UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

TRABAJO FIN DE GRADO

Detección de anomalías mediante distribuciones alfaestables y técnicas de aprendizaje automático

Autor: Alejandro Muñoz García
Tutor: Luis de Pedro Sánchez

marzo 2025

Alejandro Muñoz García
Detección de anomalías mediante distribuciones alfaestables y técnicas de aprendizaje automático

Alejandro Muñoz García
C\ Francisco Tomás y Valiente Nº 11

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

Agradecimientos

Quisiera agradecer a varias personas que me han ayudado durante la realización de este Trabajo Fin de Grado, especialmente a Luis y a Jorge por acogerme y aguantarme, dedicando su tiempo y conocimientos en las innumerables reuniones que hemos tenido.

También agradecer a los demás profesores de la universidad que han contribuido en mi formación académica y personal. Así como a mis compañeros de universidad y amigos que hicieron que todo este proceso fuera mucho más llevadero.

A mis padres, Isidro y Carmen, por su amor y comprensión durante los buenos y malos momentos, y por enseñarme que con esfuerzo todo se puede conseguir.

Resumen

Durante los últimos años el uso de internet ha tenido un crecimiento exponencial, lo que ha supuesto también un incremento en los ataques que se realizan, estos deben ser controlados y frenados a tiempo para poder tener un uso satisfactorio de la red. Este trabajo se centrará en los ataques de denegación de servicio, este tipo de ataque es uno de los más conocidos y recurrentes, los cuales buscan interrumpir el acceso a recursos y servicios en línea.

En este trabajo se ha estudiado la aplicación de una red neuronal para que diga con suficiente exactitud si se está recibiendo un ataque o no. Para extraer información y analizar el comportamiento de la red se ha usado un modelo estadístico basado en distribuciones alfa-estables, de los cuales se han extraído 4 parámetros (alfa, beta, gamma y delta) al conjunto de datos, el cual esta formado por diferentes flujos extraídos de una red real durante unas ciertas semanas y meses y por otros flujos de ataque creados por nosotros. A estos parámetros se les han añadido otros más para poder facilitar el trabajo de la red neuronal a la hora de clasificar. Con todos estos parámetros se ha configurado y puesto a punto la red neuronal, en la que se han analizado y estudiado los diferentes resultados obtenidos para poder evaluar el modelo.

Palabras clave

Ciberataques, denegación de servicio (DoS), distribución alfa-estable, red neuronal, *bootstrap*, distancia Kolmogorov–Smirnov.

Abstract

Over the past few years, internet usage has grown exponentially, which has also led to an increase in cyberattacks. These attacks must be monitored and mitigated in time to ensure a satisfactory use of the network. This study focuses on denial-of-service (DoS) attacks, one of the most well-known and recurrent types of attacks, which aim to disrupt access to online resources and services.

In this work, a neural network has been studied to determine with sufficient accuracy whether an attack is occurring. To extract information and analyze network behavior, a statistical model based on alpha-stable distributions was used. Four parameters (alpha, beta, gamma, and delta) were extracted from the dataset, which consists of various traffic flows captured from a real network over several weeks and months, along with additional attack flows created by us. Additional parameters were incorporated to facilitate the neural network's classification task. Using all these parameters, the neural network was configured and fine-tuned, and various results were analyzed and studied to evaluate the model's performance.

Keywords

Cyberattacks, denial of Service (DoS), alpha-stable distribution, neuronal network, bootstrap, Kolmogorov–Smirnov distance.

ÍNDICE

Listas

Lista de ecuaciones

Lista de figuras

Lista de tablas

1

Introducción

En este apartado se expondrán los principales motivos para la realización de este trabajo y los principales objetivos a cumplir. Además de una descripción de las fases que se han ido realizando hasta concluir el trabajo, así como un resumen de la estructura del documento.

1.1. Motivación

En los últimos años la tecnología se ha ido instaurando en nuestra vida cotidiana y por lo tanto también ha habido una exponencial expansión del uso del internet y del volumen de datos que circulan por la red. Una de las principales preocupaciones tanto para las organizaciones como la gente de a pie es que todos estos datos estén seguros.

A la par de este crecimiento también se ha incrementado la frecuencia y la sofisticación de los ciberataques, estos pueden tener consecuencias devastadoras, desde la pérdida de datos personales hasta el compromiso de grandes infraestructuras, en este caso se trabajará con un tipo de ataque que destaca por su capacidad de paralizar sistemas y servicios afectando a la disponibilidad. Por lo que no poder parar este tipo de ataques puede significar la pérdida de millones de euros para estas compañías, para poder pararlo las empresas suelen tener una gran variedad de estrategias y tecnologías para poder prevenir y mitigar estos ataques, una de estas técnicas es el monitoreo continuo de datos, en el cual se analizan y recopilan datos en tiempo real para poder detectar y responder las amenazas rápidamente. El primer enfoque para este trabajo fue realizar un sistema que realice algo parecido a esta técnica de monitoreo, es decir, que trabajando con unos datos se pudiera rápidamente extraer información de ellos y poder clasificarlos como una amenaza o no.

1.2. Objetivos

El objetivo principal es tener un mecanismo que detecte los ataques, para ello se desarrollará una red neuronal en la que se usan los parámetros obtenidos de la distribución alfa-estable además de otros complementarios para ayudar a la red. También se deberán de cumplir los siguientes subobjetivos:

O-1.– Extraer las series temporales de los ficheros proporcionados y generar los ficheros de ataque.

O-2.– Comparar y entender los métodos que se tienen disponibles para la obtención de los 4 parámetros alfa-estables.

O-3.– Configurar y poner en marcha una red neuronal que se ajuste a el trabajo propuesto.

O-4.– Analizar los resultados para determinar si se está produciendo un ataque.

1.3. Fases del trabajo

Este trabajo comenzó en octubre de 2023 con un periodo de formación y búsqueda del tema a realizar, se plantearon múltiples opciones, pero se decantó por la aplicación de técnicas de aprendizaje automático en la monitorización de redes ya que sobre este tema se habían realizado numerosos trabajos que sirvieron de punto de partida. Entre ellos se pueden destacar los Trabajos Fin de Grado de Alberto Ruiz [1], de Eric Crusi [2] o el TFM de Benjamín Martín [3] , los cuales aplicaban distribuciones alfa-estables al tráfico para poder realizar estas tareas de monitorización.
Tras haber elegido continuar con los trabajos previamente mencionados, se pasó a la parte de aplicar la distribución alfa-estable a los ficheros (*datasets* proporcionados por la Universidad de Granada [4]) para poder caracterizar el tráfico. Esta parte fue una de las que más tiempo se dedicó por la cantidad de métodos que había disponibles y también debido a que cada uno necesitaba su propio estudio previo, su puesta en marcha y su comparación con los otros métodos. Finalmente, con los parámetros alfa-estables calculados para todos los ficheros, se pasó a la parte de la red neuronal, donde se tuvo que diseñarla de cero y ponerla en marcha para que nos proporcionara el porcentaje de ataque o la energía que tenía el ataque de esa ventana. Por último, se realizaron unas pequeñas pruebas para poder definir unos umbrales que indicaran cuando había ataque y cuando no.

Figura 1.1: Diagrama de Gantt con las fases del trabajo

1.4. Organización de la memoria

La estructura del documento está organizada de la siguiente manera:

- **Capítulo 1 - Introducción:** en este apartado se expone brevemente el motivo de realización, el enfoque tomado para realizarlo y una explicación de los pasos realizados.
- **Capítulo 2 – Estado del arte:** explicación de los conceptos necesarios para comprender este trabajo, entre ellos están los ataques en la red, el aprendizaje automático o las distribuciones alfa-estables.
- **Capítulo 3 – Diseño e implementación:** en esta sección se explica todo el proceso realizado en el trabajo desde la obtención de los datos hasta la red neuronal.
- **Capítulo 4 – Pruebas y resultados:** parte en la que se exponen los resultados de las diferentes pruebas realizadas.
- **Capítulo 5 – Conclusiones y trabajo futuro:** análisis de los resultados mencionados anteriormente y una observación de cómo se podría continuar con este trabajo.

2

Estado del arte

2.1. Introducción

En esta sección se presentan los conceptos clave necesarios para comprender y seguir el desarrollo de este trabajo, abarcando desde la definición de un ciberataque hasta la caracterización del tráfico, para luego aplicar técnicas de aprendizaje automático que permitan la detección de dicho ataque. Se tratarán los siguientes temas:

- **Ataques en la red (Ciberataques).**
- **Distribución alfa-estable.**
- ***Machine Learning.***
- **Prueba de Kolmogórov-Smirnov.**
- ***Bootstrapping.***
- **Estudios anteriores relacionados.**

2.2. Ataques en la red (Ciberataques)

Los ataques en la red [5], también denominados ciberataques, son acciones maliciosas realizadas por personas o grupos, los cuales tienen como objetivo comprometer la seguridad de sistemas informáticos para poder acceder sin permiso a información sensible o datos personales, comprometiendo la integridad y confidencialidad de estos sistemas.

Existen numerosos tipos de ataques, como pueden ser los ataques de *phishing* (suplantación de identidad), ataques de *malware* (distribución de *software* malicioso) o los ataques de denegación de servicios (DoS, *Denial Of Service*) entre otros.

En este estudio se va a trabajar con los ataques de denegación de servicio (DoS) [6], este tipo de ataque vuelve inutilizable un recurso en línea, normalmente estos tienen como objetivo servidores web, servidores de correo electrónico, servidores de DNS o redes institucionales. Este ataque se puede realizar desde un origen, pero este sería más fácil de parar bloqueando el tráfico de este, por eso existe una variante que tiene varias fuentes de origen, denominado DoS Distribuido (DDoS, *Distributed Denial*

Of Service), donde varias fuentes comprometidas se coordinan para atacar al objetivo, haciendo que se amplifique el ataque y que sea aún más difícil de pararlo.

Estos ataques se pueden dividir en varias categorías:

- Inundación de conexiones: el atacante envía numerosas conexiones TCP para que el *host* no pueda aceptar las conexiones legitimas.
- Ataques de vulnerabilidad: envío de mensajes diseñados específicamente para una aplicación o sistema operativo, los cuales pueden hacer que el servicio se detenga o sufra un fallo.
- Inundación de ancho de banda: el atacante envía numerosos paquetes al *host* con el objetivo de que se inunde el acceso, para que los paquetes legítimos no lleguen al servidor.

Para evitar un ataque de denegación de servicios, se pueden tomar muchas medidas como tener un monitoreo constante del tráfico y de la actividad de red que entra en tu sistema para poder tener una detección y respuesta rápida, tener el suficiente ancho de banda para poder procesar estos altos niveles de tráfico o tener sistemas de detección de intrusiones (IDS/IPS) como *firewalls*, etc.

2.3. Distribuciones alfa-estables

Una distribución se denomina alfa-estable [7] si es una combinación lineal de dos o más copias independientes de una muestra aleatoria que tiene la misma distribución de probabilidad, exceptuando algún parámetro de localización o factor de escala.

Para completar esta definición se peude recurrir a las siguientes ecuaciones enunciadas en *"Univariate Stable Distributions"* de John P.Nolan [8]:

$$aX_1 + bX_2 \stackrel{d}{=} cX + d \tag{2.1}$$

$$X_1 + \cdots + X_n \stackrel{d}{=} c_n X + d_n \tag{2.2}$$

Se puede observar que para que se cumpla lo anteriormente mencionado, X1 y X2 tienen que ser copias independientes de X y que estén multiplicadas por dos constantes positivas, siendo $c > 0$ y $d \in R$. Además el símbolo de la igualdad significa igualdad en la distribución. Para que sea estrictamente estable la variable d debe ser igual a 0.

Esta definición también se cumple en otras distribuciones como la distribución normal, la distribución de Cauchy o la de Lévy, para que la distribución normal o Gaussiana sea estable se debe tener un α = 2 y β = 0, para que se cumpla con la de Cauchy α = 1 y β = 0 y para la de Lévy α = 1/2 y β = 1.

También se puede definir usando la función característica, ya que la función de densidad de probabilidad de esta distribución no tiene una fórmula analítica cerrada.

Una variable aleatoria X es estable si su función característica se puede escribir como:

$$E[\exp(itX)] = \begin{cases} \exp\left\{-\sigma^{\alpha}|t|^{\alpha}\left[1 - i\beta\tan\left(\frac{\pi\alpha}{2}\right)\text{sign}(t)\right] + i\mu t\right\}, & \alpha \neq 1 \\ \exp\left\{-\sigma|t|\left[1 - i\frac{2\beta}{\pi}\text{sign}(t)\log(|t|)\right] + i\mu t\right\}, & \alpha = 1 \end{cases} \tag{2.3}$$

Donde se usa una transformación del parámetro de localización, tal que:

$$\mu = \begin{cases} \mu_0 - \beta\tan\left(\frac{\alpha\pi}{2}\right)\sigma, & \alpha \neq 1 \\ \mu_0 - \beta\frac{2}{\pi}\sigma\ln\sigma, & \alpha = 1 \end{cases} \tag{2.4}$$

Esta distribución va a servir de gran utilidad para caracterizar el tráfico, la distribución alfa-estable es capaz de gestionar datos con colas pesadas y con comportamientos extremos por lo que se tendrá una gran flexibilidad a la hora de caracterizar.
En este caso se usan los siguientes cuatro parámetros:

- **Alfa (α):** describe la estabilidad de la distribución. Es el parámetro más importante y tiene un rango de $(0, 2]$. Caracteriza la forma de la distribución: cuando se acerca a 2, se tiene una curva gaussiana, y cuando se acerca a 0, se tiene una distribución degenerada.
- **Beta (β):** describe la asimetría de la distribución y comprende el rango de $[-1, 1]$. Si se tiene un valor de -1, la distribución es asimétrica a la izquierda; es asimétrica a la derecha cuando es 1, y en 0 se tiene una distribución simétrica.
- **Gamma (γ):** define la escala, es decir, la expansión o estrechamiento de la función, comprendido entre $(0, \infty)$.
- **Delta (δ):** representa la localización relativa de la función, en el rango de $(-\infty, \infty)$.

2.4. Machine Learning

El aprendizaje automático (*Machine Learning*) es un campo de estudio que da a las maquinas la habilidad de aprender sin ser programadas explícitamente. Es decir, los algoritmos utilizan los datos para identificar patrones, hacer predicciones o tomar decisiones directamente.

Hay varios tipos de algoritmos [9], dependiendo de la salida de estos, algunos de los modelos más importantes son:

- **Aprendizaje supervisado:** etiqueta los datos previamente. El algoritmo aprenderá en función de la capacidad que tenga de predecir la etiqueta para un conjunto de datos nuevos.
- **Aprendizaje no supervisado:** los datos no están etiquetados y el algoritmo debe de encontrar pistas para agrupar los datos.
- **Aprendizaje por refuerzo:** donde el objetivo es que el algoritmo aprenda a tomar decisiones interactuando con el entorno a partir de la experiencia, es decir tomar decisiones de acuerdo con un proceso de prueba y error [10].

Dentro de estos tipos se tienen varios modelos como pueden ser las redes neuronales, de las cuales se hablará más adelante, los árboles de decisión, la regresión lineal (predecir valores numéricos), la regresión logística (hace predicciones en base a respuestas de “sí/no”), agrupaciones en *clusters* o

Random Forest.

Las redes neuronales (NN: *Neural Networks*) [11] son un tipo de modelo de computación inspirado en la estructura y funcionamiento del cerebro humano. Están compuestas de neuronas artificiales interconectadas que se organizan en capas, cada una toma una entrada, realiza una computación y devuelve un resultado, que pasa a otras neuronas en otras capas. El primer modelo de red neuronal fue propuesto en 1943 por McCulloch y Pitts en términos de un modelo computacional de actividad nerviosa. En aprendizaje automático son muy útiles para analizar y reconocer patrones en los datos. Se pueden entrenar sobre conjuntos de datos etiquetados para realizar tareas de clasificación, regresión o agrupación en *clusters*. Existen diversos tipos de redes neuronales, como las Redes Neuronales *Feedforward* (FNN), en las que la información fluye en una única dirección, las Redes Neuronales Convolucionales (CNN) y las Redes Neuronales Recurrentes (RNN). En este proyecto, se utilizará una Red LSTM (*Long Short-Term Memory*), un tipo específico de RNN, debido a que se está trabajando con datos secuenciales. Estas redes se caracterizan por sus conexiones recurrentes, que les permiten recordar información pasada.

2.5. Prueba de Kolmogórov-Smirnov

La prueba de Kolmogórov-Smirnov (K-S) es una de las pruebas no paramétricas más conocidas a la hora de comparar dos distribuciones, esta prueba determina la bondad de ajuste de dos distribuciones, es decir, permite contrastar una hipótesis nula para saber si un conjunto de datos sigue o no una distribución [12].

En este contraste se calcula la máxima diferencia absoluta entre la función de distribución empírica de los datos(EDF,*Empirical Distribution Function*) con la función de distribución teórica(TDF, *Theoretical Distribution Function*).

$$D_n = \max_{1 \leq i \leq n} |F_E(a_i) - F_T(a_i)| \tag{2.5}$$

$$F_E(x) = \frac{h(x)}{n} \tag{2.6}$$

El primer elemento es la función de distribución empírica, donde h(x) es el número de datos menores que x y n es el numero total del conjunto de datos y el sustraendo es la función de distribución teórica.

Si el estadístico de la distancia excede un cierto valor crítico determinado anteriormente se rechazará la hipótesis.

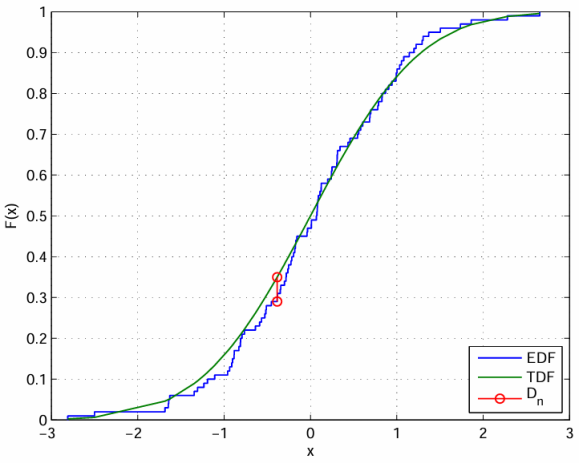

Figura 2.1: Representación de la EDF, TDF y su máxima diferencia absoluta entre ellas [12].

2.6. Bootstrapping

El bootstraping es una técnica estadística de remuestreo propuesta por Bradley Efron en 1979 que ayudará a romper la dependencia estadística de las muestras, es decir, en reducir el sesgo. Será de gran utilidad a la hora de calcular la distancia de Kolmogorov-Smirnov.

Básicamente el comportamiento de una población a partir de una muestra puede ser simulada mediante un proceso de remuestreo de esa muestra y realizando la inferencia sobre la muestra. Como la población es desconocida, el error en una muestra contra su valor poblacional es desconocido, en las re-muestras del *bootstrap*, la población es la muestra y por lo tanto se puede medir la calidad de la inferencia de la muestra a partir de los datos remuestreados [13].

2.7. Estudios anteriores relacionados

Durante este trabajo se ha realizado un proceso extenso de investigación para poder comprender y realizar este proyecto, los componentes principales de este trabajo son las distribuciones alfa-estables y las redes neuronales como método de detección de estos ataques. En muchos de los trabajos leídos, hablan de las distribuciones estables, las cuales para comprenderlas es indispensable haber leído *"Univariate Stable Distributions"* de John P. Nolan [8], donde aborda con profundidad la teoría matemática detrás de estas distribuciones y ofrece herramientas prácticas para su modelado, simulación y análisis.

Una de las principales referencias fue el trabajo realizado por Eric Crusi y Alberto Ruiz que se resumió en el artículo llamado: "Detección de ciberataques mediante análisis estadístico con distribuciones α-estables", presentado en las Jornadas de Ingeniería Telemática de Zaragoza en 2019, en

el cual se resumía los Trabajos de Fin de Grado de ambos [2] [1], donde explicaban que los análisis de los estadísticos de tráfico de una red permiten caracterizarlo e incluso identificar ataques, en este caso utilizando ajustes estadísticos de series temporales con distribuciones alfa-estables se consiguieron resultados más que notables identificando ciberataques de tipo DoS, que no eran detectables con otros métodos. Otro de los trabajos que sirvió de ayuda fue *"Network Anomaly Detection With Stable Distributions"* realizado por C. A. Bollmann [14], que comenta también el buen desempeño de estas distribuciones frente a otras como las *gaussianas*, gracias a su precisión de detección, manteniendo bajas las tasas de falsas alarmas, debido a sus estimadores de localización y dispersión que añaden un nivel más de precisión, entre otros recursos.

También cabe destacar el trabajo realizado por Benjamín Martín en sus trabajos de Fin de Grado y de Máster [3], donde presenta una técnica para detectar anomalías sutiles de corta duración en tiempo real (como pueden ser los ataques denominados L(D)DoS (o D(D)oS a baja tasa)), usando modelos predictivos basados en un histórico de datos, además de combinarlos con otras técnicas estadísticas.

En la parte del cálculo de los alfa-estables se usó el artículo de Federico Simmross-Wattenberg, "*Fast calculation of α-stable density functions based on off-line precomputations. Application to ML parameter estimation*" [12], donde se propone un rápido algoritmo para calcular funciones de densidad de probabilidad (PDFs) y funciones de distribución acumulada (CDFs) arbitrarias para distribuciones alfa-estables mediante precomputaciones realizadas externamente de los valores alfa-estables en una cuadrícula de puntos dentro del espacio de parámetros α–β, así como en un conjunto de puntos de abscisa. Los cuales demostraban que este método era más rápido que con las ecuaciones propuestas por Nolan [8]. Más tarde esto se implementaría en una librería en C/C++ con soporte para MATLAB, la cual está diseñada para realizar de manera paralelizada, eficiente y precisa, la evaluación de funciones de densidad, distribución y cuantiles, además de generar variables aleatorias y estimar parámetros de distribuciones alfa-estables en la totalidad de su espacio de parámetros. En el artículo realizado por Federico y otros compañeros de la Universidad de Valladolid, "*libstable: Fast, Parallel, and High-Precision Computation of α-Stable Distributions in R, C/C++, and MATLAB*" [15], se comenta el buen desempeño en términos de rapidez y precisión de cálculo de esta librería.

Para terminar se quiso realizar un mecanismo para poder elegir un valor de los porcentajes (umbral) que nos dijera con más o menos exactitud si hay ataque en esa ventana o no, para ello nos sirvió de inspiración el proyecto realizado por Jorge Iraizoz y Alejandro Peña llamado "Proyecto de Clasificación de Ataques sobre dispositivos IoT" [16] donde utilizaban las siguientes curvas para poder determinar un umbral:

- **Curva ROC (*Receiver Operating Characteristic*):** gráfico que muestra la capacidad de un modelo de clasificación para distinguir entre clases. Representa la relación entre la tasa de verdaderos positivos y la de falsos positivos.
- **Curva FAR (*False Acceptance Rate*):** mide el porcentaje de veces que el modelo acepta incorrectamente una instancia negativa como positiva.

- **Curva FRR (*False Rejection Rate*):** indica el porcentaje de veces que el modelo rechaza incorrectamente una instancia positiva como negativa.

El umbral se determina gracias al EER (*Equal Error Rate*) que es donde las curvas FAR y FRR coinciden, indicando un equilibrio entre la tasa de falsos aceptaciones y falsos rechazos.

2.8. Conclusiones

La decisión de utilizar una distribución alfa-estable para extraer información de los datos de tráfico de redes es debido a las características únicas de estas distribuciones. Las distribuciones alfa-estables son capaces de modelar datos con colas pesadas y comportamientos extremos, lo cual es crucial cuando se analizan patrones de tráfico de red, ya que en el conjunto de datos hay trafico normal como tráfico en el que se ha producido ataques de denegación de servicio. Los ataques DoS, afectan a la disponibilidad de servicios y sistemas, esto suele provocar fluctuaciones abruptas y anómalas en los flujos de datos. Estas distribuciones permiten capturar de manera eficaz tanto las variaciones normales del tráfico como las anomalías que producen este tipo de ataques.

Además de la decisión de usar este tipo de distribuciones, también es necesario usar el método correcto para calcular sus parámetros, para ello se comparan diferentes métodos mirando el tiempo de ejecución, la distancia de Kolmogórov-Smirnov...
Esta información extraída, que incluye parámetros como la dispersión, asimetría y pesadez de las colas, refleja la dinámica del tráfico y proporciona características valiosas que serán utilizadas como entradas para la red neuronal, las cuales son conocidas por su capacidad de realizar tareas de clasificación y por lo tanto ayudarán al objetivo de detectar los ataques.

3

Diseño y desarrollo

3.1. Introducción

En este capítulo se explica todo el proceso desde cómo se obtienen los *datasets* hasta llegar a cómo se ha generado la red neuronal y qué parámetros se han utilizado en ella.
Todo este proceso se muestra en la siguiente figura:

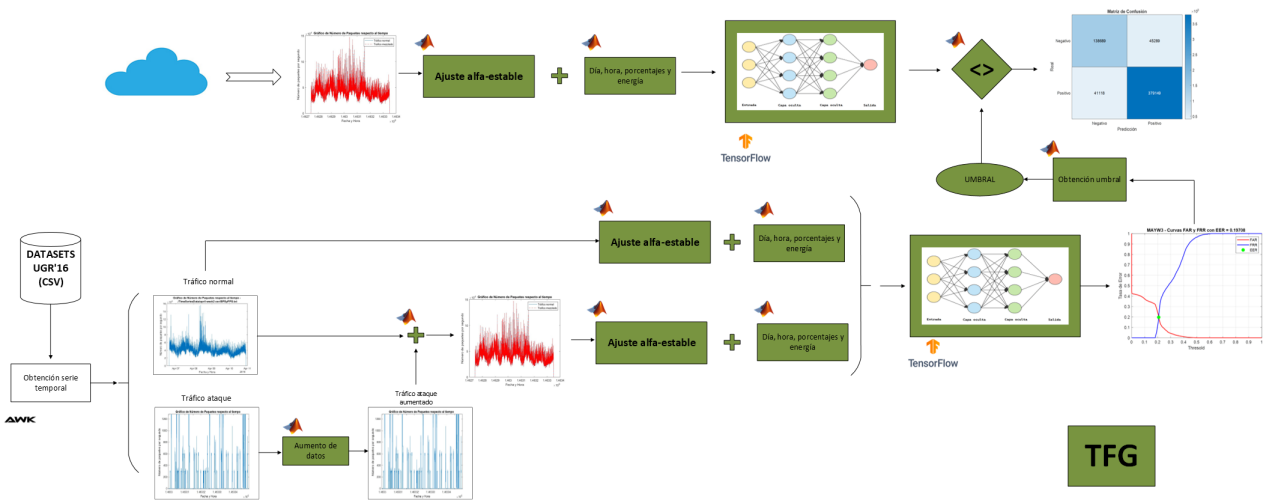

Figura 3.1: Diagrama con las fases realizadas en el trabajo.

Primero, se obtiene la serie temporal de los datasets UGR'16 [4] mediante scripts en AWK. Una vez generados estos ficheros, se procesan para obtener ficheros de tráfico normal y de tráfico con ataque. Luego, se ajustan los datos utilizando un modelo alfa-estable y se añaden parámetros adicionales para su posterior procesamiento en la red neuronal. Finalmente, con los resultados obtenidos y el uso de umbrales, se evalúa la capacidad del modelo para detectar ataques y se determina su viabilidad para la implementación en un sistema real.

3.2. Obtención del conjunto de datos

Para la obtención del *dataset* se sigue el método usado en el TFM realizado por Benjamín Martín Gómez [3]. Primeramente, se obtienen las series temporales de la página web de la Universidad de Granada, el conjunto de datos UGR'16 [4] es un *dataset* construido con tráfico real obtenido de un ISP, el cual está divido en dos conjuntos, un conjunto de datos de calibración y otro de prueba. Además, estos ficheros también están divididos por meses y semanas, el conjunto de calibración va de marzo a junio de 2016 y el de test está formado por los meses de julio y agosto. Respecto a los ataques solo hay un par de ficheros de ataques de DoS, de los cuales se comentará más adelante. De todos estos ficheros solamente se cogieron algunos debido a que otros ficheros estaban contaminados con alguna anomalía en el tráfico.

Para poder empezar a trabajar con ellos, se debe estar en un sistema operativo Linux, para poder ejecutar un par de *scripts* desarrollados por Benjamín, que con la herramienta AWK consiguió pasar de un fichero de datos en formato de flujos a un fichero de texto. Lo primero es descargar los ficheros de las semanas que queramos, que tienen extensión .tar, y tras esto, descomprimirlos en un directorio cualquiera. Una vez los ficheros estan en un directorio, se llevan a ese directorio dos *scripts*, "tref.sh" el cual nos imprime por el terminal el primer segundo en POSIX del que se tienen los datos, es decir el tiempo de referencia y también se mueve "process.sh" del cual se modifica la variable "tref" por el valor obtenido por pantalla tras ejecutar el *script* anteriormente comentado, con este se obtiene el fichero "BPSyPPS.txt".

El fichero de texto que se obtiene es la serie temporal con el ancho de banda, tiene tres columnas, la primera es el tiempo en formato UNIX, el segundo el numero de bytes/seg trasmitidos y la tercera columna el número de paquetes/seg transmitidos.

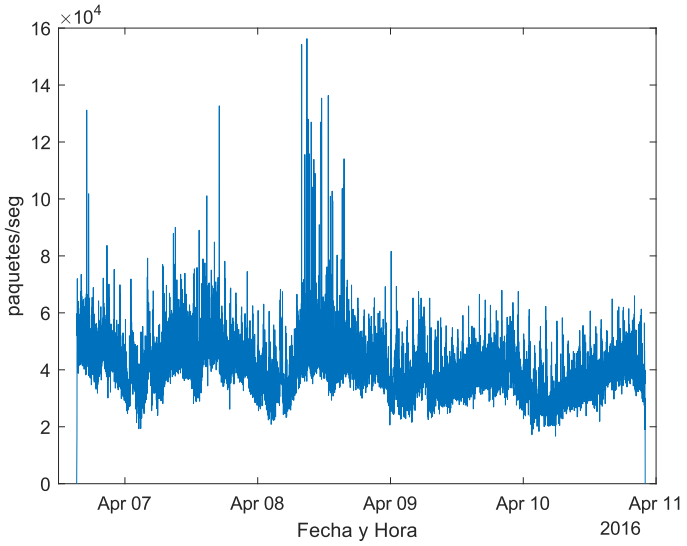

Figura 3.2: Representación de una serie temporal en paquetes por segundo.

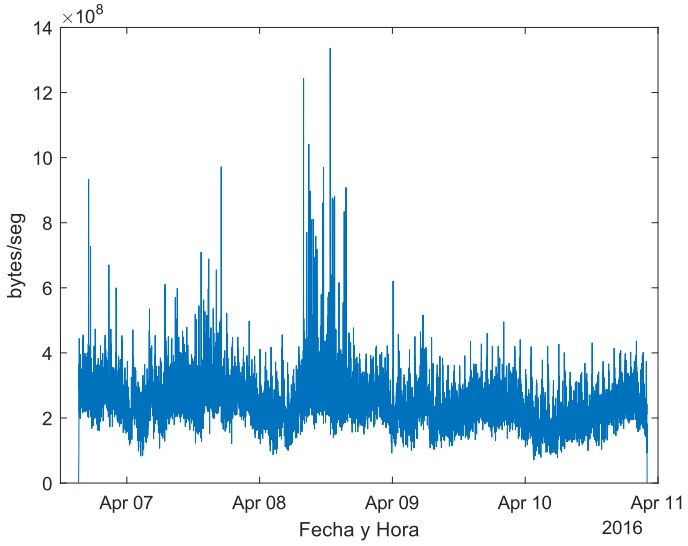

Figura 3.3: Representación de una serie temporal en bytes por segundo.

3.2.1. Obtención y generación de los ficheros de ataque

Una vez obtenidas las series temporales del tráfico normal, es decir sin ataque, se necesita obtener las que tienen los ataques DoS comentados anteriormente. En la página de la Universidad de Granada solamente hay ficheros de estos ataques para las dos primeras semanas de agosto y para la quinta semana de julio, tras realizar el mismo procedimiento de antes se obtienen unas series temporales tal que así:

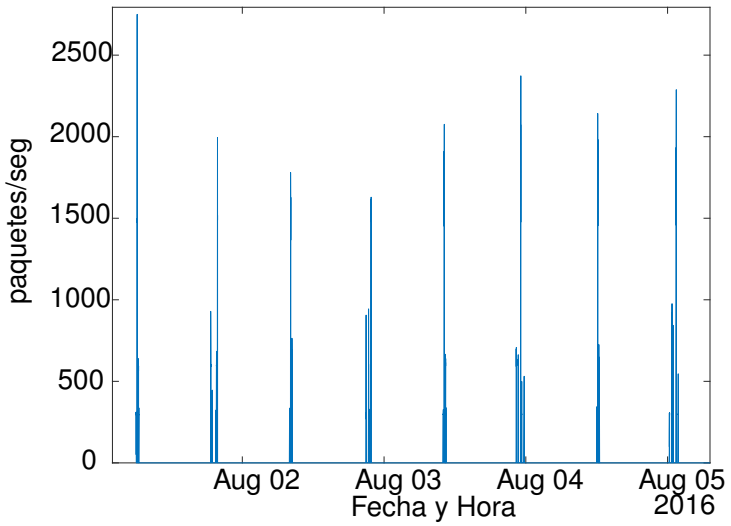

Figura 3.4: Representación de una serie temporal de un ataque en paquetes por segundo.

Tras ver que no hay ataques para todos los meses y semanas se toma la decisión de mezclar estos ataques. Para ello con el uso de Matlab de estos tres ficheros se obtienen unas 119 bandas de ataque, estas se distribuyen aleatoriamente sin que se junten unas con otras en un *array* con el tamaño de la semana que fuera, para que al final se tuvieran 1000, 500 o 250 bandas por cada semana (dependen del tamaño de los ficheros).

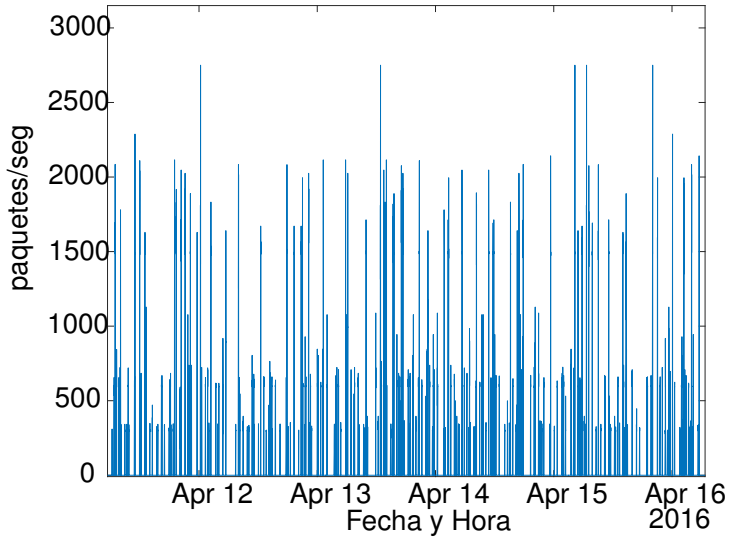

Figura 3.5: Ejemplo de la representación de un tramo del array de las 500 bandas, de las cuales no se junta una respecto a otra

Primeramente, se añadieron a los ficheros unas mil bandas por semana pero a la hora de realizar la red neuronal se observó que para los ficheros con menos valores no predecía bien la red neuronal, daba una predicción de los porcentajes bastante menor respecto a los valores reales. Esto era debido a que en los ficheros pequeños se tenían más ataques y por lo tanto los porcentajes subían, lo que hacía que se tuviera que nivelar para tener en todos los ficheros valores parecidos. Este es un error común a la hora de introducir datos a cualquier sistema de aprendizaje automático. Es importante tener los datos balanceados antes de introducirlos en una red neuronal porque un conjunto de datos desbalanceado puede sesgar el modelo y afectar a la capacidad de generalización.

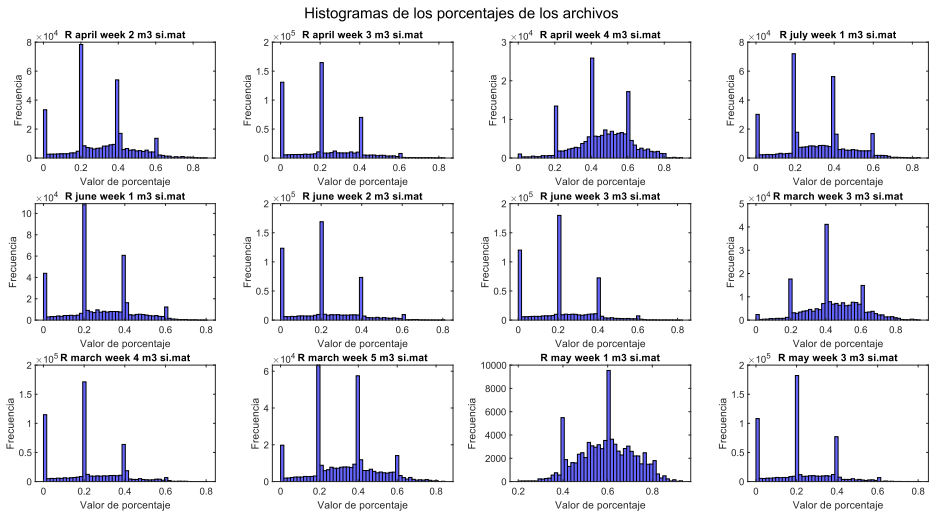

Figura 3.6: Histograma con la distribución de los porcentajes de ataque que hay en cada ventana antes de ajustarlos

Como se puede apreciar hay varios ficheros que casi no tienen zonas en las que no hay ataque y por lo tanto tienen más densidad en porcentajes más altos, estos son los ficheros de la semana 4 de abril (179 mil valores), la tercera semana de marzo (219 mil valores) o la primera de mayo (85 mil). Para poder hacer el reajuste, en vez de generar el *array* con 1000 bandas, se utilizaron 500 o 250 bandas para que esos ficheros tuvieran unas densidades parecidas a los otros ficheros, que tienen tamaños entre 300 mil y 600 mil valores.

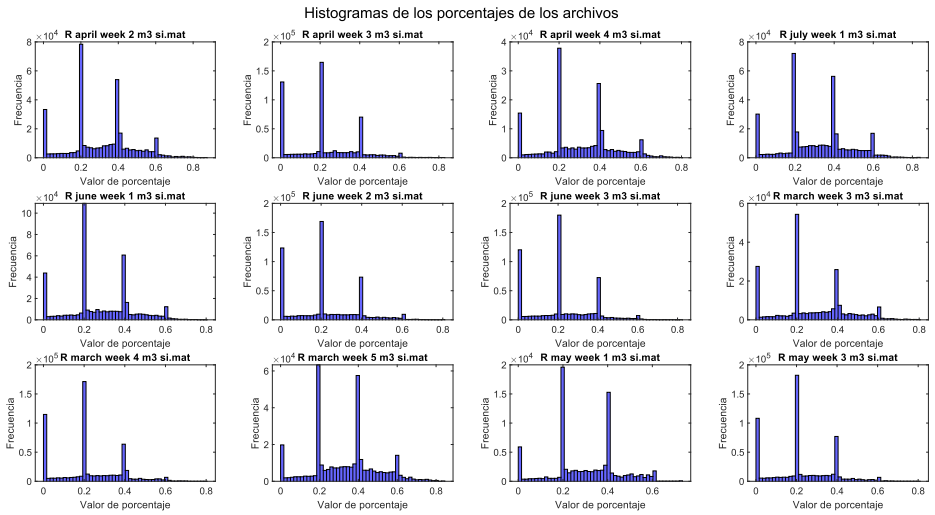

Figura 3.7: Histograma con la distribución de los porcentajes de ataque que hay en cada ventana después de ajustarlos

Una vez distribuidos de una forma más uniforme ya los se pueden juntar con el tráfico normal, donde nos queda la siguiente representación:

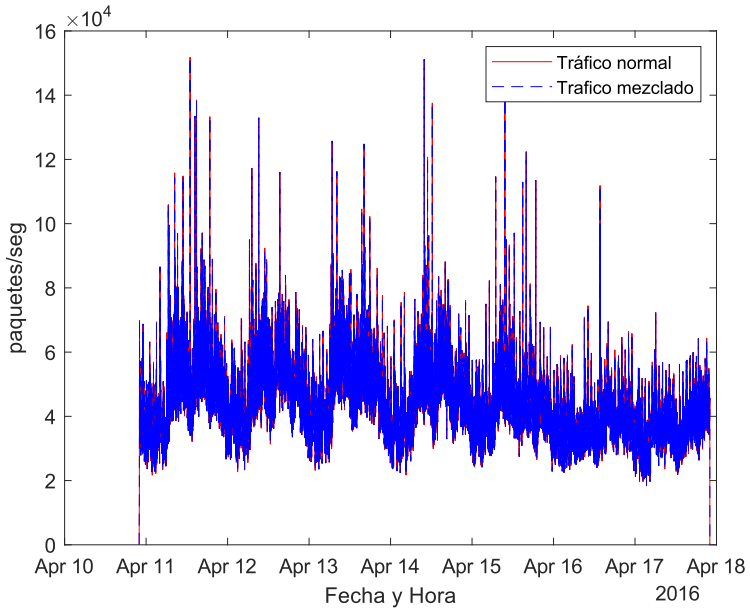

Figura 3.8: Representación de los paquetes/s del tráfico normal con el tráfico mezclado para la tercera semana de mayo

En esta gráfica se puede apreciar que casi no hay diferencias entre el tráfico normal y el mezclado, esto es debido a que los ataques son muy sutiles, es decir que se introduce al tráfico muy pocos paquetes/seg.

3.3. Cálculo de los parámetros necesarios de la red neuronal

Las redes neuronales son modelos computacionales inspirados en el cerebro humano que aprenden patrones a partir de datos. Para su correcto funcionamiento, requieren parámetros que describen las entradas y ayudan en la toma de decisiones y predicciones. Estos parámetros permiten generalizar a nuevos datos y optimizar el entrenamiento.

En el caso bajo estudio, los parámetros más importantes son los cuatro parámetros alfa-estables: *alfa, beta, gamma* y *delta*, estos dan información sobre la estabilidad de los datos, su simetría, la escala o información sobre la localización. Con estos datos ya se podría apreciar la diferencia entre unos datos con y sin ataque, por lo que podría ser suficiente para pasarlos a la red neuronal, pero en este caso para que tenga más datos y para poder afinar más la red se optó por añadirle otras variables como el día y la hora en la que empieza la ventana o unos parámetros que indican la evolución del ataque dentro de la ventana.

3.3.1. Cálculo de los parámetros de una distribución alfa-estable

Para poder realizar todos estos cálculos se ha utilizado MatLab, que es la herramienta más sencilla para poder trabajar y representar este tipo de datos. Para ello, del fichero “BPSyPPS.txt” obtenido antes, se extraen las tres columnas y se usa solamente la columna de paquetes/seg, aunque también se podría trabajar con la de *bytes/seg*, pero se comentó en trabajos anteriores que era mejor operar con paquetes por segundo.

Con una de estas columnas se pasa a la función creada, la cual tiene 3 argumentos, los paquetes/seg o *bytes/seg* como primer argumento, el tamaño de la ventana que se desee y por último un número dependiendo del método que se quiera usar. Esta función recorre los datos usando ventanas temporales, es decir que a los datos que se pasen se irán calculando sus parámetros por partes que se van deslizando segundo a segundo. Por ejemplo, si se tiene una ventana de 900 valores, primero se calcularán los parámetros de array de datos del 1 al 900, en la siguiente iteración se calculan los de 2 al 901, etc. Para este trabajo se han utilizado ventanas de 15 minutos (900 valores) porque es suficiente tiempo para poder recoger en ellas los patrones y características del tráfico sin excedernos de tiempo, pero se puede elegir el valor que se desee.

Primeramente, esta función calcula el número de ventanas totales mirando la longitud de los datos pasados y restándoles el tamaño de una ventana para no tener ventanas incompletas, con esto se genera un bucle que recorrerá todas las ventanas y calculará sus parámetros.

Los cuatro parámetros que se obtienen son los siguientes:

- **Alfa (α):**describe la estabilidad y tiene un rango de $(0, 2]$.
- **Beta (β):**describe la asimetría de la distribución y comprende el rango de $[-1, 1]$.
- **Gamma (γ):** define la escala y está comprendida entre $(0, \infty)$.
- **Delta (δ):** representa la localización relativa de la función, en el rango de $(-\infty, \infty)$.

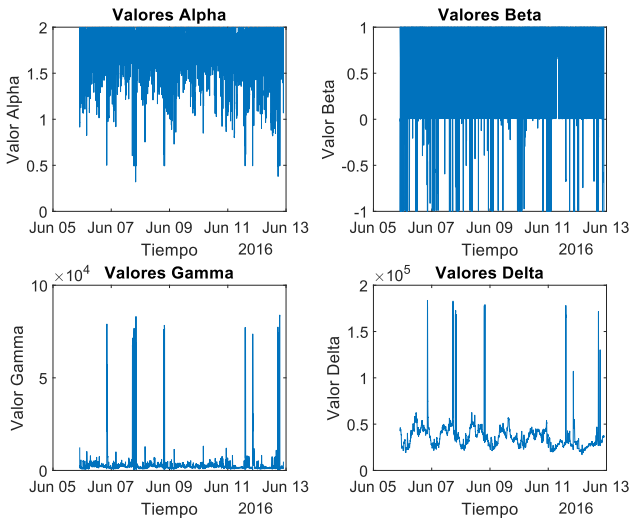

Figura 3.9: Representación temporal de los 4 parámetros alfa-estables.

Durante la duración de este trabajo se han ido probando diferentes métodos para la obtención de los parámetros alfa-estables, entre ellos se han usado los siguientes:

Se comenzó a trabajar con el método que proporcionaba Matlab: "*Alpha stable distributions for MATLAB by Mark Veillete*" [17], con la función "stblfit". En este método debes de especificar qué método quieres que se use para estimar los parámetros, si se pasa como argumento "ecf", ajusta los 4 parámetros a la función característica empírica de los datos basándose en el estudio de Koutrouvelis durante 1980 y 1981 [18] [19], y si se pasa como argumento "percentile", estima 4 parámetros en función de varios percentiles de los datos, siguiendo el modelo de McCulloch [20], de esta forma se obtenían los parámetros de una forma más rápida pero algo menos exacta.

También Matlab proporciona la función "fitdist" que ajusta la distribución que se desee a los datos que se pasen, pero esta función llama a otra función llamada "stablefit", la cual con estos datos daba el siguiente error:

"*Parameter estimation does not converge to a valid range. The shape parameter ALPHA must be a real numeric scalar such that 0<ALPHA<=2.*"

Este error no era coherente ya que con los otros métodos que el valor de alfa se mantenía en ese rango y no se pudo solucionar, por lo que este método se descartó.

Después también se utilizó otro método desarrollado por investigadores de la Universidad de Valladolid: "*Libstable: Fast, Parallel and High-Precision Computation of-Stable Distributions in C/C++ and MATLAB*" [21].

En este caso se tiene una librería de C/C++ y un *frontend* para Matlab en el se puede evaluar de forma rápida, paralelizada y de alta precisión funciones de densidad y distribución, generar variables aleatorias siguiendo una distribución o estimar parámetros alfa-estables. Al trabajar en Matlab, primeramente, se ha tenido que cargar la librería y sus diferentes dependencias. Con todo esto ya realizado se proporcionan 4 funciones, la primera es para realizar una estimación inicial de los parámetros usando el método de McCulloch, estos parámetros son una primera aproximación lo que hace que no sean muy precisos y no se puedan tomar como un método válido, estos parámetros iniciales se necesitan para los otros 3 métodos.

Uno de los métodos principales es el de Koutrouvelis, que utiliza la función característica empírica y aplica regresiones lineales recursivas en su gráfica log-log. En un primer paso, estima los parámetros alfa y gamma, y en un segundo paso, beta y delta. Aunque este método ofrece ventajas, requiere un mayor esfuerzo computacional en comparación con el método de los cuantiles. Los dos métodos restantes son los de Máxima verosimilitud (ML, *Maximum Likelihood*), los cuales son considerados como los más precisos. En estos se utilizan métodos numéricos para aproximar la función de densidad de probabilidad (PDF) para maximizar la verosimilitud de la muestra, lo que conlleva un alto costo computacional debido a las numerosas evaluaciones de las PDFs para maximizar la verosimilitud en

el espacio de parámetros en 4 dimensiones. Para reducir el coste computacional hay otra función en la que se trabaja con un espacio de 2 dimensiones (en el espacio de alfa y beta), en la que cada iteración de la maximización, gamma y delta son estimados con el método de McCulloch acorde con las estimaciones de alfa y beta.

Por último, se usó el trabajo realizado por Federico Simmross-Wattenberg llamado: "*Fast calculation of alpha-stable density functions based on off-line precomputations. Application to ML parameter estimation*" [15].

Para este caso se proponía un método en el que se necesitaba cargar una tabla que tenía almacenada distribuciones estables normalizadas en puntos estratégicos dentro del espacio de parámetros, para que se consiguiera calcular cualquier conjunto de parámetros, pero no se profundizará en ello. En este caso los datos necesitaban estar normalizados, para normalizarlos se utilizaron dos estimaciones, una para gamma y otra para delta. La delta es la moda de los datos para usarla como media y la gamma es la desviación típica. La desviación típica se calcula después de eliminar los valores atípicos, es decir, aquellos que quedan fuera del rango [Q1 - 1.5(Q3 - Q1), Q3 + 1.5(Q3 - Q1)], siendo Q1 el percentil 25 y el Q3 el percentil 75.

$$x_{\text{norm}} = \frac{x - \mu}{\sigma} \tag{3.1}$$

Con este método se ha trabajado de dos formas, la primera quedándote con los estimadores como los propios valores de gamma y delta y la otra forma que es usando el mismo método pero aplicando al final un estimador de máxima verosimilitud de esos valores, con el cual se han obtenido los mejores resultados. Este estimador es bastante utilizado en diferentes aplicaciones, por ejemplo se usa en la aplicación Distribution Fitter de Matlab [22] pero este usa la función “fitdist” comentada antes, la cual tiene un error para algunas ventanas, por lo que en esas ventanas se usó el método de máxima verosimilitud de dos dimensiones como solución porque seguía dando unos valores parecidos a los que se obtenían con un tiempo de ejecución reducido.

Método	**Nombre**
1	STBL: Alpha stable distributions for MATLAB by Mark Veillete (ECF)
2	STBL: Alpha stable distributions for MATLAB by Mark Veillete (Percentile)
3	Koutrouvelis
4	Modified 2D Maximum Likelihood Estimation
5	Maximum Likelihood Estimation
6	Fast calculation by Federico Simmross using the estimations
7	Fast calculation by Federico Simmross with MLE, if error MLE 2D

Tabla 3.1: Tabla con los métodos disponibles.

Ahora con todos lo métodos para obtener los parámetros se pasa al proceso de comparación de los métodos para conseguir los parámetros alfa-estables que se usarán en la red neuronal. Para este proceso se ha comparado la distancia de Kolmogorov-Smirnov, el tiempo de ejecución y la evolución de los parámetros.

Las distancias de Kolmogorov-Smirnov se obtienen primeramente aplicando la técnica de *bootstrapping* a los datos para romper la dependencia estadística y después las distancias se calculan con la función “kstest2”. Para evaluarlo se usa la distancia media como la evolución de las distancias en cada paso de las ventanas. Los otros parámetros que se evalúan son simplemente la desviación de esas distancias, el tiempo que tarda en ejecutarse el cálculo y la comprobación de que representado los valores en un histograma se puede ver que el ajuste se parece a la forma que se muestran los datos.

	Método 1	Método 2	Método 3	Método 4	Método 5	Método 6	Método 7
Tiempo(s)	0.022524	0.0060582	0.190234	0.378927	24.28004	0.438584	0.53579
Distancia	0.167742	0.170109	0.075736	0.043859	0.080636	0.110412	0.038696
Desviación	0.0733273	0.0720275	0.01352	0.0125238	0.0650268	0.0188197	0.0121453

Tabla 3.2: Tabla para la semana 3 de abril.

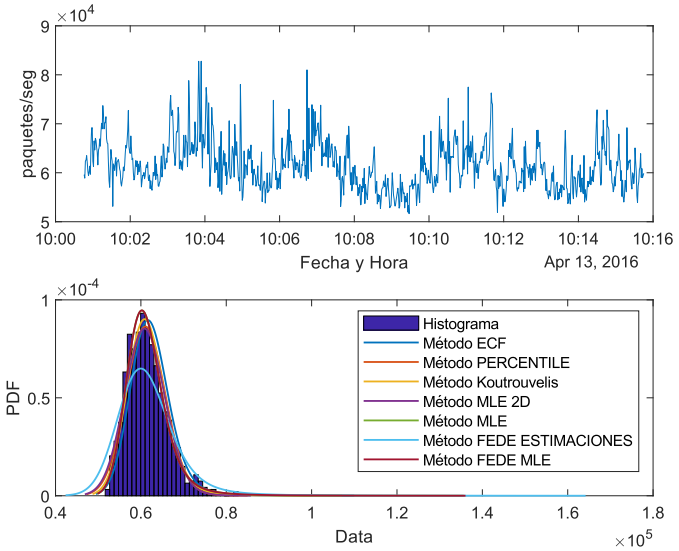

Figura 3.10: Representación temporal de un histograma y sus ajustes para una ventana de la semana 3 de abril.

Este es un ejemplo de la forma de comparar los diferentes ajustes, en este caso se ha representado una ventana de la tercera semana de abril. Por norma general, el que mejor ajusta en el histograma y tiene la menor distancia de K-S es el método 7, en el histograma es "Método FEDE MLE", en el que se usa el estimador de máxima verosimilitud y si salta el error el método de MLE en dos dimensiones, pero este grado de acierto se consigue con un tiempo computacional mayor.

La tabla muestra la media de lo que tarda en ejecutarse una ventana, la distancia K-S media y desviación media de las distancias de K-S que tiene cada método para cada ventana. Tras observar estos valores, finalmente se decide que el método que más interesante era el método 7, el que usa la técnica de Federico con el MLE añadiendo la corrección de errores ya que se obtiene la mayor precisión para poder decir si se tiene ataque o no, además de una desviación media reducida. El único inconveniente de este método es el tiempo de ejecución, pese a no ser excesivamente elevado es un tiempo notable, pero se concluye que se podría realizar la red neuronal trabajando con estos tiempos ya que al tener en este caso tiempos de ejecución de algo más de medio segundo por ventana en ventanas de 15 minutos, sería suficiente para detectar el ataque a tiempo.

3.3.2. Cálculo del resto de parámetros

Además de los parámetros anteriores, se decidió que para aumentar la capacidad de aprendizaje de la red neuronal, para mejorar la precisión y para evitar el fenómeno del *underfitting*(subajuste) entre otros motivos se debería de añadir algunos parámetros más. Entre esos parámetros nuevos están:

- Día de la semana en la que empieza la ventana (`es_lunes`, `es_martes`, `es_miercoles`, ...).
- La hora en decimal en la que empieza la ventana.
- Porcentajes.
- Energía.

Para determinar los días de la semana en los que comienza la ventana, se ha utilizado la columna de tiempos en formato UNIX de los ficheros de Granada. Estos valores se convierten a un formato legible y, mediante una simple evaluación de si corresponden a los días 1, 2, 3,..., 7, se genera un *array* binario con ceros y unos, indicando la presencia o ausencia de cada día.

Para obtener la hora en formato decimal, se emplea el *array* previamente mencionado. A partir de los tiempos convertidos a un formato legible, se extraen las horas en valores decimales, que posteriormente se normalizan para facilitar su procesamiento en la red neuronal.

El parámetro de porcentajes calcula en cada ventana cuanto tiempo del total de la ventana había un ataque, este se obtenía con el *array* donde estaban los ataques sin mezclar y donde había un valor diferente a 0 (hay ataque) se pone un 1 en otro *array*, por lo que iterando todo el fichero y haciendo el sumatorio de los unos que había en esa ventana respecto al tamaño de ella, se obtenía el porcentaje.

En cuanto a la energía, ha habido varias ideas de como implementarla, las cuales no han llegado a unos buenos resultados. El primer enfoque fue obtener la energía de la ventana calculando la suma de sus paquetes/seg al cuadrado (potencia de la señal). Otro enfoque que se obtuvo más tarde debido a los malos resultados, fue obtener un factor entre 1 y 0, para que se pareciera lo máximo a como se estaba trabajando en la red con los porcentajes, este se obtenía de dividir la energía del ataque con la energía de todo el tráfico mezclado.

3.4. Diseño de la red

Una vez calculados todos los parámetros necesarios, se pasa a la parte de la detección. El primer enfoque iba dirigido a que una vez se tuvieran los estadísticos del tráfico se pudiera detectar con precisión y con rapidez si se está produciendo un ataque. Para ello se decidió que una buena forma para conseguir este mecanismo de detección sería usando una red neuronal, estas son capaces de detectar patrones complejos, tienen la capacidad de procesar grandes volúmenes de datos, si están bien entrenadas tienen tasas bajas de falsos positivos y negativos, son fáciles de adaptar y ajustarse gracias a la infinidad de documentación y librerías que hay sobre ellas.

La red fue realizada en JupyterLab [23], que es una interfaz de usuario en la que se trabajan con Jupyter Notebooks, los cuales son documentos interactivos donde se puede trabajar con diferentes lenguajes (en este caso con Python), se puede dividir el código por celdas ayudando a la depuración y detección de errores, permiten la integración de diferentes bibliotecas, se puede combinar código con texto, con visualizaciones, etc. Una vez decidido el entorno, se usaron diferentes librerías, como pueden ser "scipy.io" para leer los ficheros de Matlab, "pandas" para el manejo y análisis de datos tabulares o "numpy" para cálculos numéricos y manipulación de arreglos o matrices, pero hay que destacar dos librerías imprescindibles para realizar nuestra red neuronal:
Las librerías de "sklearn" [24] y "tensorflow" [25] están relacionadas con el *Machine Learning* y *Deep Learning* para el procesamiento de datos, evaluación de los modelos y construcción de las propias redes neuronales. En este caso se utilizó "tensorflow.keras" para la creación de la red neuronal donde se usa para apilar las capas de la red, para definir las entradas y salidas de ella, para definir tipos de la red o para usar algunas herramientas para el buen funcionamiento de la misma. También se usó "sklearn.metrics" para evaluar el modelo usando el MAE, el MSE o el R2 *Score*, "sklearn.preprocessing" para el escalado de los datos, entre otros.

En la red se han usado 24 ficheros, los cuales están divididos en semanas y en ficheros con ataque y sin ataque. Para garantizar que funcione correctamente la red se necesita separar esos datos en ficheros de entrenamiento y de prueba(*test*) ya que si no se reparten y usas de prueba los mismos datos con los cuales la has entrenado, puede ocurrir que la red memorice los datos y se produzca *overfitting*. En este caso bajo estudio, al tener 24 ficheros, se ha repartido en 75 % archivos de entrenamiento y 25 % para los ficheros de prueba, es decir 18 para entrenamiento y 6 para las pruebas.

En cada fichero hay los siguientes parámetros:

- Parámetros alfa-estables (α, β, δ y γ)
- Días de la semana en la que empieza la ventana (`es_lunes`, `es_martes`, `es_miercoles`, ...)
- La hora en decimal en la que empieza la ventana
- Energía
- Porcentajes

Con los datos cargados y adecuados para poder trabajar con ellos (convertirlos en *arrays*, escalarlos...) ya se puede empezar a construir la red neuronal. En este caso tras realizar varias pruebas se optó por realizar una red LSTM (*Long Short-Term Memory*) que es un tipo de Red Neuronal Recurrente (RNN) que está diseñada para aprender patrones sobre datos secuenciales, que es lo que se estaba buscando ya que los parámetros están siendo obtenidos de ventanas deslizantes, las cuales dependen de los valores anteriores. Para poder ajustar y mejorar esta red se pueden añadir más capas LSTM, probar diferentes funciones de activación, meter más épocas, añadir o reducir el tamaño de las capas, de las épocas, etc. Estos ajustes para mejorar la red se verán en el siguiente capítulo.

3.5. Conclusión

Tras haber extraído todos los ficheros correctamente y calculado sus 4 parámetros alfa-estables con diferentes métodos para poder caracterizar el tráfico, se puede concluir que el método que mejor se adapta al estudio es el método 7 (uso de MLE y si da error se usa el método de MLE en dos dimensiones) debido a su alta precisión y su tiempo de ejecución razonable para cada ventana. Este método primero calculaba una primera estimación de los 4 parámetros para luego usarlos en el cálculo definitivo con la función "stablefit", de esta función se obtienen alfa y beta ya que gamma y delta se obtenían de la estimación. Finalmente se usaba la función del Estimador de Máxima Verosimilitud (MLE) para precisar más los parámetros de la distribución estable a partir de los datos enviados y los 4 parámetros calculados previamente. Con los 4 parámetros calculados y otros adicionales, se pasa a la red neuronal LSTM, en la que en el siguiente apartado se comentarán las diferentes pruebas realizadas con ella y se analizarán sus resultados.

4

Pruebas y resultados

4.1. Introducción

Con los parámetros calculados y la red neuronal diseñada, se pasa a ver las pruebas y los resultados de la red neuronal. Como se ha comentado antes, a la red neuronal se le pueden cambiar ciertos parámetros para buscar un mejor rendimiento. En este caso se ha probado cambiando el tipo de red, el número de capas, de épocas, el tamaño de las capas o cambiando las funciones de activación. Este capitulo se dividirá en dos partes, la primera centrada en la predicción del porcentaje de ataque que hay una ventana y otro con la predicción de la energía. En la última parte parte se hablará sobre la elección de umbrales y otras métricas . Para poder representar estos resultados se hará el uso de matrices de confusión, representaciones de umbrales, de diagramas de dispersión o simplemente gráficas con la predicción y su valor real.

4.2. Resultados de las predicciones

4.2.1. Predicciones de los porcentajes de ataque

En esta primera parte se ha probado con varios tipos de redes neuronales. En las primeras simulaciones se empezó con una red que no era LSTM y con una salida que no daba entre 0 y 1 (más tarde se corregiría con una capa densa final con una función de activación 'sigmoid'). Además, en este caso se utilizan 3 de las funciones de activación más conocidas para poner a prueba la red, la función de activación 'tanh' (tangente hiperbólica), 'ReLu' (Rectified Linear Unit) y 'Swish'.

Las funciones de activación desempeñan un papel crucial en las redes neuronales, ya que permiten que el modelo aprenda patrones complejos. Sin ellas, la red solo podría realizar transformaciones lineales, lo que limitaría su capacidad para representar relaciones no lineales en los datos.

La función de activación tangente hiperbólica genera valores en el rango de -1 a 1, lo que la centra en 0 y proporciona gradientes más fuertes, favoreciendo una convergencia más rápida. Sin embargo, una de sus desventajas es el problema del desvanecimiento del gradiente, donde los gradientes se

vuelven demasiado pequeños para valores extremos, dificultando el aprendizaje en capas profundas. La función ReLU (*Rectified Linear Unit*) devuelve 0 para valores negativos y mantiene los valores positivos sin cambios. Se caracteriza por su eficiencia computacional y por ayudar a mitigar el problema del desvanecimiento del gradiente, lo que la hace muy utilizada en redes neuronales profundas. Por último, la función de activación 'Swish' suaviza la activación en comparación con ReLU y puede mejorar el rendimiento en ciertas arquitecturas profundas. Sin embargo, su principal inconveniente es su mayor costo computacional debido a la función sigmoide involucrada en su cálculo [26].

Para poder comparar los diferentes rendimientos de la red se han usado 3 métricas:

- **MSE (*Mean Squared Error*):** promedio de los errores al cuadrado entre las predicciones del modelo y los valores reales.
- **MAE (*Mean Absolute Error*):** promedio de los valores absolutos de las diferencias entre las predicciones y los valores reales.
- **R^2 Score:** coeficiente que determina la calidad del modelo para replicar los resultados. En este caso será la métrica que más se tendrá en cuenta.

Con este primer modelo se obtuvieron los siguientes resultados:

TIPO	MSE	MAE	R2 SCORE
Epoch = 10 y Batch_size = 64 Activación = TANH	0,02877	0,0939	0,26330
Epoch = 10 y Batch_size = 64 Activación = RELU	0,032739	0,103577	0,16186
Epoch = 10 y Batch_size = 64 Activación = SWISH	0,05021	0,13206	-0,2855

Tabla 4.1: Tabla comparando los resultados de las diferentes funciones de activación.

Se puede apreciar que se obtienen unos coeficientes de R2 Score bajos, además de que para la función de activación 'Swish' se tiene un coeficiente negativo, el cuál indica que la predicción es incluso peor que usar la media de los datos, lo que hace que esta función de activación se descartara para las siguientes pruebas. Respecto a los MSE y MAE, no se le ha dado mucha importancia ya que el conjunto de datos tiene millones de ventanas por lo que se tenga algún valor por encima o por debajo es meramente orientativo.

Tras realizar un par de pruebas más, se llega a la versión casi definitiva, resumiendo los cambios realizados, se llega a una red usando 3 capas LSTM (una capa más respecto a las pruebas anteriores y además se incrementa el número de neuronas, usando 100, 50 y 25) y una capa final densa con la función de activación 'sigmoid' para tener la salida acotada entre 0 y 1. Donde se obtienen los siguientes resultados:

TIPO	MSE	MAE	R2 SCORE
Epoch = 50 y Batch_size = 32 Activacion = RELU	0,016812	0,0692155	0,56959
Epoch = 50 y Batch_size = 32 Activacion = TANH	0,011739	0,0560805	0,699458

Tabla 4.2: Tabla de comparación de las diferentes funciones de activación para la segunda versión de la red.

Con estas métricas, se concluyó que la red se acercaba bastante a lo que se estaba buscando, alcanzando un R^2 Score cercano a 0.7.

Los mejores resultados que se obtienen tras unos pequeños cambios fueron los siguientes:

TIPO	MSE	MAE	R2 SCORE
Epoch = 50 y Batch_size = 32 Activacion = TANH	0.008013	0.0436302	0.749549

Tabla 4.3: Tabla con los mejores resultados obtenidos en la red.

Para esta versión de la red neuronal se obtuvieron estos *scatter plot* (diagramas de dispersión):

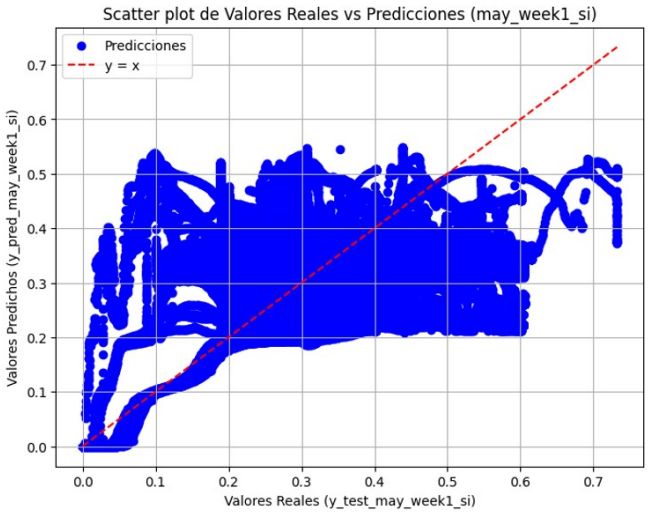

Figura 4.1: Diagrama de dispersión para la semana 1 de mayo.

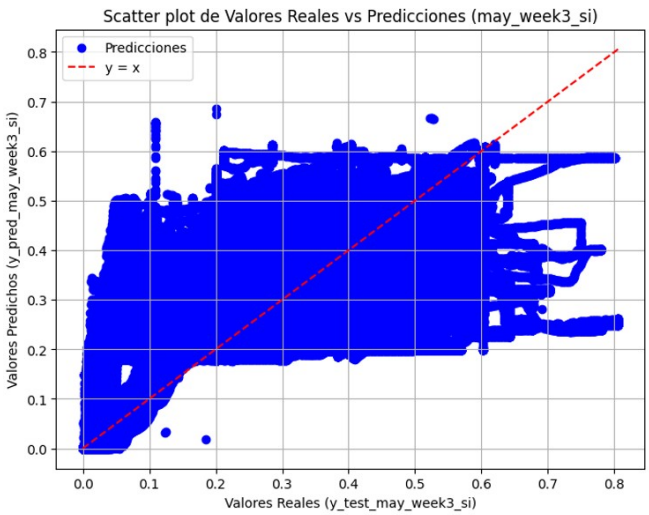

Figura 4.2: Diagrama de dispersión para la semana 3 de mayo.

Estos diagramas muestran la relación entre dos variables, en caso los valores reales y los predichos, idealmente si estos dos valores fueran iguales, la representación seguiría la línea roja. No es este caso, pero se puede ver que los datos siguen esa tendencia y tienen cierto parecido.

Ahora se continúa en Matlab para poder representar el *array* con los porcentajes reales y los sacados de la red neuronal. Primeramente, se ha separado el *array* de las predicciones en los respectivos meses que se tienen esos ficheros de test (march_w5, may_w1 y may_w3) para poder representarlos ya que la red neuronal da como salida un vector de 2 millones de ventanas (valores).

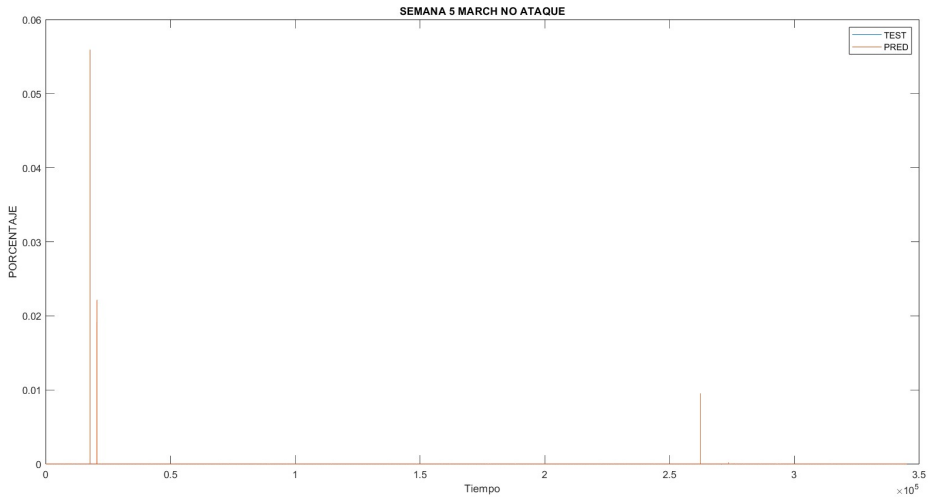

Figura 4.3: Representación de los porcentajes reales y los predichos para la semana 5 de Marzo sin ataque.

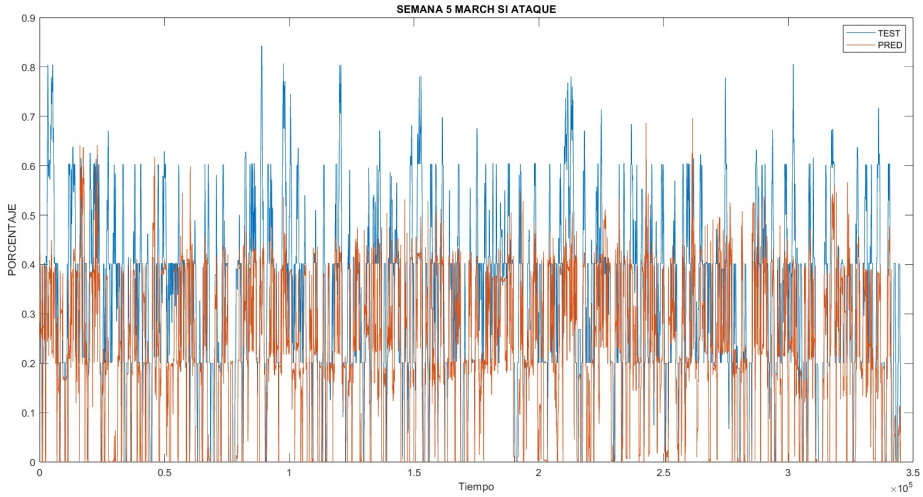

Figura 4.4: Representación de los porcentajes reales y los predichos para la semana 5 de Marzo con ataque.

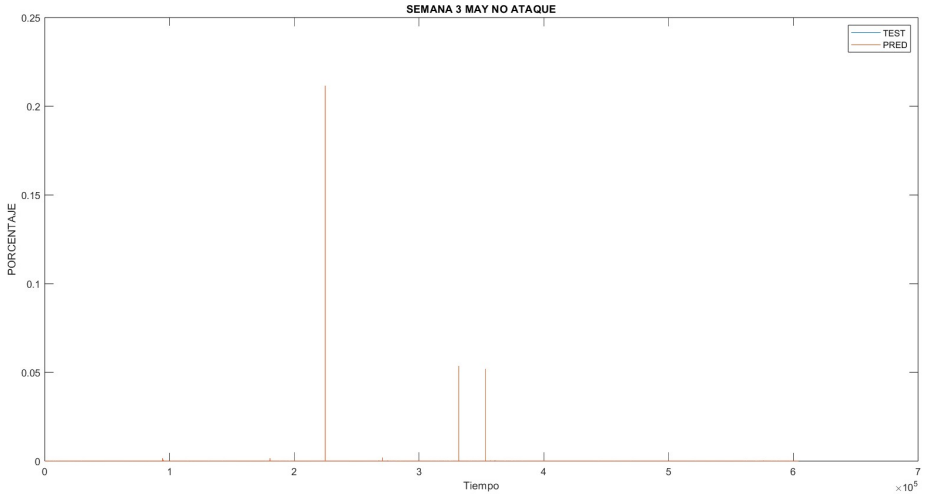

Figura 4.5: Representación de los porcentajes reales y los predichos para la semana 3 de Mayo sin ataque.

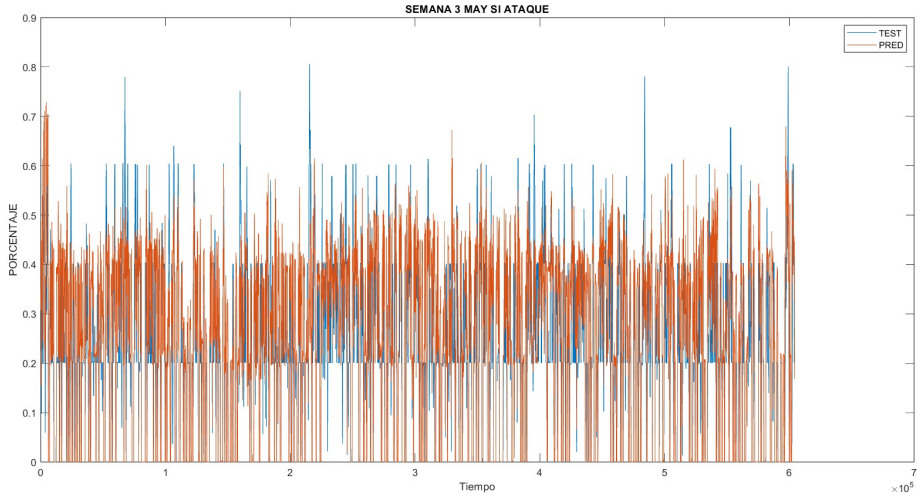

Figura 4.6: Representación de los porcentajes reales y los predichos para la semana 3 de Mayo con ataque.

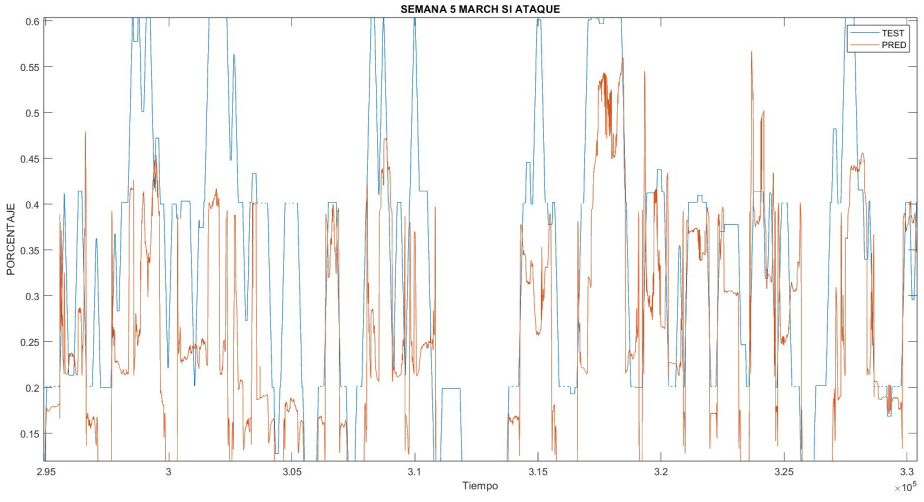

Figura 4.7: Representación de los porcentajes reales y los predichos para la semana 5 de Marzo con ataque, ampliando una zona del tráfico.

Como se puede apreciar, la red neuronal consigue que la predicción y los valores reales sean bastante parecidos, logrando detectar esos ciertos patrones y picos con bastante similitud como se ve en la imagen con el tráfico ampliado. Esto resalta el buen comportamiento de la red ya que el ataque que se ha introducido es muy sutil (contiene poca energía), es decir, se están añadiendo ataques que tienen un orden de magnitud de 10^2 a un tráfico que tiene un orden de magnitud de 10^4.

4.2.2. Predicciones de energía

En este apartado se ha intentado calcular la energía mediante dos enfoques, el primer enfoque fue básicamente obtener la energía de la ventana calculando la suma de sus paquetes/s al cuadrado (potencia de la señal) y el otro enfoque, que se obtuvo más tarde, para que se pareciera a los porcentajes de ataques, fue obtener un factor que se obtenía de dividir la energía del ataque con la energía de todo el tráfico mezclado.

Con el primer enfoque, la potencia de la señal, aparecieron varios problemas, el primero era que no se podía evaluar fácilmente los errores al igual que antes (MSE y MAE) ya que a la red se le pasaba la energía del tráfico mezclado y el que se obtenía era solamente del ataque. Otro problema fue que a la red neuronal se le tenían que pasar los parámetros normalizados y en este caso se tenía que normalizar la energía del trafico normal, el del tráfico mezclado con el ataque, el del ataque solo, el de la predicción que sacaba la red, lo que hacía que todo el proceso de normalización y desnormalización fuera muy complejo y que además en esos procesos se perdía más precisión de por sí. Igualmente, con este enfoque se obtuvieron unos resultados bastante más bajos que trabajando con los porcentajes, pero dentro de toda la problemática no eran tan malos.

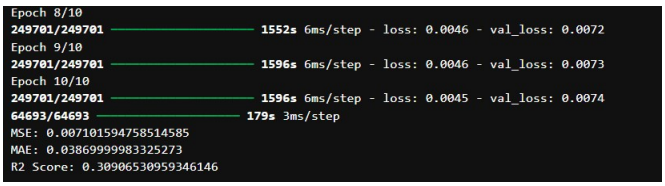

Figura 4.8: Salida de la red neuronal con los mejores resultados para el primer enfoque de energía.

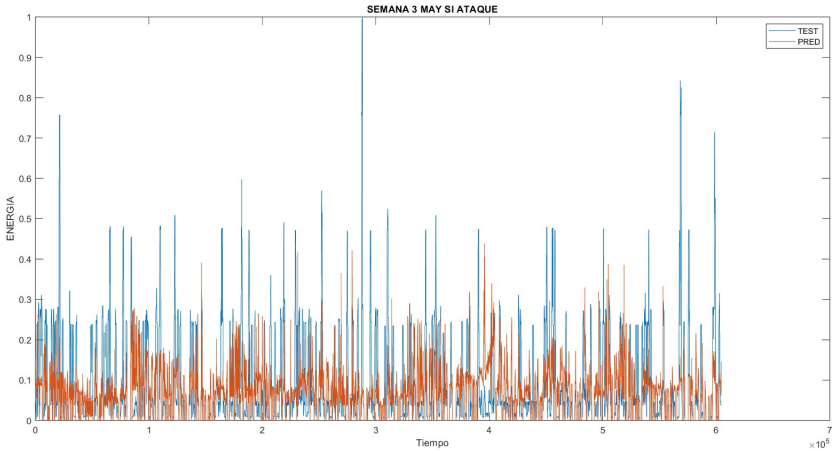

Figura 4.9: Representación de la energía real y la predicha para la semana 3 de Mayo con ataque.

En este caso se obtuvo un R2 Score de 0.3, que es bastante menor al 0.75 que se tenía prácticamente en el anterior apartado, no son muy malos resultados. Pero como se puede observar el rango de la energía está entre 1 y 0, el cuál debería ser desnormalizado pero ahí aparece el problema comentado antes de cómo se debería de realizar este proceso sin perder precisión.

El segundo enfoque pretendía simplificar las cosas ya que se estaría calculando igual que antes con los porcentajes un factor. En este caso se calculó el factor de dividir la energía del ataque con la energía de todo el tráfico mezclado, lo que hace que no hayan los problemas de des/normalización y de comparación que se ha comentado previamente. Pero para este caso no se pasó de un R2 Score de 0.25, tras probar ciertos cambios comentados antes como cambiar el número de capas, el tamaño de ellas, cambiar las épocas y sus tamaños, cambiar las funciones de activación, cambiar el tipo de capas (LSTM a GRU, por ejemplo), añadir el uso de *kernels* u optimizadores, entre otros cambios, no dieron resultado por lo que este apartado de predicción de energía no siguió hacia delante.

4.3. Umbrales de decisión

Finalmente, para complementar los resultados anteriores se decidió añadir un apartado donde mediante algún mecanismo se pudiera concretar cuando hay ataque y cuando no. Para ello se usó de referencia un proyecto realizado por Jorge Iraizoz y Alejandro Peña llamado “Proyecto de Clasificación de Ataques sobre dispositivos IoT” [16] de donde se usan las curvas ROC, FAR y FRR para poder elegir umbrales gracias al EER. Esto será de gran utilidad porque con este método se puede obtener un porcentaje que sirva de umbral para decidir si hay ataque en esa ventana o no y al ser este porcentaje el ERR (*Equal Error Rate*) se tendrá el mínimo error al usar ese porcentaje como decisor.

Primeramente, se calcula la curva ROC, donde se define un umbral para convertir los valores continuos del *test* en una clasificación binaria, con esto hecho ya se puede realizar el cálculo de la curva ROC, donde se comparan los valores reales binarios con las predicciones del modelo. El primer umbral fue uno que a partir del 20 % ya se detecta como ataque para poder ser preventivos.

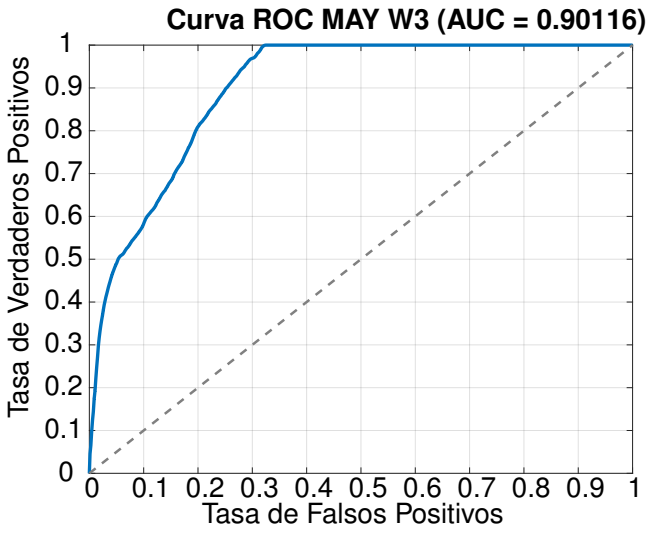

Figura 4.10: Curva ROC para un Umbral = 0.2 para la semana 3 de Mayo.

Con este porcentaje se obtiene un AUC bastante elevado por lo que de momento se va bien encaminado. Una vez elegido el porcentaje basandose en la curva ROC se puede pasar a las curvas FAR y FRR para poder afinar más este umbral.

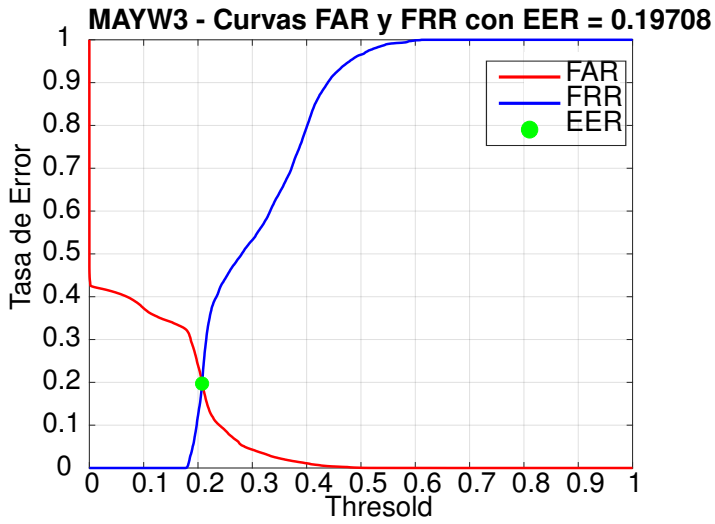

Figura 4.11: Curvas FAR y FRR donde se obtiene el ERR para la semana 3 de Mayo.

Se obtiene un EER = 0.19708 como porcentaje definitivo para detectar el ataque, ya sólo queda ver con las matrices de confusión cómo funcionaría el modelo con este umbral. Las matrices de confusión

son una gran herramienta para visualizar los resultados del modelo de clasificación, gracias a ellas se pueden ver los siguientes casos:

- **Verdaderos positivos (TP)**: Casos correctamente clasificados como positivos.
- **Falsos positivos (FP)**: Casos incorrectamente clasificados como positivos.
- **Verdaderos negativos (TN)**: Casos correctamente clasificados como negativos.
- **Falsos negativos (FN)**: Casos incorrectamente clasificados como negativos.

Y además se pueden extraer también las siguientes métricas:

- ***Accuracy* (Precisión)**: Mide el porcentaje de predicciones correctas realizadas en comparación con el total de predicciones.
- ***Recall* (Sensibilidad)**: Mide la capacidad del modelo para identificar correctamente todas las instancias positivas.
- ***F1-Score***: Métrica que combina la precisión y la sensibilidad.

Métrica	Valor
TP (Verdaderos Positivos)	379149
FP (Falsos Positivos)	45289
TN (Verdaderos Negativos)	138689
FN (Falsos Negativos)	41118
Accuracy	0,8570
Recall	0,9022
F1-Score	0,8977

Tabla 4.4: Tabla con las métricas sacadas de la matriz de confusión para la semana 3 de mayo.

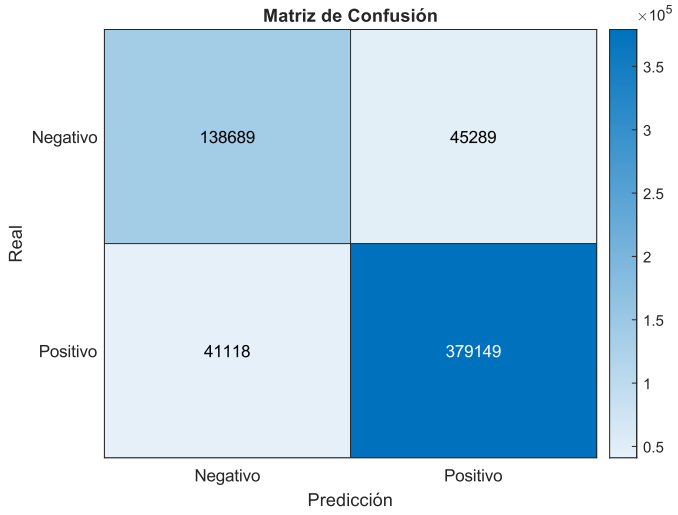

Figura 4.12: Matriz de confusión para la semana 3 de Mayo.

El modelo presenta un buen desempeño, debido a un recall alto y un F1-score elevado, lo que indica que es capaz de identificar correctamente la mayoría de los ataques. Sin embargo, se observa que la cantidad de falsos negativos es relativamente elevada, lo que representa un problema crítico para el sistema, ya que implica que algunos ataques no están siendo detectados y podrían pasar desapercibidos, comprometiendo la seguridad del sistema. Además, aunque el modelo logra una buena precisión, es importante considerar la reducción de falsos positivos. Un número excesivo de falsas alarmas podría generar alertas innecesarias, afectando la eficiencia del sistema y aumentando la carga de trabajo en la supervisión de los eventos.

Para mejorar estas métricas, se podrían explorar diferentes estrategias. Una opción sería ajustar los umbrales de clasificación para encontrar un punto óptimo entre recall y precisión, permitiendo reducir la tasa de falsos negativos sin comprometer demasiado la cantidad de falsos positivos o directamente se podría volver a tocar la arquitectura de la red neuronal. El rendimiento general del modelo es positivo, ya que la cantidad de verdaderos positivos es ocho veces mayor que la de falsos positivos. Esto sugiere que la red neuronal es efectiva en la identificación de ataques aunque con algunos ajustes adicionales se podría optimizar aún más su desempeño para su implementación en un entorno real.

4.4. Conclusión

Los resultados obtenidos muestran que la red neuronal tiene un buen rendimiento en la predicción de los porcentajes de ataque en una ventana, como se puede observar en los diagramas de dispersión, en las representaciones de los porcentajes reales y predichos o en las matrices de confusión. A pesar de que las predicciones no siguen exactamente la línea ideal, la red logra capturar tendencias y patrones relevantes en el tráfico de red, lo que valida su utilidad para la detección de anomalías. El único inconveniente es el mal comportamiento con la predicción de la energía, posiblemente debido a los problemas con las escalas, normalizaciones, datos mal balanceados, etc. Por lo que, finalmente, se puede concluir que se tiene un sistema que gracias a su precisión y rapidez podría ser utilizado en un entorno de trabajo real.

5

Conclusiones y trabajo futuro

5.1. Introducción

En este último capítulo se presentan las conclusiones derivadas del trabajo realizado, destacando los resultados más importantes y además se proponen nuevas líneas de investigación y mejoras que se podrían implementar a este trabajo.

5.2. Conclusiones

Tras terminar este trabajo se puede confirmar el buen desempeño de las redes neuronales como técnica de aprendizaje automático para detectar anomalías en la red, recalcando la sutileza de los ataques añadidos al tráfico ya que el tamaño del ataque era un par de ordenes de magnitud menor que el del tráfico normal. Estos buenos resultados se deben en gran parte a las distribuciones alfa-estables caracterizando el tráfico. Esta distribución se caracteriza por su capacidad de modelar colas pesadas, por trabajar con datos autosimilares y dependientes en el tiempo, por su flexibilidad al tener 4 parámetros que nos dejan generalizar a otras distribuciones conocidas, entre otras ventajas.

También se debería comentar que estos resultados pese a ser buenos, están sacados de un escenario creado a propósito para este trabajo, donde se han sintetizado los ficheros con los ataques DoS, se ha elegido el método para calcular los 4 parámetros alfa-estables, el tamaño de las ventanas o la forma de la red neuronal, por lo tanto, se debería de ver el desempeño de estas técnicas en otras situaciones para poder corroborar estos resultados, lo cual se podría analizar en trabajos futuros.

5.3. Trabajo futuro

De cara a un futuro se podría realizar lo siguiente:

- Llevar a cabo este estudio nuevamente, utilizando diferentes entornos y conjuntos de datos para evaluar la capacidad de generalización del enfoque propuesto.

- Intentar mejorar los resultados en el apartado de predicción de energía, con el objetivo de obtener un modelo de predicción más robusto que combine tanto los parámetros de energía como los porcentajes de ataque.
- Implementar este sistema en un entorno real para comprobar su rendimiento en condiciones prácticas.
- Mejorar el rendimiento de las predicciones explorando otros métodos de aprendizaje automático.
- Caracterizar el tráfico utilizando otras distribuciones para comparar su efectividad frente a las distribuciones alfa-estables.

Bibliografía

[1] Alberto Ruiz Santos, *Segmentación de tráfico en Internet mediante la clasificación de parámetros estadísticos. Trabajo Fin de Grado*. Universidad Autónoma de Madrid, 2019.

[2] Eric Crusi Mozota, *Estudio de ciberataques mediante el análisis de tráfico en Internet. Trabajo Fin de Grado*. Universidad Autónoma de Madrid, 2018.

[3] Benjamín Martín Gómez, *Estudio de la predictibilidad del tráfico en Internet para la detección de anomalías sutiles. Trabajo Fin de Grado*. Universidad Autónoma de Madrid, 2023.

[4] Gabriel Maciá-Fernández, José Camacho, Roberto Magán-Carrión, Pedro García-Teodoro and Roberto Theron, "UGR'16: A New Dataset for the Evaluation of Cyclostationarity-Based Network IDSs," Visitado: 28.11.2023. [Online]. Available: https://nesg.ugr.es/nesg-ugr16/

[5] ProyectamosTuWeb. Tipos de ataques en la red. Visitado: 30.03.2024. [Online]. Available: https://proyectamostuweb.com/ataques-en-la-red/

[6] James F. Kurose and Keith W. Ross, *Redes de computadoras, un enfoque descendente*, 7th ed. Pearson, 2010.

[7] Wikipedia, "Distribución estable," Visitado: 17.03.2024. [Online]. Available: https://es.wikipedia.org/wiki/Distribuci%C3%B3n_estable#cite_ref-5

[8] John P. Nolan, *Univariate stable distributions*. Springer, 2020.

[9] Wikipedia, "Aprendizaje automático," Visitado: 02.04.2024. [Online]. Available: https://es.wikipedia.org/wiki/Aprendizaje_autom%C3%A1tico

[10] Iberdrola, "Machine learning: definición, tipos y aplicaciones prácticas," Visitado: 02.04.2024. [Online]. Available: https://www.iberdrola.com/innovacion/machine-learning-aprendizaje-automatico

[11] H. P. Enterprise, "¿qué es el aprendizaje automático?" Visitado: 02.04.2024. [Online]. Available: https://www.hpe.com/lamerica/es/what-is/machine-learning.html

[12] Federico Jesús Simmross Wattenberg, *Detección de anomalías en el tráfico agregado de redes IP basada en inferencia estadística sobre un modelo alpha–estable de primer orden*. Universidad de Valladolid, 2009.

[13] Wikipedia, "Bootstrapping (estadística)," Visitado: 27.02.2024. [Online]. Available: https://es.wikipedia.org/wiki/Bootstrapping_(estad%C3%ADstica)

[14] C. A. Bollmann, *Network anomaly detection with stable distributions*. Naval Postgraduate school of Monterrey, California, 2018.

[15] Federico Simmross-Wattenberg, Marcos Martín-Fernández, Pablo Casaseca-de-la-Higuera and Carlos Alberola-López, "Fast calculation of -stable density functions based on off-line precomputations. application to ml parameter estimation," Visitado: 25.02.2024. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S1051200414003510?via%3Dihub

[16] Jorge Iraizoz Expósito y Alejandro Peña Fernández, *Proyecto de clasificación de ataques sobre dispositivos IoT, Proyecto para la asignatura de Procesado Avanzado de Señal para Multimedia*.

[17] Mark Veillete, "Alpha stable distributions for MATLAB," Visitado: 16.12.2023. [Online]. Available: https://es.mathworks.com/matlabcentral/fileexchange/37514-stbl-alpha-stable-distributions-for-matlab

[18] Ioannis A. Koutrouvelis, *Regression-Type Estimation of the Paramters of Stable Laws"Vol 75, No. 372*. JASA, 1980.

[19] ——, *An Iterative Procedure for the estimation of the Parameters of Stable Laws*. Commun. Stat. - Simul. Comput., 1981.

[20] J. H. McCulloch, *Simple Consistent Estimators of Stable Distribution Parameters"*. Commun. Stat. - Simul. Comput., 1986.

[21] Javier Royuela-del-Val, Federico Simmross-Wattenberg and Carlos Alberola-López, "Libstable: Fast, parallel and high-precision computation of-stable distributions in c/c++ and matlab," Visitado: 20.02.2024. [Online]. Available: https://www.jstatsoft.org/article/view/v078i01

[22] MatLab, "Distribution fitter," Visitado: 19.12.2023. [Online]. Available: https://es.mathworks.com/help/stats/distributionfitter-app.html

[23] "Project jupyter," Visitado: 21.08.2024. [Online]. Available: https://jupyter.org/

[24] "Scikit learn," Visitado: 02.09.2024. [Online]. Available: https://scikit-learn.org/stable/

[25] "Tensorflow," Visitado: 02.09.2024. [Online]. Available: https://www.tensorflow.org/?hl=es-419

[26] Moez Sajwani, "Introducción a las funciones de activación en las redes neuronales," Visitado: 18.9.2024. [Online]. Available: https://www.datacamp.com/es/tutorial/introduction-to-activation-functions-in-neural-networks?dc_referrer=https%3A%2F%2Fwww.google.com%2F

Apéndices

A

Comparaciones de los métodos

En el apartado de la comparación de los métodos se expone que el método 7 es el que mejor se ajusta a lo que se está buscando, donde se ha priorizado la exactitud además de tener un tiempo de ejecución razonable. En este apéndice se aportan más pruebas y resultados realizados para la comprobación de este argumento, mostrando las tablas con el tiempo medio de ejecución para una ventana, la distancia media de K-S, su desviación media y un histograma que muestra si el ajuste es parecido a los datos que se están utilizando para una ventana de ese fichero.

	Método 1	Método 2	Método 3	Método 4	Método 5	Método 6	Método 7
Tiempo(s)	0.0153	0.0042	0.1737	0.7408	23.3467	0.4402	0.5019
Distancia	0.2911	0.3954	0.0759	0.0586	0.2000	0.1291	0.0512
Desviación	0.1003	0.0827	0.0293	0.0158	0.1212	0.0308	0.0163

Tabla A.1: Tabla para la semana 1 de Julio.

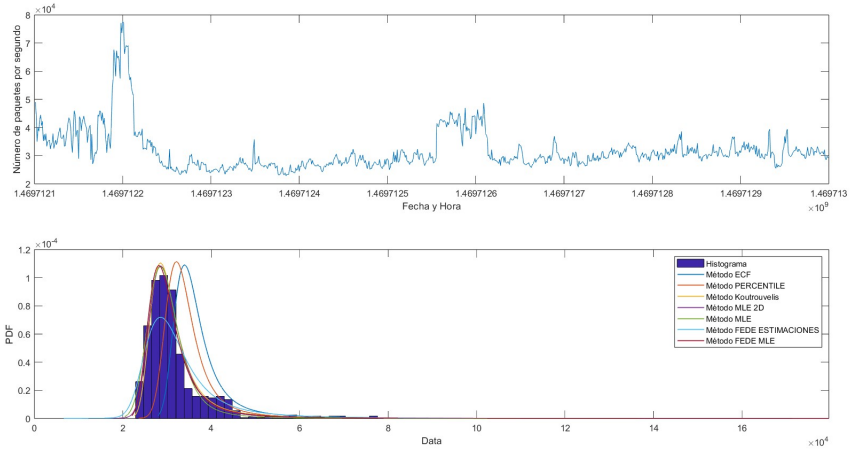

Figura A.1: Representación temporal de un histograma y sus ajustes para una ventana de la semana 1 de Julio.

	Método 1	Método 2	Método 3	Método 4	Método 5	Método 6	Método 7
Tiempo(s)	0.0125	0.0038	0.1562	0.3897	17.3949	0.3818	0.4437
Distancia	0.5294	0.1991	0.2855	0.2156	0.3179	0.2193	0.0808
Desviación	0.1233	0.0283	0.2032	0.0201	0.2013	0.0170	0.0275

Tabla A.2: Tabla para la semana 2 de Junio.

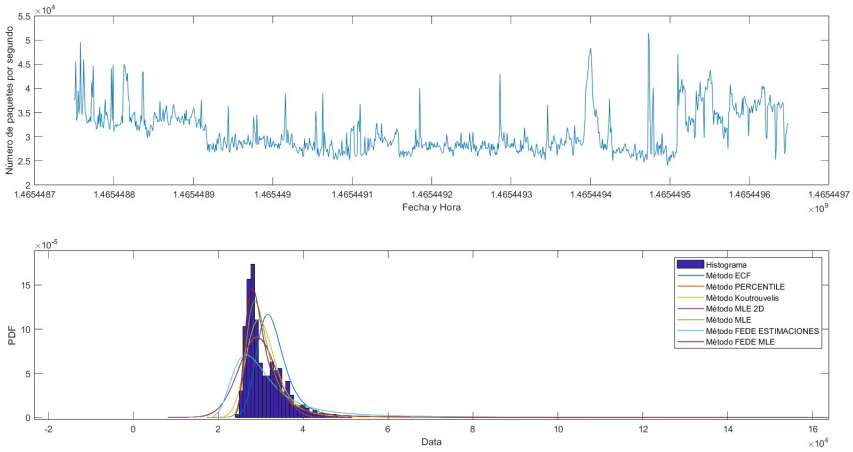

Figura A.2: Representación temporal de un histograma y sus ajustes para una ventana de la semana 2 de Junio.

	Método 1	Método 2	Método 3	Método 4	Método 5	Método 6	Método 7
Tiempo(s)	0.0198	0.0045	0.1503	0.5501	534.3579	0.4699	0.6201
Distancia	0.1127	0.0816	0.0749	0.0534	0.1019	0.1781	0.0475
Desviación	0.1127	0.0277	0.0170	0.0147	0.1426	0.0340	0.0131

Tabla A.3: Tabla para la semana 3 de Marzo.

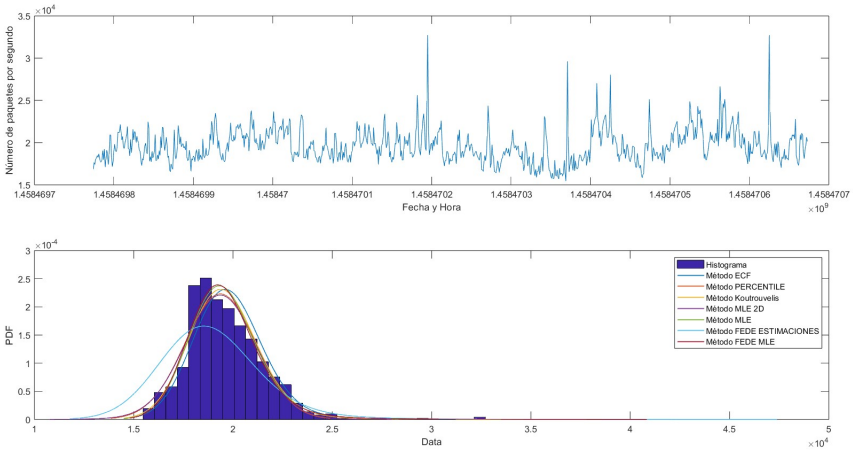

Figura A.3: Representación temporal de un histograma y sus ajustes para una ventana de la semana 3 de Marzo.

	Método 1	Método 2	Método 3	Método 4	Método 5	Método 6	Método 7
Tiempo(s)	0.0120	0.0042	0.1365	0.3148	24.7798	0.6046	0.7250
Distancia	0.4636	0.4095	0.1203	0.0938	0.3867	0.1300	0.0715
Desviación	0.2809	0.3465	0.0955	0.0200	0.2686	0.0283	0.0174

Tabla A.4: Tabla para la semana 3 de Mayo.

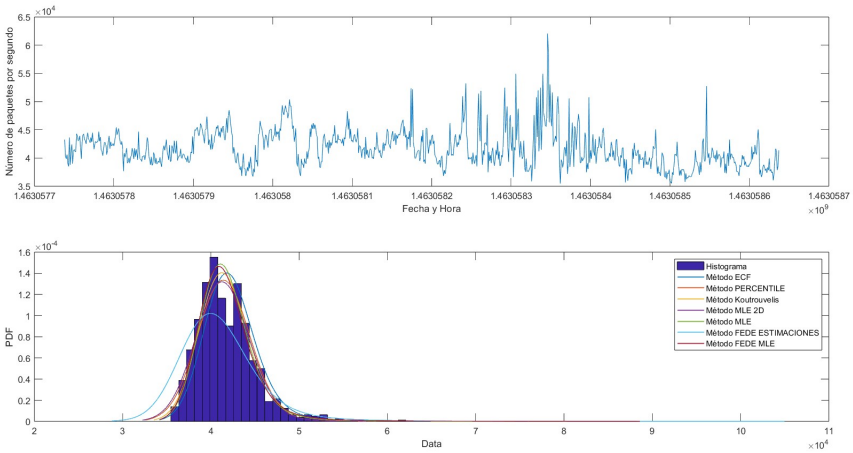

Figura A.4: Representación temporal de un histograma y sus ajustes para una ventana de la semana 3 de Mayo.

Observando todas las gráficas, se puede observar que siempre se tiene la menor distancia media y una desviación pequeña en el método 7, el único método que se acerca es el método 4 en temas de precisión y tiempos, pero como se opta por priorizar la precisión se decide por el método 7 para calcular los alfa-estables.

B

GITHUB

Se ha habilitado un repositorio de GitHub para poder probar el código desarollado en este proyecto.

`https://github.com/aalejandromgg/TFG_AMG`


UAM
UNIVERSIDAD AUTONOMA
DE MADRID