denegación de un servicio (DDoS), o la introducción de algún malware en el dispositivo.

Todo esto tiene lugar por el aumento del uso de internet y es por ello, que para solventar este problema y evitar estos ataques han surgido varias ramas de la seguridad informática que la investigan y las evitan, entre ellas el análisis forense,el peritaje judicial, el hacking ético, la protección de base de datos y seguridad en las redes informáticas.

El análisis forense se entiende como un área a la que pertenece la seguridad informática, nacida del incremento de incidentes en esta. Su importancia reside en que reduce la carga de trabajo y los tiempos de resolución en las investigaciones criminales cibernéticas.

De todo ello, nace la importancia de identificar el contenido del tráfico de redes y deriva en la necesidad de identificar el contenido para el análisis forense. Este aspecto, ha fomentado a desarrollar este trabajo con el fin de aportar una nueva herramienta que ayude a analizar el tráfico de la red.

1.2. Objeto de la Investigación

A través de la red viaja una elevada cantidad de información, entre ella, mucha perjudicial o dañina para otros usuarios. Se ha descubierto que la inteligencia artificial bien entrenada permite encontrar esta información a través de la identificación de patrones en la información.

Este trabajo propone distintas técnicas, algoritmos y modelos que permiten implementar el análisis forense de tráfico de datos con el fin de mejorar la seguridad cibernética en este aspecto.

1.3. Plan de Trabajo

El desarrollo del TFG esta divido en las siguientes fases:

1. **Investigación**: Primero nos centramos en la investigación del tema, pues antes de comenzar necesitabamos adquirir unos conocimientos básicos sobre el tráfico de mensajería y sobre técnicas de Machine Learning. Se nos indicó desde un principio cual era el objetivo del proyecto y cuales eran las aplicaciones y técnicas que ibamos a tener que utilizar. Durante los meses de Septiembre a Diciembre nos centramos en esta fase, trabajando de manera individual cada integrante del grupo en la búsqueda y análisis de estas técnicas y aplicaciones y otras que considerasemos que podian ser útiles para mejorar el funcionamiento de nuestra herramienta. Para esta búsqueda se utilizaron herramientas como el *Google Scoolar* las cuales facilitaban encontrar documentos académicos sobre los temas que nos interesaban. Se realizaban reuniones semanales por *Google Meet* donde cada uno exponía los avances en la investigación que habia realizado esa semana y los directores de la tesis nos resolvían las dudas que pudieramos tener. Al final de esta fase fuimos capaces de extraer conclusiones de toda la investigación realizada sobre la manera más óptima que había para desarrollar el código para obtener los resultados esperados. Cabe destacar que a lo largo de toda esta fase fuimos generando un dataset propio para el entrenamiento de nuestra herramienta por medio de capturas realizadas con el *PCAPandroid*, el cual permitía la captura de tráfico de aplicaciones específicas.

2. **Desarrollo**: La segunda fase de este proyecto fue el desarrollo del código. Cuando ya logramos decidir como deseabamos que fuera este y en que modelos ibamos a basarnos para la construcción del nuestro propio nos toco estudiar de que manera era posible programar esas ideas. Por esta razón a lo largo de esta fase se investigaron sobre los lenguajes de programación como *Python* y dentro de este las librerías como *Keras* y *Tensor Flow* que necesitabamos como apoyo. Esta no fue la unica investigación que se realizó a lo largo de este periodo debido a que segun progresabamos con el código surgían problemas o dificultades los cuales requerían de una investigación para encontrar la manera más optima de resolverlos. Para terminar con esta fase y una vez que el código era lo suficientemente solido creamos conjuntos de entrenamiento para el modelo.

3. **Experimentación**: Por último tenemos la fase de experimentación en la cual nos centramos en los prototipos del modelo que ibamos creando. Durante esta fase también se continuó con el desarrollo, necesario para mejorar los prototipos que se iban creando. A lo largo de esta fase, buscamos optimizar el modelo mediante la realización de pruebas y la comparación de los resultados obtenidos en cada una de ellos y con los obtenidos en los trabajos académicos que habiamos analizado. De esta manera se fueron ajustando los parámetros del modelo hasta que logramos unos resultados óptimos. Para la comparación de los distintos modelos se utilizaron las herramientos proporcionadas por las librerías de *Python*, las cuales investigamos en la fase anterior.

1.4. Estructura del Trabajo

El resto del trabajo está organizado en 6 capítulos y 4 anexos con la estructura que se comenta a continuación: El Capítulo 2 se presenta todo lo relacionado con la investigación realizada. Se introducen los conceptos más importantes del Machine Learning y del Aprendizaje Automático. Así como, el proceso que se realiza para crear un modelo del Machine Learning y las diferentes variables que influencian. Posteriormente, se presenta lo investigado de los diferentes algoritmos relacionados con nuestro proyecto y de los que se van a utilizar. En el Capítulo 3 se presentan distintos trabajos relacionados con el proyecto desarrollado.

El Capítulo 4 presenta las contribuciones de este trabajo. En primer lugar, se presenta diferentes algoritmos relacionadas con varias ramas de la inteligencia artificial, en concreto, Machine Learning y Deep Learning.

El Capítulo 5 describe los experimentos realizados para evaluar la efectividad de los algoritmos propuestos en el capítulo y presenta los resultados obtenidos.

El Capítulo 6 muestra las principales conclusiones de este trabajo, las líneas futuras de investigación y las publicaciones derivadas del presente trabajo. El Capítulo 7 presenta las aportaciones realizadas por cada miembro del equipo, en concreto, en que se ha especializado y que ha aprendido durante el desarrollo del mismo.

Los Capítulos 8 y 9 son las traducciones al inglés de la Introducción y de las Conclusiones.

Capítulo 2

Contexto de la Investigación

2.1. Introducción

A lo largo de este capítulo se va a presentar la investigación realizada a lo largo de nuestro proyecto. Primero se realizó un analisis profundo sobre los tipos de ínteligencia artificial, las distintas ramas de esta. Posteriormente, se investigó los algoritmos que componían los distintos tipos y subtipos de ínteligencia artificial.

Tras tener claro todos los conceptos anteriormente investigados, se entendió la metodología para la construcción de un modelo de Machine Learning así como los distintos modos para evaluar el modelo resultante de entramiento realizado al algoritmo. Y por último, se realizó una investigación entorno a las ventajas de personalizar los hiperparámetros de los algoritmos y como se debe realizar esta tarea.

2.2. Funcionamiento de envío de datos por la red

Existen numerosos métodos de establecer conexiones entre usuarios en la red. Con el tiempo esto se ha ido optimizando, dando lugar al modelo OSI. Los paquetes son enrutados para llegar a su destino correcto, es decir, es el proceso por el cual se encuentra la ruta por la que debe ir el paquete para llegar a su destino final.

2.2.1. Modelo OSI

Los paquetes de datos viajan por la red, de un usuario a otro. Para poder realizar este proceso, el paquete contiene numerosos datos organizados en capas con el fin de realizar una comunicación eficaz.

La mayoría de los conjuntos de protocolo de red se clasifican en capas. La Organización Internacional para la Estandarización ha diseñado el modelo de referencia OSI [Ora21], este tiene siete capaz, que representan las operaciones de transferencia de datos comunes. Estas capas son las siguientes: [Ora20]:

1. **Red física**: se encarga de la conexión física de los elementos, es decir, medio físico de transmisión. [Imp21]

2. **Enlace**: se encarga de proporcionar los medios funcionales como los switches o el router.[Imp21]

3. **Red**: encargada de que los paquetes llegen del transmisor al receptor, es decir, se encarga del enrutamiento. Además, debe descomponer los paquetes.[Imp21]

4. **Transporte**: garantiza que los paquetes llegan en secuencia y sin erorres, para ello se va confirmando la recepción de cada paquete. [Imp21]

5. **Sesión**: tiene como objetivo mantener el enlace entre los participantes de esta. Se encarga de crear canales, conocido como sesiones, entre dispositivos. [Imp21]

6. **Presentación**: encargada de representar la información que se transmite. Esta define como se debería codificar, descodificar y comprimir los datos de manera que el otro lado de la conexión reciba los datos de manera correcta.[Imp21]

7. **Aplicación**: se encarga de definir las aplicaciones de red y los servicios de internet que puede utilizar un usuario. Contiene protocolos que permite al sotfware enviar y recibir la información.[Imp21]

Cada capa tiene una serie de protocolos distintos como se puede observar en la figura 2.1.

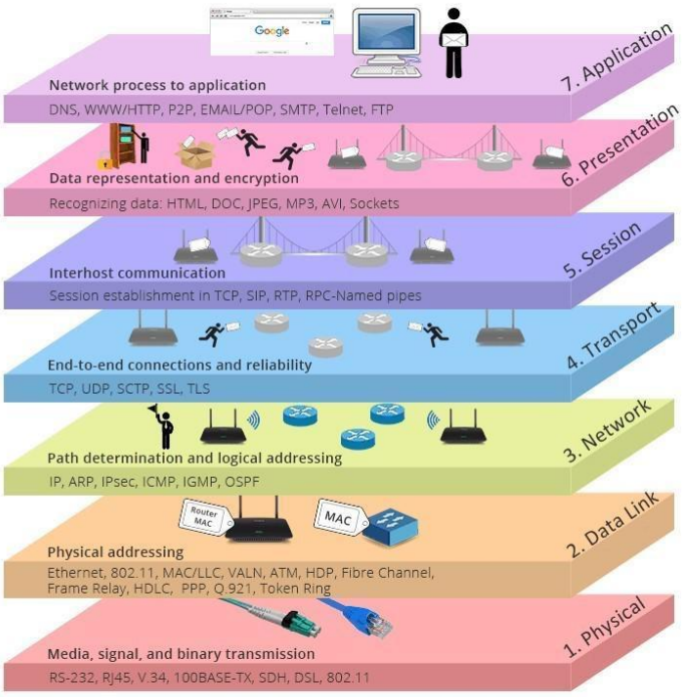

Figura 2.1: Protocolos de las capas del modelo de la arquitectura OSI[Par21]

2.3. Manejo y almacenamiento de paquetes

La información que se genera en las aplicaciones web y que permite que las personas se conecten e interactuen entre ellas contiene muchos datos los cuales se almacenan en

paquetes con una estructura específica segun cual sea su función. Los paquetes que se mueven por la red se pueden guardar en archivos, en concreto, en formato archivo PCAP. Este tipo de archivo se puede abrir con WireShark y analizar el contenido de este [Rev20].

2.4. Ínteligencia artificial

Según John McCarthy [IBM20] define la ínteligencia artificial como la ciencia y la construcción de programas informaticos que realizan tareas similares a las que realizaría de manera racional un ser humano.

De manera simple, la ínteligencia artificial, conocida también como IA, combina la ciencia informática y conjunto de datos robustos que permiten la solución de problemas.

Existen dos tipos de IA [IBM20]: la robusta y la débil. La IA débil, conocida también como estrecha, está enfocada a la realización de tareas más específicas. Este tipo de IA se encuentra implementada en tecnología como la de Siri de Apple o Alexa de amazon. La IA robusta es más general. Compuesta por la ínteligencia artificial general y la superinteligencia artificial. Este tipo de IA, es una tecnología teórica en la que una maquina tendría la misma ínteligencia que un humano. De esta rama, se despliega varios subcampos como el aprendizaje automático y el Deep Learning.

2.5. Aprendizaje automático

Es un subcampo del IA que usa modelos matemáticos de datos para aprender sin usar intrucciones directas del ser humano [Azu]. Usa algoritmos que identifican patrones en los datos y usarlos para crear un modelo con el que luego se realizan predicciones. Cuanto mayor sea la cantidad de datos más preciso seran las predicciones. La IA se puede usar en numerosos ámbitos o tareas como reconocimiento de voz, generación de texto o imágenes, etc.

El funcionamiento del aprendizaje automático consiste en recopilar y preparar los datos, posteriormente se entrena el algoritmo, para ello los datos se dividen en un conjunto de datos de entrenamiento, que se usa para ajustar el modelo, y otro de testeo. Luego, se evalúa el rendimiento del modelo y posteriormente se interpretan los datos que arroja. Esta interpretación del modelo puede ayudar a entender los datos y tomar decisiones sobre ellos.

Por lo general, los algoritmos de Machine Learning se clasifican en [SM21]:

1. Aprendizaje supervisado.
2. Aprendizaje no supervisado.
3. Aprendizaje semi supervisado.
4. Aprendizaje reforzado.

En términos generales, los algoritmos de aprendizaje supervisados y no supervisados son los más comunes y esto se debe a que llevan más tiempo disponibles. También, podemos encontrar otros algoritmos más nuevos pero más complejos los cuales permiten

obtener excelentes resultados, estos son el aprendizaje semi supervisado y de refuerzo. El aprendizaje automático se basa en el teorema de "No Free Lunch" que se basa en la idea de que cada problema es diferente al anterior y por lo tanto tiene unas características diferentes que hacen que haya un algoritmo que se ajuste a ellas mejor que el resto. En resumen, no existe un algoritmo que sirva para resolver todos los problemas sino muchos algoritmos para cada situación específica. [Bro21]

2.6. Tipos de aprendizaje automático

2.6.1. Aprendizaje automático supervisado

El aprendizaje supervisado es el más utilizado hoy en día[SM21]. Este se da cuando tenemos dos variables, una de entrada y otra de salida a las cuales denominamos X e Y respectivamente.

El objetivo de este método es aproximar la función de mapeo de la mejor manera posible de tal manera que cuando se reciban nuevos datos de entrada X este sea capaz de predecir de manera exitosa cuales son las variable de salida Y. Los algoritmos más modernos como el XGBoost y las redes neuronales, cuando se le proporcionan las respuestas correctas, van realizando predicciones de manera iterativa con los datos de entrada. En cada iteración, se comparan las predicciones realizadas con las respuestas correctas proporcionadas. De esta manera, el algoritmo reduce el error de predicción aprendiendo con mayor precisión los patrones o relaciones entre las variables de entrada y el objetivo.

2.6.2. Aprendizaje automático no supervisado

El aprendizaje no supervisado es aquel [SM21] que solo se tienen los datos de entrada X y se carece de las variables de salida. El objetivo del este método consiste es encontrar patrones ocultos o agrupaciones con las mismas características. Se denominan aprendizaje no supervisado porque, a diferencia del supervisado anterior, no se busca obtener un resultado exacto y no requiere de un maestro.

2.6.3. Aprendizaje automático semi supervisado

El aprendizaje semi supervisado es una unión de los dos anteriores pues se dispone de una gran cantidad de datos de entrada X pero de todos estos solo algunos estan etiquetados Y. [SM21] Cuando aplicamos el aprendizaje para la resolución de problemas reales nos encontramos con una falta de datos etiquetados debido a que son más costosos y suelen requerir de la ayuda de expertos en dominios mientras que los no etiquetados son más sencillos de conseguir y almacenar y por lo tanto mucho más baratos. Esto implica que la mayoría de los problemas que se intentan solucionar requieren de este método de aprendizaje automático. Puede utilizar técnicas de aprendizaje sin supervisión para descubrir y aprender la estructura en las variables de entrada.

Para ser capaz de resolver los problemas utiliza los dos tipos de aprendizaje previamente analizados, aplicando primero el no supervisado para de esta manera encontrar y aprender sobre las variables de entrada. también utiliza los supervisados para mejorar los predicciones que realiza con los datos que no estaban etiquetados, introduciendo los

resultados al algoritmo como si fueran datos de entrenamiento. De esta manera logra ser capaz de utilizar el modelo para realizar predicciones sobre nuevos datos que no habia tratado con anterioridad.

2.6.4. Aprendizaje automático reforzado

Por último, vamos a analizar el aprendizaje reforzado, el cual aunque es menos común que los demas debido a su gran complejidad, es capaz de genera excelentes resultados. La principal peculiaridad de este algoritmo es que no utiliza etiquetas como los anteriores, en su lugar utiliza recompensas, es decir, se podría decir que aprende de los errores. Para que este algoritmo funcione de manera correcta se debe lograr que el algoritmo relacione los comportamientos que estamos buscando con una señal positiva y en su lugar aquellos comportamientos que son contraproducentes y deseamos que evite con una señal negativa [SM21]. Despues se le enseña que debe elegir siempre las señales positivas antes que las negativas, es decir, que debe elegir siempre realizar los comportamientos que estamos buscando. Segun pase el tiempo se logra que el algoritmo realice menos errores de lo que solía mejorando su eficiencia.

2.7. Metodología para la construcción de un modelo de Machine Learning

Para realizar un modelo de Machine Learning se deben seguir una serie de pasos, en los cuales se puede ir aplicando distintos algoritmos o metodologías. Los pasos son los siguientes:

1. **Recolección de datos**: Se deben recolectar los datos con los que se va a entrenar el algoritmo. Se pueden utilizar distintas fuentes como APIs, bases de datos, etc.
2. **Preprocesamiento de datos**: Para que los pasos siguientes tengan el funcionamiento correcto, todos los datos deben tener el correcto formato, es decir, se transforma los datos crudos en datos útiles. En este paso, se analizan todos los datos y se eliminan los datos atípicos o las tuplas en los que faltan valores. Es decir, se dejan los datos listos para pasar a las fases siguientes.
3. **Exploración de datos**: El objetivo de este paso es entender los datos con los que se va a trabajar.
4. **Entrenamiento del algoritmo**: En esta etapa, se alimenta el algoritmo con los datos anteriormente preprocesados y explorados. La idea es que se extraiga información útil para que luego se puedan realizar las predicciones.
5. **Evaluar el modelo**: Se realiza una evaluación del modelo anteriormente entrenado. Si no se esta satisfecho con el modelo, se debe volver al paso anterior. En este paso se utilizan métricas de clasificación.

2.7.1. Datasets de Entrenamiento,Test y Validación

Para saber que el modelo que se esta realizando funciona bien, se utilizan distintos conjuntos de datos: los de entrenamiento y los de test.

Para conseguir ambos conjuntos de datos, se selecciona el conjunto de datos con el que se va a trabajar y se divide en [Na820]:

1. **El conjunto de entrenamiento**: Se usará para entrenar el algoritmo que se ha seleccionado.

2. **Etiquetas del conjunto de entrenamiento**: Son los resultados esperados del conjunto anterior.

3. **El conjunto de testeo**: Se usará una vez el modelo este entrenado.

4. **Etiquetas del conjunto de testeo**: Son los resultados para validarlo.

El funcionamiento general, mostrado en la ilustración 2.2, consiste en entrenar el algoritmo con el conjunto de entrenamiento y con dichas etiquetas asociadas a dicho conjunto. Posteriormente, se generan una serie de métricas para su evaluación. Una vez el modelo esta entrenado, se introduce en el modelo el conjunto de testeo para comprobar el resultado que produce con el esperado. Una vez el modelo esta entrenado, se introduce en el modelo el conjunto de testeo para comprobar el resultado que produce con el esperado, que se almacena en el conjunto de etiquetas de testeo.

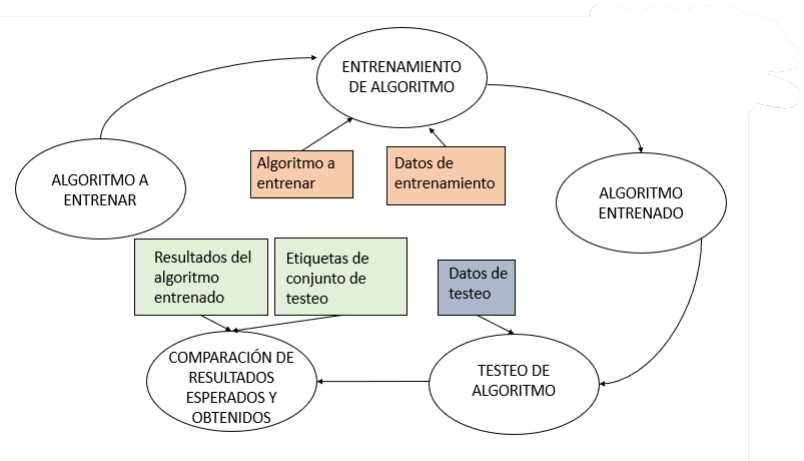

Figura 2.2: Entrenamiento de algoritmo con set de Entrenamiento y Testeo

Si la acurracy del testeo y del entrenamiento es muy parecido significa que el modelo funciona de manera correcta. Si estas métricas difieren, es porque existe sobreajuste. Esto significa, que el modelo entrenado se ajusta demasiado a los datos de entrenamiento y no generaliza bien para conjuntos que no se han visto anteriormente.

Para poder solventar esto, se tuneara el algoritmo a través de los hiperpárametros, que permiten determinar como aprende el algoritmo: número de hojas, profundidad, etc. para determinar el mejor modelo posible.

El set de validación se emplea cuando existe pocos datos para que el azar no intervenga a la hora de entrenar el algoritmo.

2.8. Algoritmos de clasificación supervisados

2.8.1. Métodos de ensamblado o combinados

Estos métodos son técnicas que generan un gran número de modelos para luego combinarlos y de esta manera obtenener un mejor resultado[SM21]. Los métodos de ensamblado suelen obtener resultados más precisos que los modelos individuales pero también tienen sus desventajas. Las técnicas son bagging, Random Forest y boosting.

2.8.1.1. Bootstrapping

Para ser capaces de comprender como funcionan las tecnicas de clasificación supervisadas, primero debemos de entender en que consiste el bootstrapping. El bootstrapping es un método de remuestreo aleatorio con reemplazo el cual tiene como fin mejorar la distribución de los datos en un muestreo estadistico[SM21]. Para ello se selecciona dentro de un conjunto de datos, un subconjunto de menor tamaño sobre el que vamos a aplicar un muestreo aleatorio y realizamos este proceso n veces. De esta manera logramos estimar con un número menor de datos la media y la desviación estandar del conjunto y también nos permite aproximar el sesgo y la varianza. Para ser capaz de de aplicar este método se deben de cumplir dos hipótesis:

1. *Representatividad.* Es necesario un tamaño N del conjunto para que de esta manera se mantenga las mismas propiedades y características que podemos encontrar en la población de la que provienen los datos. Por ejemplo si queremos estudiar la población de una ciudad no usaremos a todos sus habitantes, solo un conjunto de personas lo suficientemente grande como para que todas las demas se sientan representadas. Esto es necesario para que el muestreo sea una buena aproximación a la distribución real.

2. *Independencia.* Es necesario que los datos no esten demasiado correlacionados entre si pues de esta manera se podria generar un sesgo. Esto puede suceder si las muestras de arranque, subconjuntos que extraemos de manera repetida y con reemplazo del conjunto, son muy parecidas entre si [Ben]. Para evitarlo es necesario que el tamaño N del conjunto de datos sea bastante más grande que el tamaño B de las muestras, evitando asi que se repitan los mismos datos. Por ejemplo si queremos estudiar la altura media de los estudiantes de un instituto y escogemos un conjunto de 20 estudiantes y muestras de 5 se nos generara un sesgo, pero si el conjunto es de 1000 y la muestra sigue siendo de 5 logramos que la probabilidad de que se repitan los datos disminuya y reducimos cualquier sesgo que pueda haber.[Bos20]

Las técnicas de ensamble son: bagging y boosting. Random Forest emplea bagging y xgboost y gradient boosting usa boosting.

2.8.1.2. Bagging

La primera de las tecnicas que vamos a analizar se trata del bagging, la cual se basa en la idea de utilizando varios modelos en pararelo aunque cada uno de ellos tengo un error si al final se realiza un promedio de sus resultados se puede lograr reducir el error individual y obtener de esta manera una varianza menor. El problema que nos encontramos

cuando tratamos de llevar esto a la práctica reside en la falta de datos necesarios para un funcionamiento óptimo de la técnica. Aquí es donde encontramos la razón por la cual el bootstrapping es tan importante, si no podemos obtener tantos datos como los que serian necesarios podemos usar subconjuntos de una muestra de datos. Como estos subconjuntos mantienen las características del conjunto original podemos confiar en que el resultado que se obtiene es el mismo que si se hubieran utilizado todos los datos [SM21].

La figura 2.3 ejemplifica el proceso que se sigue cuando es utilizado el bagging. Primero creamos los subconjuntos por medio del Bootstrapping y con ellos realizamos un modelo de arbol débil de decisiones por cada subconjunto. Al final se unen de tal manera que el resultado promedio de todos ellos tiene una varianza menor que los resultados individuales.

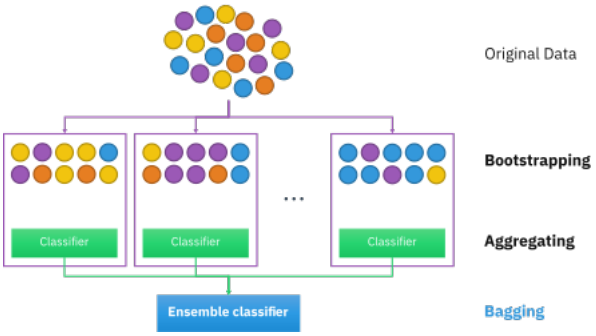

Figura 2.3: Bagging

Si aplicamos esta técnica junto con árboles de decisión logramos unos arboles más ajustados los cuales tienen pocas muestras de entrenamiento en cada nodo hoja del árbol y no se podan. El resultado que obtenemos en comparación con los anteriores son árboles con una alta varianza y un bajo sesgo los cuales mejoran la eficiencia del modelo. Otra de las ventajas de este modelo es que al utilizar modelos paralelos podemos aplicar distintos algoritmos en cada uno de ellos y de esta manera tratar de mejorar el rendimiento.

Como conclusión este es un modelo eficiente el cual nos permite conseguir resultados con una menor varianza y nos libramos del sobreajuste que genera el utilizar un único modelo pero nos encontramos con la desventaja de que se pierde interpretabilidad y al tener que utilizar algoritmos trabajando en parelelo también aumenta el coste computacional.

Random Forest se basa en este algoritmo y utiliza la técnica aprendizaje por conjunto creando tantos árboles como el número de subconjunto de datos y al final agrega el resultado de todos ellos. Obtenemos unas predicciones con una menor varianza que las individuales como sucedia con el bagging. Pero este modelo nos aporta unas ventajas adicionales como pueden ser la estimación de valores ausente o la manera en la que maneja los valores atípicos de manera automÁtica. Otra ventaja, es la gestión que hace de los valores de entrada siendo capaz de seleccionar los más relevantes entre una gran cantidad de ellos. Este modelo cuenta con dos grandes desventajas y es que se pierde control sobre el funcionamiento y además, el coste en tiempo es muy superior pues es necesario entrenar todos los arboles que genera.

2.8.1.3. Tree Boosting

El tree boosting se basa en la generación de arboles débiles, los cuales genera en serie y cada uno centrado en los errores del anterior[SM21]. Cada vez que se añade un arbol

utiliza un subconjunto de datos que es un poco diferente del que ha utilizado el arbol anterior y no completamente distinto. De esta manera por medio de la unión de muchos arboles débiles se obtiene un resultado como si fuese generado por un modelo más fuerte.

Por lo tanto la principal diferencia que encontramos entre los dos modelos es que el Tree Boosting al generar de manera secuencial los modelos, de modo que cada uno de ellos corrige los errores del anterior nos ofrece una mayor eficiencia y precisión. El modelo funciona como se muestra en la figura 2.4

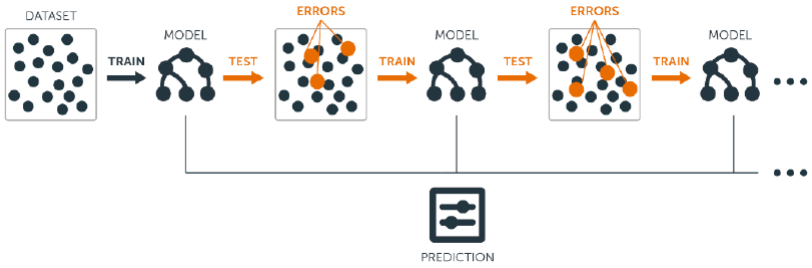

Figura 2.4: Funcionamiento del boosting[SM21]

Hay varios tipos de algoritmos que utilizan esta técnica de generación de modelos débiles de manera secuencial pero con algunos cambios en las técnicas que utilizan para generarlos. Los principales modelos los cuales explicaremos más adelante son [SM21]:

1. AdaBoost (Adaptive Boosting).
2. Gradient Boosting.
3. XGBoost Algoritm.

2.8.1.3.1. Gradient Boosting

El primero de los algoritmos basados en tree boosting que vamos a estudiar es el Gradient boosting. La principal utilidad de este es analizar la regresión y en problemas de clasificación estadística[Wik20b]. Al igual que el tree bosting se encarga de generar modelos débiles en serie de tal manera que cada uno aprende de los errores del anterior y obtiene un mejor resultado.

Este modelo [ich20] es un modelo intuitívo en el que cada nuevo modelo que se genera puede mejorar si se combina con modelos anteriores ya que minimíza el error de predicción. Por esta razón se debe indicar a cada modelo cual es su resultado objetivo y que de esta manera tratar de reducir el error. El aumento de gradiente se puede utilizar tanto para clasificación como para regresión.

Entre las ventajas del gradient boosting en [ich20] se destaca el tipo de datos que puede procesar. La variables pueden ser tanto numéricos como categóricos sin tener que crear variables dummy o one-hot-encoding y no necesitan ningún tipo de distribución estadística específica. Es decir, es bastante amplio al valor de los datos. Por lo general, este tipo de algoritmo de aprendizaje automático no necesita que los datos sean preprocesados ni limpiados, ni se ve afectado por valores anormales. Además, es capaz de seleccionar predictores de forma automática, es decir, identifica las variables automáticamente lo que

permite un mejor entrenamiento. Un gran punto a favor del gradient boosting, se basa en que si el valor de una variable no está disponible, podemos lograr que este realice una predicción gracias a las observaciones que pertenecen al último nodo alcanzado, pero tiene una desventaja y es que la predicción que obtenemos es menor que si tuviéramos los datos disponibles. Este algoritmo, tiene una alta escabilidad, es decir, puede aplicarse a grandes números de observaciones y puede aplicarse este algoritmo tanto en problemas de regresión como en clasificación.

El gradient boosting tiene varias desventajas. La primera de ellas es que cuando las variables son continuas, se pierde información al categorizarlas cuando se realiza la división de nodos. Otro punto en contra se encuentra a la hora de ramificar los árboles debido a que identifica y evalúa la división de cada variable dependiendo de una determinada medida(Gini, entropía...) y las variables continuas tienen mayor probabilidad de tener algún punto de corte. Esto favorece la creación de árboles. Por último, no son capaces de extrapolar las observaciones que realizan con los datos de entrenamiento fuera del rango de las variables observadas.

Gradient Boosting se ha convertido en el principal algoritmo ha utilizar cuando se trata con datos tabulares debido a sus excelentes resultados. Esto ha favorecido su desarrollo y ha supuesto que haya múltiples implementaciones. Cada una de ellas cuenta con unas características propias las cuales hacen que sean más óptimas dependiendo del caso de uso. Scikit-learn, tiene dos implementaciones nativas[dd20a]:

1. GradientBoostingClassifier y GradientBoostingRegressor: Son las primeras implementaciones que se hicieron de este modelo. En estas primeras se preprocesaban los datos para lograr mínimizar los errores de observación[Lim20]. Tampoco trabajaba con partes del algoritmo en paralelo. Eran capaces de funcionar con matrices cuya densidad de ceros es predominante. Por último, esta implementación no necesita realizar el one-hot-encoding de variables categóricas, es decir, no necesita crear una columna para cada variable distinto que exista en la categoría dándole un valor 1 o 0, dependiendo de si pertenece o no a la categoría [**?**].
2. HistGradientBoostingClassifier y HistGradientBoostingRegressor: Es una nueva implementación, es todavía experimentar pero tiene grandes ventajas frente a la primera, entre las que destaca su mayor rápidez. A diferencia de la anteriormente explicada, esta realiza binning, permite la parelización del algoritmo y admite valores missing. Sin embargo no se pueden utilizar variables sparse.

Este algoritmo presentado en [sl22] puede ser adaptado a las necesidades del usuario según diferentes parámetros. Se puede modificar el número de estimadores que usar a través del párametro *n_estimators*, que determina el número de etapas a realizar. Además, se puede determinar el ratio de aprendizaje a través del *learning_rate* el ritmo de aprendizaje del algoritmo, cuanto menor sea más árboles se necesitan para alcanzar resultados óptimos pero menor es el ritmo de sobreajuste [Ama20]. Además, se pueden personalizar más factores como la profundidad que se quiere explorar en cada árbol o el número de hojas mínimo que tiene que tener el árbol para ser considerado una hoja válida para explorar.

2.8.1.3.2. XGBoost

El XGBoost Extreme Gradient Boosting es un algoritmo predictivo supervisado que utiliza el principio de boosting[Bos20]. El funcionamiento de XGBoost se basa en la

creación de modelos de predicción débiles donde cada uno de ellos recibe los resultados del anterior y con ellos es capaz de crear un modelo más fuerte el cual obtiene un mejor resultado en su predicción, así como más estabilidad en los resultados, como se puede observar en la figura 2.5.

En este proceso, se emplea el descenso del gradiente como algoritmo de optimización. Para entrenar el modelo, los parámetros que utilizamos en cada uno de los modelos débiles es ajustado iterativamente buscando encontrar el mínimo de una función objetivo. Si el modelo que se está comparando es mejor que el anterior, se utiliza para realizar modificaciones en los siguientes, en caso contrario se rechaza este modelo y se regresa al mejor modelo anterior. Este proceso se itera, hasta que se la diferencia entre modelos es insignificante, lo que significa que se ha encontrado el mejor modelo proceso. Este proceso también finaliza si se llega al número máximo de iteraciones posibles definidas por el usuario.

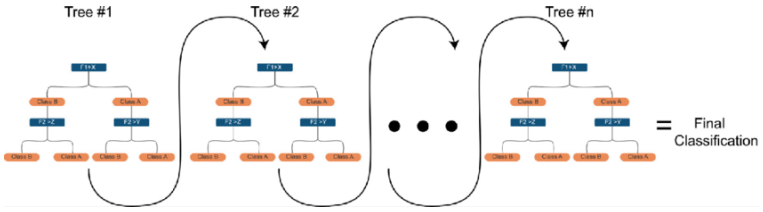

Figura 2.5: Esquema funcionamiento XGBoostl

Los modelos débiles que utiliza XGBoost son árboles de decisión de varios tipos, los cuales pueden ser utilizados tanto para tareas de clasificación, como de regresión. Se debe usar este algoritmo cuando los datasets sean grandes y el número de variables sea menor que el número de observaciones. Además, se puede aplicar en problemas que se mezclen variables categóricas y numéricas, o aunque solo hay presencia de variables numéricas.

Las principales ventajas que ofrece este algoritmo es que es capaz de manejar grandes bases de datos con múltiples variables, e capaz de manejar valores perdidos, ofrece unos resultados son muy precisos y posee una excelente velocidad de ejecución.

Pero también tiene ciertas desventajas. Puede llegar a consumir muchos recursos computacionales en grandes bases de datos, por lo que se recomienda reducir el número de variables sobre las que se aplica, se necesita ajustar los parámetros para lograr mínimizar el error de precisión y evitar sobreajuste del modelo y por último, solo sirve para vectores númericos.

2.8.2. Regresión logística

La Regresión Logística Simple es un método de regresión que estima la probabilidad de una variable cualitativa binaria en función de una variable cuantitativa [dd20b]. Uno de sus principales usos es para la de clasificación binaria, es decir, para clasificar las observaciones realizadas según el valor que ha tomado la variable usada como predictor en un grupo u otro.

Si una variable cualitativa con dos niveles se codifica como 1 y 0, es posible ajustar un modelo de regresión líneal por mínimos cuadrados $\beta_0 + \beta_1 X$. El problema de esta aproximación es que, al tratarse de una recta los valores extremos del predictor se saldrían de los niveles codificados, es decir, 0 y 1. Este hecho, contradice la idea de

que la probabilidad debe estar dentro del rango 0 y 1. La regresión logística solventa este problema. La regresión logística transforma el valor devuelto por la regresión líneal aplicando una función para que se comprenda entre 0 y 1. Existen varias funciones que cumplen esta descripción, una de las más utilizadas es la función logística, también conocida como función sigmoide.

$$\sigma(t, a, m, n, r) = \frac{1 + me^{-}t/r}{1 + ne^{-}t/r} \quad \text{Función sigmoide} \tag{2.1}$$

Esto es porque para aquellos valores de x muy grande, la función sigmoide devuelve un 1 y para aquellos valores de x muy pequeños, la función anteriormente mencionada devuelve un 0. Si se emplea la función logarítmica de esta, se obtiene lo que se conoce como LOG of ODDs.

$$f(x) = \log_a b \quad \text{Función logarítmica} \tag{2.2}$$

En la regresión líneal simple, el valor de la variable dependiente Y se ve directamente afectado por el valor de la variable independiente X. En cambio la regresión logística es diferente, la probabilidad de que la variable respuesta Y pertenezca al nivel de referencia 1 depende del valor que adquieran los predictores, mediante el uso de LOG of ODDs que consiste en la inversion de la función sigmoide. Si consideramos que la probabilidad de que suceda un evento sea cierta es de 0. 7, y por lo tanto la de que falso es 0. 3. Los ODDs o razón de probabilidad de verdadero es la división entre la probabilidad del evento cierto entre la de que sea falso *p/q*. En este caso los ODDs de verdadero son *0.7 / 0.3 = 2.3*, es decir que se esperan 2.3 eventos verdaderos por cada uno de los falsos.

Si la probabilidad es mayor también lo son los ODDs, y viceversa. El rango de valores entre los que se pueden encontrar son [0, ∞]. Pero como la probabilidad solo puede estar entre [0, 1] se utiliza a una trasformación logarítmica para convertir el rango de esta a [∞, $+\infty$].

$$\log(ODDs) = \log(P) = \ln\left(\frac{P}{1-P}\right) \quad \text{Función LOG de ODDs} \tag{2.3}$$

Una vez que se ha obtenido la relación líneal entre el logaritmo de los ODDs y la variable predictora X, se debe de estimar los parámetros β_0 y β_1. Se busca una combinación de valores óptimos los cuales ofrezcan la mayor verosímilitud posible, es decir el valor de los parámetros 0 y 1 con los que se maximiza la probabilidad de obtener los datos observados. También se puede ajustar el modelo por medio del descenso de gradiente pero este no es el método de optimización más adecuado para resolver la regresión logística.

Existen diferentes técnicas estadísticas para calcular la significancia de un modelo logístico en su conjunto. Pero todos darán al modelo como útil si logra demostrar que este es mejor en algún aspecto que el modelo nulo, es decir, aquel que carece de predictores y usa solo 0.

Existe diferencia entre la interpretación de un modelo de la regresión líneal y la logística. En el modelo de regresión líneal la variable β_1 mide el cambio promedio en la variable dependiente Y por cada unidad que aumenta el predictor X. En la regresión logística, 1 indica el cambio que sufre el logaritmo de ODDs cada vez que aumenta en una unidad la variable dependiente X. En este modelo, a medida que aumenta la probabilidad

de la variable dependiente por unidad de variable de la independiente, depende de la posición en la curva logística en la que se encuentre.

Este algoritmo permite el tuneado a través de varios hiperpárametros. Los más importantes son los siguientes [sl21c]:

1. class_weight: Permite determinar si los datos son balanceados o no.
2. solver: Con este hiperpárametro se determina para optimizar el algoritmo. Para determinar cual usar hay que tener en cuenta si son tamaños de datos grandes o pequeños, balanceados o no o incluso el tipo de penalty.
3. penalty: Nivel de aprendizaje.

Entre las ventajas de este algoritmo, destaca lo sencillo que es entenderlo así como que su rápidez de entrenamiento. Además, trabaja de manera rápida con grandes volúmenes de datos, aprende rápido. Otro gran punto a favor para su uso, es que rara vez produce sobreajuste. Entre sus desventajas el hecho de que es complicado que se ajuste a datos lineales y no gestiona bien los valores atípicos.

2.9. Redes neuronales

Las redes neuronales es un tipo de aprendizaje automático que imita a las redes neuronales cerebrales y se compone de neuronas. Una neurona es una unidad básica de red, es una unidad de procesamiento de información. Le entra un dato y lo procesa como se ve en la figura 2.6. Un conjunto de neuronas forman redes neuronales [MB19].

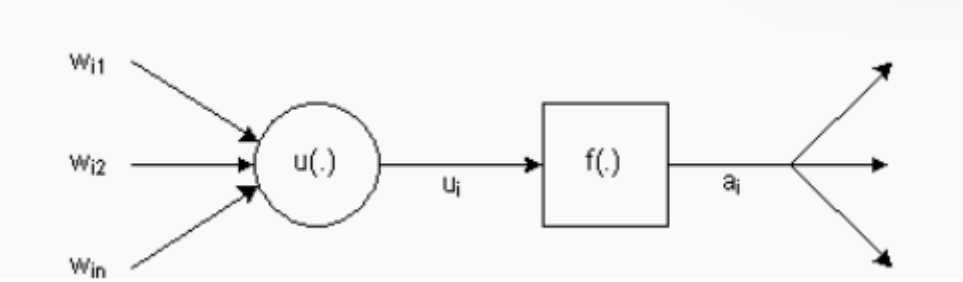

Figura 2.6: Esquema de red neuronal

Las redes neuronales se pueden clasificar dependiendo del número de capas que tiene y su estructura. En el caso que tenga pocas capas seria una red neuronal multilayer perceptron, conocida como, MLP. Cuando tienen más se entiende como Deep learning. En cuanto a su estructura, existen redes convuncionales y recurrentes.

2.9.1. Entrenamiento de las Redes Neuronales

El objetivo de entrenar una red neuronal minimizar el error que este genera cuando realiza una predicción, es decir, reducir el número de veces que predice un resultado pero en la vida real el resultado fue diferente. Para lograrlo vamos a utilizar un vector de pesos el cual ajustaremos segun el error que genere cada neurona. El problema que nos

encontramos es que este no es el único parámetro de la red neuronal, hay muchos más los cuales debemos de tener en cuenta. [IBM]

El entrenamiento de las redes neuronales se realiza a través de ciertos algoritmos cuyo objetivo es mínimizar la función de error. El más importante de estos algoritmos es el backpropagation.[IBM]

Lo primero que se debe hacer antes de entrenar una red es identificar cuales son las condiciones de parada [IBM]. Esta pueden ser:

- Se ha alcanzado una cuota de error que consideramos suficientemente pequeña.
- Se ha alcanzado el número máximo de iteraciones previamente definidas como limite para el entrenamiento.
- Se ha obtenido un error cero y ya no es necesario continuar.
- Se ha llegado a un punto de saturación a partir del cual error no disminuye.

Ahora veremos cuales son los elementos principales en el entrenamiento de las redes neuronales:

2.9.1.1. Pesos Sinápticos

Normalmente una neurona recibe varias entradas al mismo tiempo de otras neuronas. Cada entrada tiene un peso relativo el cual indica la importancia de la entrada dentro de la función de agregación de la neurona, es decir, de la información recibida cuanto impacto tendrá en la salida de dicha neurona. [AT21]

El problema de los pesos consiste en su inicialización, por lo que se suele optar por inicializar los pesos aleatoriamente a valores cercanos a 0. Y a lo largo del entrenamiento estos pesos se van ajustando hasta lograr optimizarlos.

2.9.1.2. Regularización

La regularización consiste en una modificación en el modelo para impedir que este se ajuste demasiado a los ejemplos de entrenamiento y debido a esto se genere un sesgo por sobreentrenamiento.

Para entender mejor este concepto utilizamos la L2 o L1, las cuales tratan de modificar la función de coste añadiendo un nuevo término. Este busca que el algoritmo encuentre un equilibrio entre mínimizar su error original y mantener la simplicidad del modelo [Ber].

2.9.1.3. Forward propagation

Es la manera en la cual las redes neuronales generan las predicciones. Para lograrlo los datos de entrenamiento pasan por las distintas capas neuronales hasta llegar a la capa de salida donde se predice la clase a la cual pertenecen los datos de entrenamiento. [Set21]

Una vez que hemos realizado la predicción debemos de comparar los resultados obtenidos con los resultados deseados para conocer la función de error de la salida.

2.9.1.4. Back propagation

Cuando se realiza la predicción y se obtiene la función de error, se utiliza el backpropagation para optimizar la predicción que genera la red neuronal.

Este proceso busca que cada neurona conozca el error que ha cometido y como ha afectado esto en el resultado final, para ello se envía desde la salida donde se encuentra el resultado final hacia todas las neuronas de la capa oculta las cuales han participado en el proceso. Cada una de ellas reciben una fracción de la señal total del error que depende de cuanto ha aportado al resultado. Se repite este proceso en cada capa oculta hasta que todas las neuronas han recibido una señal de error sobre su aportación relativa al error final [Set21].

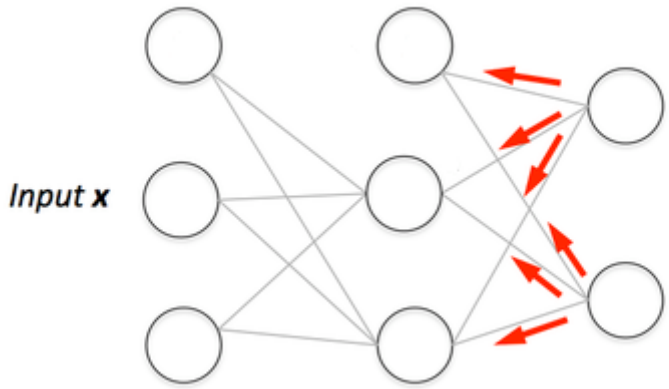

Figura 2.7: Esquema propagación red neuronal

A medida que entrenamiento avanza, las neuronas de las capas intermedias van organizándose entre sí de tal modo que son capces de aprender a reconocer distintas características del espacio total de entrada.

2.9.1.5. Función de coste

La función de coste o de error calcula la diferencia entre la salida que nos ofrece el modelo y la salida correcta. [Vil20]

$$\frac{1}{2}\sum(t_n - a_n)^2 \quad \text{Función de coste} \tag{2.4}$$

La calculamos de la siguiente donde t representa el resultado obtenido y a el esperado.

2.9.1.6. Conjunto de datos

Para lograr entrenar un algoritmo se requiere de un conjunto de datos los cuales es importante dividir en dos o tres conjuntos diferentes.

Comenzamos con el conjunto de entrenamiento el cual tiene como objetivo que las neuronas aprendan patrones sobre los datos [wik20a].

El segundo, de validación, se encarga de comprobar como actúa el modelo una vez que ha sido entrenado y se encuentra con datos que no ha visto previamente. Por esta razón es importante que los datos utilizados en este set sean diferentes al de entrenamiento o se podría generar un sesgo [wik20a].

También nos permite definir cuales son los hiperparámetros optimos para la red neuronal, algunos ejemplos de estos son: learning rate, número de capas, número de nodos, funciones de activación, etc.

Estos hiperpÁrametros se van modificando hasta lograr los mejores resultados y para ello comprueba el resultado del set de validación, se cambian los valores de los hiperparámetros y se vuelve a comparar con los nuevos resultados hasta encontrar aquellos que mejor resuelven el problema.

El último conjunto de datos que se tiene en cuenta cuando se entrena una red neuronal es el conjunto de prueba. Este set se usa al final para comprobar que tan bueno es el entrenamiento de la red neuronal y asi encontrar los mejores parámetros.

Normalmente, se emplean el conjunto de train y test. En caso de tener pocos datos, se utiliza el de validación [wik20a].

Si no se tienen los suficientes datos, estos se ignoran y solo se toman los datos de entrenamiento y de validación.

2.10. Redes Gráficas Convolucionales

2.10.1. Definición redes neuronales convolucionales

Lo primero que debemos de entender es en qué consiste una red neuronal convolucional. Estas son una serie de redes que fueron creadas tratando de imitar el funcionamiento del cerebro humano, siendo capaces de aprender en los diferentes niveles de abstracción [Gon19].

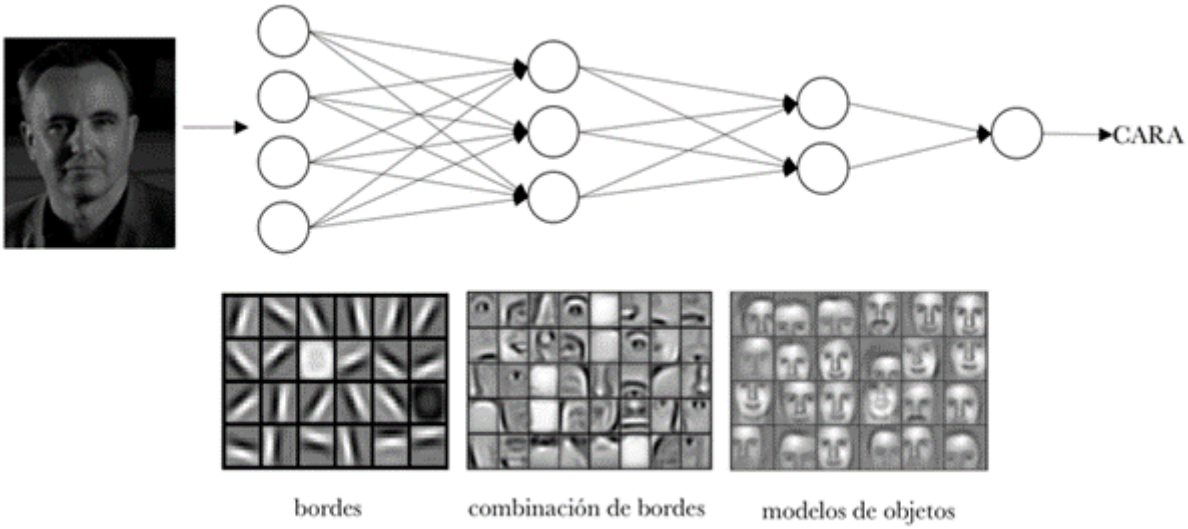

Figura 2.8: Ejemplo de red neuronal de manera sencilla

Para lograrlo utilizamos la convolución que consiste en colocar un filtro sobre una imagen y desplazarlo por todas las zonas de la imagen hasta que cada pixel de esta ha pasado por el filtro. Cada vez que movemos el filtro sobre la imagen se calcula un valor el cual proviene de la suma de todas las multiplicaciones de los pixeles que se encuentran el ese momento en el filtro. Cada uno de ellos tiene un valor dependiendo de la posición del filtro en donde se encuentran. Con todo esto generamos un mapa de características.

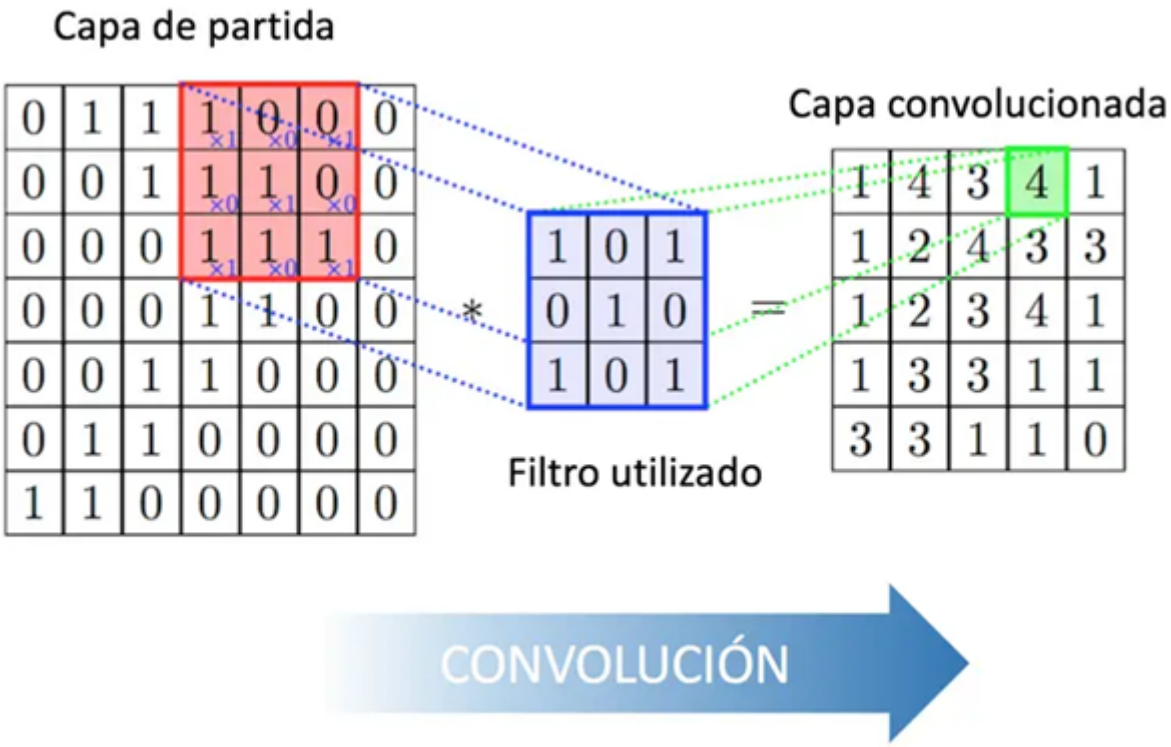

Figura 2.9: Proceso de convolución

2.10.1.1. Estructura redes convolucionales orientadas a imagenes

La estructura de las redes convuncionales, reflejado en la figura 2.10 es la siguiente:

(a) Input: Serán los píxeles de la imagen. Alto, ancho y profundidad, 1 para color o 3 para RGB. La entrada a una capa convolucional suele ser una imagen de tamaño m x n x c donde m y n son el ancho y alto y c, la profundidad.

(b) Convolution: como hemos visto previamente permite reducir el tamaño utilizando un filtro el cual desplazamos por la imagen.

(c) Pooling: técnica para la generalización de las características extraidas por el filtro y de paso, ayuda a la red neuronal ha reconocer características independientemente de su localización en la imagen.

(d) Fully connected: Se encarga de la clasificación de la imagen según las probabilidades de pertenecer a cada clase.

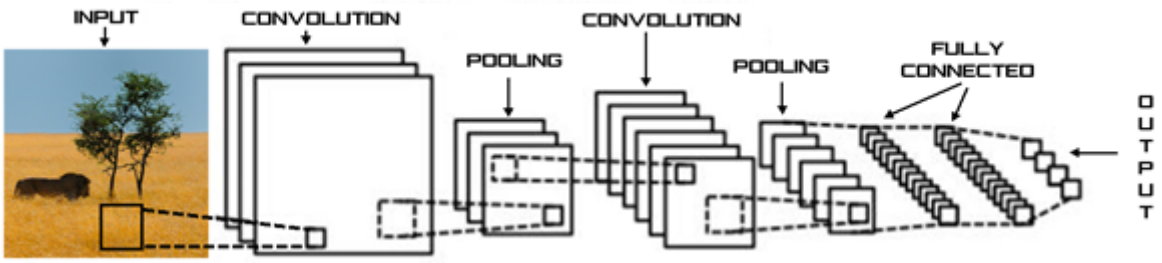

Figura 2.10: Proceso de convolución

2.11. Deep Learning

Este tipo de aprendizaje automático se basa en redes neuronales. El deep learning se caracteríza por la automatización de la extracción de características del proceso eliminando la intervención de los seres humanos para la realización de este proceso de manera manual. Es por ello que permite el uso de datos más grandes. Esta es la mayor diferencia con el Machine Learning, que si necesita la ayuda de la capacidad humana para aprender[IBM20].

El funcionamiento básico del deep learning consisten en niveles jerárquicos. El primer nivel aprende algo sencillo y simple. El segundo, coge la información del primer nivel y aprende algo más complicado. Esta información se la pasa al tercer nivel, que la usa y aprende algo todavia más difícil y así sucesivamente con todos los niveles jerárquicos que la componen[Ind20].

En la vida real se usa este tipo de tecnología para identificar marcas a través de logos en fotos de redes sociales, anuncios orientados y predicción de preferencias de los clientes o incluso identificación de clientes potenciales.

2.12. Grafos

Es una estructura de datos muy útil a la hora de usar la ínteligencia artificial ya que permite representar información muy heterogénea, como por ejemplo la comunicación entre dos aplicaciones. Mediante el uso de estos, se puede extraer de manera eficaz y rápida características para refórmular los datos y analizarlos en conjunto, incrementando así la eficacia. Otro gran punto a tener en cuenta al usar grafos, es que aporta escabilidad. Esto permite, el uso de grandes capacidades de datos de manera automática.

2.13. Redes neuronales convolucionales gráficas

Uno de los principales métodos para el tratamiento de datos son los gráficos, debido a la comodidad y claridad que ofrecen al interpretarlos. Por esta razón es comprensible que los modelos de aprendizaje automático en muchas ocasiones utilicen datos que aparecen en gráficos prediciendo sus resultados. También, se pueden crear modelos más avanzados los cuales utilizando elementos más complejos predicen las relaciones entre ellos. Este tipo de algoritmos se aplican para solventar problemas como el procesamiento de lenguaje natural o el procesamiento de imagenes. [Hon21] la La estructura es parecida a la del CNN pero recibe un grafo como input y las operaciones de convolución y pooling son diferentes. El

objetivo final será extraer las características espaciales de los gráficos topológicos.

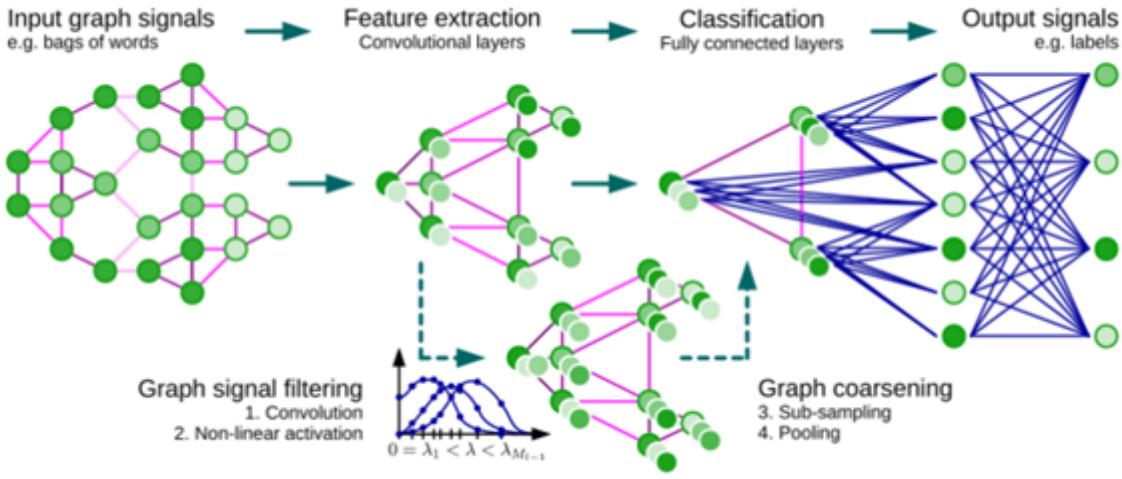

Figura 2.11: Grafo convuncional

2.13.0.1. Extraccion de características

Hay dos maneras de obtener las características:

(a) Spatial: extrae las características espaciales en el gráfico topológico, luego busca los vecinos adyacentes a cada vértice.

No requiere una estructura gráfica homogénea, a su vez, requiere preprocesamiento del gráfico para permitir el aprendizaje en él y todas las estructuras de grafos heterogéneos deben asignarse a una salida de tamaño fijo antes de realizar el aprendizaje en ella.

Clasificamos las redes convolucionales de gráficos espaciales en los modelos clásicos basados en CNN y en modelos basados en propagación.

Los modelos clásicos de CNN sobre datos en forma de cuadrícula, como imágenes, han demostrado tener un gran éxito en muchas aplicaciones relacionadas, incluida la clasificación de imágenes, detección de objetos, segmentación semántica, etc. Las propiedades básicas de los datos en forma de cuadrícula que son explotados por las arquitecturas de convolución incluyen: el número de píxeles vecinos para cada píxel y el orden espacial de escaneado de imágenes.

Los modelos basados en propagación surgen de la teoría de que para gráficos arbitrariamente grandes, el número de valores únicos por nodo suele ser un número muy grande. En consecuencia, habrá muchas matrices de peso para aprender en cada capa, lo que posiblemente conduzca al problema de sobreajuste. Se propone una red convolucional de gráficos basada en difusión que evoca las propagaciones y agregaciones de representaciones de nodos mediante procesos de difusión de gráficos.

Pero nos enfrentamos a la desventaja de que los vecinos extraídos de cada vértice son diferentes, por lo que el procesamiento de cálculo debe ser para cada vértice y el efecto de extraer características puede no ser tan bueno como la convolución.

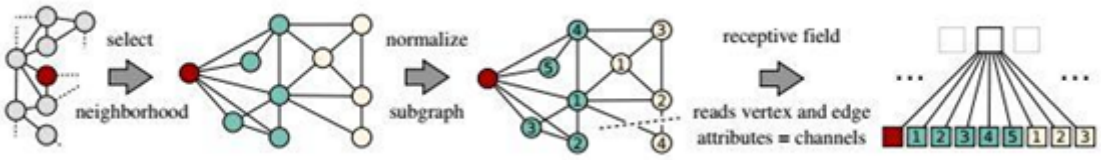

Figura 2.12: Explicación de spatial

(b) Spectral: Consideramos que los métodos basados en espectros son aquellos que comienzan con la construcción del filtrado de frecuencia y utilizan la teoría de gráficos para realizar operaciones de convolución en gráficos topológicos.

Para ello usamos los autovalores y autovectores de la matriz laplaciana del gráfico para estudiar las propiedades del gráfico.

2.14. Evaluación del modelo

2.14.1. Métricas de clasificación

Existen diferentes resultados que puede arrojar el modelo, que un dato encaje respecto a lo predicho o no. Para evaluar el rendimiento de un modelo, podemos emplear diferentes técnicas dependiendo el modelo a resolver. Existen diferentes términos estandar para medir el desempeño del clasificador [Shi20]:

1. Exactitud: también conocido como accurracy, es la proporción de las predicciones correctas de las totales realizadas.
2. La Razón de Verdaderos Positivos: Denominado como TPrate o Recall, es la proporción de positivos que realmente lo son.
3. La Razón de Falsos Positivos, conocido como FPrate, muestra la proporción de casos negativos que son incorrectamente identificados como positivos.
4. La Razón de Verdaderos Negativos, es una métrica básica, que muestra la proporción de casos negativos y positivos correctamente identificados.
5. La Razón de Falsos Negativos, conocido como FNrate, es la proporción de casos positivos que fueron incorrectamente clasificados como negativos.
6. La precisión es la proporción de casos predichos positivos que fueron correctos.

Frecuentemente son utilizados también los términos de sensibilidad y especificidad. El primero de ellos se centra en la capacidad del modelo de al ser afectado por los casos positivos. Mientras que la especifidad es la complentariedad a los falsos positivos. Los clasificadores buscarán aumentar su sensibilidad y tener una gran especificidad para de estar manera optimizar su entrenamiento.

Además, existen métricas de clasificación que combinan varias de las anteriormente explicadas, haciéndolas así más eficaces, como F1 score y la matriz de confusión.

2.14.2. F1 score

Esta herramienta se basa en dos conceptos [Kor21]: la precisión y la especificidad. El primer concepto, la precisión, se basa en la fórmula 2.5.

$$Precision = \frac{\#TruePositives}{\#TruePositives + \#FalsePositives} \tag{2.5}$$

Esta fórmula arroja la interpretación siguiente: Si es cierto que aunque puede indicar que lo predecido es correcto puede ser por dos razones. La primera de ellas porque su método tiene alto nivel de ruido, es decir, un alto nivel de falsos positivos. La segunda razón, es porque el modelo es muy exacto, es decir, encuentra pocos positivos pero los positivos que encuentra son muy correctos al haber sido clasificados como tales.

El segundo principio es la especificidad, que se basa en la fórmula 2.6, que se puede interpretar de dos maneras. Una alta especificidad puede ser porque identifica altos casos como positivos, en los que se puede haber identificado casos negativos como positivos. Otra interpretación puede ser que la fórmula de un bajo recall porque encuentra pocos casos positivos.

$$recall = \frac{\#TruePositives}{\#TruePositives + \#FalseNegatives} \tag{2.6}$$

Si es cierto que al predecir queremos ambos fundamentos, es decir, encontrar muchos positivos que sean muy probables de serlo, pero el medio entre ambas se llama The precisión-Recall Trade-Off.

El f1 score combina ambos fundamentos, combina la precisión y la especificidad en una sola métrica. Esta herramienta fue diseñada para trabajar en datos desembalanceados dando lugar a la fórmula 2.7.

$$F1score = \frac{Precision * Recall}{Precision + Recall} \tag{2.7}$$

Esta fórmula le da la misma importancia a ambos factores, es por ello que si se obtiene un alto f1 score es porque ambos parámetros son altos y por el contrario, si el f1 indica un valor bajo es porque la precisión y el recall también lo son. Si da un valor medio, es porque uno es más alto que otro.

Esta métrica de clasificación es buena usarla para datos desequilibrados, si usamos la acurrencia en este caso seria un error pues no distingue entre falsos positivos y negativos. Sin embargo, la precisión y la especificidad tienen en cuenta esto y por tanto puede trabajar con este tipo de datos. Sin embargo, si se tiene un alta precisión pero una baja especificidad se puede obtejer un f1score bajo. Para solventar este problema, existe otra métrica llamada FBetaScore que te permite determinar cuantas veces es más importante la especificidad que la precision.

2.14.3. F1 score aplicado a Python

La función de python correspondiente a la métrica de clasificación de F1 score dispone de varios parámetros que atienden a diferentes factores de funcionamiento de este.

El primero de ellos es el “average”, este determina la manera de realizar la media de los datos pasados a la función[sl21a]. Los valores posibles son:

1. binary: Al usar este párametro solo se devuelve los resultados para una clase específicada en otro parámetro.
2. micro: Calcula la métrica teniendo en cuenta todos los verdaderos positivos, falsos negativos y falsos positivos.
3. Macro:calcula la metrica para cada etiqueta y encuentra su media no ponderada. Ademas, no tiene encuenta el desequilibrio de las etiquetas.
4. samples: Calcula la métrica de cada uno de los datos y hace la media.

Además, se puede determinar si los datos a valorar son binarios o son multiclases.

2.14.4. Matriz de confusión

La matriz de confusión evalúa un modelo y dónde se cometen errores, es decir, [Shi20] determina cuando una clase es confundida con otra lo cual permite trabajar de forma separada con distintos tipos de errores. Es por ello, que muestra mucha más información que una métrica de precisión. Existen dos tipos de errores, el tipo 1, que tiene lugar cuando se da un falso positivo, y el tipo 2, que corresponde a un falso negativo. Si a estas dos circunstancias, se le suma los verdaderos positivos y negativos tiene lugar la matriz de confusión.

A la hora de aplicar la matriz de confusión se puede tunear a través de los hiperpárametros. Lo más importante que se puede determinar sobre que normalizar los resultados(los verdaderos positivos, lo predicho o la popularidad) [sl21b].

2.14.5. Resumen de las métricas de clasificación

Se ha observado que la exactitud no valora realmente el modelo, lo realiza de manera engañosa[Mar21]. Hay otras medidas, como la precisión, el recall y el F1 score, realizan buenas valoraciones tanto con datos balanceados como no. En conclusión de lo explicado, la precisión nos indica la calidad de la predicción, es decir, que proporción de lo que se ha predicho como positivo lo es. El recall nos indica la cantidad de positivos que se han identificado. Como se ha explicado anteriormente, el F1 score combina la precisión y el recall. Y por último, la matriz de confusión nos indica el tipo de errores que se cometen durante las predicciones.

2.15. Personalización de hiperpárametros

La personalización de hiperpárametros, conocido como tuneo, permite que el algoritmo que se esta entrenando sea mejor pues prueba con todos los parámetros que permite la personalización del algoritmo. Esta búsqueda la realiza de manera autómatica y no aleatoria.

Este algoritmo presentando por [Rod18] realiza la búsqueda indicandole el algoritmo que queremos entrenar, los parámetros que queremos optimizar. Para evaluar se realiza validación cruzada, permitiendo tener modelos más estables al evitar el sobreajuste.

Capítulo 3

Estado del Arte

3.1. Análisis de aplicaciones móviles

En esta sección se realizará un análisis de estado del arte sobre trabajos realizados sobre análisis de tráfico de aplicaciones móviles.

3.1.1. Características del tráfico de datos generado por dispositivos móviles

A lo largo de los años se han utilizado diferentes tipos de redes, las cuales han traído funcionalidades novedosas en consonancia con los avances tecnológicos del paso de los años. Actualmente,la mayor parte de la información que se comparte en Internet a través de los dispositivos móviles está asociada a páginas web, plataformas de y redes sociales. Por esta razón, el tráfico de red móvil corresponde mayoritariamente al protocolo HTTP, del cual se hablará más en detalle a continuación *streaming*.

En el estudio [Zr12] se consideran tres categorías en las que se clasifican las características: estadísticas a nivel de paquete, de nivel de flujo y de sesión.

Para observar las diferencias de tráfico de datos wireline y wireless se recolectaron dos datasets: *wireless dataset*, con 186 horas de tráfico con 3.9 millones de sesiones de 50.000 direcciones IP; *wireline dataset*, con 31 horas de tráfico con 19 millones de sesiones de 21.000 direcciones IP.

Tras analizar los resultados, se concluye que para el 70 % de las aplicaciones el tráfico de datos *wireline* es de mayor tamaño que el *wireless*; alrededor del 15 % de las aplicaciones generan la misma cantidad de datos; y, el 15 % restante ocurre al revés. el tráfico de datos *wireless* es de mayor tamaño que el *wireline*.

A nivel de nivel de tráfico, los proveedores de servicios también pueden optimizar las características del tráfico de datos para móviles. Analizando el tiempo entre llegada de paquetes los autores concluyeron que el tráfico de datos en una red móvil tiende a tener unos paquetes cercanos entre si y otros alejados en el espacio temporal. Respecto al tamaño de los paquetes, los paquetes *wireline* suelen ser más grandes que los paquetes *wireless* y para estos últimos los tamaños suelen ser parecidos.

Analizando el volumen de tráfico y los tiempos de acceso a diferentes aplicaciones en diferentes horas del día. En general, los usuarios son más activos durante el día, pero en las

conexiones *wireline* se observa menos actividad durante la noche respecto a las conexiones *wireless*. Esto posiblemente se asocie a que los dispositivos *wireline* son más usados para trabajar, mientras que los dispositivos móviles se usan fuera del trabajo.

Para identificar las diferencias entre aplicaciones con tráfico de datos *wireless* primero se clasificaron las aplicaciones en diferentes grupos. El objetivo es averiguar si esos grupos tenían requisitos distintos.

Las aplicaciones se agruparon en tres categorías y para cada grupo de aplicaciones se calculó el tiempo medio de llegada entre paquetes del mismo flujo, que llamaremos *inter-packet gap* , indicando la desviación estándar dentro de cada grupo con barras verticales.

En general, se observa que el siguiente paquete de un flujo llega un par de segundos después que su predecesor, pero esto varía en función de la aplicación, como es el caso de Facebook y Apple. Así pues, para Apple un período de silencio de menos de 10 segundos significa que el flujo ha terminado con un 90 % de probabilidad, mientras que para Facebook la probabilidad es de alrededor del 50 %.

A pesar de que algunas aplicaciones pertenezcan a la misma categoría estas seguramente estarán optimizadas e implementadas de forma diferente. Para analizar las diferencias, se aplicó el algoritmo de *clustering K-means* a un subconjunto de las aplicaciones para las cuales se realizó un análisis de componentes principales (ACP) con el objetivo de analizar el impacto de diferentes variables en los clusters.

Los resultados para las 10 aplicaciones más populares muestran que la primera componente está dominada por el tamaño del flujo, el tamaño de la sesión y el *inter-packet gap*, mientras que la segunda componente refleja diferentes tamaños de paquete. Facebook y Tuenti están en diferentes clusters porque sus tamaños de flujo y sesiones son diferentes, así como Tuenti y Twitter están en clusters separados por la misma razón.

Respecto al número medio de flujos en sesiones del mismo tamaño en Twitter y Tuenti se concluye que el número de flujos tiende a crecer con el tamaño de la sesión en Twitter, mientras que en Tuenti no parece existir tal relación. Para una sesión del mismo tamaño, la duración de la misma tiende a ser mayor en sesiones cortas debido a que produce tiempos de espera mayores, aunque esto puede depender de la implementación de la aplicación.

3.1.2. Análisis de aplicaciones de redes sociales usando Autoencoder

Un autocodificador,en concreto, es un software de tipo de red neuronal artificial que se utiliza para aprender sobre datos no etiquetados [RU20]. La codificación se válida y ajusta al intentar regenerar la entrada a partir de la codificación. Autoencoder es usado para extraer las características del paquete. Estas fueron agrupadas usando K-means. Los aspectos conductuales considerados fueron inter-arrival-time, longitud del paquete, numero de paquetes ACK observados y numero de retransmisiones, tiempo de ida y vuelta y la varianza.

3.1.2.1. Resultado y análisis

Se observó que la longitud de los paquetes y la duración de los flujos es menor en los paquetes de TCP que en los de UDP. Analizando las direcciones IP de destino, se vio claramente que los paquetes TCP están diseñado para varios servidores. Pero los paquetes

de UDP están diseñados para un set concreto de IPs. De este análisis se sacó en claro que los paquetes TCP se usan para establecer la conexión entre los dispositivos y el servidor del destino y los paquetes UDP se emplean para el intercambio de media.

Después de la recolección de datos se ha encontrado 84 atributos. Algunas características fueron descartadas atendiendo a su importancia y correlaciones usando el heat-map plotting.

El objetivo fue clasificar los paquetes de datos encriptados dependiendo del tipo de archivos. Inicialmente, las 16 características fueron aportadas a través del algoritmo K-means. Posteriormente, se introdujo nuevas dimensiones de esas 16 características que aporto el PCA.

Los paquetes a analizar de una aplicación fueron capturados usando el Wireshark y el port mirroning. Los datos intercambiados fueron archivos de texto, imágenes y videos. Para la clasificación de paquetes se utilizo el algoritmo de K-means y pero los resultados fueron pobres. Posteriormente, se empleo el PCA. Se introdujo esas características en el PCA y el Autoencoder de manera separada. Se estudio la reducción que aportaban ambos aplicando el algoritmo Kmeans. Se observo que Autoencoder trabaja menor que PCA.

3.1.3. Identificación de la firma y análisis de la actividad de aplicaciones móviles a través del análisis de datos

El Whatsapp utiliza el protocolo SSL para su comunicación. Los datos se monitorizan a través del puerto del WIFI y luego es capturado[RS21]. Tras ser capturado, se guarda como un PCAP file a través del whireshark. El PCAP contiene datos de todas las aplicaciones de la red y luego se filtra los datos de la aplicación a través de la cabecera SSL. Posteriormente, se analiza la red de tráfico. Para identificar las firmas, se ha creado un algoritmo que clasifica las confirmaciones de lectura. La metodología,que se muestra en la figura 3.1, sigue los siguientes pasos:

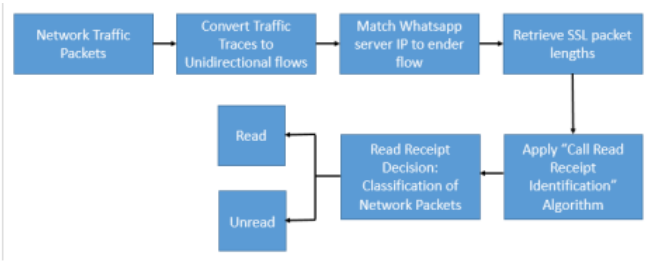

Figura 3.1: Metodología aplicada para identificar firmas la aplicación

1. *Creación de flujo unidireccional.* Dado que la firma se obtiene de una dirección de la comunicación , los rastros de tráfico se convierten a flujos unidireccionales. Un flujo es una tupla de 5 que consta de IP de origen, puerto de origen,IP de destino, puerto de destino y protocolo de transporte.

2. *Recuperación de longitud SSL.*Muchos flujos unidireccionales se crean desde cada IP de origen a cada IP de destino. De este conjunto de flujos, solo el servidor de la aplicación (IP de origen) al flujo del remitente (IP de destino) se extrae cuidadosamente como el la firma se encuentra solo en este flujo.

3. *Extracción de la aplicación Server to Sender Flow.* flujos unidireccionales constan de varios encabezados información. Sin embargo, solo las longitudes de SSL se recuperan del flujo porque el firma identificada en base a ella.

4. *Implementación del algoritmo.* El propósito de este algoritmo es verificar si las longitudes de SSL recuperadas se agrupan en una matriz que se alimenta como entrada al algoritmo.

3.1.3.1. Colección de tráfico

El principal requerimiento es que los datos recogidos deberán ser generados sin ninguna interfaz de otra aplicación con el fin de evitar ruido. Además, se debe confirmar que WhatsApp únicamente esta corriendo en la maquina host. La versión de WhatsApp utilizada es v0.3.2390. La comunicación debe ser únicamente entre dos dispositivos.

3.1.3.2. Análisis del resultado

1. *Transferencia de media* . Usa TLSv1.2 y TLSv1.3. Posteriores inspecciones de tráfico usando protocolos TLSv1.3 que revela una firma especifica a través de la longitud del SSL. El patrón identificado siempre empieza con la longitud del paquete SSL de 69, que ocurre entre el quinto y el decimotercer paquete del client hello Packet de la aplicación. Esta longitud puede estar seguida de las longitudes 26 y/o 30. El patrón se muestra en la ilustración **??**.

2. *Intercambio de comunicaciones.* Utiliza TLSv1.2, pero no se haya un patrón.

3. *Status.*Usa TLSv.1.3 durante la visualización de status y sigue el mismo patrón que la transferencia de media.

4. *Localización.* Usa el GQUIC protocolo. Como la localización emplea Google maps. La localización se transmite a través de los servicios de Google usando la IP.

La aplicación se encripta con SLL, el payload de la aplicación no revela ninguna firma, pero después de analizar la longitud de los TLS record layer se ven firmas. Solo se observan firmas en el tráfico que utiliza TLSv.1.2.

3.2. MAppGraph

MAppGraph es una técnica novedosa para la clasificación de aplicaciones móviles para Android que aborda:

- El tráfico encriptado.
- El cambio dinámicamente del comportamiento del usuario.
- El hecho de que las aplicaciones modernas se diseñan y desarrollan utilizando microservicios que comparten las mismas librerías y utilizan redes de distribución contenido y servicios de terceros (ej de servicios compartidos: análisis de fallos, redes publicitarias móvil, redes sociales) lo que hace que sus transmisiones de sean similares y difíciles de reconocer.

MAppGraph construye un grafo de comunicación cuyos nodos están definidos por tuplas de tipo (*Internet Protocol* (IP)) y aristas(la correlación entre nodos) y extrae la información de los paquetes [DP21].

3.2.1. Background

MAppGraph recopila el tráfico de red de aplicaciones móviles en diferentes momentos, que resulta en una gran cantidad de tráfico de red que hay que procesar para cada aplicación. Cada tramo de tráfico forma un grafo con su respectiva tupla. El enfoque utilizado es la correlación cruzada para establecer las aristas entre los nodos y calcular su peso. La información se extrae de las cabeceras de los paquetes y derivar características estadísticas que se utilizan como atributos de los nodos del grafo para representar el comportamiento del tráfico de la comunicación entre la aplicación y un servicio. Una aplicación tendrá un conjunto de grafos, cada uno representando el comportamiento del tráfico en diferentes momentos. Las redes neuronales de convolución de grafos profundos tienen la capacidad de aprender comportamientos complejos y extraer características de alto nivel de big data para crear el fingerprint de los datos analizados. Se puede utilizar para problemas de aprendizaje supervisados y no supervisados.[DP21]

Se compara el rendimiento con técnicas recientes, específicamente AppScanner y FlowPrint.

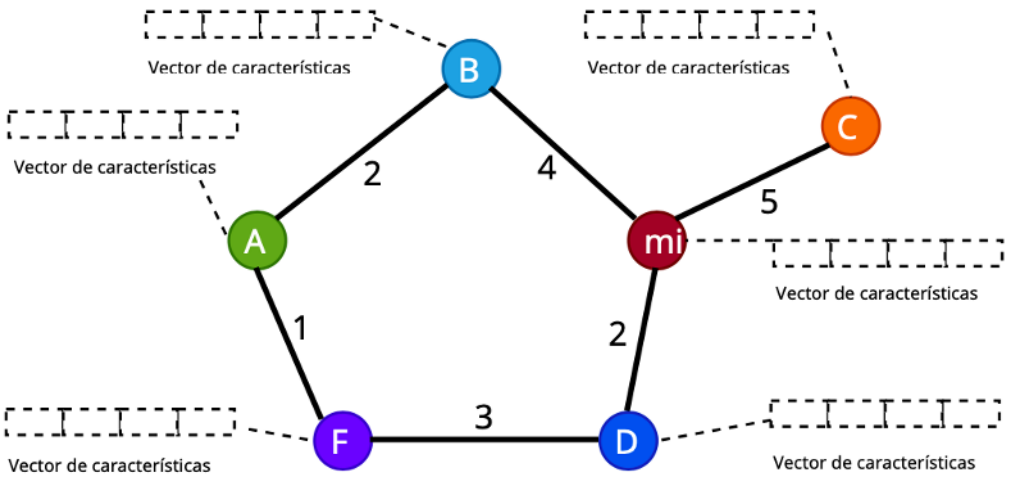

Figura 3.2: Grafo estructurado de datos

Hay tres etapas secuenciales en el proceso de análisis de grafos:

- Primero, un grafo dado se pasa a través de las capas de convolución del grafo que extraen características locales y definen un orden consistente de vértices.
- En segundo lugar, las capas de agrupación procesarán la salida de la primera etapa y construirán una representación vectorial de todo el grafo con un orden y tamaño predefinidos.
- En tercer lugar, emplean capas tradicionales convolucionales o densas

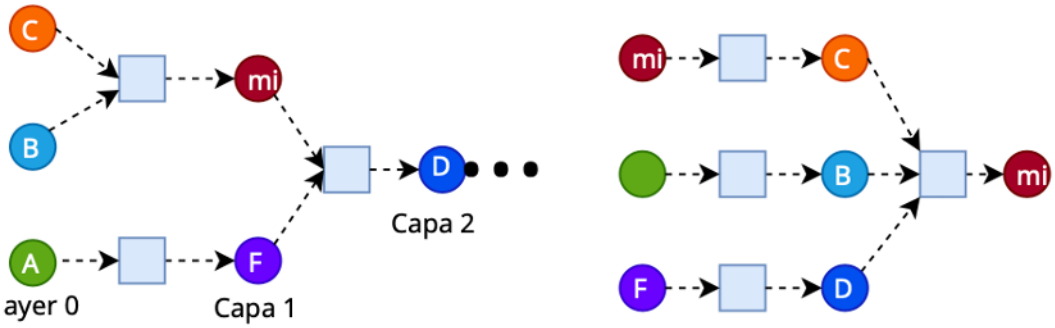

Figura 3.3: Grafo estructurado de datos

(completamente conectadas) que toman la salida de la segunda etapa y hacen la predicción.

3.2.2. Colección de tráfico de red

Para preparar el conjunto de datos de entrenamiento para el clasificador, es necesario recopilar y procesar una gran cantidad de tráfico. Esta tarea se puede realizar sin conexión, pero la fase de inferencia debe realizarse en tiempo real para reaccionar ante el comportamiento anormal de las aplicaciones móviles.

3.2.3. Construcción de grafos de comportamiento del tráfico

Un nodo del grafo se define mediante una dirección IP de destino. Los servicios de terceros generalmente se implementan en múltiples servidores (con diferentes direcciones IP) para proporcionar equilibrio de carga. Dado el tráfico capturado en una ventana de tiempo que da como resultado la lista de nodos, la ventana de tiempo se divide en varios segmentos con una duración de segmento predefinida. Durante cada segmento, se considera que un nodo esta activo si la aplicación móvil ha enviado o recibido al menos un paquete de tráfico al servicio implementado en la dirección IP de destino. La correlación cruzada entre dos nodos es alta si tienen mucha actividad en los mismos intervalos de tiempo. De lo contrario, es baja o incluso nula si no hay correlación entre ellos. Para evitar el sesgo de características al alimentar grafos a para entrenamiento y predicción, se normaliza la correlación cruzada al rango [0, 1] utilizando el *min-max scaler*.[DP21]

3.2.4. Extracción de características del nodo

Las características se extraen solo de las cabeceras de los paquetes sin analizar las cargas útiles de los paquetes, considerando únicamente las características de flujo.

Las características se clasifican en 4 categorías:

- **Funciones agregadas**: las funciones calculadas como suma, recuento total, valores máximo y mínimo y los números de de flujos.

- **Características temporales**: relacionadas con el tiempo de los paquetes y flujos.
- **Características estadísticas**: relacionadas con las características mátematicas y estadísticas de los paquetes y flujos.
- **Funciones categóricas**: como el protocolo de transmisión y la dirección IP, deben factorizarse antes de introducirse en el modelo.

3.2.5. Análisis de los resultados

Se han realizado muchos experimentos para comparar el rendimiento como el impacto del número de nodos de grafos utilizados para entrenar modelos de y de la duración de la ventana de tiempo de la recopilación de tráfico para la construcción de grafos mostrado en la figura, la duración del segmento en la correlación cruzada en la construcción de grafos, el rendimiento con y sin el uso de direcciones IP en vectores de características, la clasificación de aplicaciones móviles con funcionalidades similares y el rendimiento con diferente número de aplicaciones.

Obtiene mejores resultados que las aplicaciones semejantes a ella como son AppScanner, FlowPrint y MLP tanto en precision, recall, F1-score y tasa de aciertos logrando unos resultados en todas ellas superiores al 93 %. Tambien se observa que se trata de una aplicación que no requiere de un gran numero de nodos para ser capaz de obtener resultados lo cual facilita su implementación.

Por lo tanto hemos concluido que **MAppGraph** es una técnica novedosa para la clasificación de aplicaciones móviles que puede lidiar con el tráfico, comportamiento de comunicación dinámica y naturaleza de implementación de aplicaciones móviles.

3.2.6. Análisis código

3.2.6.1. Función para generar muestras de cada aplicaión

El notebook crea tramos de tráfico móvil que tienen la misma longitud. Esto es posible mediante la siguiente función que recorre el conjunto de los hiperparámetros. Los hiperparámetros que se pasan como parámetros de entrada están formados por 5 tuplas. Por cada tupla se crea una carpeta con el nombre de los argumentos (en caso de que no exista ya) y dentro se crea otra carpeta que va a contener las muestras. Cada aplicación tiene su correspondiente carpeta con los ficheros de muestra.

3.2.6.2. Función de entrenamiento

Los datos se dividen en entrenamiento y prueba. De las muestras generadas anteriormente el 80 % está constituido por el conjunto de entrenamiento y el 20 % restante por el conjunto de prueba. El porcentaje que corresponde al conjunto de entrenamiento indicado previamente. Las muestras de entrenamiento y de prueba se guardan en ficheros de tipo *json*.

3.2.6.3. Función generación grafos

Realiza el preprocesamiento de los datos, la extracción de características (del flujo y de los paquetes), la generación de grafos partiendo de los tramos de tráfico generados anteriormente, estandarización de las características y por último se guardan los grafos.

Un nodo del grafo está definido mediante una tupla con la información que utilizamos al conectarnos con la aplicación. El preprocesamiento elimina los números de puerto correspondientes al protocolo DNS, ya que varios servicios pueden tener la misma dirección IP pero con diferentes números de puerto.

Generar pesos

Para la asignación de pesos a la aristas del grafo se toman en cuenta tanto las características de paquete como las del flujo. Si un nodo está inactivo se elimina porque significa que no tiene conexión con ningún otro nodo.

Generar grafos

Para cada aplicación se generan 2 ficheros: uno para las características y otro para los pesos utilizando las funciones anteriores, cada uno teniendo una columna que indica el id del grafo.

Normalización

Para evitar el sesgo de características al alimentar grafos para entrenamiento y predicción y para que el algoritmo de *Machine Learning* (ML) funcione mejor, hay que normalizar los valores de las características a un rango definido.

Guardar grafos Cada conjunto de parámetros se produce un conjunto diferente de grafos. Se van a generar y guardar grafos para entrenamiento y prueba para cada una de las aplicaciones y con cada una de la combinación posible de los parámetros de entrada.

3.2.6.4. Función entrenamiento redes convolucionales

Para generar el modelo y encontrar los hiperparámetros más adecuados se utilizan distintos valores para los parámetros del GCNN y se utiliza el optimizador Adam, una extensión del descenso de gradiente estocástico, para encontrar la mejor combinación de parámetros [Bru]. Una vez seleccionado los hiperparámetros, se entrena el modelo. Por último, se evalúa el modelo con el conjunto de prueba utilizando las siguientes métricas de rendimiento: Precision, Recall, F1-Score y Accuracy.

3.3. AppScanner

Esta aplicación, es usada para hallar aplicaciones en la red. Este algoritmo debe ser entrenado para que funcione de manera correcta.Esta aplicación tiene un nivel alto de acierto y utiliza el Random Forest. Esta aplicación tiene una serie de opciones, entre ellas, cargar datos para trabajar con ellos o salvar el resultado en un fichero. La aplicación esta gestionada a través de módulos:

1. Reader. Encargado de leer los paquetes del .pcap file y extraer los campos más relevantes de estos.

2. Flow. Extraer los flows en bursts, entendido como un periodo de inactividad basado en un umbral.Es decir, se encarga de extraer los flujos de información en cada periodo de inactividad basado en el umbral pasado como parámetro al programa.

3. Preprocessor. Encargado de procesar los datos, es decir, de extraer las características de cada paquete y sus etiquetas.

4. Burst. Calcula los bursts.

5. AppScanner. Módulo encargado de entrenar y predecir el modelo siguiendo el algoritmo Random Forest.

3.3.1. Funcionamiento general

El funcionamiento de la aplicación se gestiona desde el main. El flujo de la aplicación en términos generales es el siguiente como ilustra la figura 3.4:

1. Extracción del nombre del directorio.

2. Extraer los paquetes del fichero y sus etiquetas.

3. Cargar los datos en el preprocesador si es necesario.

4. Salvar los datos en el preprocesador si es necesario.

5. Clasificar los datos y entrenar el modelo, en concreto,el módulo de AppScanner.

6. Se ajusta el modelo.

7. Muestra los resultados.

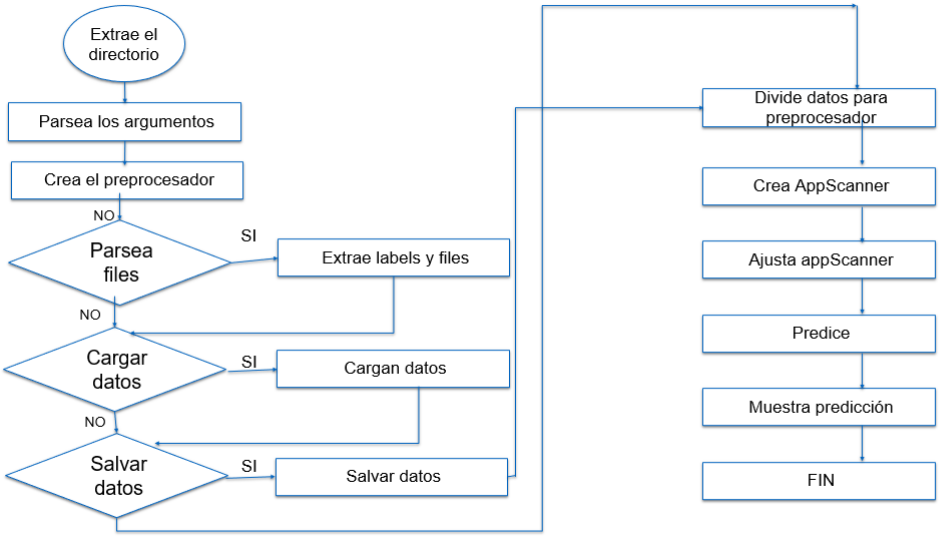

Figura 3.4: Funcionamiento general de AppScanner

A continuación, se explicara detalladamente cada módulo.

3.3.2. Módulo Reader

Es un módulo sencillo que se encarga de leer los paquetes TCP de entrada y extraer los campos relevantes de un paquete en concreto. Posteriormente se añade a la variable global de los paquetes.Se encuentra desarrollado en el archivo *reader.py*. Para ello, se desarrollaron una serie de funciones:

1. *Init(self,verbose)*.Crea e inicializa el objeto Reader. El argumento verbose es un booleano, que cuando es true imprime por pantalla que ficheros van a ser leídos.
2. *Read(self,infile)*.Se encarga de leer los paquetes TCP del fichero de entrada, que se pasa por el parámetro infile.
3. *Extract(self,packet)*.Extrae los campos relevantes de un paquete en concreto y se añade a la variable global paquetes.

3.3.3. Módulo Burst

Este módulo tiene como finalidad definir los diferentes bursts, entendido como periodos de inactividad. Para dividirlos se basa en un umbral, pasado como parámetro. Este módulo tiene únicamente una función:

1. *split(self,packets,threshold)*. El argumento packets son los paquetes que se quiere dividir, acorde con los periodos de inactividad determinados por el umbral, pasado por el parámetro threshold. El flujo de esta función,como se muestra en la figura 3.5, se basa en ir revisando los paquetes e ir viendo la diferencia entre los paquetes y los umbrales, luego estos se devuelven en una variable. La diferencia entre paquetes se calcula con la función diff del módulo difflib.

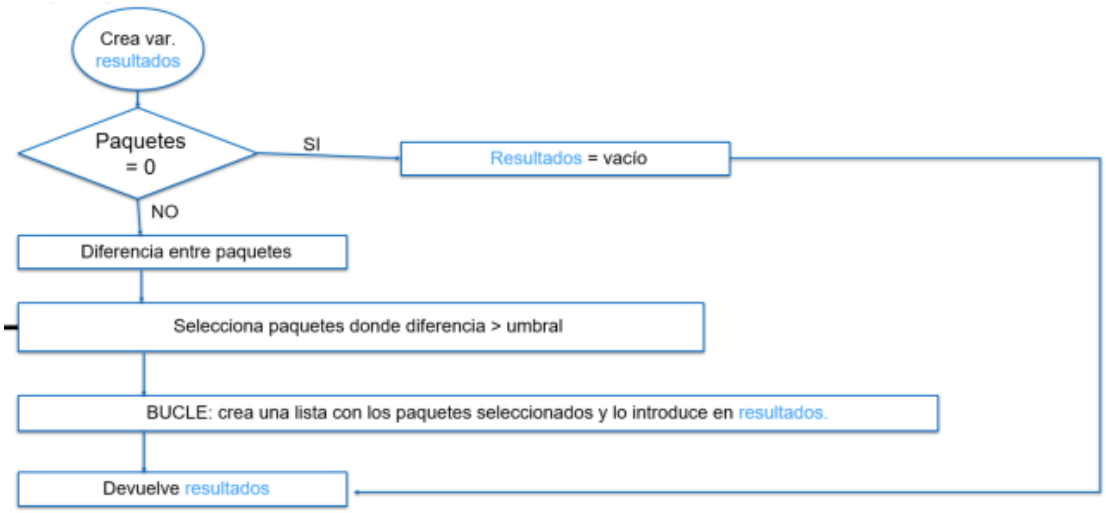

Figura 3.5: Flujo de la función split del módulo Burst

3.3.4. Módulo Flow

Encargada de extraer los flujos de información de acuerdo a los burst definidos.

1. *extract(self,burst)*. Extrae los flows de todos los paquetes de acuerdo a los burst. Esta función va extrayendo las ráfagas pasándole el correspondiente.

2. *single_extract(burst)*. Extrae un flow.

3. *key(self,timestramp,packets)*. Extrae la key del paquete y comprueba si es de entrada o salida. La key esta definida por el origen y destino del paquete. Esta función devuelve un parámetro con la estampa de tiempo y la demás información del paquete en forma de dirección IP.

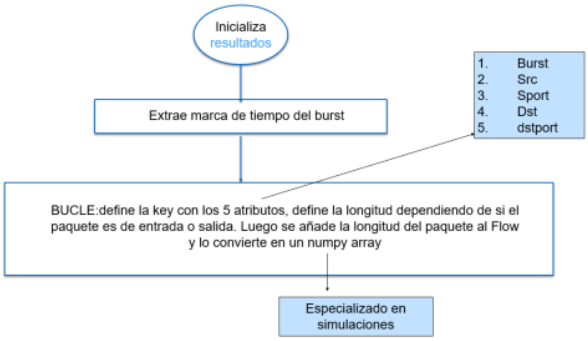

Figura 3.6: Flujo del módulo Flow

3.3.5. Módulo features

Módulo encargado de extraer las características de un paquete.Se determinan las siguientes características estadísticas y para ello dispone de varias funciones:

1. *extract(self,flows)*. Extrae las características de todos los flows que se le pasan como parámetro. Esta función va llamando a la función encargada de ir extrayendo todas las características.

2. *single_extract(flow)*.Extrae las características de cada flow de acuerdo a combinando la entrada,salida y combinación de ambas.

3. *features(self,array)*.Devuelve una lista de características.

3.3.6. Módulo preprocessor

Es uno de los módulos más importantes. Se encarga de procesar las características de los flows acorde a las rafagas y sus etiquetas. Para ello, llama a funciones de estos módulos. Además, se encarga de guardar o cargar los datos al programa, en caso de ser necesario. Para realizar todo lo explicado anteriormente posee una serie de funciones:

1. *init(self,verbose)*. Esta función se encarga de iniciar el preprocesador y para lograrlo tambien el reader, el burst, flows y features. El parámetro *verbose*, que es un booleano, cuando esta a true imprime por pantalla que ficheros se están leyendo.

2. *process(self,files,labels)*. Esta función devuelve las características extraídas de cada fichero y las etiquetas correspondientes. El parámetro *files* corresponde con el path de los datos extraidos y label con sus etiquetas. La lógica de la función se observa en la figura 3.7.

3. *features(self,array)*. Devuelve una lista de características. Coge los datos como una serie panda (estructura más simple que proporciona los datos como columnas de Excel).

4. *extract(self,files)*. Extrae los flows de cada *pcap file* y devuelve una Flow tupla. La lógica de la función se observa en la figura 3.8.

5. *save(self,outfile,x,y)*. Guarda los datos en un fichero.

6. *load(self,infile,x,y)*. Carga los datos en un fichero.

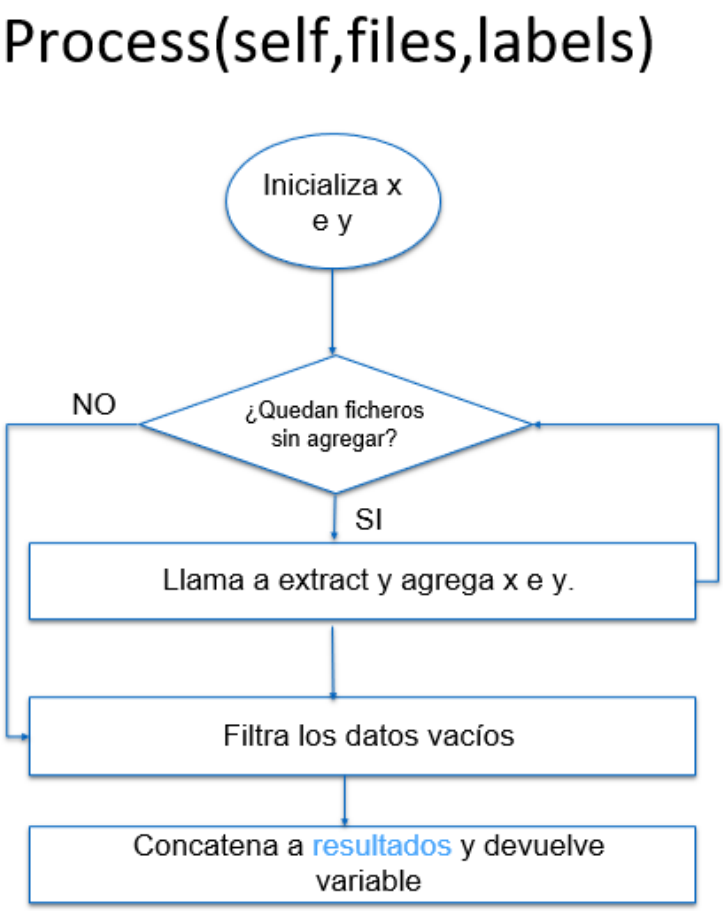

Figura 3.7: Flujo de la función process del módulo Preprocessor

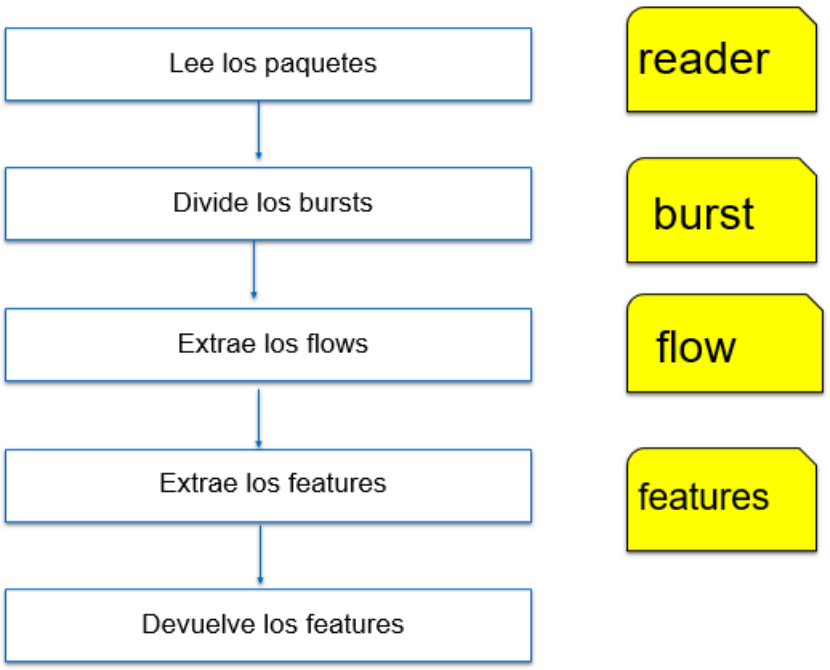

Figura 3.8: Flujo de la función extract del módulo Preprocessor

3.3.7. Módulo AppScanner

Este módulo es el encargado de entrenar y predecir el modelo. Para ello utiliza el modelo Random Forest, que es un meta estimador que ajusta el numero de clasificaciones de árboles de decisión de varios submuestras del dataset y usa el promedio para mejorar la agudeza de la predicción y controlar el sobreajuste. Para realizar la función de este módulo se implementa una serie de funciones:

1. *init(self)*. Define el umbral y crea el clasificador con Random Forest de la biblioteca. *init(sklearn.ensemble)*. A la hora de determinar el clasificador con la función. RandomForestClassifier(criterion, max_features, n_estimators) pasa una serie de argumentos[sl] :
   a) Criterion. Esta variable se encarga de determinar la calidad de la división.En este caso, se le pasa el valor gini, que indica que el criterio a seguir es la impureza del Gini.
   b) max_features. Hace referencia al numero máximo de características a considerar cuando se busca realizar la división más óptima. La búsqueda para división no finaliza hasta que se encuentra una partición de nodos valida, aunque se supere el valor de esta variable. En este caso, el valor de este parámetro es sqrt(n_features),es decir, la raíz cuadrada de las características existentes.
   c) n_estimators. Este parámetro hace referencia al numero de estimadores,es decir,al numero de arboles. En este caso, son 150.
2. *fit(self,x,y)*. X se entiende como los datos para ajustar el modelo e y las etiquetas que corresponden con las x. Este llama a fit de su clasificador y se devuelve a si mismo

para el fit_predict del mismo. La función *fit(X,Y)* del clasificador Random Forest construye arboles para el conjunto de entrenamiento, pasándole como argumentos los datos de entrenamiento y los valores objetivos.

3. *fit_predict(self,x,y)*. Devuelve las predicciones para las etiquetas de x.

4. *predict(self,x,y)*.Predice la clase de x para el modelo entrenado. Devuelve las etiquetas predecidas para x. En esta función se hace empleo de la función *predict_proba(X)* en la que se calcula las probabilidades de las clases predecidas del conjunto *X*, parámetro que corresponde con la muestra, como la media de las probabilidades de las clases predecidas de los arboles del bosque. Y posteriormente, utiliza la función *predict(X)* del Random Forest para predecir la clase que se pasa como parámetro,*X*.

En términos generales, esta aplicación emplea una serie de módulos, cada uno encargado de una función muy especifica, que colaboran todos para llegar al resultado deseado. Como se puede ver en el diagrama 3.9 la función *preprocessor.process(files,labels)* para hacer el procesamiento de los datos pasados por el argumento file, emplea una serie de funciones de los demás módulos. En esta función, se encarga de leer el archivo y extraer los paquetes (para ello utiliza el módulo Reader), calcula los burst usando el módulo correspondiente para ello, extrae los flows y posteriormente, las características estadísticas. Como se puede ver el funcionamiento es sencillo pero muy importante ya que el procesamiento de datos si se realiza mal, la predicción no es útil. Posteriormente, se entrena y se ajusta el modelo y se predicen resultados.Todo esto se realiza usando el módulo AppScanner que hace uso de Random Forest. Por ultimo, se muestran los resultados.

3.4. Resultados experimentales de los trabajos analizados

Tras analizar todos los resultados que muestran los papers, se observa que el modelo con mayor precisión es DCGNN con un valor de 98 %, y posteriormente AppScanner con un valor de 90 %. Poniendo esto junto al Recall que tiene cada una (DCGNN arroja un valor de entorno al 95 % y AppScanner de 90 %), arrojan unos valores de la métrica F1 score elevada, en concreto, del 95 % y del 90 %, respectivamente.

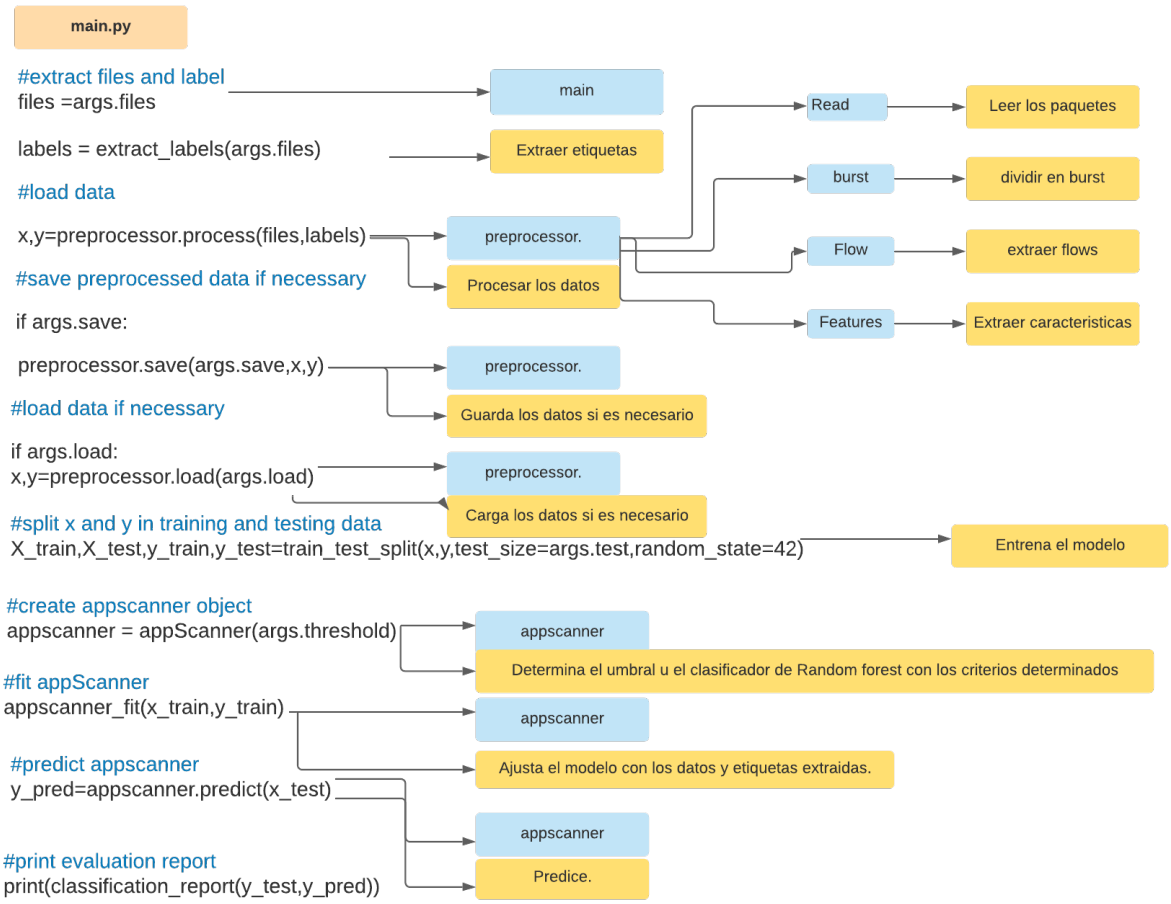

Figura 3.9: Funcionamiento general de AppScanner

Capítulo 4

Capítulo de Contribución

4.1. Introducción

A lo largo de este capítulo, se presentaran las contribuciones que se han realizado al proyecto. El objetivo de este es analizar paquetes de la red y conseguir determinar a que aplicación pertenecen a través de Machine Learning. Para ello utilizaremos tres modulos:

1. Preprocesador. El objetivo de este será: limpiar los datos de ruido, clasificarlos por flujos y obtener sus características estadistícas. Además, se obtendran datos más especifícos como la direccion IP o si es un paquete de origen o destino, entre otras.
2. Generación de gráfos. Este módulo se encargara de generar gráfos, tanto de testeo como de entreno, con la información obtenida en el módulo anterior, el preprocesador.
3. Clasificador. La finalidad de esta parte, es entrenar un modelo con diferentes algoritmos y comprobar los resultados obtenidos a través de diferentes métricas de clasificación.

4.2. Terminología

Para realizar el proyecto, nos hemos centrado en las características que tienen los paquetes y obtenerlo a través la herramienta tShark.

El primer término importante a entender es flujo. Un flujo de datos está compuesto por una serie de paquetes relacionados entre sí, en nuestro caso, serán paquetes cuya comunicación es peer to peer, es decir, entre las dos mismas personas. Como se explicará más adelante, para identificar a que flujo pertenece cada paquete se utilizará el atributo ”stream”.

4.3. Módulo de preprocesador

El objetivo del módulo mencionado es analizar los archivos y clasificarlos por aplicación con el fin de extraer las características de los paquetes siguiendo el pseudocódigo 1.

algpseudocode

Algoritmo 1: Pseudocódigo de la lógica general de módulo Preprocesador

while *i ≠fin de archivos* **do**
    Combinar archivos agrupado por aplicaciones
    Convertir .PCAP a .csv
    Extraer las características
    Calcular los hiperparámetros

En este módulo se realiza el primer paso del Machine Learning, es decir, el preprocesamiento de datos. A continuación se detallara como se ha realizado el proceso.

Primero, se combinan todos los archivos pcap de una misma aplicación a uno solo utilizando el comando *mergecap*. Luego, se realiza una conversión del archivo pcap a csv. De este modo se puede extraer las características de los paquetes a través de la ejecucion de una serie de comandos de tShark, volcando la información conseguida en un DataFrame. Luego se realiza un análisis de cada paquete para identificar el flujo y se refleja dicha información en el dataframe anteriormente mencionado. Para analizar los paquetes de entrada y salida, se hace uso de una función del módulo *ipaddress* que nos indica si una dirección es privada o no. Si es privada significa que el paquete es de entrada. Una vez identificados los paquetes de entrada y salida se realiza el cálculo de las características necesarias.

Tras eso, se clasifican los paquetes dependiendo del flujo al que pertenezcan. Para ello, se utilizan una serie de características de los paquetes como el protocolo al que pertenecen, el stream y el tiempo. A la hora de implementar este método hay que utilizar la característica stream de cada uno de ellos. Es decir, en la situación que se presenta en la Figura 4.1, se observa que se tienen distintos paquetes de distintos protocolos y con distintos valores de índice de flujo. Se realiza la clasificación en función del valor de este último. Esta función realiza el análisis para los protocolos TCP y UDP de manera independiente aplicando la misma lógica como se puede observar en la Figura 4.1. Primero, separamos los paquetes por protocolos y se ordenan por el stream. Una vez realizado eso, se calcula el tiempo de cada flujo haciendo una diferencia entre el mayor y el menor de cada paquete con mismo índice stream. Para calcular el tamaño total de cada flujo se realiza una suma de los tamaños de los paquetes con el mismo índice de flujo.

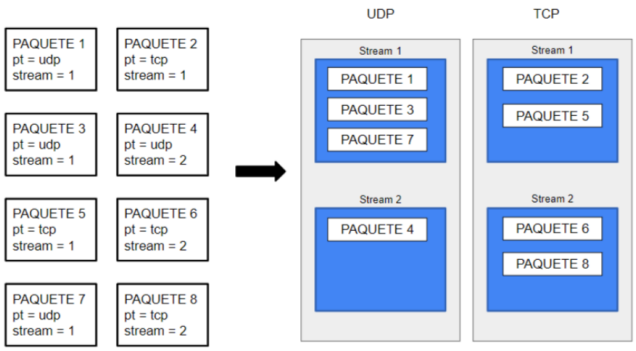

Figura 4.1: Clasificación de paquetes en función del flujo al que pertenecen

Una vez entendido como se clasifican los paquetes en flujos y con los paquetes divididos en paquetes de entrada y salida, se preprocesan todos los paquetes para calcular las

características, Para ello, se ha implementado la función explicada en el pseudocódigo 2.

Algoritmo 2: Pseudocódigo del análisis de flujo

```
UDP ←paquetes protocolo UDP
TCP ←paquetes protocolo TCP
while i≤ fin UDP do
    b ←paquetes UDP que pertenece al flujo analizado
    b1 ←menor tiempo de b
    b2 ←mayor tiempo de b
    Actualiza información de los flujos.
Calcula las características de los flujos.
```

Posteriormente, se calculan las características de los flujos, de los paquetes de entrada y de salida. Y por último, se divide el archivo de salida por hiperparámetros.

4.4. Trainer

El objetivo principal del trainer es obtener la muestra entrenada para posteriormente generar los gráfos.

Algoritmo 3: Pseudocódigo que presenta la lógica general del módulo Trainer

```
archivo sin repetición ←muestra de archivos con funcion set y sample
archivos de entrenamiento ←archivos que pertenecen a archivos sin repeticion
archivos de testeo ←archivos que no se encuentra en el archivo de entrenamiento
```

Para ello, primero se obtiene la ruta de las samples. Este dato es utilizado para entrenar los datos dependiendo de la aplicación. Primero, se obtiene una muestra de los archivos sin repetición. Posteriormente, se le resta al conjunto de archivos totales estos, con el fin de eliminar ruido y se define el conjunto de archivos para entrenamiento. Una vez obtenido este conjunto, se traslada a un archivo JSON.

4.5. Generacion de gráfos

En términos generales, este módulo genera, para unos hiperparámetros dados, los gráfos de entreno y los de testeo. Para crearlos se sigue una serie de pasos:

1. Se obtiene la ruta del archivo que contiene la información de entrenamiento y testeo.
2. Se obtiene la ruta del directorio de los archivos correspondientes, conocidos como samples.
3. Se recorre todo el directorio para cada app.
4. Se generan los gráfos para cada app.

4.5.1. Generación de gráfos por app

Este proceso se encuentra implementado en la clasificación. Para ello, se realiza un bucle que realiza lo siguiente:

1. Se genera la ruta de cada archivo por aplicación uniendo el path de la misma con el nombre del archivo.
2. Se devuelven estos datos a un dataframe que se ordena por tiempo ascendente.
3. Se define el tiempo base cogiendo el primer valor de tiempo presente, dado a que esta ordenado de manera ascendente.
4. Se generan los pesos, como se explica en el punto 2.
5. Se añade este gráfo al conjunto de todos los gráfos.

4.6. Generación de pesos

En términos generales, esta función preprocesa una serie de datos, en concreto, los archivos y los paquetes. Posteriormente, se extraen las características de ambos y se combinan para omitir las relaciones débiles. Se realiza lo mismo con los flujos. Una vez realizado eso se generan los pesos y se elimina el ruido. Además, como parte del preprocesamiento de datos, estos se normalizan y se estandarizan. Para que el algoritmo funcione de manera correcta, todos los datos deben estar dentro del mismo rango de valores, es decir, de 0 a 1. La razón de realizar la normalización es que, aparte de que los datos se modelen de manera correcta, todos los datos esten en esa escala y para ello, la normalización modifica los valores numericos que se salen de esa escala para que esten dentro. La normalización es un aspecto muy relevante en ciertos algoritmos como la regresión logistica, las redes neuronales y los algoritmos basados en distancia. La normalización optimiza y mejora el funcionamiento de estos algoritmos. Primero se

Algoritmo 4: Pseudocódigo que presenta la lógica general de la generación de los pesos

```
preprocesa los archivos
preprocesa los paquetes
extrae las características de todos los paquetes en términos generales
extrae las características de los paquets de entrada
extrae las características de los paquets de salida
realiza merge sobre los paquetes
preprocesa los flujos
extrae las características de los flujos
realiza merge sobre los flujos
generaliza pesos
normaliza
estandariza
```

preprocesan los archivos todas las características del archivo. Posteriormente, se realiza un preprocesamiento de los datos con el fin de eliminar los datos irrelevantes. Tras eso, se extraen las características de los paquetes en términos generales y posteriormente a los

paquetes. Luego realiza un merge de todos los paquetes, con el fin de eliminar paquetes. Luego, se pre-procesan los flows. Para ello, se agrupan los paquetes en función del stream al que pertenecen y el protocolo. Posteriormente, se calcula la duración de cada flow. Luego, se sacan las características y se realiza merge para eliminar aquellos datos son innecesarios.

Luego, se realiza un preprocesamiento de los pesos y se agrupan en función del puerto destino.

Posteriormente, se eliminan los destinos que crean ruido y se normalizan tanto los pesos como las características. Una vez generado los gráfos, se guardan en carpetas por aplicaciones y se convierten a formato csv.

4.7. Clasificador

Este módulo, como se ha explicado anteriormente, tiene como objetivo principal el entreno del algoritmo y el testeo del modelo construido. Dado a que se aplica machine learning tradicional, utilizamos los algoritmos:XGboost, regresión logistica y gradient boosting. Además, se aplica el GCNN, que pertenece al Deep Learning y se realiza con redes neuronales y gráfos. Estos algoritmos fueron investigados anteriormente.

Este módulo se divide en dos partes. La primera de ellas, es la realización del proceso completo sin tuneo de los parametros. Y la segunda parte, consiste en repetir el mismo proceso tuneando los hiperparámetros, es decir, a través del empleo de GridSearch se iran cambiando los hiperparámetros para ver que solucion es mejor. Por ultimo, se evaluará todos los modelos juntos y se vera cual es la mejor combinación.

Antes de entrenar a cada uno de los modelos, se cargan los datos de entrenamiento y los del test, que se usaran posteriormente para comparar las predicciones.

Una vez cargado los datos, se entrenan los modelos. Todos siguen la misma dinámica: se ajustan los datos de entrenamiento, se entrena el modelo y se realizan las predicciones sobre ese modelo entrenado. Posteriormente, se seleccionan varias métricas de clasificación para ver que modelo es mejor.

Tras realizar la predicción, se compararán los resultados obtenidos con las etiquetas reales del test para medir el rendimiento del modelo. Para realizar esto, se emplean métricas de confusión. En concreto, tras nuestra investigación, hemos optado por aplicar Matrix Confussion y el f1 Score.

Capítulo 5

Experimentos y Resultados

Se han realizado una serie de experimentos para comprobar el resultado de nuestro modelo. Se han realizado la prueba con los sets de entrenamiento y testeo. Para realizar la comprobación de nuestros resultados, se han realizado sin emplear el algoritmo GridSearch y usándolo. Se ha empleado las metricas de comprobacion de Matriz de confusión y F1 Score.

5.1. Regresión logística

Como se puede observar en la imagenes relativas a los resultados de regresión logística, mostrado en la figura 5.1 la mejora en términos de matriz de confusión y F1 score es la misma con que sin tuneo. Si es cierto, que a la hora de pasarle hiperpárametros se observan pequeñas variaciones.

Esto significa que dado a que nuestros valores son pequeños el parámetro de fuerza de regularización es alto (es valor por defecto es 1 y la mejor solución es con 10, es decir, 10 veces mas de fuerza) . El algoritmo de solución que mejor funciona es con newton-cg. Aunque el parámetro por defecto también trabaja bien con multiclases (que es nuestro caso) , el mejor es este dado a que soporta un penalty que concuerda mejor con el algoritmo que es el l2, como ha concluido también GridSearch. Como se ha explicado con anterioridad, el penalty indica cuanto debe aprender el algoritmo, controla el sobre ajuste, y en este caso solo se añade el término L2.

Observando la matriz de confusión, se puede observar que arroja un 100 % de verdaderos positivos y un alto porcentaje de falsos negativos, en concreto el 72 %, unicamente acertando en un 2 %. Acerca del f1 score muestra un valor de 58, 6 %.

5.2. Gradient Boosting

En el caso de este algoritmo, hay muchas diferencias si se emplea el tuneo o no (datos observables en las figura 5.3) , esto es porque este algoritmo posee muchos parámetros que permiten la modificación de su comportamiento. Se observa que se arroja mejores resultados sin tuneo que con el, eso es porque el GridSearch realiza validación cruzada reduciendo asi los datos de comprobación.

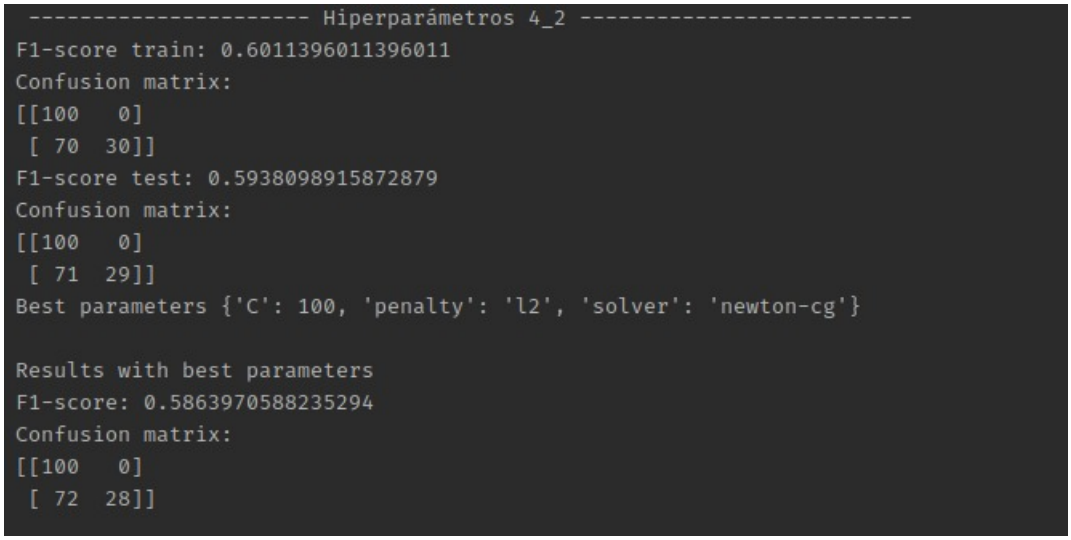

Figura 5.1: Resultados obtenidos de Regresión logística

Con tuneo, el modelo presenta un F1 score del 54, 8 %. La tasa de falsos negativos es alta, de un 77 %. Analizándolo sin tuneo, se observa que tiene un F1 score mayor, del 61, 55 % y una tasa de falsos negativos del 68 %, es decir, es mejor realizar este modelo sin tuneo que con el.

Figura 5.2: Resultados obtenidos de Gradient Boosting

5.3. XGBoost

Este algoritmo tras su entrenamiento arroja los mismos resultados tuneándolo o no. Lo que indica que los parámetros de serie estan elegidos de manera que arroja los mejores resultados, sin necesidad de adoptarlos. Este algoritmo arroja un F1 score de 79, 4 %.

5.4. Análisis de resultados

Los resultados en conjunto se muestran en 5.1, que seran analizados en el posterior apartado.

```
F1-score train: 0.8139727159983464
Confusion matrix:
[[25  0]
 [ 9 16]]
F1-score test: 0.794745484400657
Confusion matrix:
[[24  1]
 [ 9 16]]
Best parameters {'eta': 0.01, 'gamma': 1, 'max_depth': 2}
Results with best parameters
F1-score: 0.794745484400657
Confusion matrix:
[[24  1]
 [ 9 16]]
```

Figura 5.3: Resultados obtenidos de XGboost

Tabla 5.1: Resultados arrojados tras los experimentos

	Sin tuneo	Con tuneo
	F1Score	F1Score
Regresión Logística	0. 59	0. 58
XGBoost	0. 79	0. 79
Gradient Boosting	0. 61	0. 36

Capítulo 6

Conclusiones y Trabajo Futuro

6.1. Conclusiones

Poniendo todos los resultados juntos, se observa que sin utilizar el GridSearch, el algoritmo que arroja mejores resultados es elXGBoost. Este presenta grandes diferencias que los otros. Utilizando el GridSeach, se observa que el XGBoost sigue siendo el mejor algoritmo probado al arrojar un F1Score mejor. Esto se debe dado a varias razones. La primera de ellas, es que dado a los pocos datos que se introducen en el modelo,tanto para entrenarlo como para testearlo, por eso XGboost es el mejor pues es el que mejor trabaja con datasets pequeños. Se observa que los resultados sin tuneo son mejores que con el, es dado a que el GridSeach,empleado para la elección de los mejores parámetros, trabaja con validación cruzada y al tener datasets pequeños no es optimo. Otra razón por que los resultados sin tuneo son mejores es porque los parámetros por defecto se eligieron porque daban los mejores resultados.

Por lo explicado anteriormente, nuestro modelo de Machine Learning elegido es el que se encuentra entrenado con el XGBoost.

Comparando estos resultados con los que arrojan los trabajos investigados, se observa que el resultado F1 score que produce nuestro modelo se aleja del que tiene AppScanner y DGCNN, por lo explicado anteriormente.

6.2. Trabajo Futuro

En el futuro, se piensa investigar con otras tecnicas de aprendizaje automático e inteligencia artificial que procesen datos de mayor tamaño con una mayor rapidez y que trabaje bien tanto con datasets pequeños y tambien grandes. Además, se comprobara los datos con el set de validación, con el fin de obtener resultados de las métricas de clasificación que tiene AppScanner y DGCNN.

Capítulo 7

Aportaciones individuales

En este capítulo, se va a listan las distintas aportaciones realizadas por cada miembro de los participantes del proyecto. Aunque la mayoría del proyecto se ha realizado de manera conjunta, ha habido módulos en los que la participación de un integrante ha sido mayor.

7.1. Marcos Gálvez Santamaría

- **Fase de investigación**:

  Se investigó acerca de la huella digital de TLS mediante la utilización de información determinada, en este caso concreto, se utilizaba el destinatario de la conexión y las bases de conocimiento.

  Se realizó una investigación y análisis de diversas herramientas de clasificación de tráfico producido por diferentes aplicaciones.

  Se investigó y estudió acerca de las conexiones establecidas por las diferentes aplicaciones móviles para conocer las características de éstas, implicando esto el estudio de los diferentes protocolos como pueden ser el protocolo TCP o el protocolo UDP.

  Se realizó una investigación acerca de las distintas técnicas de evaluación de resultados obtenidos mediante aprendizaje automático, en concreto se llevó a cabo una investigación acerca de la matriz de confusión y el f1-score, en colaboración con Patricia Martín-Salas.

  Se investigó acerca de las redes neuronales como modelo de aprendizaje automático y su aplicación práctica en el lenguaje de programación Python mediante la librería tensorflow.

  También se realizó una investigación acerca de deep learning y sus apliciones prácticas en el lenguaje de programación Python.

  Se investigó acerca de la creación de grafos con sus correspondientes pesos para modelos de aprendizaje automático tanto supervisado como no supervisado, además de realizar una investigación acerca del preprocesamiento de datos para prepararlos con el objetivo de realizar modelos de aprendizaje automático que los procesen.

  En cuanto a los diferentes formatos de ficheros que servirán de apoyo en la implementación y ejecución del algoritmo, se ha realizado una investigación acerca

del formato json para el guardado de los datos generados por el algoritmo, y una investigación acerca de los mejores formatos para guardar los modelos de aprendizaje automático ya entrenados.

Por último, se ha llevado a cabo una investigación acerca de la búsqueda de los mejores parámetros de cada uno de los modelos con la librería GridSearchCV con el objetivo de maximizar los resultados de la ejecución de los diferentes modelos de aprendizaje automático implementados.

Adicionalmente, se ha llevado a cabo una investigación de la herramienta LaTex Overleaf para llevar a cabo la realización de esta memoria.

- **Desarrollo del algoritmo:**

  En cuanto al desarrollo del algoritmo, en primer lugar se realizó la implementación del preprocesamiento de los datos extrayendo las características necesarias y generado los diferentes ficheros de apoyo para la correcta ejecución del algoritmo.

  Posteriormente, se llevo a cabo la creación de los grafos con sus correspondientes pesos utilizando el preprocesamiento llevado a cabo con anterioridad.

  Una vez realizado el preprocesamiento y la creación de los grafos, se llevo a cabo la implementación de los siguientes modelos de aprendizaje automático: GCNN, XgBoost y regresión logística, en colaboración con Patricia Martín-Salas.

  Cuando se terminó de implementar los diferentes modelos de aprendizaje automático, se llevo a cabo la implementación de la librería de Python GridSearchCV para la búsqueda de los mejores parámetros de cada algoritmo con el objetivo de alcanzar las mejores métricas posibles.Esta tarea se realizo en colaboración con Patricia Martín-Salas.

  Por último, se implementaron las diferentes métricas de evaluación de los resultados de los diferentes modelos. En esta parte se llevaron a cabo tanto la matriz de confusión como el f1-score, siendo ésta última en colaboración con Patricia Martín-Salas.

- **Conceptos adquiridos:**

  Gracias a la elaboración de este proyecto se han adquirido ciertos conocimientos que procedo a explicar a continuación.

  En primer lugar, se han adquirido conocimientos de programación en Python, tanto generales debido a que ha sido una de las primeras veces en las que he utilizado este lenguaje, como específicos de tareas concretas, como puede ser la creación de grafos o la implementación de modelos de aprendizaje automñatico.

  También se han aprendido conceptos teóricos acerca de los diferentes modelos de aprendizaje automático utilizados en la implementación del algoritmo.

  Además, se ha profundizado en los conceptos teóricos y prácticos sobre redes y conexiones entre diferentes dispositivos.

  Se han adquirido conocimientos teóricos y prácticos acerca del preprocesamiento de datos para modelos de aprendizaje automático.

  Se ha aprendido a utilizar diferentes librerías relacionadas con el aprendizaje automático y el procesamiento de datos en Python como sklearn, matplotlib, stellargraph y tensorflow. Además, también se han adquirido conceptos sobre la

evaluación de modelos con métricas como el f1-score, el accuracy o las matrices de confusión.

Por último, se han adquirido conceptos y conocimientos acerca de formatos de ficheros para el guardado de datos, como puede ser el formato json, y formatos para el guardado de los diferentes modelos entrenados.

Adicionalmente, se ha aprendido a realizar y llevar a cabo una correcta utilización de la herramienta LaTex Overleaf para la elaboración de la presente memoria.

7.2. Patricia Martin-Salas Gesteira

- **Fase de investigación**:

  Desarrollo del capitulo 1 y el resumen y su posterior traducción al ingles conformando el capitulo 9 y el abstract, respectivamente. Posteriormente, fue corregida por una experta en el lenguaje con el fin de que no haya errores de ningún tipo y se entienda a la perfección.

  Se investigó sobre el funcionamiento del modelo OSI así como de la transferencia de paquetes a lo largo de la red y se plasmo en la memoria.

  Se investigo en profundidad sobre que era la inteligencia artificial así como los distintos tipos de aprendizaje automático. Se desarrollo en la memoria lo anteriormente explicado, así como las ventajas y desventajas de cada uno de los tipos y metodologías que implementa. Para apoyar la explicación, se realizo una búsqueda exhaustiva de imágenes que lo ilustrarán e incluso se desarrollo alguna con el fin de explicar todo de manera fácil para el lector.

  Se realizo una búsqueda inteligente y profunda sobre los algoritmos de machine learning que se iban a implementar: Gradient Boosting y XGBoost. En concreto, se investigo sobre como funcionaba, sus ventajas y desventajas y los posibles parámetros con los que se podia modificar.

  Además, se realizó una búsqueda con el fin de comprender las diferencias entre set de entrenamiento,text y validación, asi como la metodología necesaria para contruir un modelo de machine learning y los conceptos básicos para entender que es un grafo. Posteriomente, se realizo una investigación sobre los métodos de ensamblado, pues era de vital importancia entenderlo para posteriormente aplicarlo. Se investigaron distintos proyectos similares al que iba a ejecutar el grupo, en concreto App Scanner, el analisis de aplicaciones de redes sociales y usando Autoenconder e identificación de la firma y análisis de la actividad de Whatsappa a través del análisis de datos.

  Se colaboró con Marcos Galvéz en la investigación de métricas de clasificación, siendo Patricia la responsable de identificar el funcionamiento, las ventajas y desventajas de F1 score y posteriormente le transmitió la información a su equipo. Posteriomente, se trabajo junto para realizar un resumen de todas y comprenderlas en conjunto.

  Las últimas investigaciones realizadas fueron entorno a la personalización de los hiperparámetros. Una vez comprendida toda la información existente en la memoria, se realizó el análisis de resultados y se redacto los experimentos y la conclusión.

- **Desarrollo del algoritmo:**

  - Se trabajo con Marcos Galvéz en el la lectura de los archivos, asi como de la limpieza de datos para su procesamiento.

- Se aporto la información necesaria sobre la contrucción de grafos a Marcos Galvéz que llevo la carga de este módulo.
- Se desarrollo el entrenamiento y el testeo de gradient boosting. Se colaboró con Marcos Galvéz con el fin de desarrollar los demás, en especial regresión logística pues se realizo una investigación en profundidad que les ayudo a comprender y detectar los errores.
- Desarrollo de la métrica de validación f1 score. Se colaboró con Marcos Galvéz en el desarrollo del GridSearch.
- Se analizaron los experimentos realizados y con todo lo aprendido se desarrollo las conclusiones.

- **Conceptos adquiridos:**

  Para la realización de este proyecto se ha tenido la obligación de realizar una investigación previa al desarrollo del mismo con el fin de aprender las diferentes tecnologías que se iban a utilizar en la aplicación.

  Es por ello que aún sabiendo las ventajas que aporta Python frente a otros lenguajes de programación respecto al aprendizaje automático y otros algoritmos de inteligencia artificial, hasta este año no se ha tenido la posibilidad de emprender un proyecto de estas características para aprender dicho lenguaje. Es por ello, que para el desarrollo de este proyecto se ha tenido que investigar cómo funciona este lenguaje asi como sus principales bibliotecas. Este aprendizaje se ha visto apoyada por la asignatura de Big Data y minería de Datos que se ha cursado paralelo al desarrollo de esta aplicación facilitando, en mayor medida, el aprendizaje de diferentes librerías utilizadas en cada uno de los proyectos asignados por esa asignatura. Además, para aumentar todo conocimiento posible, se consultó muchos manuales de estudio de Python con el fin de entender los errores que iban surgiendo a lo largo de este proyecto.

  En cuanto al conocimiento en las tareas relacionadas con la inteligencia artificial y sus algoritmos, ya sea el uso de Machine learning o Deep learning, que son los principales algoritmos utilizados en este proyecto, se carecía de la experiencia previa necesaria para desarrollarlo. Por lo que se tuvo que estudiar desde cero en que consistía, qué beneficios aportaba la utilización de dichas técnicas, sus tipos para seleccionar el más indicado y como se confeccionaba este. Una vez adquirido los conceptos básicos necesarios, se aprendió y se realizó la investigación correspondiente de XGBoost y gradient Boosting. De este modo, una vez entendido tanto el funcionamiento como las ventajas y desventajas que tenía, se pudo desarrollar y concluir sobre sus resultados elaborándose de la manera más eficiente posible. Este proceso, se realizó con la ayuda de un investigador de Machine Learning y Deep Learning con el que tuvo reuniones informales con el fin de ir aclarando dudas y orientando cuando no sabía que sucedía.

  Anteriormente, se realizaron diferentes capturas de tráfico en la red desde ordenadores mediante tecnologías como Wireshark y se realizó un análisis breve de los paquetes obtenidos. Dado que el proyecto se orientaba a tecnologías móviles, se tuvo que hacer una investigación previa para aprender a capturar tráfico de datos entre diferentes dispositivos móviles. Una vez adquirido y aplicado lo aprendido, se hizo un análisis en profundidad de lo que contenía cada paquete entendiendo el significado de cada campo del mismo con el fin de poder determinar que datos son relevantes y cuales se podían omitir.

Dado a que el desarrollo de la memoria se realizó en látex, en concreto, Overleaf. Se aprendió el manejo total de este a través de la lectura de un manual con el fin de desarrollar una memoria de alto nivel. Se aprendió como maquetar el proyecto así como insertar citas, referencias y leyendas.

7.3. Carlos Ramírez

- **Fase de investigación**:

  Desarrollo junto con Patricia Martín-Salas del capitulo 1, asi como de la traducción de este para el capitulo 6. Tambien se realizo la traducción de las conclusiones y de futuros trabajos.

  Se investigó sobre MAppGraph, cómo funciona su técnica de clasificación de aplicaciones de Android, método que utiliza para la generación de grafos por medio de nodos y aristas, y técnica de extracción de información de cabeceras que ha desarrollado. Para complementar también se investigó sobre AppScanner para el entrenamiento de modelos de reconocimiento de patrones de tráfico y clasificación de estos por aplicaciones, y sobre FlowPrint para el reconocimiento de las distintas aplicaciones.

  Se investigo junto a Patricia Martín-Salas sobre los algoritmos de clasificación supervisados para la construcción del modelo. Centrandose en el Bootstrapping, Bagging, Tree boosting, Gradient Boosting y XGBoost. Tambien se investigo sobre la regresión logística.

  Se investigó sobre los modelos de aprendizaje no supervisados y como obtener resultados sobre los paquetes de datos con su implementación en Python. También se investigó sobre las características comunes de las aplicaciones web más utilizadas para ser capaz de reconocerlas, así como los protocolos que utilizan para sus datos.

  Se investigó sobre la creación de grafos de pesos, así como su implementación en Python para modelos de aprendizaje supervisados y no supervisados. Para complementar también se ha estudiado sobre métodos de regularización para lograr reducir el error que se obtiene de los modelos, así como el funcionamiento de la propagación hacia delante y hacia atrás para los modelos de aprendizaje automático.

  Se analizó StellarGraph para conocer el método que este utiliza para la generación de gráficos no supervisados y su implementación en el modelo.

  Se investigó sobre la extracción de características y cuales son los dos métodos más eficientes para lograrlo: Spatial y Spectral. Así como las ventajas y desventajas de implementarlas en el modelo. También investigó sobre el funcionamiento de la regresión logísticas y su implementación en los modelos supervisados de Python.

  Por último, se investigó sobre las redes convolucionales de grafos como método para automatizar el proceso de detección de datos y cómo implementarlo de manera práctica en Python.

- **Desarrollo del algoritmo:**

  Al principio se aportó apoyo a Marcos Galvéz con la implementación del preprocesador, así como con la identificación y extracción de las características necesarias.

También se ayudó con la generación de los grafos y sus correspondientes pesos, así como con los resultados obtenidos de estos.

Se colaboró con Marcos Galvéz en el desarrollo del módulo GCNN tras la investigación realizada para la comparación de modelos supervisados. Apoyo a Marcos Galvéz durante la implementación del clasificador con la resolución de errores y optimización del programa.

Por último, generación de resultados para la comparación de los modelos así como de diagramas de flujo del código para su interpretación.

- **Conceptos adquiridos:**

  A lo largo de este proyecto se han ido adquiriendo nuevos conocimientos de los que previamente no disponía debido a la complejidad y profundidad de los temas desde los que parte este proyecto.

  Para comenzar ha sido necesario la investigación en profundidad de artículos académicos, así como de trabajos previos sobre el análisis de tráfico que no se encuentran con facilidad. Para ello ha sido necesario la utilización de portales con acceso a este tipo de documentos. También ha requerido de una profundización de algunos conceptos previamente vistos como los paquetes TCP y UDP o las características de los paquetes, los cuales hemos tenido que analizar para el desarrollo de los modelos.

  Para la realización del proyecto también ha sido necesario aprender desde cero sobre los modelos supervisados y no supervisados de grafos, como funcionan, para que se utilizan y sobre todo como se puede diseñar uno propio. Otro de los conceptos teóricos necesarios ha sido aprender sobre las redes neuronales convolucionales de grafos o GCNN que hemos implementado en el proyecto, cómo funciona la convolución y como se puede aplicar para el tráfico cifrado.

  También se ha requerido comenzar desde cero el análisis de las aplicaciones que se analizan en el trabajo. Para ello se ha sido necesario aprender sobre características de estas y los métodos que existen para extraerlas. Para lograrlo hemos necesitado aprender a utilizar aplicaciones de captura de tráfico y a evaluar la validez de estas capturas.

  Para la generación del modelo se ha requerido aprender a utilizar Python, lenguaje de programación con el cual no se había trabajado antes. Para ello se han utilizado diversos manuales y cursos online los cuales nos recomendaron nuestros tutores y de esta manera alcanzar un dominio sobre el lenguaje. También me ha sido muy útil para conocer las oportunidades que te ofrece Python frente al resto de los lenguajes que había utilizado previamente, en especial el uso de las bibliotecas que es muy cómodo y ofrece una gran variedad de soluciones a problemas que uno pueda encontrar programando.

  También se ha aprendido sobre escritura en LaTeX para la realización de la memoria. Como no se había utilizado previamente se han utilizado guías y videos de internet para aprender cómo escribir y lograr un diseño apropiado de este. Se ha aprendido también a como citar y meter referencias de manera óptima para evitar cualquier problema de copia o plagio respecto a los artículos originales de los que proviene.

  Para la creación de los diagramas de flujos se ha aprendido a utilizar diagramans.net y se han utilizado guias online para aprender como realizarlos de manera que sean sencillos de entender pero al mismo tiempo contengan toda la información relevante.

Por último, también recalcar el aprendizaje que supone realizar un proyecto de tal envergadura en equipo y las habilidades de gestión y comunicación que se requieren.

Capítulo 8

Introduction

8.1. Motivation

Our experience, encouraged by contact with society and the visualization of news, has allowed us to see that today there are many individuals who have been attacked by some type of cybercrime, whether financial fraud, identity theft, falsification of information, theft of intellectual property and cyberbullying.
Today, there are many ways to attack a user by using the computer network, either by impersonating an identity with techniques such as *man in the middle* or denial of service (DDoS), or by the introduction of some malware on the device.
All this takes place due to the increase in the use of the Internet, and that is why, to solve this problem and avoid these attacks, several branches of computer security have emerged to investigate and avoid them, including studies such as forensic analysis, judicial expertise, ethical hacking, database protection and security in computer networks.
Forensic analysis is defined as an area inside computer security, born from the increment of the incidents on it. Its importance resides in the fact that it reduces the workload and resolution times in cyber criminal investigations.
From all this, the importance of identifying the content of network traffic becomes a necesity and this derives in the need to identify the content for forensic analysis. This aspect has encouraged the development of this work in order to provide a new tool that helps analyze network traffic.

8.2. Object of the investigation

A large amount of information travels through the network, including a lot of content which can be harmful to other users. It has been discovered that well-trained artificial intelligence makes it possible to find this information through the identification of patterns in the information.
This work proposes different techniques, algorithms and models that allow the implementation of forensic analysis of data traffic in order to improve cybersecurity in this aspect.

8.3. Workplan

The development of this work has been carried out in three phases:

1. **Research**: First we focused on the investigation of the topic, because before starting we needed to acquire some basic knowledge about mobile device traffic and about Machine Learning techniques. We were told from the beginning what the main objective of the project was and the applications and techniques that we were going to use. During the months of September to December we focused on this phase, each member of the group working individually in the search and analysis of those techniques and applications and others that we considered could be useful to improve the functioning of our tool. For this research, we used tools like *Google Scholar*, which made it easier to find academic documents about the topics that interested us. Weekly meetings were held via *Google Meet* where each one presented the advances in the research they had carried out that week and the thesis directors resolved any doubts we might have. At the end of this phase we were able to draw a few conclusions from all the research carried out that helped us on the most optimal way to develop the code and to obtain the expected results. Also throughout this phase we generated our own dataset for training our tool through captures made with *PCAPdroid*, which allowed us the capture of traffic from specific applications.

2. **Development**: The second phase of this project was code development. When we managed to decide how we wanted this to be and on which models we were going to base our own, model, we had to study how it was possible to program those ideas. For this reason, throughout this phase, programming languages such as *Python* were investigated, and inside of it, the libraries such as *Keras* and *Tensor Flow* that we needed as support. This was not the only investigation that was carried out throughout this period because as we progressed with the code, we have to face some problems or difficulties which required an investigation to find the most optimal way to solve them. To finish this phase, and once the code was functional, we created training sets for the model.

3. **Experimentation**: Finally we have the experimentation phase in which we focus on the prototypes of the model that we were creating. During this phase, we also continue with the development of the code because it was necessary to improve the prototypes that we were creating. Throughout this phase, we seek to optimize the model by carrying out tests and comparing the results obtained in each of them and with those obtained in the academic works that we had analyzed. In this way, the parameters of the model were adjusted until we achieved optimal results. For the comparison of the different models, the tools provided by the *Python* libraries were used, which we investigated in the previous phase.

Capítulo 9

Conclusions and Future Work

9.1. Conclusions

Putting all the results together, it can be seen that without using GridSearch, the algorithm that gives the best results is theXGBoost. It is much different from the others. Using GridSeach, it is observed that XGBoost is still the best algorithm tested by giving a better F1Score. There are several reasons for this. The first one is that due to the small amount of data that is introduced in the model, both for training and testing, XGboost is the best because it is the one that works best with small datasets. It is also observed that the results without tuning are better than with it, because the GridSeach, used to choose the best parameters, works with cross validation and having small datasets is not optimal. Another reason why the results without tuning are better is because the default parameters were chosen because they gave the best results.

As explained above, our chosen Machine Learning model is the one trained with the XGBoost.

Comparing these results with those of the works investigated, it can be observed that the F1 score produced by our model is far from the one produced by AppScanner and DGCNN, as explained above.

9.2. Future Work

In the future, we plan to investigate with other machine learning and artificial intelligence techniques that process larger data more quickly and work well with both small and large datasets. In addition, the data will be tested with the validation set, in order to obtain results of the classification metrics that AppScanner and DGCNN have.

Bibliografía

[Ama20] J Amat. Gradient boosting con python, Octubre 2020.

[AT21] J Areli-Toral. Redes neuronales, 2021.

[Azu] Azure. ¿qué es el aprendizaje automático?

[Ben] L Benites. Muestra de arranque.

[Ber] F Berzal. Regularización del deep learning.

[Bos20] J Bosco. Xgboost aplicado a pyhton, 2020.

[Bro21] J Brownlee. No free lunch theorem for machine learning, Octubre 2021.

[Bru] Brutalk. Introducción suave al algoritmo de optimización de adam para el aprendizaje profundo.

[dd20a] Ciencia de datos. Gradient boosting en python, 2020.

[dd20b] Ciencia de datos. Regresión logística simple y múltiple., 2020.

[DP21] Tram Truong-Huu Tien-Dung Cao Dien Pham, Thien-Lac Ho. Mappgraph: Mobile-app classification on encrypted network traffic using deep graph convolution neural networks, 2021.

[Gon19] L Gonzalez, November 2019.

[Hon21] S Hong. An introduction to graph neural network(gnn) for analysing structured data, 2021.

[IBM] IBM. Reglas de parada (redes neuronales).

[IBM20] IBM. What is artificial inteligence?, 2020.

[ich20] ichiPro. ¿en que se diferencia el gradient boosting y el ada boost?, 2020.

[Imp21] Imperva. Osi model, 2021.

[Ind20] Indra. ¿qué es el deep learning y para qué sirve?, 2020.

[Kor21] J Korstange. F1 score, 2021.

[Lim20] A Lima, 2020.

[Mar21] J Martinez. Precision, recall, f1, accuracy en clasificación, 2021.

[MB19] A. Martínez Bowen. *Inteligencia artificial.Redes neuronales y aplicaciones.* PhD thesis, Unversidad Carlos III de Madrid, 2019.

[Na820] Na8. Sets de entrenamiento, test y validación, 2020.

[Ora20] Oracle. Modelo de arquitectura del protocolo tcp/ip, 2020.

[Ora21] Oracle. Modelo de referencia osi, 2021.

[Par21] M Parada. ¿qué son las capas del modelo osi y cómo funcionan?, 2021.

[Rev20] Reviversoft. Extensión de archivo .pcap, 2020.

[Rod18] D Rodríguez. Gridsearchcv, 2018.

[RS21] Dr Usha G Ramraj S. Signature identification and user activity analysis on whatsapp web through network data. 2021.

[RU20] S Ramraj and G Usha. Unsupervised feature learning for whatsapp network data packets using autoencoder. pages 546–549, 2020.

[Set21] N Seth. How does backward propagation work in neural networks?, 2021.

[Shi20] T Shin. Comprensión de la matriz de confusión y cómo implementarla en python, 2020.

[sl] scikit learn. Random forest classifier.

[sl21a] scikit learn. F1 score en python, 2021.

[sl21b] scikit learn. Párametros de la matriz de confusión, 2021.

[sl21c] scikit learn. Regresión logística en python, 2021.

[sl22] scikit learn, 2022.

[SM21] A. Serra Marrugat. *Comparación de algoritmos de clasificación supervisada.* PhD thesis, Escola Tècnica Superior d'Enginyeria Industrial de Barcelona, 2021.

[Vil20] J Villanueva. Redes neuronales desde cero, 2020.

[wik20a] wiki. Training set, 2020.

[Wik20b] WikiPedia. Gradient boosting, 2020.

[Zr12] Y. Zhang and A. Årvidsson. Understanding the characteristics of cellular data traffic. CellNet '12, page 13–18, New York, NY, USA, 2012. Association for Computing Machinery.