VNIVERSITAT DE VALÈNCIA

Doctorado en Tecnologías de la Información, Comunicaciones y Computación

VNIVERSITAT DE VALÈNCIA

Tesis Doctoral

Técnicas de aprendizaje automático para la detección de dominios maliciosos generados algorítmicamente

Autor: Jose F. Selvi Sabater

Directores:

Dr. Ricardo J. Rodríguez

Dr. Emilio Soria Olivas

Noviembre 2021

VNIVERSITAT DE VALÈNCIA

Escola Tècnica Superior d'Enginyeria ETSE-UV

Tesis Doctoral

Técnicas de aprendizaje automático para la detección de dominios maliciosos generados algorítmicamente

Autor

Jose Francisco Selvi Sabater

Directores

Dr. Emilio Soria Olivas	Universitat de València
Dr. Ricardo Julio Rodríguez Fernández	Universidad de Zaragoza

Tutor

Dr. Miguel Arevalillo Herráez	Universitat de València

Tribunal

Presidente:	Dr. Manuel Domínguez González	Universidad de León
Vocal:	Dr. Juan Manuel Estévez Tapiador	Universidad Carlos III de Madrid
Secretaria:	Dra. María Roser Benavent García	Universitat de València
Presidente (suplente):	Dr. Ignacio Díaz Blanco	Universidad de Oviedo
Vocal (suplente):	Dra. Marta Beltrán Pardo	Universidad Rey Juan Carlos
Secretaria (suplente):	Dra. Esther de Ves Cuenca	Universitat de València

A mis niñas, Leyre y Nerea.

Never walk on the traveled path because it only leads where others have been.
(Nunca vayas por el camino trazado, porque conduce a donde otros ya han ido)
(Alexander Graham Bell)

Do or do not. There is no try.
(Hazlo o no lo hagas, pero no lo intentes)
(Maestro Yoda)

Resumen:

Durante los últimos años, el desarrollo de software malicioso (*malware*) ha pasado de ser un sector donde la principal motivación era el reconocimiento de la comunidad a convertirse en un auténtico negocio multimillonario. Entre las diferentes técnicas que los desarrolladores de malware han empleado para dificultar su detección, está el uso de algoritmos de generación de dominios (en inglés, *Domain Generation Algorithms* o DGA). En el trabajo plasmado en esta tesis doctoral se ha analizado la problemática de esta técnica, se han revisado los diferentes tipos de algoritmos que se pueden encontrar y se ha estudiado el estado del arte en la detección de este tipo de algoritmos en cuanto a técnicas de aprendizaje automático.

Además, esta tesis propone dos soluciones diferentes que aportan ciertas ventajas y también ciertos inconvenientes. Esto permite elegir la técnica más adecuada según la necesidad del entorno concreto en el que sea utilizado. En primer lugar, se propone el uso de nuevas características que, junto con otras ya descritas en la bibliografía, son usadas para la construcción de un clasificador basado en *Random Forest*.

En segundo lugar, se utiliza una aproximación más directa mediante aprendizaje profundo para prescindir del proceso de ingeniería de características. Esta aproximación permite reentrenar el modelo de forma automática ante nuevas técnicas que pretendan evadir la detección. Para este propósito se emplea un modelo de red neuronal LSTM diseñada de forma minimalista para que sea capaz de aprender empleando el mismo conjunto de datos que ha sido empleado para el entrenamiento del modelo *Random Forest*.

Ambos modelos han sido entrenados con el mismo conjunto de datos y probados con un conjunto de datos que no ha sido empleado ni en el entrenamiento ni para la selección de hiperparámetros, obteniendo una exactitud en la clasificación del 97-98 % en dicho conjunto de datos. El modelo *Random Forest* obtiene una exactitud levemente mayor, aunque es también el modelo menos flexible ante necesidades futuras de reentrenamiento.

Por último, todo el código fuente de las herramientas y contenedores empleados en los experimentos se encuentran disponible de forma pública y abierta, por lo que cualquier investigador puede acceder a los mismos y reproducir los resultados.

Agradecimientos:

En primer lugar quiero agradecer a mi mujer Eva y a mis hijas Leyre y Nerea, por su paciencia durante estos años por las horas en las que han tenido que prescindir de su marido y padre para poder realizar esta tesis doctoral, por las horas de parque en las que no les pude acompañar, y por las semanas y semanas de vacaciones durante años que no se pudieron invertir haciendo viajes, excursiones, y demás actividades propias. En especial a las niñas, que son pequeñas y muchas veces ha sido difícil de explicar por qué papá "estaba de vacaciones" y aún así tenía que quedarse en casa trabajando. Os lo compensaré, lo prometo. Perdón, y gracias.

En segundo lugar, a mis padres, Pepe y Pepa, que siempre nos inculcaron a mi hermano y a mi una cultura del esfuerzo, de intentar ir siempre un paso más allá, y de acabar lo que se empieza cueste lo que cueste. Si no fuera por esta educación, es muy probable que la exigencia de realizar una tesis, junto con el trabajo a tiempo completo y la familia, hubiera podido conmigo. Espero ser capaz de inculcar a mis hijas esta misma cultura, así como otros tantos valores que tanto mi hermano como yo hemos recibido.

Por último, a mis directores de tesis Emilio y Ricardo, y a mi tutor Miguel, que me han aguantado durante años y me han ayudado mucho desde el primer día, tanto a nivel técnico como a nivel personal, y han sabido aconsejarme en todo momento teniendo en cuenta mis situaciones personales y familiares. También a todo el personal de la *Universitat de València*, tanto docente como administrativo, porque todos sois fantásticos profesionales y me habéis ayudado mucho desde el primer día que llegué con 18 años a estudiar primero de Ingeniería Informática, hasta el día de hoy. Gracias por esta formación que me habéis dado y por hacerme sentir que esta universidad es mi casa.

Índice general

Capítulo 1

Introducción

En este capítulo se describe una introducción al problema del malware en general y sitúa los motivos por los que existen los algoritmos de generación de nombres de dominio y el problema que ello ocasiona en la industria de la seguridad. Seguidamente se describe cómo esta tesis doctoral plantea ayudar a la resolución de este problema y las contribuciones que aporta a este campo científico. Finalmente se referencia la organización de sus capítulos.

1.1. Introducción

La escena del malware ha cambiado desde principio de los años 90, cuando los objetivos principales de los grupos de investigación como 29A [1] pasaban fundamentalmente por conseguir el reconocimiento de la comunidad, mejorar o adquirir conocimiento técnico y exponer los riesgos de seguridad a los que se enfrentaba la población general. Lejos han quedado aquellos días en los años 80 y 90 cuando malware como "Viernes 13" [2] o "barrotes" [3] atemorizaban a los usuarios de ordenadores personales, hasta el punto, por ejemplo, de no encender el ordenador en fechas señaladas como viernes 13 para evitar los estragos que este conocido malware causaba en los equipos infectados. Sin embargo, el malware desarrollado en esta época, a pesar de tener en ocasiones un objetivo bastante destructivo, no generaba ningún beneficio económico para los desarrolladores o distribuidores de malware, al contrario de lo que sucede en la actualidad.

Hoy en día desarrollar malware se ha convertido en un negocio extremadamente rentable. En 2013, Europol publicaba que el impacto global del cibercrimen creció hasta cerca de los $3 trillones de dólares americanos, convirtiéndolo en un negocio más lucrativo que el tráfico de marihuana, cocaína y heroína juntos [4]. El malware es, habitualmente, una parte imprescindible del negocio del cibercrimen. Este es el caso de los troyanos bancarios, que capturan las credenciales en el ordenador infectado una vez que la víctima conecta a la banca on-line de su banco, o de los ransomware, que cifran el contenido del disco duro de la víctima y piden cierta cantidad por proporcionar la clave de descifrado, así como otros tipos de malware más o menos sofisticados [5].

Otro de los cambios fundamentales que se ha sufrido durante estas últimas décadas es la popularización de Internet y el aumento exponencial de usuarios conectados. En las décadas de los 80 y 90 la mayor parte de los ordenadores personales se encontraban aislados unos de otros y el único medio de transmisión de la infección de unos equipos

a otros era el uso de dispositivos de almacenamiento como los disquetes. Esto hacía que la curva de infección de un malware fuera mucho menor y creciera de una forma mucho más lenta. En la actualidad el escenario es muy diferente, ya que la mayor parte del malware está diseñado para propagar la infección a través de la red y obtener unos ratios de infección, y por tanto de beneficios, mucho mayores. Como resultado, la cantidad de muestras de malware a ser analizadas por los profesionales del sector crece año tras año [6].

Según AV Test Statistics [5], la cantidad de muestras de malware ha aumentado exponencialmente desde 2006, a pesar de que la cantidad de nuevo malware ha tenido un aumento más lineal. El análisis manual de todas estas muestras, con el fin de entender el comportamiento de cada malware y poder diseñar mecanismos de detección y desinfección, es un proceso complejo y costoso denominado ingeniería inversa. Este proceso ha aumentado su complejidad con el tiempo hasta el punto de convertirlo en una tarea imposible para las compañías anti-malware debido al gran volumen de muestras que deben ser analizadas día tras día y al aumento exponencial en el volumen de dichas muestras. Por ejemplo, la empresa de seguridad y desarrolladores de productos anti-malware Kaspersky Lab manifestó haber analizado diariamente unas 350 000 muestras de malware en 2013 [7]. Por este motivo existe un creciente interés, tanto en la industria como en el mundo académico, de diseñar y perfeccionar mecanismos de análisis automático de malware que ayuden a manejar los volúmenes con los que la industria se está encontrando en las últimas décadas.

Existen diversas maneras de analizar o detectar malware. La primera de ellas es lo que se llama un análisis estático [8]. Este análisis consiste en una inspección del fichero o ficheros sospechosos. Durante esta inspección se recolecta toda la información que pueda ser extraída de ellos. Esto incluye metadatos del tipo de formato, como puedan ser la información de la cabecera PE en el caso de ficheros ejecutables para Microsoft Windows, y también secciones del fichero y contenido de cada una de las secciones, entre las que se encuentra el código binario que será ejecutado. Muchos desarrolladores de malware emplean técnicas de ofuscación de su código para dificultar la inspección estática. Una de las técnicas comúnmente empleadas consiste en cifrar el código ejecutable y que el fichero ejecutable realice un descifrado en tiempo de ejecución [9].

Por otro lado existe el análisis dinámico [8] en el que, al contrario de lo que ocurre con el análisis estático, la información recogida no se obtiene a partir de una inspección de los ficheros sino que la muestra es ejecutada en un entorno controlado (*sandbox*) y se registra toda la información referente a las acciones realizadas por esa muestra como puede ser creación de otros procesos o ficheros, creación de entradas en el Registro de Windows, conexiones de red, etc. La aproximación mediante análisis dinámico tiene la ventaja de que se registran las acciones reales que la muestra realiza, por lo que técnicas como la de cifrar el código malicioso resultan inefectivas ya que las acciones realizadas a priori deberían delatar la naturaleza maliciosa de la muestra. No obstante, algunos desarrolladores de malware emplean técnicas de detección de estos entornos controlados [10] ya que mayoritariamente se trata de entornos virtualizados o con unas características concretas que el malware es capaz de detectar, ocultando de esta manera el comportamiento malicioso que lleva a su detección.

Por último, como caso particular del análisis dinámico pero de especial interés para esta tesis doctoral, se encuentra la detección en la red [11]. Esta aproximación consiste en la captura de la información transmitida por la red en la que se encuentra el equipo infectado y en la identificación de características en dicho tráfico que puedan constatar la presencia del malware. Estas características pueden ser establecidas en base a patrones

de tráfico de malware conocidos o descubiertos como parte del análisis dinámico, o bien basarse en algún tipo de anomalía con respecto a un patrón normal de tráfico de red. Por ejemplo, en una red en la que no debieran existir comunicaciones con protocolo *Internet Relay Chat* (IRC) [12], la presencia de este tipo de comunicaciones se consideraría una anomalía y debería ser notificada para su análisis en profundidad a pesar de que no exista una detección de un tipo de malware en concreto.

Con la popularización de Internet el malware ha dejado de ser software ejecutándose en equipos informáticos aislados a pasar a formar parte de una estructura más amplia generalmente denominada *botnet*. Estas *botnet* buscan tener un control centralizado sobre un número indeterminado de equipos infectados y obtener información de ellos o forzar la ejecución de tareas coordinadas. Un ejemplo de estos tipos de infraestructura es la *botnet Mirai* [13], que en 2017 provocó una de las mayores denegaciones de servicio de la historia a varios de los principales proveedores y servicios de Internet, empleando para ello cientos de miles de dispositivos infectados que actuaron de forma coordinada para provocar tal situación de colapso.

Una arquitectura comúnmente usada en este tipo de malware es hacer que el equipo víctima conecte a un nodo central, bien sea directamente o a través de una red de equipos intermedios que ocultan la localización del nodo principal. Dicho nodo principal desde el cual se coordinan las acciones de la *botnet* se denomina servidor de mando y control o en inglés, *Command & Control* (C2).

En el pasado, las diferentes familias de malware han empleado una variedad de protocolos para comunicarse con sus C2. Uno de los ejemplos más típicos de finales de los 80 y durante la década de los 90 fue el uso del protocolo IRC. Este protocolo era ampliamente usado para la comunicación interactiva entre personas pero, prácticamente tras su aparición, fue así mismo usado por diversas familias de malware que conectaban a servidores y canales de IRC específicos donde su *bot master* podía interactuar con ellos [14].

Sin embargo, desde finales de los años 90 el uso de comunicaciones empleando los protocolos HTTP (*Hypertext Transfer Protocol*) y su versión segura HTTPS se ha convertido en la práctica más extendida [14], ya que dicho protocolo es el empleado por el tráfico de navegación web. Por este motivo el tráfico malicioso pasa más desapercibido entre el tráfico legítimo que empleando otros protocolos como pueda ser IRC, por estar éste mucho más en desuso en la actualidad.

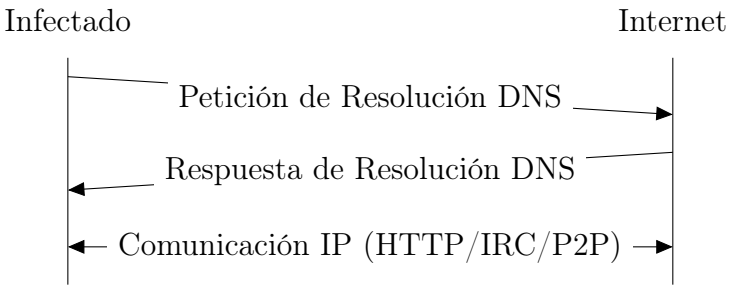


Figura 1.1: Esquema de comunicación típica en redes TCP/IP.

En cualquiera de estos casos, la comunicación en la red entre un malware y su C2 suele ser como se describe en la figura 1.1. Obviamente cualquier tipo de software o dispositivo podría conectarse directamente a una dirección IP, por lo que no sería necesaria la resolución DNS que se muestra en la figura. Sin embargo esto no es lo habitual en la actualidad para ningún tipo de software y tampoco para el malware, ya que resta mucha

flexibilidad al imposibilitar un cambio de receptor de la conexión.

Como medida de defensa es habitual establecer una cooperación entre todos los actores dedicados a la lucha contra el malware (como puedan ser los Centros de Respuesta a Incidentes, empresas privadas, etc.) para notificar tanto a los ISP (*Internet Service Providers*) como a los registradores de dominio cuando una dirección IP o nombre de dominio está siendo utilizado de forma maliciosa, con el fin de proceder a su bloqueo. También se usa esta información para la creación de listas negras de IPs y dominios que pueden ser usados por cualquier entidad para bloquear su acceso mientras estas son bloqueadas por los proveedores de servicio de Internet.

1.2. Motivación y objetivos

Con el fin de evitar la detección de conexiones contra los servidores C2, los desarrolladores de malware han empezado a usar una estrategia de comunicación más sofisticada. En concreto una de estas estrategias consiste en la generación, a través de un algoritmo, del nombre de dominio donde puede estar alojado el servidor C2. De este modo una muestra de malware genera un nombre de dominio candidato e intenta conectar a él. Si la conexión se produce, la comunicación comienza con ese equipo y se sigue el funcionamiento normal de dicha muestra. En el caso contrario, se genera un nuevo nombre de dominio y se repite el proceso. Estos nombres de dominio son generados mediante una técnica llamada DGA (*Domain Generation Algorithm*), la cual se popularizó en 2008 gracias al conocido gusano Conficker [15] que empleaba esta técnica para conectar a su C2. Dado que estos algoritmos generan nombres de dominio de forma dinámica, las compañías anti-malware tienen más dificultades para generar las listas negras y notificar a los proveedores de servicio ya que requiere la extracción del algoritmo mediante análisis estático de la muestra (generalmente manual), lo cual es mucho menos eficiente y consume muchos más recursos [16].

Sin embargo, los nombres de dominios generados mediante DGA tienen un aspecto muy diferente a los nombres de dominio de otro tipo de servicios. Normalmente los nombres de dominio están basados en una palabra o un conjunto de palabras que son representativas del servicio que se pretende ofrecer. También suelen ser elegidos de forma que son fáciles de recordar y de escribir. La cantidad de nombres de dominio que un DGA puede generar, por el contrario, es proporcional a la longitud del dominio y a la aleatoriedad empleada por el algoritmo generador. Como consecuencia, la mayoría de los nombres de dominio generado por los DGAs parecen aleatorios y de aspecto extraño a la vista de un humano.

El objetivo de esta tesis es estudiar la aplicación de métodos de aprendizaje automático para resolver el problema de la detección de nombres de dominio que hayan sido generados mediante un DGA y, por tanto, la detección de la presencia en la red de malware que haga uso de este tipo de técnicas. Esto evita la necesidad de realizar el complejo y costoso proceso de obtener el algoritmo específico empleado mediante técnicas de ingeniería inversa. También evita mantener y manejar listas negras de gran tamaño y con poca posibilidad de escalar.

En concreto, en esta tesis se aborda este problema desde dos aproximaciones diferentes. La primera realizando el proceso manual de ingeniería y extracción de características, y la segunda empleando técnicas de aprendizaje profundo que permiten el desarrollo del modelo sin ningún tipo de procesamiento previo de las entradas.

1.3. Contribuciones

El trabajo de esta tesis doctoral incluye una serie de contribuciones que se describen brevemente a continuación:

- En primer lugar, se describe la volatilidad de ciertas características que se pueden extraer de los nombres de dominio generados algorítmicamente y la problemática asociada a su uso para la detección de estos dominios. Para solventar estos problemas, en esta tesis se propone el concepto de *N-Gram* enmascarado. Este concepto define la extracción de nuevas características de un nombre de dominio que proporcionan un nivel de abstracción intermedio entre los *N-Grams* y su información estadística (media, varianza y desviación estándar). Su importancia en la clasificación y detección de dominios DGA se ha respaldado mediante el algoritmo de Boruta (explicado en más detalle en el capítulo 5).

- Para realizar la experimentación necesaria, en esta tesis se crea y se publica de manera abierta un conjunto de datos compuesto por 64 000 nombres de dominio, repartidos entre dominios legítimos y generados por un DGA al 50 %. La distribución de los nombres de dominios generados por DGAs también se genera de forma balanceada con respecto a la familia de malware que usa cada algoritmo [17]. La construcción de este conjunto de datos utilizado se explica en más detalle en el capítulo 4.

- Se evalúa el uso de un modelo *Random Forest* para la clasificación, empleando una combinación de características léxicas usadas en la literatura junto con la propuesta de *N-Gram* enmascarados. El capítulo 5 describe el modelo utilizado y discute los resultados obtenidos en mayor profundidad.

- Se evalúa el uso de un modelo basado en redes neuronales recurrentes LSTM que posibilita evitar el costoso proceso de la ingeniería de características, inherente al modelo evaluado anteriormente. El diseño minimalista de este modelo reduce la cantidad de parámetros entrenables y posibilita entrenar la red con conjuntos de datos de tamaño reducido, permitiendo además comparar los resultados de ambos modelos evaluados y discutir en qué escenarios resulta más adecuada cada una de ellas. Todo este trabajo se describe en más detalle en el capítulo 6.

- Se emplean técnicas de inteligencia artificial explicable (XAI) para determinar si el aprendizaje profundo identifica las reglas que un analista de malware o en detección de intrusiones puede establecer de forma intuitiva. Este estudio se detalla en los capítulos 5 y 6.

- A modo de conclusión, se proponen dos arquitecturas que combinan ambas propuestas de esta tesis doctoral y permiten mejorar las capacidades defensivas de una organización, facilitando la detección de dominios generados algorítimicamente mediante análisis de tráfico. Estas arquitecturas se describen en el capítulo 7.

Por último, cabe destacar que todo el código fuente derivado de los trabajos de programación realizados en esta tesis doctoral se encuentran disponibles de forma pública y abierta [18, 19], por lo que cualquier investigador puede acceder a los mismos y reproducir los resultados.

1.4. Organización de la tesis

La tesis doctoral está organizada de la siguiente forma: en primer lugar el capítulo 2 repasa los conceptos previos sobre los cuales esta tesis se fundamenta, incluyendo tanto información específica sobre DGAs como información detallada sobre el tipo de modelos empleados. En segundo lugar el capítulo 3 muestra el estado del arte alrededor de este tema y comenta las aproximaciones y resultados obtenidos por diferentes autores. El capítulo 4 muestra la construcción del conjunto de datos empleado que posteriormente es utilizado para el entrenamiento y validación de los modelos cuyo diseño se detalla en los capítulos 5 y 6. En el capítulo 5 se propone el uso de *N-Grams* enmascarados en un modelo de aprendizaje no profundo *Random Forest* y se evalúan sus resultados en comparación con otras. En el capítulo 6 se propone el uso de un modelo de aprendizaje profundo LSTM que no requiere el proceso de ingeniería de características manual, y se estudia también su interpretabilidad empleando técnicas de XAI. Por último, las conclusiones de esta tesis, incluyendo posibles futuros trabajos y arquitecturas de red para mejorar la capacidad defensiva, se encuentran en el capítulo 7.

Capítulo 2

Conceptos previos

En este capítulo se describe el contexto y los conceptos previos necesarios para poder entender las propuestas contenidas en esta tesis doctoral. Dado que el campo estudiado es una intersección entre los campos de conocimiento de las tecnologías de la información, la ciberseguridad y del aprendizaje automático, este capítulo se separa en tres secciones que dan cobertura a cada uno de dichos campos. Por un lado, la primera sección describe el problema de la detección de malware tal y como tiene lugar en la industria de la ciberseguridad. Por otro lado, la segunda sección explica en qué consiste el protocolo de resolución de nombres DNS y qué información contiene su tráfico de red que resulta interesante para detectar dominios generados algorítmicamente. Por último, la tercera sección detalla los conceptos previos referentes al aprendizaje automático, centrándose en aquellas técnicas que serán usadas en esta tesis o cuyos conceptos son necesarios para entender los que aquí se manejan.

2.1. Algoritmos de generación de dominios

Tal y como se menciona anteriormente, existen diferentes técnicas que se emplean para detectar la presencia de malware, siendo una de ellas a través de la detección en la red. La detección en la red consiste en capturar tráfico de red y analizarlo para identificar el tráfico generado por una actividad maliciosa [11]. Para ello existen una gran variedad de herramientas, tanto de código abierto como comerciales, que deben ser desplegadas en puntos estratégicos de la red para ser capaces de monitorizar la mayor cantidad de tráfico de red posible o al menos el tráfico más relevante.

De forma habitual la detección se realiza por medio de una serie de reglas que describen el comportamiento malicioso. En el listado 2.1 se observa una regla del conocido sistema de detección de intrusiones (*Intrusion Detection System* o IDS) de código abierto Snort [20]. En concreto, esta regla pertenece al repositorio abierto de reglas *Emerging Threads* [21] y tiene como objetivo detectar la actividad en la red del malware Conficker [15] cuando dicho malware se comunica con dispositivos en Internet mediante el protocolo HTTP. El formato de la regla puede ser diferente dependiendo del producto IDS para el que ha sido generada, ya que dichos productos tienen diferentes capacidades y pueden emplear un lenguaje distinto para definir sus reglas de detección.

Aunque existen diferentes diseños con los que se puede desplegar un IDS, una arquitectura típica es la mostrada en la figura 2.1, en la que el detector de intrusos recibe el

```
alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS (
    msg:"ET TROJAN Conficker/MS08-067 Worm Traffic Outbound";
    flowbits:isset,ET.ms08067_header;
    flow:established,to_server;
    content:"If-None-Match|3A| |22|60794|2D|12b3|2D|e4169440|22|";
    nocase;
    reference:url,doc.emergingthreats.net/bin/view/Main/2008739;
    classtype:trojan-activity;
    sid:2008739;
    rev:6;
    metadata:created_at 2010_07_30, updated_at 2010_07_30;)
```

Listado 2.1: Regla del IDS de red Snort.

tráfico (o una copia del mismo) y se comprueba si se da alguna actividad descrita por sus reglas de detección. En tal caso, se genera una alerta o se procede a bloquear el tráfico si se dispone de las capacidades para hacerlo.

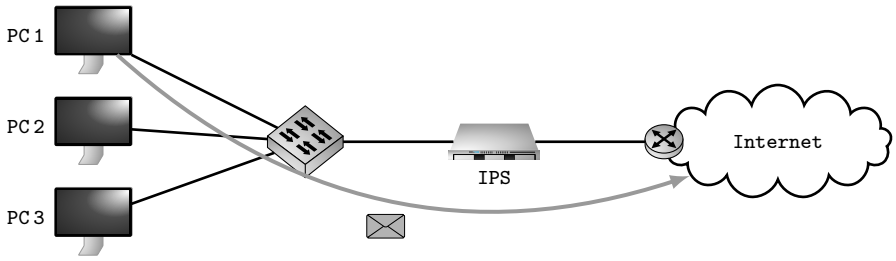


Figura 2.1: Diseño de red de un IDS.

Aunque la mayoría de dispositivos IDS permiten crear reglas de detección muy complejas que detectan actividad maliciosa muy específica, el uso de dichas capacidades de detección avanzada implica un fuerte impacto en el rendimiento del dispositivo [22]. Por este motivo es habitual emplear mecanismos de detección basados en listas negras [23] por ser estas mucho más sencillas y eficientes. Estas listas negras contienen direcciones IP, nombres de dominio o direcciones web que se han visto involucradas en actividades maliciosas. Su eficiencia con respecto a otros tipos de detección se debe a que el acceso a esta información en el tráfico de red no requiere inspeccionar dicho tráfico en profundidad sino que es accesible de forma muy eficiente a través del propio paquete de red o empleando un servicio *proxy* como el mostrado en la figura 2.2. Estos servicios *proxy* son empleados habitualmente para gestionar tráfico saliente y permitir o denegar las peticiones en función de listas negras o de ciertas reglas definidas. Por tanto, el uso de esta aproximación tiene como consecuencia una mejora en el rendimiento con respecto al uso de reglas de detección más complejas en un IDS.

Tanto el uso de listas negras como de dispositivos IDS han sufrido en estas últimas décadas el problema del incremento de uso del tráfico cifrado, que dificulta monitorizarlo e inspeccionarlo en profundidad [24]. Esto provoca que las listas negras que han gozado de más popularidad hayan sido aquellas que pueden ser utilizadas incluso cuando el malware emplea protocolos de comunicación cifrados como HTTPS, como por ejemplo las listas negras de direcciones IP maliciosas. Las direcciones IP de origen y destino del tráfico de red se encuentran en la cabecera IP del paquete y por tanto no se encuentran cifradas ni ocultas por protocolos como TLS, que actúa en capas superiores de la pila TCP/IP.

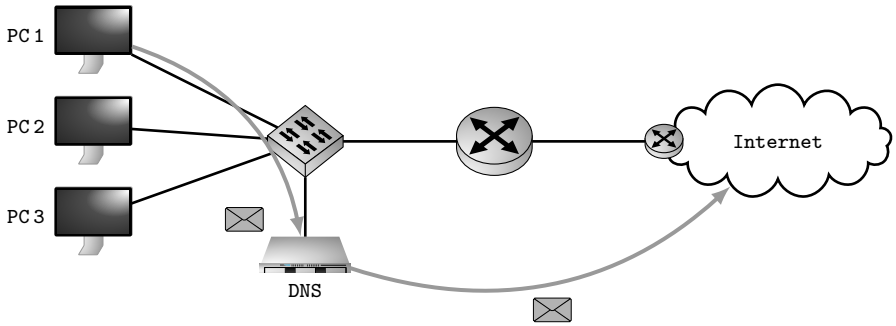


Figura 2.2: Diseño de red usando detección de DNS.

Otro de los tipos de listas negras más utilizadas son las listas negras basadas en nombres de dominio, que también son fácilmente monitorizables en la red debido a que el protocolo DNS estándar no implementa ningún tipo de cifrado y a que el uso de su alternativa cifrada (protocolo DNSSEC [25]) aún no se utiliza de forma significativa en Internet. Por tanto resulta sencillo y eficiente extraer el nombre de dominio a partir del tráfico de red o de un servidor DNS intermedio y verificar si está incluido en la lista negra de dominios maliciosos.

Con respecto a los nombres de dominio, se llama FQDN (*Fully Qualified Domain Name*) al nombre completo que identifica a un dispositivo de forma inequívoca en Internet. Este FQDN incluye el nombre del dispositivo y el dominio al que pertenece. Por ejemplo, si el FQDN de un dispositivo es "www.informatica.uv.es", eso significa que el dispositivo recibe el nombre "www" y pertenece al dominio "informatica.uv.es". El dominio a su vez se divide en varias partes separadas por el carácter punto ('.'), estando ordenados de derecha a izquierda en cuanto a jerarquía. En este ejemplo, "es" representa la jerarquía más alta en el nombre del dominio y se conoce como el TLD (*Top-Level Domain*). El segundo componente "uv" es conocido como dominio de segundo nivel y se emplea para identificar una entidad de algún tipo. Junto con el TLD forma lo que en el lenguaje técnico habitual se conoce como dominio y es lo que los usuarios pueden registrar y gestionar. Por ejemplo, el dominio "uv.es" representa en Internet a la entidad *Universitat de València* que es la propietaria del nombre del dominio y se encarga de gestionarlo de forma autónoma. Por último, existen los dominios de niveles superiores (tercer nivel, cuarto nivel y así sucesivamente), como en este caso "informatica", que se conoce como subdominio y se emplea para ordenar jerárquicamente dentro de una entidad a los diferentes dispositivos o para delegar la gestión de ese subdominio en un departamento concreto u otra subestructura de la entidad principal.

En la tabla 2.1 se muestra varios ejemplos de cómo se construye un FQDN en base a cada una de sus partes descritas anteriormente. Estos FQDN no incluyen el nombre de protocolo TCP/IP empleado (por ejemplo, "http://" o "ftp://").

FQDN	Dispositivo	Subdominio	Dominio	TLD
www.informatica.uv.es	www	informatica	uv	es
jselvi.blogs.uv.es	jselvi	blogs	uv	es
www.google.com	www		google	com

Tabla 2.1: Ejemplos de la estructura de un FQDN.

Durante mucho tiempo se ha popularizado el uso de listas negras de nombres de

dominio mediante los llamados *sumideros DNS* [26]. Estos sumideros son servidores DNS intermedios que actúan como *proxy* de las peticiones DNS que se generan dentro de una red. Una vez reciben una petición de resolución DNS comprueban si dicho nombre existe en su lista negra. En el caso de que no se encuentre, el servidor procede a realizar una resolución DNS de la forma estándar. En caso contrario, el servidor DNS responde con una resolución falsa (típicamente la dirección IP virtual del interfaz *loopback*). En ocasiones, los equipos de seguridad informática emplean esta misma técnica para resolver a la dirección IP de un servidor falso que suplanta al servidor malicioso y así investigar su comportamiento.

Por lo tanto, una muestra de malware que emplea el nombre "mihostmalicioso.pentester.es" puede tener un tiempo de vida como el siguiente:

1. El malware es desarrollado.
2. Se registra el dominio "mihostmalicioso.pentester.es" y se despliega su infraestructura C2.
3. El malware se actualiza para conectar a la infraestructura recientemente creada.
4. Se inicia la campaña para infectar a víctimas con dicho malware.
5. Alguna muestra del malware acaba llamando la atención de las entidades de referencia y se procede a su análisis.
6. Se determina que se trata de malware y que conecta con el dominio "mihostmalicioso.pentester.es".
7. Se procede a desplegar protecciones contra dicho malware. Entre ellas se incluye al nombre de dominio detectado en las correspondientes listas negras.
8. Una vez que los sumideros DNS actualizan sus listas negras, estos empiezan a bloquear las comunicaciones del malware y, en principio, lo inactivan.

Para eludir el uso de las listas negras de nombres de dominio, los desarrolladores del malware deben registrar un nuevo nombre de dominio, desplegar de nuevo la infraestructura en una ubicación diferente y actualizar su software para emplear el nuevo dominio. Una vez hecho esto tienen que repetir la campaña para infectar a las víctimas que es, sin duda, la parte más costosa y menos determinista, ya que su éxito es impredecible.

En este contexto, los desarrolladores de malware crearon una nueva técnica llamada DGA (*Domain Generation Algorithm*) que se pudo ver por primera vez siendo utilizada en el malware Conficker [15]. Con el uso de esta técnica las muestras de malware no emplean un único nombre de dominio para conectar a su infraestructura C2 sino que generan el nombre mediante un algoritmo que tiene una gran cantidad de posibles resultados y que, a priori, únicamente dicho malware y su desarrollador pueden predecir. Algunos ejemplos de los nombres de dominio generados por Conficker pueden verse en la tabla 2.2.

El uso de esta técnica tiene dos grandes ventajas para los desarrolladores de malware. En primer lugar, incluso cuando un nombre de dominio está incluido en una lista negra y su infraestructura desmantelada, el desarrollador del malware tiene la capacidad de registrar otro de los nombres de dominio que serán generados por el algoritmo y desplegar una nueva infraestructura. Con ello, los equipos infectados que se encontraban conectados

tvxwoajfwad.info	blojvbcbrwx.biz	wimmugmq.biz
fwnvlja.org	umgrzaybbf.ws	btgoyr.cc
zboycplmkhc.cc	qsqzphbn.biz	xqdvmavs.cn

Tabla 2.2: Nombres de dominio empleados por Conficker.

a la infraestructura antigua pasarán a conectarse a la nueva sin necesidad de realizar modificaciones en el malware ni de volver a realizar una nueva campaña para infectar a las víctimas.

La segunda gran ventaja de esta técnica es que la gran cantidad de nombres de dominio que pueden llegarse a generar dificulta enormemente el uso de listas negras, ya que no se puede crear una lista negra con todos los posibles nombres de dominio generados por cada uno de estos algoritmos. El tamaño de dicha lista sería tal que con toda probabilidad imposibilitaría su uso en la práctica.

Con el tiempo, los desarrolladores de malware han ideado diferentes aproximaciones para crear dinámicamente los nombres de dominio [27]. Por este motivo no se puede hablar de una única manera de hacerlo y mucho menos de implementarlo, sino que cada familia de malware generalmente usa una técnica con ciertas diferencias con respecto a otras familias e incluso puede ir cambiando con respecto a las versiones anteriores de esa misma familia.

No obstante, habitualmente los DGAs tiene un comportamiento similar al de los algoritmos generadores de números pseudoaleatorios (PRNG, de sus siglas en inglés) [28], utilizados ampliamente en diferentes campos del mundo de las ciencias de la computación para obtener eficientemente secuencias de valores con apariencia aleatoria pero generados de forma determinista a partir de un valor inicial llamado semilla. Esta característica tiene como consecuencia que sea posible que varios dispositivos diferentes inicialicen el mismo PRNG con la misma semilla y generen la misma secuencia de valores pseudoaleatorios sin ningún otro tipo de comunicación o sincronización.

Esta propiedad es aplicada en el diseño de los DGAs para que tanto el desarrollador del malware como las muestras distribuidas entre los equipos infectados puedan inicializar su generador del mismo modo y que el desarrollador sea capaz de predecir qué nombres de dominio debe registrar para que los dispositivos infectados conecten a ellos. Esta inicialización se hace en base a una semilla que se encuentra incluida dentro del propio código del malware y que el desarrollador conoce. En muchas ocasiones, además de la semilla, también se emplea otro tipo de información que el desarrollador es capaz de predecir como puede ser la fecha de ejecución. Esto se realiza para no repetir la misma secuencia cada vez que el malware se ejecuta, ya que ello facilitaría su detección.

En el listado 2.2 se observa una reimplementación en Python publicada por Bader [29] del DGA empleado por el malware Corebot, que muestra un aspecto muy similar al que se espera de un algoritmo PRNG. En primer lugar (línea 1) se genera el valor de inicialización a partir de la semilla (*seed*) y de los valores de la fecha actual (*day*, *month* y *year*). Tal y como se comentaba, este valor será predecible por el desarrollador puesto que conoce tanto el valor de la semilla como la fecha del día actual. A partir de ese momento el DGA actualiza su estado para obtener nuevos valores pseudoaleatorios (líneas 7 y 11). El primero de estos valores es usado para determinar la longitud del nombre que genera (línea 8) y a partir de este valor se inicia un bucle (línea 10) que genera cada uno de los caracteres del nombre de dominio por medio de la obtención de nuevos números pseudoaleatorios (línea

```
1   state = (seed + year + ((1 << 16) + (month << 8) | day)) & 0xFFFFFFFF
2   charset = [chr(x) for x in range(ord('a'), ord('z'))] + [chr(x) for x in
        range(ord('0'), ord('9'))]
3
4   for _ in range(10):
5       len_l = 0xC
6       len_u = 0x18
7       state = (1664525*state + 1013904223) & 0xFFFFFFFF
8       domain_len = len_l + state % (len_u - len_l)
9       domain = ""
10      for i in range(domain_len, 0, -1):
11          state = ((1664525 * state) + 1013904223) & 0xFFFFFFFF
12          domain += charset[state % len(charset)]
13      domain += ".ddns.net"
14      print(domain)
```

Listado 2.2: DGA del malware Corebot.

```
4chij1xk8axsrite.ddns.net
alg81hc07rq4a6kdmnw.ddns.net
klkran5no83vwpotu8qxqro.ddns.net
u8kvadodg2evul5xsn3vgrk.ddns.net
54kjo6sxe4qtwn7.ddns.net
```

Listado 2.3: Ejemplo de nombres generados por Corebot.

11) y la conversión de dichos números a caracteres válidos (línea 12). Una vez obtenido el nombre se concatena con la cadena "ddns.net" (línea 13) para completar el nombre del dominio. Aunque en este caso el dominio empleado es estático, en otros algoritmos existe una lista de TLDs ("biz", "cc", "cn", etc.) de la cual se elige un elemento en base al valor de uno de estos valores pseudoaleatorios. El listado 2.3 muestra un ejemplo de 5 nombres de dominio generados empleando este DGA con una semilla establecida y para una fecha de ejecución determinada.

Una de las características que muestra la calidad de un PRNG es la cantidad de valores pseudoaleatorios que es capaz de generar sin producirse un bucle en los valores generados. Esta misma propiedad se observa también en los DGAs, puesto que un algoritmo que no esté adecuadamente diseñado puede producir una pequeña cantidad de nombres de dominio únicos y por tanto arruinar la estrategia para la que los DGAs fueron pensados. En el caso del ejemplo anterior, las pruebas con el DGA de Corebot mostraron que es capaz de generar al menos 100 millones de nombres de dominio sin que se produzca ninguna repetición, por lo que cumple el propósito para el cual fue diseñado.

Tal y como se mencionaba anteriormente, tener que incluir más de 100 millones de nombres de dominio en una lista negra y actualizar dicha lista con una periodicidad diaria hace imposible su uso en la práctica. Es cierto que el malware puede limitar dicha generación a una cantidad menor de dominios diarios. Por ejemplo, una muestra de malware podría generar únicamente los 100 o 1 000 primeros dominios para el día en curso y, en el caso de no encontrarse ninguno registrado, simplemente esperar al siguiente día para repetir el proceso. En cualquier caso, añadir todos esos dominios dificulta en gran medida el uso de listas negras además de que requiere que las instituciones relevantes en materia anti-malware empleen una mayor cantidad de recursos en extraer mediante ingeniería

```
unuinstrumentation.com
missiondeveloped.com
contentfortheweiler.com
numberforscience.com
numberworkshop.com
surfacethetheandmissions.com
februaryrecent.com
spacesciencehopeseeking.com
provideatmoscomp.com
provenmeritspacecraftco.com
launchlauheldnathe.com
distkuideproposals.com
```

Listado 2.4: Ejemplo de nombres generados por Gozi.

inversa el algoritmo y su semilla. Este proceso se debería realizar para cada una de las familias de malware que empleen esta técnica de DGA, ya que es necesario para predecir los nombres de dominio que se utilizarán en el futuro.

Plohmann et al. [30] clasifican los diferentes DGAs según dos categorías diferentes: la fuente de la semilla y el esquema de generación. Estas dos categorías se combinan entre ellas formando un total de 16 posibles familias de algoritmos, aunque solo 6 de ellas se han encontrado siendo implementadas por DGAs reales.

De entre esos dos métodos de categorización, el esquema de generación es el único que afecta de forma directa al aspecto de un nombre de dominio, y por tanto es la única de las dos categorizaciones que es de interés para esta tesis. Esta categoría se divide en cuatro tipos de equemas:

- *Aritméticos*: Se calcula una secuencia de valores pseudoaleatorios que son convertidos a caracteres ASCII para formar el nombre de dominio.
- *Basados en hash*: Emplean una función criptográfica de resumen (*hash*) en su formato hexadecimal.
- *Basados en diccionario*: Concatenan una secuencia de palabras provenientes de uno o más diccionarios embebidos en la propia muestra u obtenidos de Internet.
- *Basados en permutaciones*: Derivan todos los dominios posibles a partir de permutar los caracteres de un dominio inicial.

De entre todos ellos el más común es el método de generación aritmética del nombre (37 de los 43 DGAs analizados en [30]). Perteneciente a este tipo son Conficker (tabla 2.2) o Corebot (listado 2.3), ya descritos con anterioridad. El segundo de los métodos más empleados (3 de los 43 DGAs analizados) es el de DGAs basados en diccionario. El listado 2.4 muestra un ejemplo de varios nombres de dominio generados por el malware Gozi [29] que emplea este tipo de DGA. En concreto, este DGA se basa en un diccionario de palabras que elige pseudoaleatoriamente hasta alcanzar una longitud mínima en el nombre del dominio construido.

Como se puede observar, los nombres de dominio generados empleando esta técnica son visualmente muy diferentes a los generados de forma totalmente pseudoaleatoria ya

que a simple vista no dan al observador la misma sensación de aleatoriedad. Dominios como "missiondeveloped.com" podrían haber sido registrados por un ser humano y usarse para ofrecer en Internet servicios legítimos. Sin embargo, el uso de estas aproximación basada en diccionario reduce el espectro de posibles nombres de dominio que un DGA es capaz de generar ya que se encuentra limitado por la longitud del nombre de dominio que genera y el tamaño del diccionario empleado. Por ejemplo, al generar 100 millones de nombres de dominio con el DGA de Gozi se produjeron un 14.26 % de repeticiones. Esto no ocurre con Corebot, que emplea un DGA aritmético, con el que se generaron los 100 millones de nombres de dominio sin que se produjera ninguna repetición.

2.2. Protocolo y tráfico DNS

El protocolo DNS [31] describe los mecanismos necesarios para obtener una dirección IP a partir de un nombre de dominio. Esta resolución se solicita por parte de los dispositivos (cliente DNS) y es respondida por los servidores DNS. De entre los servidores DNS se pueden distinguir dos grandes tipos. En primer lugar están los servidores autorizados, que son los encargados de gestionar uno o varios dominios concretos. En segundo lugar están los servidores recursivos, que gestionan las peticiones DNS realizadas por una subred y se encargan de obtener de los servidores autorizados correspondientes la respuesta a las peticiones y de transmitirla al cliente DNS que la realiza.

Los paquetes DNS que se pueden encontrar en el tráfico de red tienen una estructura como la mostrada en la figura 2.3 en la que el valor de uno de los campos especifica si se trata de una petición o una respuesta. En concreto, la mayor parte de la información útil para identificar un dominio malicioso se encuentra en los registros de recurso cuyo formato se muestra en la figura 2.4.

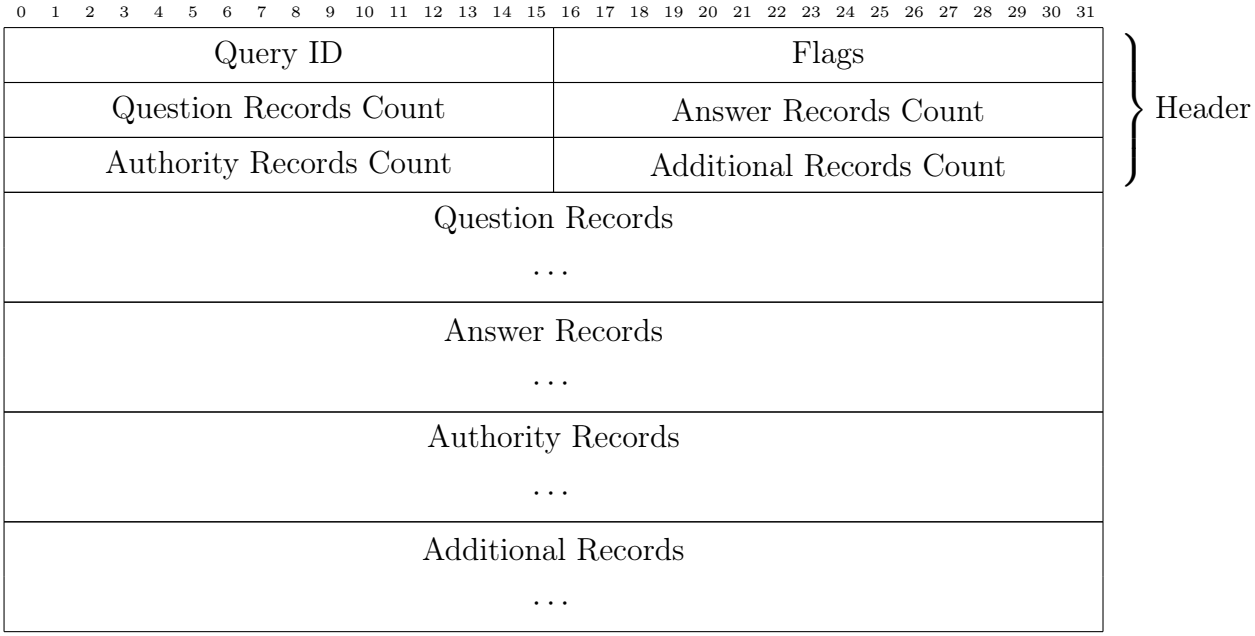


Figura 2.3: Estructura de un paquete DNS.

Estos registros de recurso muestran información que refleja cómo está configurado el servidor DNS autorizado para el dominio malicioso, el cual se encuentra bajo el control

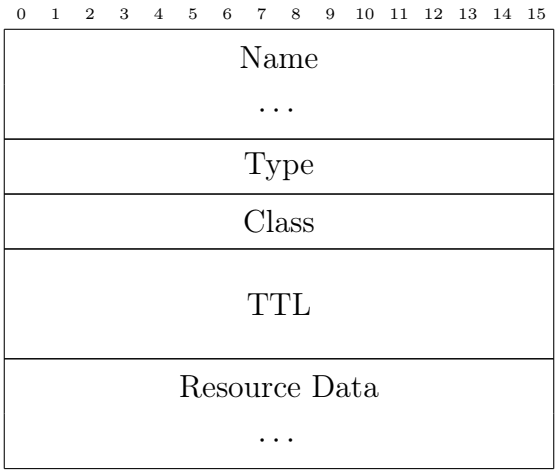


Figura 2.4: Registro de recurso DNS.

del desarrollador u operador del malware. Por tanto en esta información pueden verse reflejadas características que permitan identificar un dominio malicioso.

Como parte de esta información se encuentra también el nombre del dominio cuya resolución se ha solicitado. Este dato es especialmente útil en el caso de los nombres de dominio generados mediante DGAs, ya que dicho nombre tiene unas características especiales que pueden hacerlo identificable con respecto a otros nombres de dominio creados por un ser humano.

Por otro lado, es habitual también obtener lo que podrían considerar metadatos sobre el dominio. Esta información no se encuentra en el tráfico de red en sí mismo sino que se obtiene mediante los servidores de *whois* o directamente de los agentes registradores, que son las entidades encargadas de gestionar el registro de los dominios.

Tal y como se puede observar en el listado 2.5, que muestra la información del dominio "uv.es", es posible recolectar parte de la información proporcionada al registrar el dominio como pueden ser los datos de contacto, el agente registrador, la fecha de registro y la fecha de última actualización. Esta información, al igual que ocurre con la información extraída directamente del tráfico de red, resulta útil para identificar características propias de los dominios maliciosos ya que estos tienden a mostrar una información más opaca, a emplear agentes registradores con controles más laxos o actualizar su contenido de una forma inusual.

2.3. Modelos

2.3.1. Árboles de decisión

Un árbol de decisión es una manera de representar reglas obtenidas a partir de los datos en una estructura jerárquica y secuencial que dividen los datos de forma recursiva [32]. En esta representación cada nodo evalúa una característica y la estructura se ramifica en cada una de sus posibles respuestas, dando lugar a un resultado o a un nuevo nodo que evalúa alguna de las características relevantes. La figura 2.5 muestra un ejemplo clásico de árbol de decisión, en el que se usan diferentes características asociadas al tiempo atmosférico para decidir si es un buen momento para jugar. En cada nodo de este árbol se evalúa

```
refer:        whois.nic.es
[...]
contact:      administrative
name:         David Cierco Jiménez de Parga
organisation: Red.es
address:      Edificio Bronce
address:      Plaza Manuel Gomez Moreno
address:      Madrid 28020
address:      Spain
phone:        +34 91 212 76 24
fax-no:       +34 91 555 76 64
e-mail:       esnic-admin@red.es

nserver:      A.NIC.ES 194.69.254.1 2001:67c:21cc:2000:0:0:64:41
nserver:      C.NIC.ES 194.0.34.53 2001:678:44:0:0:0:0:53
nserver:      F.NIC.ES 130.206.1.7 2001:720:418:caf1:0:0:0:7
nserver:      G.NIC.ES 2001:500:14:7001:ad:0:0:1 204.61.217.1
nserver:      H.NIC.ES 194.0.33.53 2001:678:40:0:0:0:0:53
nserver:      NS-ES.NIC.FR 194.0.9.1 2001:678:c:0:0:0:0:1
nserver:      NS1.CESCA.ES 2001:40b0:1:1122:ce5c:a000:0:3 84.88.0.3
ds-rdata:     50252 8 2
    dd2b515da6dbc0e826006e6ae02864c05825694a84b3e4e0e59918c78c1bc8eb
ds-rdata:     50252 8 1 9bf0115b43a4ee8f7223fbf5b294d91cdeed2dbe

created:      1988-04-14
changed:      2021-04-27
source:       IANA
```

Listado 2.5: Información del dominio “uv.es”.

una característica concreta y se da una respuesta a la pregunta planteada (por ejemplo, *¿Podemos jugar?*) o bien se plantea una nueva pregunta. En este caso el proceso se repite hasta que el árbol proporciona una respuesta. Este tipo de estructuras son ampliamente usadas en muchos campos ya que es una manera muy intuitiva de establecer un proceso de preguntas y respuestas que llevan a una solución. Es, de hecho, una de las formas que emplean los seres humanos de forma instintiva para la toma de decisiones.

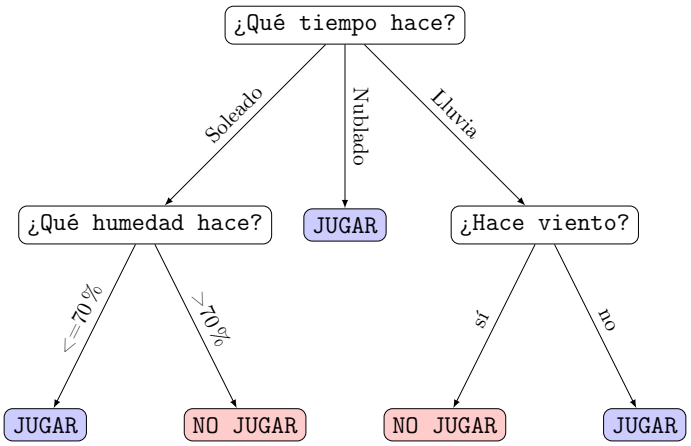


Figura 2.5: Ejemplo clásico de árbol de decisión.

A pesar de que este tipo de diagramas han sido ampliamente utilizados durante décadas, las técnicas para la creación de ellos a partir de los datos (inducción de árboles de decisión) son de especial interés para esta tesis. Según Sammut et al. [33], la inducción de árboles de decisión es una de las técnicas más antiguas y populares en modelos de aprendizaje discriminativo. Estas técnicas han sido desarrolladas de forma independiente tanto en el campo de la estadística [34, 35] como en el del aprendizaje automático [36, 37, 38].

La inducción de estos árboles se basa fundamentalmente en ser capaces de encontrar una característica o combinación de características que tienen una buena capacidad para discriminar el conjunto de datos [32]. Por este motivo, una tarea esencial para la construcción de un árbol de decisión es dar con algún tipo de técnica que permita establecer una clasificación de las características (o combinaciones de ellas) que proporcionan una mejor capacidad para discriminar los datos. Una vez obtenida esta clasificación, la construcción de árbol consiste en elegir la característica más discriminante para la creación del primer nodo, separar los conjuntos de datos en base a este primer discriminante y repetir el proceso para cada uno de las ramas del árbol.

Ser capaces de evaluar esta capacidad de las características para discriminar el conjunto de datos es la parte clave de esta técnica y tiene un impacto directo en la capacidad de inducir un árbol de decisión que modele los datos de la forma más correcta posible. Por ello han existido a lo largo de los años diferentes técnicas, siendo las siguientes las más empleadas:

- *Iterative Dichotomiser 3* (ID3) [38]: Esta técnica calcula la entropía resultante tras la división del conjunto de datos para cada una de las características y elige la característica que minimiza dicha entropía con respecto a la entropía del conjunto de datos antes de la división. Gran parte de las técnicas usadas hoy en día se basan en mayor o menor medida en esta propuesta.
- C4.5 [39]: Propuesto como extensión de ID3. En ella se introduce la capacidad de poder utilizar valores continuos como características. También se introducen nuevos

mecanismos para mejorar la capacidad de generalización del método, al contrario de lo que ocurre con ID3, que tiene tendencia al sobreentrenamiento.

- *Classification and Regression Trees* (CART) [40]: En lugar de emplear una medida de discriminación basada en la entropía como ID3 y C4.5, este método emplea la llamada *impureza de Gini* (ecuación 2.1). Este valor se calcula a partir de la probabilidad p que un elemento de cada una de las divisiones del conjunto de entrenamiento pertenezca a una clase i (p_i en la ecuación). Por tanto, la impureza de Gini es cero cuando el conjunto de datos es homogéneo e incrementa su valor cuanto más heterogeneidad existe en los datos.

$$Gini = 1 - \sum_{i=1}^{C} p_i^2 \tag{2.1}$$

Además del uso de la entropía o de la impureza de Gini, estos métodos tienen algunas otras diferencias que les hace construir árboles de decisión diferentes para el mismo conjunto de datos. Una de las diferencias más visuales es que CART siempre divide el conjunto de entrenamiento en dos subconjuntos, construyendo por tanto un árbol binario. Por el contrario, C4.5 divide el conjunto de datos en múltiples subconjuntos, por lo que en función de la característica evaluada los nodos del árbol pueden tener una cantidad arbitraria de ramas. Por este motivo, dado un árbol de decisión binario, lo más probable es que haya sido generado mediante el método CART o un método similar, mientras que un árbol no binario es muy probable que haya sido generado utilizando el método C4.5 o alguna de sus variantes.

En general, las técnicas usadas para la inducción de árboles de decisión tienen tendencia a producir árboles sobreentrenados, es decir, árboles muy grandes y de mucha profundidad que se ajustan a cada uno de los datos del conjunto de entrenamiento de forma exacta. Esto reduce la capacidad de generalización de este método de aprendizaje y, por tanto, producen resultados inesperados cuando se pretende clasificar un dato que no existía en su conjunto de entrenamiento.

Con el fin de evitar este problema se introduce el concepto de poda. Este mecanismo permite reducir el árbol resultante al introducir un factor que penaliza la creación de una cantidad excesiva de nodos hoja. Gran parte de los hiperparámetros que se pueden ajustar están relacionados con factores que permiten podar el árbol y mejorar la generalización del árbol resultante. Sin embargo, muchos de estos hiperparámetros son valores absolutos que no son introducidos basándose en los datos. Una configuración inadecuada de los mismos puede tener como consecuencia un subentrenamiento, que consiste en la situación opuesta al sobreentrenamiento, es decir, la creación de un árbol excesivamente pequeño que no se ajusta bien a los datos y no proporciona buenos resultados en la clasificación. Por este motivo, estos valores deben configurarse tras un profundo estudio y validación, con el fin de encontrar aquellos valores que proporcionan los resultados óptimos para resolver el problema.

2.3.2. Random Forest

El método *Random Forest* [41] emplea una estrategia en la que se inducen varios estimadores de forma independiente para posteriormente seguir una aproximación de tipo

ensemble. Esta aproximación consiste en combinar el resultado de cada uno de los estimadores para proporcionar un resultado único. La manera de combinar cada uno de estos estimadores depende del propósito del modelo. En un modelo clasificador, por ejemplo, se emplearía un esquema de votación en el cual se cuenta cuantos estimadores dan como resultado cada una de las clases candidatas, eligiendo aquella clase que obtuvo más votos. Otros tipos de problemas diferentes a la clasificación adoptan otras estrategias para combinar los resultados de cada uno de los estimadores, como por ejemplo calcular un promedio de los resultados para obtener el valor de una regresión [42].

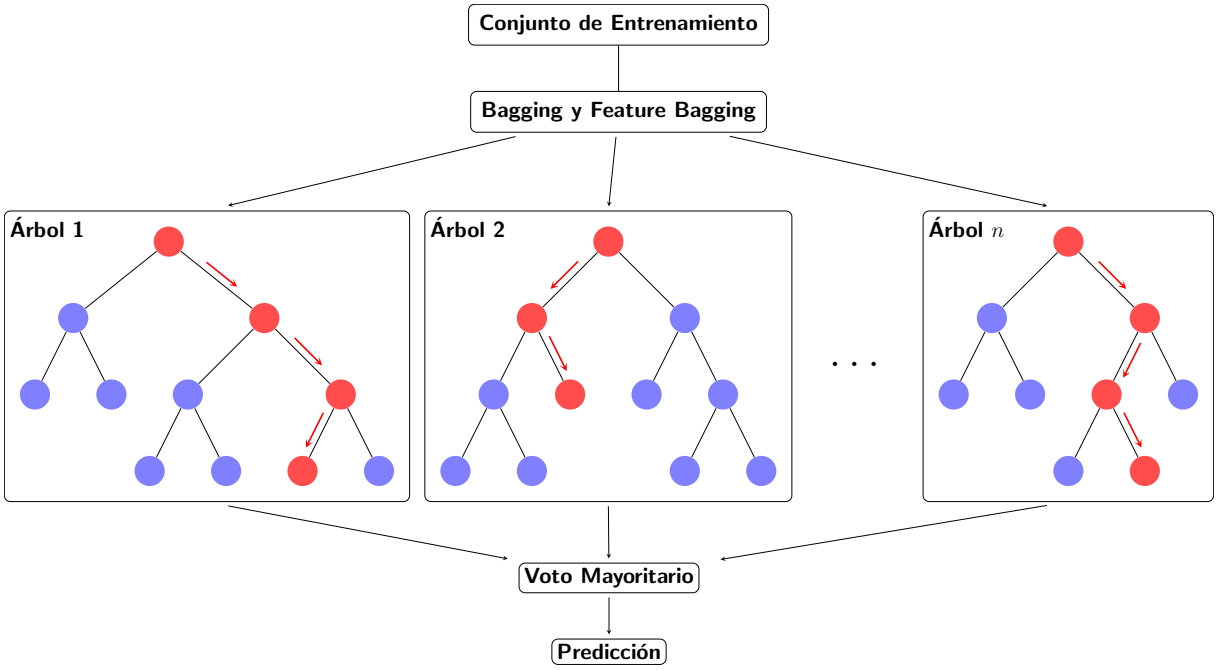


Figura 2.6: Esquema de un *Random Forest*.

Cada uno de estos estimadores puede denominarse *Random Tree* ya que son árboles de decisión inducidos como se explicó en la sección 2.3.1 pero con ciertas modificaciones que les dotan, en su conjunto, de una mejor capacidad de generalización que un árbol de decisión simple, que como se comentó tienen tendencia al sobreentrenamiento. Este diseño puede apreciarse de forma esquemática en la figura 2.6.

En primer lugar, cada uno de estos *Random Trees* es entrenado con una muestra aleatoriamente extraída del conjunto de entrenamiento. Esto produce que la inducción de los árboles, incluso aún produciéndose sobreentrenamiento, sea diferente para cada uno de ellos. Cada uno de los árboles inducidos se ajusta a un conjunto de datos de entrenamiento diferente o parcialmente diferente, por lo que las clasificaciones propias del sobreentrenamiento dejan de ser significativas al quedar diluidas entre las clasificaciones del resto de árboles. A esta técnica se la conoce como *bagging* y es ampliamente usada en *ensemble learning* para evitar que todos los estimadores sean entrenados con el mismo conjunto de datos y, por tanto, repliquen los mismos problemas [42].

Por otro lado, el método *Random Forest* también incluye lo que se llama *feature bagging*. Esta técnica consiste en elegir una muestra aleatoria de las características y calcular el valor de heterogeneidad de cada una de las características elegidas, quedando el resto excluidas de la decisión. A partir de esos valores se selecciona la característica que produce la división del conjunto de datos de entrenamiento y se prosigue con el algoritmo de inducción. Esto favorece que si existe alguna característica que resulta ser prevalente

para la clasificación, esta no sea siempre elegida de forma inicial en la inducción de los árboles. El uso de *feature bagging* crea más diversidad entre los estimadores y, por tanto, mejora la generalización del modelo.

Gran parte de los hiperparámetros que se pueden ajustar en un modelo *Random Forest* son idénticos a los de un árbol de decisión. Esto es debido a que, tal y como se mencionaba anteriormente, *Random Forest* está formado por varios árboles de decisión que se inducirán en base a dichos parámetros. Sin embargo, algunos de los hiperparámetros son específicos de *Random Forest*, y se describen a continuación:

- Cantidad de estimadores (árboles) empleados. Las diferentes implementaciones tienen un valor configurado por defecto aunque dicho valor puede ser reducido o ampliado en función de las características del problema a resolver.
- Uso de *bagging* para la elección del conjunto de entrenamiento de cada uno de los árboles o no.
- Cantidad de elementos o porcentaje de los mismos elegidos de forma aleatoria del conjunto de entrenamiento para realizar el *bagging*.
- Cantidad máxima de características o porcentaje de las mismas elegidas de forma aleatoria para realizar el *feature bagging*. En algunas implementaciones toma por defecto el valor de la raíz cuadrada de la cantidad total de características.

2.3.3. Boruta

Llamado así por el Dios eslavo de los bosques [43], Boruta es una técnica para selección de características que emplea como base un modelo *Random Forest*. El propio modelo *Random Forest*, de forma natural, ya facilita una manera de determinar qué características resultan ser más importantes, por lo que Boruta aprovecha esta característica para construir su método de selección de características apoyándose en ella.

Para determinar esta importancia en problemas de clasificación se calcula la tasa de error en la predicción de cada árbol en un conjunto de datos diferente al *bag* con el que dicho árbol fue entrenado. Una vez obtenido este valor se repite el cálculo tras realizar una permutación en cada una de las características en ese conjunto de datos y se calcula también la tasa de error para cada una de estas pruebas. Con estos datos se calcula para cada característica la diferencia entre la tasa de error inicial y tras la permutación y se obtiene un promedio de todos los árboles. Este valor se normaliza empleando la desviación estándar de las diferencias para obtener la puntuación final de la importancia, llamada *Z score*. Existen otras técnicas alternativas para medir la importancia como por ejemplo la basada en el descenso medio de la impureza de los nodos al construir los árboles, si bien la primera de ellas sería la implementación por defecto más habitual de las librerías más ampliamente utilizadas [44].

En concreto, la tabla 2.3 muestra un ejemplo de la importancia de las características tras el entrenamiento de un modelo *Random Forest* en la que se observa como existen ciertas características con una importancia mucho mayor que otras a la hora de determinar el resultado de dicho modelo.

Este valor de la importancia que se muestra en la tabla 2.3 muestra valores entre 0 y 100. Esto es debido a que la puntuación es escalada para facilitar la interpretación de

V18	**V12**	**V4**	**V6**	**V10**	**V5**	**V8**	**V9**
100.00	81.64	61.51	51.17	49.02	48.84	38.21	35.75
V17	**V11**	**V13**	**V16**	**V15**	**V14**	**V7**	
8.63	7.61	5.84	4.29	3.23	0.85	0.00	

Tabla 2.3: Resultado de importancia en las características en un modelo *Random Forest*.

qué características serían más importantes que otras y ver los órdenes de magnitud entre ellas. No obstante, da lugar a confusión, puesto que se puede interpretar que en este caso la característica V18 tiene un 100 % de importancia para la clasificación y que por tanto permitiría clasificar con total efectividad empleando esta única característica. Esta es una conclusión incorrecta ya que esta puntuación de 100 únicamente quiere decir que esta característica es la de mayor importancia. El resto de importancias son escaladas en base a la puntuación de esta, motivo por el cual este valor es 100.

No obstante, aunque permite comparar la importancia entre características, no proporciona una manera intuitiva de analizar la importancia real que cada una de las características tienen en la clasificación y, por tanto, facilitar la selección de características. Por este motivo, Kursa y Rudnicki diseñaron la técnica Boruta como un método para facilitar este análisis y la selección de características de una forma mucho más intuitiva [43]. Con este fin, Boruta genera una nueva característica artificial (*shadow*) por cada una de las características reales. Los valores de estas características *shadow* se obtienen de barajar los valores de la característica original de diferentes elementos del conjunto de datos, lo cual produce que los valores resultantes para cada una de las características *shadow* pierdan la correlación con el resultado de la clasificación.

Una vez generadas estas características *shadow* se realiza el entrenamiento del modelo *Random Forest* empleando estas características junto con las características originales, para posteriormente evaluar la importancia de ellas utilizando la técnica descrita anteriormente.

Por tanto, las características *shadow* actúan como referencia externa contra la que comparar la importancia de las características reales y determinar si estas aportan un valor significativo para resolver el problema de clasificación. Al ser las características *shadow* aleatorias por diseño, según [43], las características reales con valores de *Z score* cercanos a los de las características *shadow* son características que aportan una información cercana a ser aleatoria. Estas características, por tanto, pueden ser descartadas sin que afecten de forma significativa al resultado del entrenamiento y de las predicciones posteriores.

Una manera intuitiva de realizar la selección de características es generar un gráfico de cajas (*boxplot*, en inglés) a partir de las puntuaciones de importancia obtenidas mediante el algoritmo Boruta, como el mostrado en la figura 2.7. Este gráfico muestra de forma visual todas las características reales, así como un resumen de las características *shadow*. En concreto, los siguientes tres elementos muestran información de las características *shadow*:

- *shadowMin*: Importancia más baja de entre todas las características *shadow*.
- *shadowMean*: Importancia media calculada a partir de todas las características *shadow*.
- *shadowMax*: Importancia más alta de entre todas las características *shadow*.

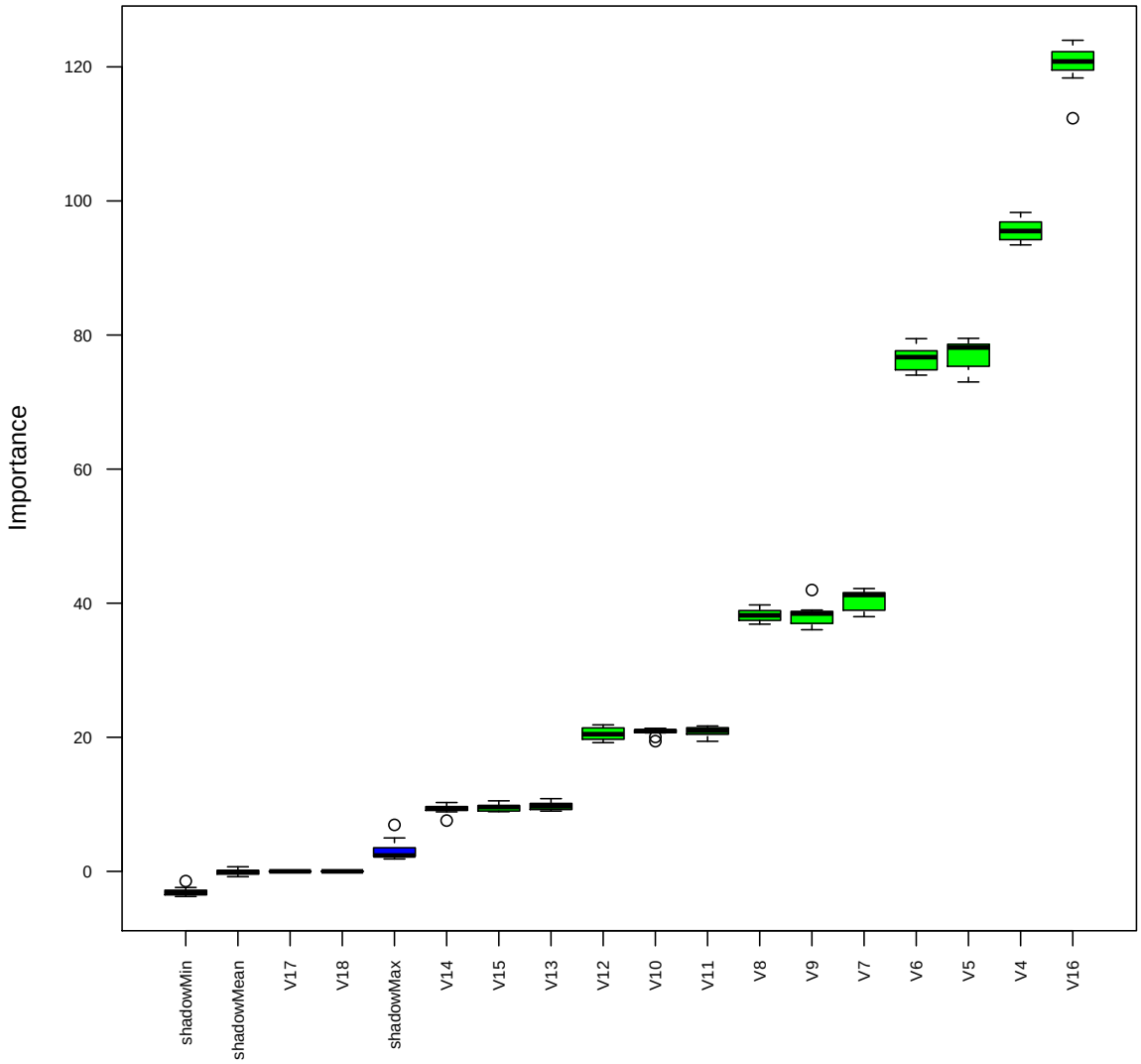


Figura 2.7: Ejemplo de un *boxplot* generado por Boruta.

Según lo mostrado en el gráfico, las características V17 y V18 son claramente no relevantes dado que su importancia es inferior a *shadowMax*, es decir, inferior a características consideradas aleatorias. No obstante, la gráfica también muestra que V13, V14 y V15 tienen una importancia levemente superior a las de las características *shadow* y, por tanto, también se puede prescindir de ellas para construir el clasificador sin un gran impacto en la calidad del mismo. Es a partir de V8 que la importancia comienza a crecer de forma exponencial. Por tanto, son esas 7 características las que resultan más relevantes para el problema y, en menor medida, V10, V11 y V12. El resto de características serían pues potencialmente descartables.

2.3.4. Redes neuronales

Una red neuronal artificial, también conocida simplemente como red neuronal o *Neural Network* (NN), es un modelo de aprendizaje automático inspirado en la organización de

las neuronas que se da en los cerebros de animales, que se interconectan entre ellas para realizar tareas complejas.

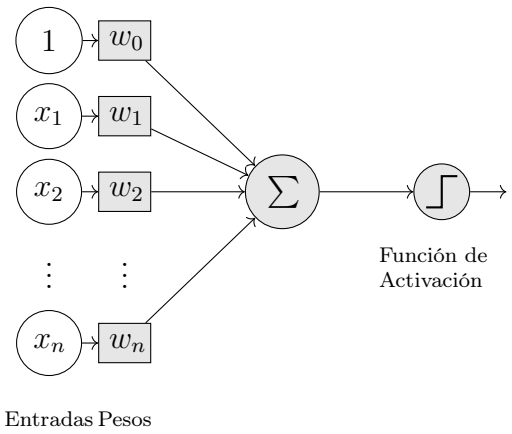


Figura 2.8: Diagrama de un perceptrón.

La primera propuesta de neurona artificial fue denominada perceptrón y su diseño se muestra en la figura 2.8. La arquitectura del perceptrón consiste en una serie de coeficientes (pesos sinápticos, $w_i, 0 \leq i \leq n$) asociados a cada una de las entradas, que multiplican el valor de las mismas para luego sumar el resultado de los productos y obtener el valor de z (ecuación 2.2). En dicha ecuación, el valor $1 \cdot w_0$, que representa al *bias* del diseño del perceptrón, es substituido por b. Finalmente la ecuación 2.3 muestra como se aplican una función lineal σ (función de activación) que produce la salida a del perceptrón. Este modelo es capaz de aprender funciones lineales sencillas como por ejemplo las funciones lógicas OR y AND, pero no es capaz de aprender funciones más complejas que no son linealmente separables como XOR [45].

$$z = \sum_{i=1}^{n} x_i w_i + b \tag{2.2}$$

$$a = \sigma(z) \tag{2.3}$$

Estas neuronas pueden ser distribuidas en capas y ser empleadas para construir una arquitectura que se conoce como un perceptrón multicapa (figura 2.9), capaz de sobrepasar las limitaciones del perceptrón tradicional y aprender funciones mucho más complejas gracias al uso de las múltiples capas.

Para implementar el aprendizaje de estas redes neuronales se diseña la técnica de la propagación hacia atrás [46]. Conceptualmente, esta técnica describe un mecanismo por el cual el error en los valores de salida de la red neuronal es propagado hacia atrás, determinando el grado de responsabilidad δ que cada uno de los nodos de la red tiene sobre dicho resultado. A partir de este valor se calcula un nuevo valor de los pesos que mejora el resultado global de la red.

El primer paso para la ejecución del mecanismo de propagación hacia atrás es el cálculo del error en la salida. En una red neuronal compuesta de L capas, las salidas de la red se denotan como a_j^L, siendo j el índice que especifica a qué salida concreta de la red se refiere. Este valor a_j^L, junto con el valor esperado en esa salida y_j, son utilizados para calcular el error en la predicción por medio de la función de coste. La función de coste es uno de los

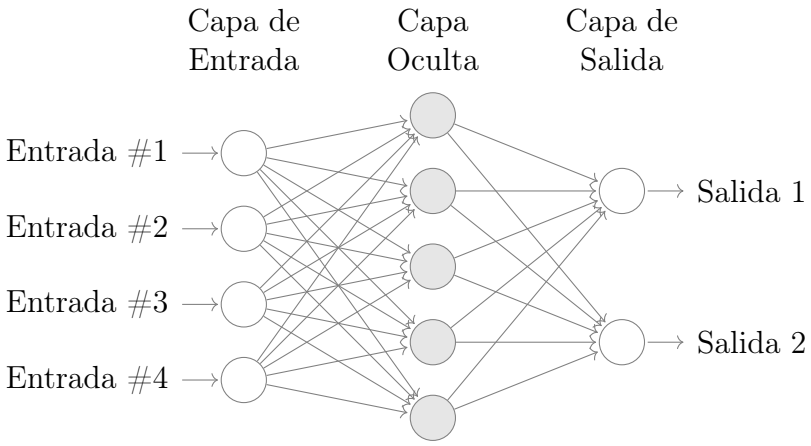


Figura 2.9: Perceptrón multicapa.

hiperparámetros de la red ya que existen diferentes alternativas de función de coste que se pueden emplear para configurar una red neuronal. Un ejemplo de función de coste es el error cuadrático medio, que se describe mediante el cálculo de $\left(y_j - a_j^L\right)^2$, o de la media de todos los valores obtenidos tras evaluar el error de más de una entrada. A partir de este cálculo del error C_j, el error de cada una de las neuronas de la capa de salida L se calcula mediante la ecuación 2.4, donde z_j^L representa el resultado de la suma ponderada que se produce en la neurona j de la capa L. Este error proporciona una intuición del grado de responsabilidad que una neurona concreta tiene en el resultado global de la red. Un valor grande de δ en una neurona significa que lo ocurrido en dicha neurona tiene una gran relevancia en el resultado final que produce el error, mientras que un valor bajo significa que el resultado de salida de dicha neurona apenas tiene relevancia en el error en la salida de la red.

$$\delta_j^L = \frac{\partial C_j}{\partial z_j^L} = \frac{\partial a_j^L}{\partial z_j^L} \cdot \frac{\partial C_j}{\partial a_j^L} \tag{2.4}$$

La derivada parcial $\frac{\partial a_j^L}{\partial z_j^L}$ es la derivada de la función de activación de la neurona que, al igual que la función de coste, es un hiperparámetro de la red y puede variar de unas configuraciones de red a otras. Por ejemplo, en el caso de la función de activación sigmoide σ su derivada es $\sigma\left(z_j^L\right) \cdot \left(1 - \sigma\left(z_j^L\right)\right)$. Al ser $\sigma\left(z_j^L\right)$ el resultado de aplicar la función de activación a la salida de la suma ponderada que tiene lugar en la neurona, es igual a a_j^L.

La derivada parcial $\frac{\partial C_j}{\partial a_j^L}$ es la derivada de la función de coste que, en el caso del error cuadrático medio, es $2\left(y_j - a_j^L\right)$ aunque se puede simplificar como $\left(y_j - a_j^L\right)$. De resolver ambas derivadas para los hiperparámetros concretos de la red se obtiene la ecuación 2.5, que es válida para calcular el valor δ de las neuronas de la capa de salida [42].

$$\delta_j^L = a_j^L\left(1 - a_j^L\right)\left(y_j - a_j^L\right) \tag{2.5}$$

La ecuación 2.6 generaliza este resultado para cualquier capa l distinta a L, es decir, toda las capas menos la capa de salida de la red. El grado de responsabilidad de la neurona j de la capa l cuando esta no es la capa de salida está en función de la derivada de la función de activación, al igual que ocurre en el caso particular de la capa de salida L. Además, esta ecuación incluye un factor que substituye a la derivada de la función de coste de la

ecuación 2.5, debido a que dicho valor solo puede ser usado en la capa de salida por ser la capa de la que es posible calcular el error de forma inmediata. Este factor es substituido por el sumatorio, para toda neurona k con la que la neurona j se encuentra interconectada en la siguiente capa de la red (representado como $downstream(j)$), del resultado de multiplicar el valor δ de dicha neurona k de la capa $l+1$ por el peso de la interconexión entre dicha neurona y la neurona j de la capa l, denotado como w_{jk}. Se deduce de esta ecuación que la responsabilidad de una neurona cualquiera de la red está en función de la responsabilidad de las neuronas de la capa siguiente con las que está interconectada y del peso de dichas interconexiones. Dicho de otro modo, cada neurona transmite su responsabilidad (o error) hacia atrás de una forma proporcional a los pesos de sus interconexiones con las neuronas de la capa anterior [42]. Esto responde a la intuición tras el algoritmo de propagación hacia atrás, dado que una interconexión entre dos neuronas con un peso grande tiene mayor efecto en el resultado final y por tanto en el error que una interconexión entre dos neuronas con un peso menor.

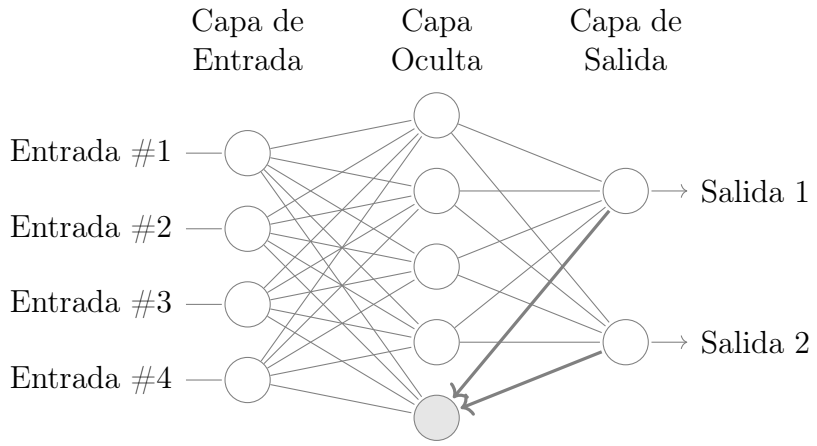
$$\delta_j^l = a_j^l \left(1 - a_j^l\right) \sum_{k \in downstream(j)} w_{jk} \cdot \delta_k^{l+1} \tag{2.6}$$

Este proceso se repite de forma iterativa desde la capa de salida L hasta la capa de entrada, obteniendo los valores δ para cada una de las neuronas de la red. La figura 2.10(a) muestra de forma gráfica como en primer lugar las neuronas de la capa de salida propagan hacia atrás el error a cada una de las neuronas de la capa anterior. Una vez calculado δ para todas las neuronas de esta capa se repite el proceso con la capa anterior, tal y como se observa en la figura 2.10(b). El proceso termina cuando se recorren todas las capas de la red y se obtienen los valores δ de todas y cada una de las neuronas.

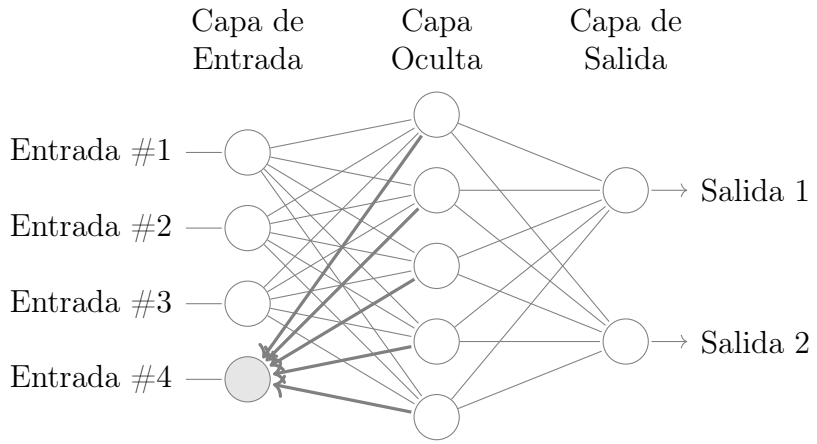
Una vez que se dispone del valor δ para todas las neuronas de la red es posible calcular la actualización de los pesos. La formula exacta para calcular el valor de los pesos varía en función del algoritmo de optimización escogido. Por ejemplo, en el caso del descenso por gradiente estocástico la ecuación de la actualización de los pesos es $w_{ij} = w_{ij} - \eta\delta_j a_i$, siendo w_{ij} el peso de la interconexión entre la neurona i y la neurona j, a_i la entrada de la interconexión (la salida de la neurona i) y η un factor llamado tasa de aprendizaje. Esta tasa de aprendizaje es un ratio que determina cuánto efecto tiene el cambio en el peso de dicha interconexión. Cuando este valor es cercano a 1, la actualización del peso se realiza de tal forma que se minimiza el error en un solo paso. Esto ocasiona que el aprendizaje sea más rápido pero, al mismo tiempo, aumenta la probabilidad de que el aprendizaje nunca llegue a converger. Del mismo modo, cuando esta tasa de aprendizaje es cercana a cero, la actualización de los pesos es muy leve, por lo que son necesarias muchas más rondas de entrenamiento para completar el aprendizaje, es decir, el aprendizaje es más lento.

Este cálculo del error y actualización de los pesos puede realizarse tras haber procesado uno solo de los datos del conjunto de entrenamiento (*online*), todos los elementos del conjunto de entrenamiento (*batch*) o un subconjunto de dichos datos (*mini-batch*). El uso de un lote de tipo *mini-batch* de tamaño N implica que N datos son procesados por la red y el cálculo del error δ_k de las neuronas de salida se calcula en base a la media de los errores de los N elementos del lote. El tamaño del lote, por tanto, también condiciona la manera en la que los pesos son actualizados, ya que un lote pequeño ocasionará cambios muy bruscos en los pesos mientras que un lote grande diluye sus efectos al realizarse la media del error de N elementos.

El entrenamiento de una red neuronal se produce en varias rondas llamadas épocas.



(a) Primer paso.



(b) Segundo paso.

Figura 2.10: Propagación del error.

En cada una de estas épocas el conjunto de datos de entrenamiento se emplea de forma completa, separado en sus respectivos lotes. En función del tamaño del lote y de la tasa de aprendizaje, los pesos se actualizan de forma más o menos rápida para ajustarse a lo observado en el conjunto de entrenamiento según el mecanismos explicado con anterioridad. Una vez completada una época, la exactitud (*accuracy*) de la red se evalúa empleando el conjunto de datos de validación, que contiene datos que no fueron usados para el entrenamiento, y se inicia una nueva época. Este proceso continúa hasta alcanzar el número máximo de épocas establecido como hiperparámetro o hasta que se cumpla una condición de salida. Las condiciones de salida pueden ser varias y diversas, pero las más habituales son no obtener mejores resultados en una cantidad determinada de épocas o llegar a un cierto grado de exactitud establecido como objetivo.

Otro de los hiperparámetros de una red neuronal que tiene gran importancia en el resultado es la función de activación. Tal y como se muestra en la figura 2.8, la salida de cada una de las neuronas es procesada por una función de activación. Al tener influencia en el valor de salida de la neurona, dicha función también influye en la actualización de pesos durante el proceso de entrenamiento. Tradicionalmente las funciones de activación más empleadas en redes neuronales han sido la función sigmoide y la tangente hiperbólica, aunque existe una amplia variedad de funciones que pueden ser empleadas. Como muestra de ello, el conocido entorno para desarrollo de redes neuronales Keras soporta diferentes funciones de activación [47]. Algunas de las usadas más frecuentemente se listan

a continuación, si bien las figuras 2.11 muestran las diferencias entre ellas de un modo más visual e intuitivo:

- *Sigmoide*: Recibe una entrada de cualquier valor y la transforma a una salida en el rango entre 0 y 1, siguiendo la función $sigmoid(x) = \frac{1}{1+e^{-x}}$. Es una función ampliamente usada en el campo de las redes neuronales, puesto que garantiza una salida acotada del perceptrón independientemente de las entradas.

- *Tanh*: Tiene un aspecto similar a la sigmoide pero devuelve una salida entre -1 y 1, y su transición es más brusca al converger a sus valores máximo y mínimo. Está descrita por la ecuación $tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$. Es junto con la función sigmoide una de las funciones de activación más ampliamente usadas en la literatura [42], especialmente hasta la popularización de las funciones ReLu.

- *Softsign*: Función similar la tangente hiperbólica. También devuelve una salida entre -1 y 1 pero es una función que converge de forma polinómica, al contrario que la tangente hiperbólica que converge de forma exponencial. A nivel práctico, *softsign* presenta una transición más progresiva mientras que la tangente hiperbólica es practicamente lineal en valores no muy cercanos a -1 y a 1, para luego converger de forma rápida a estos valores. Está definida por la ecuación $softsign(x) = \frac{x}{|x|+1}$.

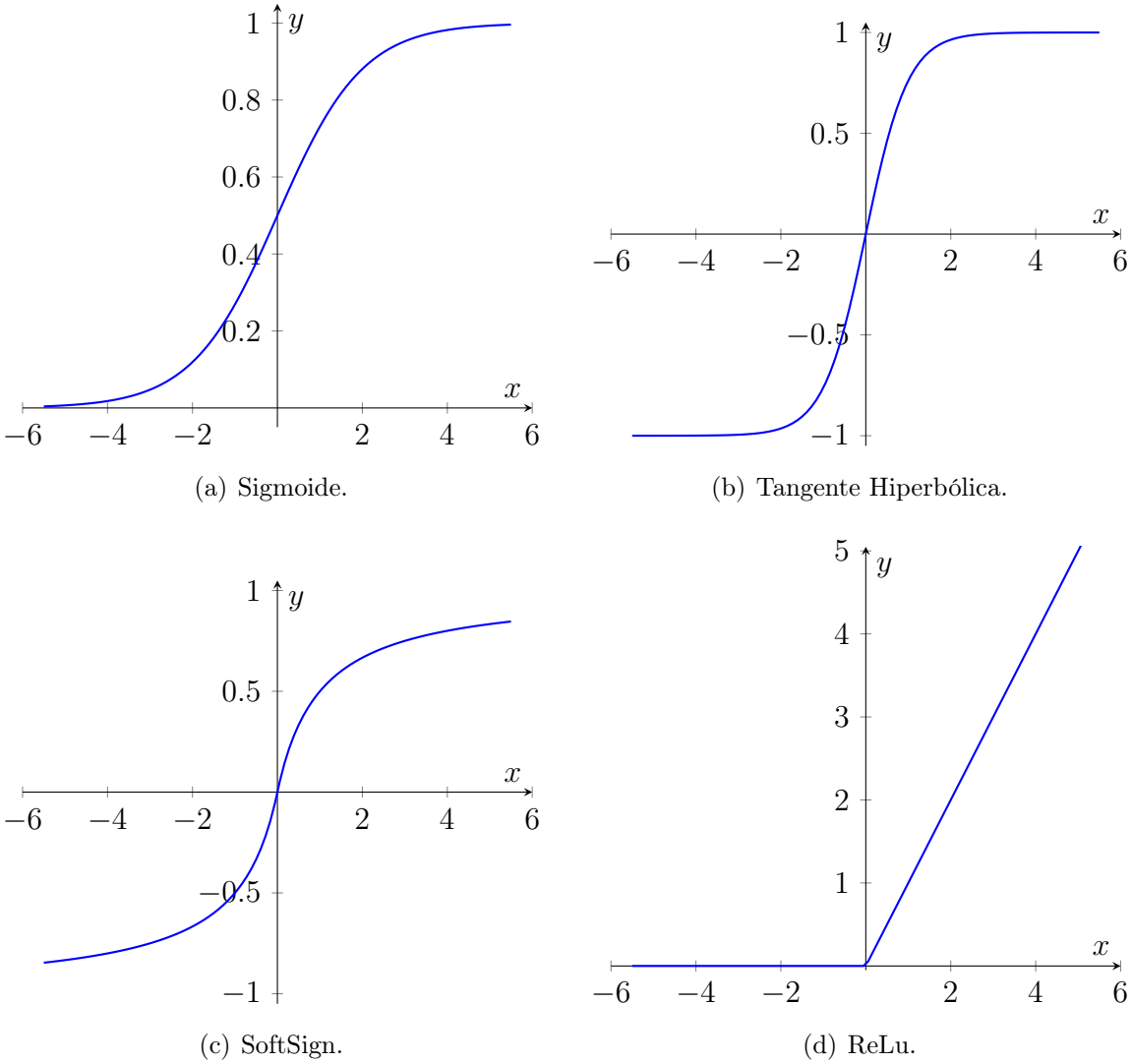
- *ReLu*: Creada como alternativa a otras funciones para evitar el problema del desvanecimiento del gradiente consistente en una ralentización en la velocidad del aprendizaje para valores cercanos a los límites de la función, en los que la derivada de la función es cercana a cero. Se utiliza con frecuencia como función de activación de las capas ocultas porque evita la saturación de las neuronas. Se define según la ecuación $relu(x) = max(0, x)$.

A partir de estos conceptos se han desarrollado multitud de propuestas de diseños de redes neuronales que tiene sus particularidades con respecto a la propuesta original. Ejemplos de estas propuestas son las redes neuronales profundas, redes convolucionales o redes neuronales recurrentes, entre otras.

2.3.5. Redes neuronales *Long Short-Term Memory* (LSTM)

Por lo general, cuando se piensa en el diseño de una red neuronal se piensa en una red neuronal de tipo *feed-forward* como las explicadas en la sección 2.3.4. En este tipo de redes los datos de entrada alimentan la primera capa de neuronas, sus resultados son proporcionados a la siguiente capa y así sucesivamente hasta que ese flujo llega a la capa de salida sin que se produzca ningún bucle ni realimentación en este flujo. Como consecuencia de este diseño, estas redes neuronales procesan un determinado dato de entrada siempre de la misma forma, independientemente de los datos anteriores o posteriores. Esto es un comportamiento deseable en muchos escenarios pero puede no ser deseable cuando se manejan secuencias ordenada de datos en los que la posición en la secuencia tiene importancia para el resultado de la red.

Las redes neuronales recurrentes (*Recurrent Neural Networks* o RNN) [48], tal y como se muestra en la figura 2.12, contienen un bucle que reintroduce la salida de la red como entrada. Esto permite al valor de salida de la red estar condicionado no solo por los valores de entrada sino también por el valor de las salidas anteriores.



(a) Sigmoide.

(b) Tangente Hiperbólica.

(c) SoftSign.

(d) ReLu.

Figura 2.11: Funciones de activación.

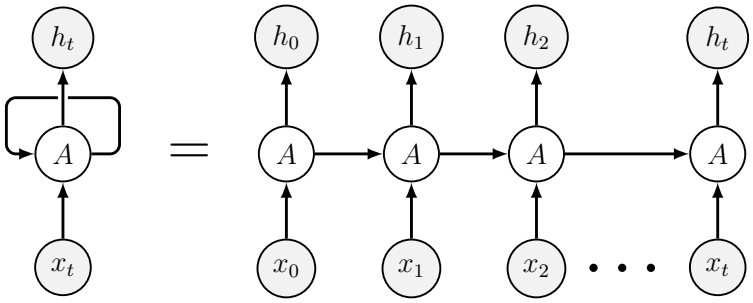


Figura 2.12: Red neuronal recurrente.

Además, el uso de RNN permite una mayor flexibilidad a la hora de procesar entradas de datos de tamaño variable, al contrario de lo que ocurre con redes neuronales *feed-forward* en las que la dimensionalidad en la entrada y la salida de la red debe estar forzosamente definida en el diseño de la red y permanecer inalterable. Las RNN, por el contrario, permiten entradas de tamaño arbitrario, ya que están diseñadas para procesar secuencias de cualquier tamaño. Del mismo modo pueden devolver como salida otra secuencia de longitud no preestablecida en la fase de diseño de la red. Algunos ejemplos de redes con diferentes tamaños de salida y entrada se pueden ver en la figura 2.13 y son

comentados a continuación [49]:

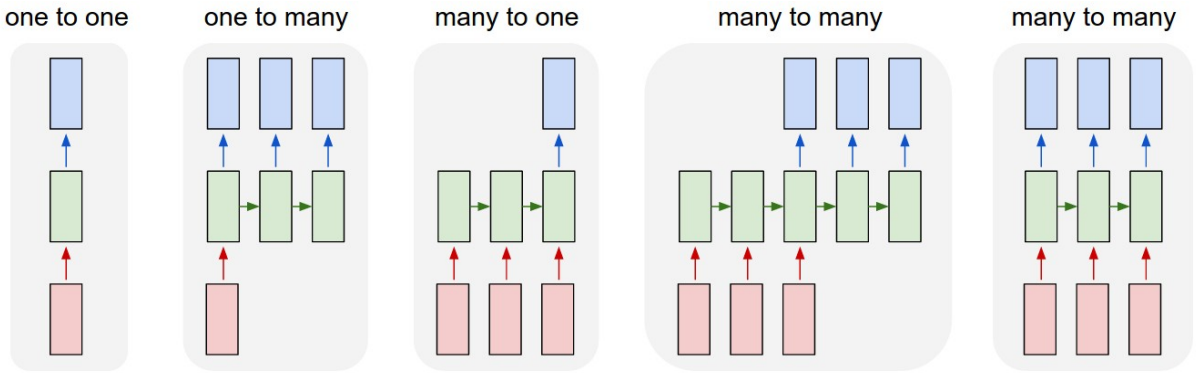


Figura 2.13: Diferentes arquitecturas de RNNs. Fuente [50].

- *One to one*: Modo de funcionamiento que simula una red neuronal *feed-forward* en la que se tiene un vector de características de entrada de tamaño fijo y un vector de resultados de salida.
- *One to many*: Modo de funcionamiento para generar secuencias a partir de un dato individual como puede ser la generación de música o la obtención de la descripción en lenguaje natural de una imagen proporcionada como entrada.
- *Many to one*: Clásico ejemplo de clasificación de una secuencia en el que se procesan todos los elementos de la secuencia y la red proporciona un resultado único para dicha secuencia. Por ejemplo, un clasificador de análisis de sentimiento puede tener esta forma, ya que procesa una secuencia de palabras y da como resultado una puntuación (sentimiento negativo o positivo).
- *Many to many*: Clásico ejemplo de la traducción automática de lenguaje natural. La entrada es una secuencia de palabras en un idioma y el resultado es otra secuencia de palabras en un idioma diferente con el mismo sentido que tenía la secuencia original. En ocasiones puede interesar que ambas secuencias se encuentren sincronizadas cuando se quiera generar una salida que esté asociada a un elemento concreto de la secuencia de entrada.

Este tipo de redes, en su versión tradicional, no consiguen retener dependencias a largo plazo debido a problemas de desvanecimiento/explosión del gradiente. Por este motivo se idearon diferentes propuestas de alternativas a las RNN tradicionales como por ejemplo las redes neuronales LSTM (*Long Short-Term Memory*) [51]. Estas redes LSTM han sido ampliamente usadas para resolver problemas relacionados con el procesamiento de lenguaje natural, reconocimiento del habla y otros problemas donde el concepto de secuencia es importante para la clasificación [52].

La figura 2.14 muestra de forma simplificada una célula LSTM que, como se puede observar, tiene 4 elementos diferentes:

- *Input gate*: Puerta que determina si la entrada actual se tiene en cuenta para generar la salida y en qué medida afecta al resultado.

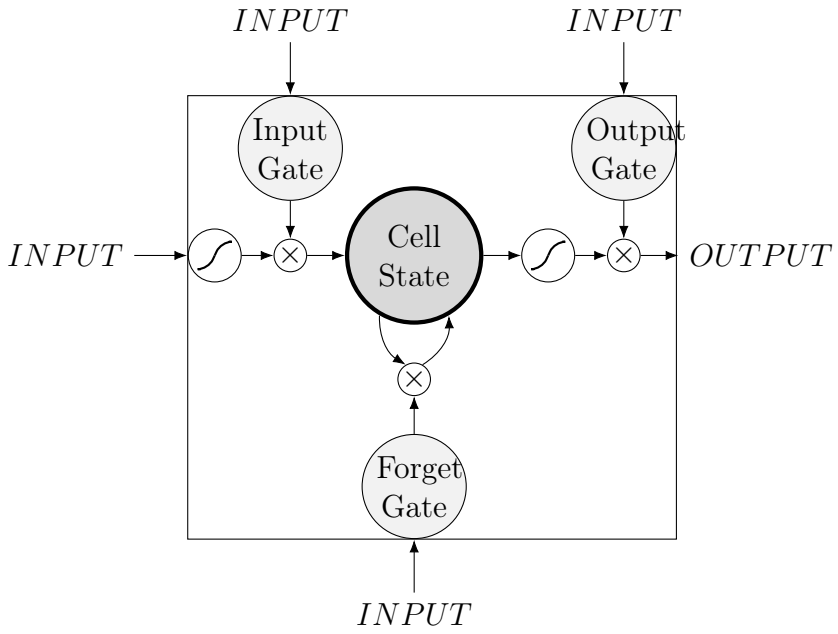


Figura 2.14: Célula LSTM (simplificada).

- *Cell state*: El estado de la célula, que guarda la información recolectada a partir de las entradas previas de esa secuencia.
- *Forget gate*: Puerta que determina si la información contenida en el estado de la célula es tenida en cuenta para generar la salida y en qué medida afecta al resultado.
- *Output gate*: Puerta que determina en qué medida el valor resultante de combinar la entrada y el estado de la célula se proporciona como salida. Esta puerta no existe en otras propuestas como las *Gated Recurrent Units* (GRU) [53].

En el contexto de las redes LSTM, una puerta es una operación que controla si un valor dado es transmitido a la siguiente operación (o no) y como su valor se ve afectado en el proceso. Por ejemplo, si se recibe un valor de entrada 2, un valor de la *input gate* de 0 implica que el valor de esa entrada no es transmitido a la siguiente operación. En ese caso se dice que la puerta se encuentra cerrada. Por el contrario, si la puerta tiene un valor de 1, el valor de la entrada es transmitido a la siguiente operación. Se dice entonces que la puerta está abierta. La figura 2.15 muestra la arquitectura de una célula LSTM en detalle. En ella el estado de cada puerta está controlado por una neurona estándar que realiza una suma ponderada de sus entradas y a la que se aplica una función de activación sigmoide σ. Esta función de activación provoca que la salida tenga valores comprendidos entre 0 y 1, el cual es el rango adecuado para determinar el grado de apertura de las puertas. Estas puertas, por tanto, están destinadas a que la célula aprenda la información que es importante y debe conservar, y la que no es relevante y puede olvidar.

Por su parte, existe una cuarta neurona que realiza del mismo modo una suma ponderada de sus entradas pero a cuya salida se aplica una función de activación tangente hiperbólica. Esta neurona procesa el dato de entrada y proporciona un valor que posteriormente se ve afectado por los resultados de las diferentes puertas y que da como resultado la salida y el estado de la célula.

Siguiendo paso a paso el funcionamiento de una red LSTM (figura 2.15), una célula tiene las siguientes entradas:

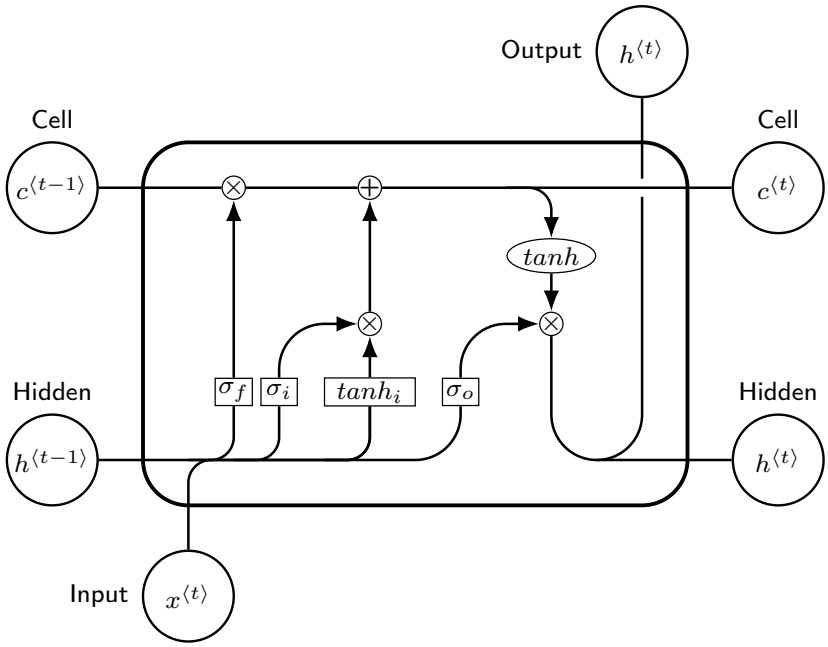


Figura 2.15: Célula LSTM.

- $x^{\langle t \rangle}$: Dato de entrada introducido en el instante temporal presente t en contraposición con los datos de entrada que fueron introducidos previamente ($t-1$, $t-2$, etc.) y con los que se introducirán próximamente ($t+1$, $t+2$, etc.).
- $c^{\langle t-1 \rangle}$: Estado interno de la célula obtenido para la entrada en el instante anterior al actual ($t-1$). Se emplea para conservar información a más largo plazo.
- $h^{\langle t-1 \rangle}$: Resultado de la salida de la célula para la entrada en el instante anterior al actual ($t-1$). Llamado frecuentemente *hidden* (oculto) puesto que en ciertos diseños de red puede permanecer oculto hasta que el procesamiento de toda la secuencia concluye, siendo el último de estos valores el resultado final de la red. Se puede considerar a este valor como la salida temporal de la secuencia procesada hasta ese momento.

La primera operación que se realiza dentro de una célula LSTM es la concatenación de $h^{\langle t-1 \rangle}$ y $x^{\langle t \rangle}$. Este vector de elementos de entrada combinados alimenta a todas las neuronas que activan las puertas (en la figura 2.15 marcadas como σ_f, σ_i y σ_o) y a la neurona que procesa los datos de entrada (marcada como $tanh_i$) [51].

Tras este paso, la *forget gate* establece en que proporción el valor del estado de la célula proveniente de la iteración anterior, $c^{\langle t-1 \rangle}$, se tiene en consideración para esta nueva iteración, mientras que la *input gate* hace lo propio con la salida de la neurona $tanh_i$. Ambos valores se suman para obtener el nuevo estado de la célula $c^{\langle t \rangle}$ que es propagado a la siguiente iteración. Este mecanismo permite, en función de la configuración de los pesos de las diferentes puertas, elegir un estado que codifique información tanto de los últimos datos de entrada como de iteraciones anteriores [51].

Por su parte, la *output gate* establece del mismo modo qué proporción del valor $c^{\langle t \rangle}$, tras serle aplicada una función de activación, será tomado como valor de salida de la célula $h^{\langle t \rangle}$.

La dimensionalidad del vector de salida $h^{\langle t \rangle}$, llamado *unidades* en algunas implementaciones, se considera uno de los hiperparámetros de este tipo de redes y tiene un impacto

directo sobre la cantidad de parámetros entrenables de la célula. El motivo es que este valor condiciona la dimensionalidad de la entrada de cada una de las neuronas internas de la célula, al ser esta entrada el resultado de concatenar el vector de entrada de la célula $x^{\langle t \rangle}$ con la salida de la célula proveniente de la iteración anterior. Esta dimensionalidad, por tanto, también afecta a la cantidad de parámetros entrenables de la red, ya que a más entradas en las neuronas son necesarios más coeficientes que deben ser ajustados durante el entrenamiento.

2.3.6. Inteligencia artificial explicable (XAI)

Se llama inteligencia artificial explicable (*Explainable Artificial Intelligence* o XAI) al conjunto de métodos y técnicas que buscan facilitar que un humano pueda entender los resultados de un modelo de aprendizaje automático [54]. Este concepto es opuesto al concepto de caja negra que existen en algunos modelos de aprendizaje automático y en especial en el aprendizaje profundo, en los que resulta muy complicado explicar el funcionamiento exacto del modelo. Esta falta de interpretabilidad de algunos modelos resulta ser muy problemática en muchos contextos, ya que no es posible explicarle al usuario los motivos de la clasificación. Esta situación crea un problema de confiabilidad en el modelo que reduce en gran medida su utilidad práctica. Arrieta et al. [17] describen en detalle una taxonomía de los tipos de técnicas XAI que han sido publicadas en los últimos años y que se pasan a explicar de forma resumida a continuación.

Los llamados modelos transparentes son modelos que, por su propia naturaleza, son explicables sin emplear ningún tipo de técnica o método adicional. Los modelos de regresión lineal o logística, modelos bayesianos, K vecinos más cercanos (KNN) y árboles de decisión son ejemplos de este tipo de métodos [17]. En el caso de un árbol de decisión es posible recorrer el árbol para ver qué características han sido evaluadas y de qué manera. En otros modelos como KNN se puede explorar los K vecinos que produjeron la clasificación y eso permite explicar el resultado.

Por otro lado existen un gran grupo de técnicas llamadas de explicabilidad a posteriori. Estas técnicas son empleadas en modelos no transparentes para facilitar su explicabilidad y se pueden categorizar en las siguientes familias de técnicas [17]:

- *Text explanations*: Generan una explicación del motivo de la clasificación mediante una descripción textual.
- *Visual explanation*: Muestra una explicación visual del resultado. Muchas de estas técnicas han sido usadas tradicionalmente para visualizar la importancia de las características y realizar una reducción de la dimensionalidad, pero también facilitan la explicabilidad de los resultados.
- *Local explanations*: Consiguen la explicabilidad por medio de reducir el espacio de soluciones a aquellas cercanas al resultado que se pretende explicar. Se busca una explicación a dicho subconjunto del espacio, no al espacio completo, lo cual reduce su complejidad.
- *Explanations by example*: Extracción de ejemplos representativos similares a la solución que se pretende explicar. Con ellos se intenta explicar dicha solución.
- *Explanations by simplification*: Construcción de un modelo alternativo más sencillo, generalmente transparente, que se desarrolla empleando el modelo original con el fin

de obtener unos resultados lo más similares posibles a dicho modelo. La explicación se obtiene a través de este segundo modelo, por ser éste transparente.

- *Feature relevance*: Obtiene una puntuación de relevancia de las características usadas en el modelo. A partir de esta puntuación se puede determinar la importancia de cada una de las características en comparación con las otras y, de una forma indirecta, explicar el funcionamiento del modelo.

Adicionalmente, las técnicas se dividen entre aquellas que pueden ser usadas empleando el modelo como una caja negra (agnósticas al modelo) y aquellas que han sido diseñadas para ser utilizadas en un tipo concreto de modelo (específica al modelo) [17].

De entre las técnicas específicas al modelo esta tesis se centra en *Random Forest* y en redes neuronales recurrentes (RNN), que son los modelos relevantes para este trabajo de investigación y que han sido descritos en las secciones 2.3.2 y 2.3.5.

A pesar de que los modelos de aprendizaje no profundo son intrínsecamente más sencillos y con menos parámetros que los modelos de aprendizaje profundo, no todos estos modelos se consideran modelos transparentes, ya que algunos de ellos presentan una gran complejidad que dificulta su explicabilidad. Un ejemplo son los modelos de tipo *ensemble* que obtienen sus resultados al aglutinar el resultado de diferentes predictores. En concreto *Random Forest* no resulta tan sencillo de explicar al ser el resultado final una combinación de resultados intermedios, a pesar de que cada uno de estos resultados intermedios sea generado por un modelo transparente como es un árbol de decisión.

Con el fin de mejorar la explicabilidad de los modelos *Random Forest* se han propuesto diferentes técnicas que pertenecen a las categorías de simplificación del modelo (*explanations by simplification*) y la importancia de características (*feature relevance*).

En el campo de la simplificación, por ejemplo, Deng propone un entorno de trabajo llamado *inTrees* (*interpretable trees*) [55] que consiste en la simplificación del modelo previo mediante diferentes técnicas como la poda, para finalmente crear un modelo basado en reglas que modela una versión simplificada del modelo original. La explicabilidad se consigue empleando este modelo transparente y extrapolando dicha explicabilidad al modelo original.

En cuanto a la importancia de características, Auret et al. [56] proponen el uso de técnicas similares a Boruta, explicada en la sección 2.3.3. Otros autores como Tolomei et al. [57] emplean una técnica llamada *Feature Tweaking* que consiste en observar el resultado de cada uno de los árboles de decisión que forman el *Random Forest* y modificar las características de la entrada a explicar, de tal forma que se busque un cambio en la clasificación. Estos cambios pueden realizarse observando cada uno de los árboles de decisión de forma individual. Al determinar las modificaciones de las características que producen un cambio global en el resultado mejora la interpretabilidad del modelo.

Estas técnicas de *Feature Relevance* también han sido aplicadas a modelos RNN utilizando la llamada *Layer-wise Relevance Propagation*. Esta técnica consiste en aplicar un concepto similar a la propagación hacia atrás para atribuir la relevancia de las características de entrada, propagando capa a capa desde la relevancia de las salidas. Este proceso es, en realidad, similar al realizado cuando se atribuye la responsabilidad del error δ durante el entrenamiento de una red neuronal, tal y como se explicó en la sección 2.3.4. Arras et al. [58] extienden esta técnica para ser utilizada en la arquitectura concreta de las redes LSTM y GRU. Otras técnicas como la propuesta por Karpathy et al. [59] consiste en un método de visualización en el que se muestra como el procesamiento de cada elemento de

la secuencia altera el resultado de la clasificación por medio de un código de colores entre azul y rojo.

Otras aproximaciones proponen modificar el diseño del modelo para proporcionar información adicional que facilite su interpretabilidad. RETAIN [60] y SISTA [61] son ejemplos de esta aproximación, ya que introducen componentes que capturan la información que posteriormente se puede utilizar para explicar los resultados.

Por último los métodos agnósticos al modelo son, como su propio nombre indica, independientes al modelo. Por este motivo pueden ser aplicados a cualquier tipo de modelo. LIME (*Local Interpretable Model-agnostic Explanations*) [62] es una de las técnicas más representativa de este tipo de modelos. Esta técnica emplea *feature tweaking*, es decir, dado una entrada concreta modifica las características de dicha entrada para obtener los valores de las predicciones. Estos valores de las predicciones son estimados mediante una regresión lineal en la que el peso de cada predicción se encuentra ponderada con respecto a la modificación realizada sobre el dato de entrada original. A más cercanía a la entrada original, más peso y por tanto más relevancia tendrá al realizarse la regresión lineal. Este proceso se muestra gráficamente en la figura 2.16, donde se aprecia que esta técnica consigue explicar el resultado simplificando el espacio de soluciones a un subconjunto que pueda ser más fácilmente explicado. Como su propio nombre indica, proporciona una explicación local [62, 17].

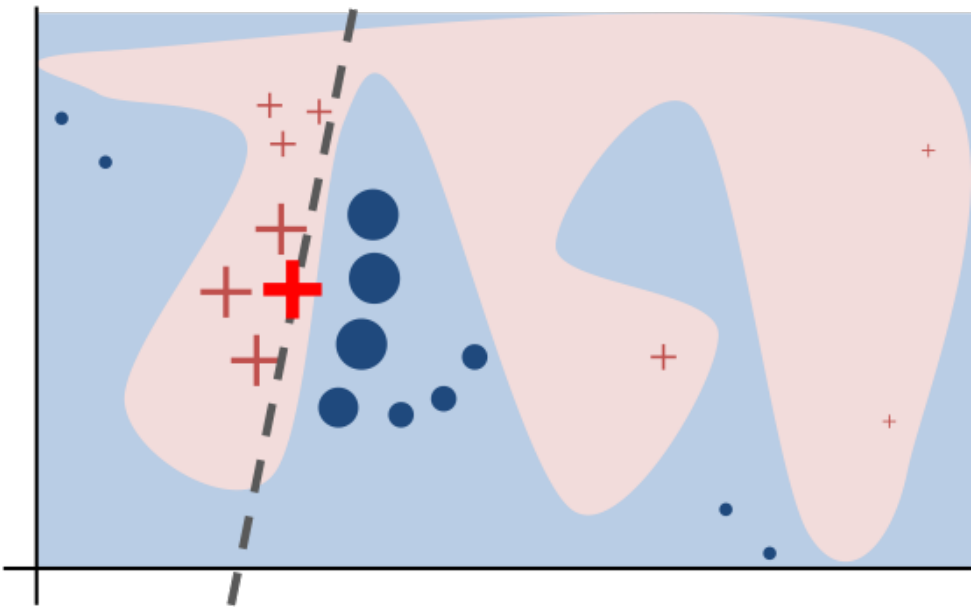


Figura 2.16: Ejemplo del *Feature Tweaking*. Fuente [62].

En concreto, la implementación de LIME incluye tres clases que pueden ser utilizadas según sea el tipo de dato de entrada y proporciona diferentes tipos de visualización de los resultados que favorecen su interpretación [62]:

- *LimeImageExplainer*: El dato de entrada es una imagen y la salida es de la forma que se muestra en la figura 2.17. Esta clase divide la imagen en componentes y sigue un proceso en el que borra ciertos de ellos para determinar qué componente de la imagen provoca que se la clasifique de una determinada manera. Un ejemplo que se cita en [62] y se puede observar en la figura 2.18 es la clasificación de "*Husky vs Wolf*". A pesar de tener buenos resultados de clasificación, esta técnica evidencia

que los componentes de la imagen que resulta relevantes para la discriminación entre husky y lobo se corresponden con la nieve del fondo de la imagen, que no son características de los propios animales.

- *LimeTextExplainer*: El dato de entrada es una cadena de texto de tamaño arbitrario y la salida es de la forma que se muestra en la figura 2.19. En ella se observan tres partes que se pasan a describir de izquierda a derecha. En primer lugar se muestra la puntuación de cada una de las posibles clases por las que el modelo puede optar y la puntuación que cada una de ellas ha recibido. En segundo lugar se muestra la visualización tabular de las características (en este caso las palabras) con la clase cuya clasificación favorece y en qué grado lo hace. Finalmente se muestra el texto de entrada con las palabras más relevantes para la clasificación resaltadas en colores de diferentes intensidades según dicha relevancia. Al igual que ocurre con *LimeImageExplainer*, los experimentos publicados en [62] muestran que en ocasiones el modelo clasifica en base a información como el nombre de la persona que escribe el mensaje y no del contenido en sí mismo [62].

- *LimeTabularExplainer*: Es la opción a utilizar cuando el dato de entrada no es una imagen o cadena de texto. Recibe como entrada las características extraídas del dato y tiene una salida como la mostrada en la figura 2.19 a excepción del componente de texto situado más a la derecha, al no tratarse de una entrada en formato texto.

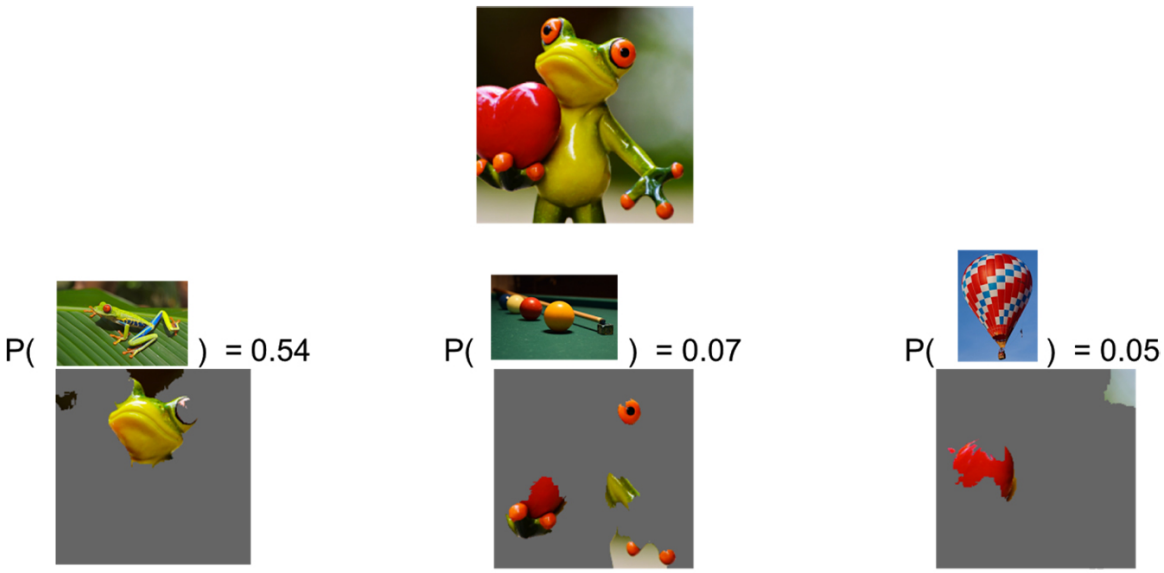


Figura 2.17: Ejemplo de LIME para explicar en imágenes. Fuente [62].

(a) Husky classified as wolf (b) Explanation

Figura 2.18: Ejemplo del problema "*Husky vs Wolf*". Fuente [62].

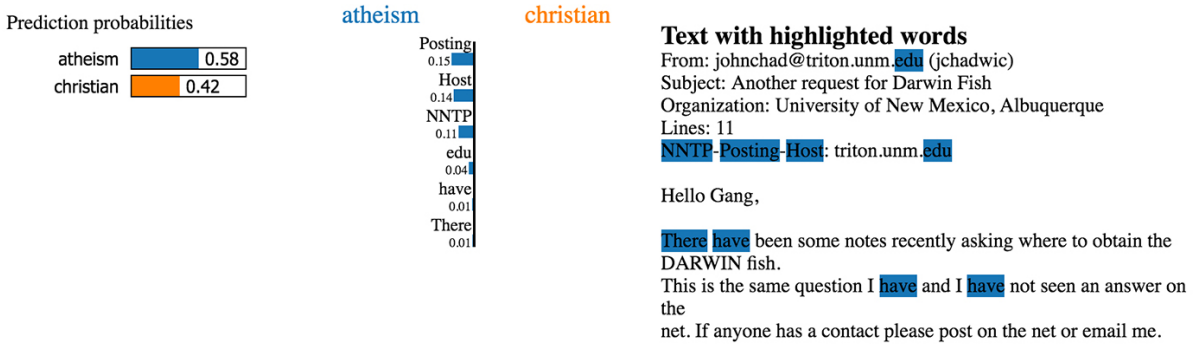


Figura 2.19: Ejemplo de LIME para explicar en texto. Fuente [62].

Capítulo 3

Estado del arte

En este capítulo se detalla el estado del arte presente en la literatura sobre el uso de técnicas de aprendizaje automático empleadas para la detección de nombres de dominio generados mediante DGA. Estas técnicas pueden dividirse en dos grandes grupos en función de su naturaleza: por un lado los que han empleado técnicas de aprendizaje no profundo y por otro lado las técnicas de aprendizaje profundo.

Respecto a las técnicas de aprendizaje no profundo, Bilge et al. publicaron EXPOSURE [63], más tarde mejorado en [64]. EXPOSURE es un sistema de detección de dominios maliciosos que emplea técnicas de análisis pasivo de DNS basadas en 15 características organizadas en 4 grupos: relativas al tiempo, extraídas de las respuestas DNS, basadas en el campo TTL (*Time-to-Live*) de los registros de recurso DNS y obtenidas del nombre de dominio en sí mismo. La mayor parte de estas características fueron obtenidas a partir de tráfico DNS en tiempo real (o recientemente capturado), ya que muchas de ellas son altamente volátiles y pueden cambiar a lo largo del tiempo. Estas características fueron seleccionadas después de un proceso de análisis manual, en el que se tuvo en cuenta el comportamiento habitual del malware conocido. En cuanto a las características léxicas, Bilge et al. se centraron fundamentalmente en dos valores: el porcentaje de caracteres numéricos en el nombre y el porcentaje de la subcadena de texto con significado de mayor tamaño con respecto a la longitud total del nombre. El objetivo de estas características centradas en el nombre de dominio era detectar el uso de nombres generados mediante DGA. Sin embargo, esta aproximación no proporciona un análisis léxico completo del nombre de dominio. Los resultados de EXPOSURE muestran un 98.4 % de acierto en la detección con alrededor de un 1 % de falsos positivos, empleando un árbol de decisión J48 como modelo [65].

Antonakakis et al. presentaron Pleiades en [16]. Su propuesta se centra en la detección de peticiones DNS fallidas, ya que los DGAs pueden llegar a generar y probar una gran cantidad de nombres de dominio, pero solo unos pocos de ellos son finalmente registrados. Por ello es habitual observar varias peticiones de resolución DNS fallidas antes de que una de ellas sea resuelta satisfactoriamente. Pleiades agrupa los nombres de dominio que resultan en una respuesta fallida junto con listas de nombres de dominio generados mediante DGA y dominios legítimos. Además de esta información, basada en características volátiles, también emplea una serie de características estadísticas extraídas de los nombres de dominio, como por ejemplo la longitud del nombre, nivel de aleatoriedad y frecuencia en la distribución de caracteres. Finalmente, se usa un modelo oculto de Markov (*Hidden Markov Model* o HMM) para agrupar los dominios sospechosos.

Schiavoni et al. propusieron Phoenix en [66], que consiste en un sistema por capas que detecta los nombres de dominio generados con DGA y los clasifica en base a las direcciones IP a las que resuelve. Esta propuesta emplea dos características ampliamente utilizadas en lingüística: la proporción de caracteres con sentido (*meaningful characters ratio*) y la puntuación de normalidad de *N-Grams* (*N-gram normality score*), las cuales proporcionan una medida de significado y pronunciabilidad (según el idioma inglés) como mecanismo para detectar nombres generados mediante DGA. Estos *N-Grams* son el resultado de dividir una secuencia en todas las posibles subsecuencias de longitud N, es una estructura ampliamente utilizada en este tipo de problemas y que ha sido empleada de uno u otro modo por diferentes autores. Además, obtuvieron una lista de nombres de dominio legítimos a partir de las listas de Alexa [67] para calcular la distribución de probabilidad de los nombres de dominio legítimos en cuanto a esas características lingüísticas. Una vez calculadas emplearon la distancia de Mahalanobis junto con un umbral obtenido empíricamente para clasificar los nombres de dominio sospechosos. A continuación usaron mecanismos de agrupamiento (*clustering*) para determinar la pertenencia de la muestra detectada a una familia de malware en particular.

Por último, Da Luz emplea una combinación de 18 características léxicas (estadísticas obtenidas a partir de los *N-Grams*, entropía de Shannon, longitud del nombre, proporción de vocales y consonantes, entre otros) y 17 características de red (por ejemplo estadísticas a partir del TTL y cantidad de subredes IP) extraídas de forma pasiva por una serie de servidores DNS recursivos [68]. Esta aproximación se basa en algunas de las características ya empleadas por Antonakakis et al. [16], aunque la mayor parte de ellas son simplificadas para reducir la dimensionalidad del problema y mejorar la velocidad de la solución. Dichas características son utilizadas con dos conjuntos de datos diferentes y tres modelos de aprendizaje automático: KNN, árboles de decisión y *Random Forest*. Los mejores resultados se obtienen usando un modelo *Random Forest* parametrizado para utilizar 10 árboles de decisión con una profundidad máxima de 20.

Aunque todos estos trabajos previos produjeron buenos resultados en cuanto a detección, todos ellos empleaban características basadas en información muy volátil que puede cambiar a lo largo del tiempo. Por ejemplo, una dirección IP en una respuesta DNS obtenida en el momento actual puede ser diferente a la obtenida unas semanas antes. Así mismo, no es posible entrenar al modelo para identificar nombres generados por DGAs que se encuentren inactivos en el momento de realizar la captura de los datos de entrenamiento, ya que las respuestas DNS reales no se pueden obtener. Por ejemplo, *Kraken* es una familia de malware que usa DGA. Estuvo activo durante varios años y, en cierto momento, las nuevas versiones del mismo pasaron a utilizar un DGA diferente. Cuando la primera versión del DGA dejó de ser empleada, dejó de ser posible capturar algunas de las características utilizadas por otros autores para este problema de clasificación, por lo que es imposible entrenar al modelo empleando dicha información.

Por estos motivos, esta tesis se centra en técnicas de detección basadas únicamente en características léxicas del nombre del dominio. Los nombres de dominio, al contrario de lo que sucede en otros campos de los mensajes DNS, no cambian ya que son generados desde el lado cliente y no del servidor, por lo que si se ha podido obtener una muestra del malware, se tiene toda la información necesaria para generar un conjunto de datos de entrenamiento, incluso para familias que no se encuentran activas desde hace tiempo.

Por el contrario, otros autores han propuesto técnicas de aprendizaje profundo para la detección de dominios generados algorítimicamente, basándose en redes neuronales de diferentes tipos. Estos trabajos tienen la ventaja de no requerir un proceso de ingeniería

de características tan intenso y complejo como los requeridos por otros modelos de aprendizaje automático. En concreto, las redes LSTM, que pertenecen a la familia de las redes neuronales recurrentes [49], han sido ampliamente usadas para resolver problemas relacionados con el procesamiento de lenguaje natural, reconocimiento del habla [52] y otros problemas donde el concepto de secuencia es importante para la resolución del problema.

Woodbridge et al. propuso una red neuronal compuesta de una capa de codificación (*embedding*), una capa oculta LSTM de 128 unidades de tamaño y finalmente una capa densa que combinaba las salidas de la capa anterior en una sola salida que es el resultado de la clasificación [69]. Los resultados de este modelo se comparan con otros tres modelos: un modelo HMM, una regresión logística a partir de los *2-Grams* y un modelo *Random Forest* empleando características extraídas manualmente, similares a las propuestas por otros autores mencionadas anteriormente. Sus resultados muestran que su propuesta de modelo usando LSTM es capaz de clasificar correctamente el 98 % de los nombres generados con DGA, con una tasa de falsos positivos (*False Positive Rate* o FPR) de 0.001.

Tran et al. propusieron una modificación con respecto al trabajo de Woodbridge consistente en emplear una capa *LSTM.MI* (*LSTM with Multiclass Imbalance*) que sustituye a la capa LSTM utilizada anteriormente [70]. Esta red incluye la posibilidad de introducir un coste más granular en el proceso de aprendizaje, con lo que el impacto de los errores de clasificación pueden variar en función del tipo de error. Se propone una aproximación en dos etapas. En la primera etapa una red es entrenada para clasificar los nombres de dominio en "generados por DGA" o "no generados por DGA", estableciendo un coste mayor para el error de clasificar un nombre de dominio legítimo como generado por DGA que el error contrario, evitando así que dominios legítimos sean bloqueados por error. En la segunda etapa, que solo se ejecuta si la primera ha clasificado el nombre de dominio como "generado por DGA", otra red clasifica el nombre de dominio para obtener el tipo de DGA concreto que genera dicho nombre.

Las redes neuronales convolucionales (*Convolutional Neural Networks* o CNN) también han sido usadas con éxito para resolver este problema de clasificación. Catania et al. propusieron una red neuronal compuesta por una capa de *embeddings*, una capa convolucional de una dimensión (1D-CNN) y una capa densa para obtener el resultado final de la clasificación [71]. El uso de una red CNN en lugar de una LSTM tiene como resultado un aumento del rendimiento debido a que las convoluciones son computadas de forma concurrente, mientras que el modelo LSTM requiere una ejecución secuencial al necesitar el resultado de procesamiento del elemento de la secuencia anterior para procesar el elemento actual. Catania et al. comparan su propuesta con el modelo propuesto por Woodbridge et al. pero introduciendo una pequeña modificación en la función de activación de la capa LSTM. Ambos modelos son entrenados durante 10 épocas con el 70 % del conjunto de datos y probados con el 30 % restante. El modelo LSTM obtiene una tasa de verdaderos positivos (*True Positive Rate* o TPR) de alrededor del 94 % y una tasa de falsos positivos cercana al 3 %, mientras que el modelo CNN obtiene una TPR de alrededor del 94 % y un FPR de 0.7 %.

La aproximación empleando una red CNN con un filtro (*kernel*) unidimensional (1D-CNN) es conceptualmente muy similar a una aproximación empleando *N-Grams* como características. Por ejemplo, una capa convolucional que emplea un filtro $1x3$ sobre la cadena "facebook" hace que la cadena sea dividida en "fac", "ace", "ceb", "ebo", "boo", "ook". De cada una de estas partes se obtiene un valor como resultado de la convolución de estas secuencias con el filtro, que a su vez es la entrada de una capa densa que da como salida el resultado de la clasificación. Este filtro $1x3$ hace una función equivalente

(aunque más simple) a la extracción de características manual empleando *3-Gram*, pero sus pesos son obtenidos automáticamente en el propio proceso de entrenamiento.

Liu et al. propusieron una mejora sobre el modelo de red recurrente convolucional (*Recurrent Convolutional Neural Network* o RCNN) propuesto inicialmente por Lai et al. [72], en el cual se combina una capa LSTM bidireccional (Bi-LSTM) con capas convolucionales [73]. Este modelo se llama RCNN-SPP (*RCNN with Spatial Pyramid Pooling*), e incorpora una modificación en el algoritmo de agrupación (*pooling*) de las capas convolucionales que mejora la representación de las características de un nombre de dominio. En esta aproximación se emplean varios filtros de diferentes tamaños que son usados para obtener una representación que es a su vez combinada para generar la representación final. Esta propuesta obtiene un 92 % de exactitud tanto para clasificación binaria (generado o no generado por DGA) como para clasificación multi-clase (perteneciente a uno u otro tipo de DGA).

Yang et al. propusieron el uso de una arquitectura compuesta de dos modelos diferentes a la que llamaron *Heterogeneous Deep Neural Network* (HDNN) [74]. En primer lugar emplean varias capas convolucionales con diferentes tamaños de filtro que se combinan para obtener características locales, a lo que llaman *Improved Parallel CNN*. En segundo lugar emplean una *Self-Attention based Bi-LSTM* para obtener características globales. Finalmente ambos resultados se combinan para proporcionar el resultado final de la clasificación. Sus experimentos muestran que su red HDNN funciona mejor que otras aproximaciones propuestas para un mismo conjunto de datos.

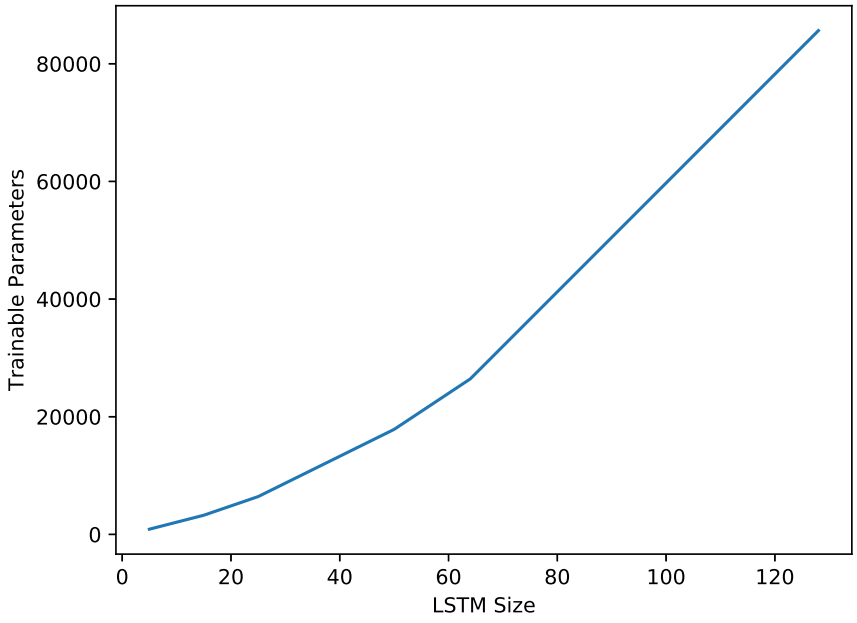


Figura 3.1: Cantidad de parámetros entrenables para diferentes tamaños de célula LSTM.

Todos estos trabajos emplean conjuntos de datos mucho mayores que el construido para esta tesis doctoral, motivo por el cual es posible entrenar modelos tan complejos como los descritos y con tal cantidad de parámetros entrenables. Dichos parámetros son los pesos de las capas que son actualizados por el mecanismo de propagación hacia atrás y contienen la información aprendida por la red al ser expuesta al conjunto de datos de entrenamiento [75]. Para ilustrar este hecho, en la figura 3.1 se observa cómo la cantidad de parámetros aumenta considerablemente al aumentar el tamaño de la célula LSTM. La cantidad de parámetros aumenta considerablemente al combinar varios modelos com-

plejos, tal y como proponen varios de los autores citados anteriormente [73, 74], ya que el uso de redes bidireccionales y de redes convolucionales utilizando diferentes filtros de diferentes tamaños multiplica la cantidad de parámetros.

Capítulo 4

Conjunto de datos

Este capítulo detalla la construcción del conjunto de datos que se emplea en toda la tesis doctoral. Con este fin se detalla las fuentes de información empleadas y el preprocesamiento realizado a dicha información para obtener el resultado final.

4.1. Nombres de dominio

Tal y como se menciona en la sección 2.1, el nombre del dominio generado mediante un DGA tiene unas características especiales que pueden hacerlo identificable con respecto a otros nombres de dominio creados por un ser humano. El uso del nombre del dominio como único dato para entrenar a un modelo que sea capaz de clasificarlo como generado por un DGA o no, tiene como ventaja su poca volatilidad. Recuérdese que en el caso de la información que se puede capturar de la red, esta solo es posible obtenerla si se realiza una captura de red de esa información en el momento que ese malware que emplea el DGA con el que se quiere entrenar al modelo se encuentra activo. Además, se requiere que se hayan compartido gran cantidad de trazas de red que pueden llegar a contener otro tipo de información por lo que, en general, no suele ser un tipo de recurso que se pueda obtener con facilidad y en la cantidad necesaria para entrenar un modelo de aprendizaje automático.

En cuanto a la información contenida en el agente registrador del dominio (conocida como *whois*), es de una volatilidad mucho menor que la información obtenible del tráfico DNS pero, del mismo modo, es más volátil que el nombre de dominio ya que dichos datos pueden cambiar a lo largo del tiempo e incluso desaparecer cuando los dominios registrados son bloqueado como respuesta a la actividad maliciosa. Por otro lado, solo es posible entrenar al modelo con los datos de aquellos dominios que el desarrollador u operador del malware haya registrado, con lo que el conjunto de datos disponible es mucho más reducido.

Los nombres de dominio generados por un DGA son, por el contrario, un dato muy poco volátil que se puede extraer de cualquier muestra de malware aunque ya no se encuentre en uso. El acceso a estas muestras es sencillo debido a plataformas como VirusTotal [76], además de existir repositorios públicos que contienen el código de DGAs de conocidas familias de malware [29]. Disponer del código del DGA permite generar una cantidad arbitrariamente grande de datos para utilizar en el entrenamiento, motivo por el cual esta tesis se centra en este tipo de dato.

4.2. Obtención de dominios legítimos

Con el fin de evaluar las propuestas de esta tesis es necesario disponer de un conjunto de datos balanceado entre nombres de dominio legítimos y generados por DGAs. Desafortunadamente, es poco habitual que los investigadores y la industria de la ciberseguridad compartan sus conjuntos de datos, tal y como dictan las buenas prácticas [77]. Por este motivo, la mayor parte de los investigadores se ven obligados a crear su propio conjunto de datos a partir de la información que ellos mismos son capaces de recoger, lo que no siempre es una tarea sencilla y dificulta la comparativa posterior entre trabajos realizados por diferentes investigadores.

Dado que los nombres de dominio legítimos (no utilizados por malware) son aquellos empleados para ofrecer servicios en Internet, y siendo que la mayor parte del tráfico de Internet y los servicios más populares están relacionados con servicios web (protocolo HTTP), el conjunto de datos de dominios legítimos (etiquetados como "CLEAN") se construye a partir del listado de portales web proporcionado por Alexa [67]. Alexa es una compañía perteneciente a la empresa Amazon que publica listas de los portales web más visitados de Internet, existiendo varias opciones. La lista "Alexa's Top 10" contiene los 10 portales web más visitados, la lista "Alexa's Top 100" contiene los 100 portales más visitados, y así sucesivamente. En estas listas se encuentran portales web como por ejemplo "google.com", "amazon.com" o "qq.com". Por otro lado también realiza agrupaciones de estos datos bajo diferentes criterios, como por ejemplo listas a nivel global, específicas de un país concreto, o incluso focalizadas en un sector específico (por ejemplo noticias, informática o salud, entre otros muchos). Por lo tanto, esta fuente de información es una muestra muy representativa de dominios legítimos, por lo que para esta tesis se ha utilizado la lista global "Alexa's Top 1 million" [78], que proporciona un listado de un millón de dominios legítimos con el que construir el conjunto de datos.

4.3. Obtención de dominios maliciosos

En cuanto a los dominios generados a partir de DGA, se ha empleado un repositorio público de Github creado por J. Bader [29] que contiene cientos de estos dominios junto con el código fuente que reproduce el comportamiento de 26 implementaciones diferentes de DGAs extraídos de malware real. Algunos ejemplos de los dominios contenidos en este repositorio pueden observarse en la tabla 4.1. Bader también publicó código fuente escrito en el lenguaje Python para emular los DGAs usados en cada una de las familias de malware incluidas en el repositorio. En dicho código se observan diferentes tipos de aproximaciones en el diseño e implementación de los DGAs que se corresponden con los tipos de DGA descritos en la sección 2.1. La tabla 4.2 lista las diferentes familias presentes en el repositorio, así como el tipo de DGA que implementan y un ejemplo de salida de dicho algoritmo.

4.4. Creación del conjunto de datos

Desafortunadamente, los ejemplos proporcionados por Bader en su repositorio son heterogéneos en cuanto a la cantidad de nombres de dominio generados para cada una de las familias de malware. Por ejemplo, existen miles de nombres generados por el código de

gvllisqi.eu	odgmmjwsdsrb.net
wfxspste.cc	uuummdqifcon.ru
pvtlkprr.co	washingtoncalanthe.net
hzsitbdm.eu	mjuwntiwmtya.net
xxlzgrom.cc	gacyzuh.com
qhbynrab.co	dbzwestnessbiophysicalohax.com

Tabla 4.1: Ejemplo de nombres de dominio generados mediante DGA [29].

Familia	**Ejemplo**	**Tipo de DGA**
Banjori	zvfdestnessbiophysicalohax.com	Basados en permutaciones.
Corebot	ybylsvo0ahwpe2i0mdinibo.ddns.net	Aritmético.
Dircrypt	ktqyrmiyvnidd.com	Aritmético.
Dnschanger	tcfejerekw.com	Aritmético.
Fobber	ugovykiwouxhdlrtj.net	Aritmético.
Gozi	ulpurgatoriopetrum.com	Basado en diccionario.
Kraken v1	ozyqosysu.dyndns.org	Aritmético.
Kraken v2	ygdcdhonlxxs.com	Aritmético.
Locky	qtysmobytagnrv.it	Aritmético.
Murofet	nwpyftn30gso51krkrnzh64f62aym29lvmtlu.biz	Aritmético.
Necurs	iqdpvmeywdb.kz	Aritmético.
Newgoz	p46prua8qn39yijcj2n2o0xq.net	Aritmético.
Nymaim	shlxqighem.com	Aritmético.
Padcrypt	nmcdnfbacafmecab.co.uk	Aritmético.
Pykspa	uyznvxlof.info	Aritmético.
Qadars	lensxmn0puz8.com	Aritmético.
Qakbot	xzzicwkdvayojtkckzzlr.biz	Aritmético.
Ramnit	ppvrnfkbarbnlm.com	Aritmético.
Ranbyus	tuusskblufagqnjan.pw	Aritmético.
Shiotob	hey9ydfb5v5o2.net	Aritmético.
Simda	purywoq.com	Aritmético.
Sisron	mjcwntiwmtya.net	Aritmético.
Suppobox	willoughbyalbertson.net	Basado en diccionario.
Symmi	ofvaucifadvukii.ddns.net	Aritmético.
Tinba	srqpkllgskhw.in	Basados en permutaciones.
Unnamed_javascript_dga	rnjaigkfu.co	Aritmético.

Tabla 4.2: Tipos de DGAs en el repositorio de Bader [29].

Murofet mientras que para *Gozi* únicamente hay 12 ejemplos de nombres. Esta descompensación entre los datos pertenecientes a cada una de las diferentes familias hace más difícil el ajuste del modelo, ya que el modelo aprendería a identificar con mayor acierto los nombres de dominio generados por *Murofet* antes de los generados por otras familias. Siguiendo el ejemplo anterior, un modelo que clasificara las 12 muestras de *Gozi* erróneamente obtendría aún así unos resultados de clasificación globales muy buenos debido al poco impacto de estas muestras. Sin embargo, el modelo sería incapaz de identificar ningún nombre de dominio generado por este DGA.

Por este motivo, en lugar de emplear los ejemplos de nombres de dominio ya proporcionados en el repositorio, se ha utilizado el código fuente contenido en el repositorio para generar desde cero una lista de 100 000 nombres de dominio por cada familia, con el fin de construir un conjunto de datos más equilibrado en la cantidad de patrones por clase. A partir de esa lista se ha extraído aleatoriamente una muestra de mil dominios por familia, resultando en un conjunto de datos de 32 000 nombres de dominio generados con DGA, etiquetados como “MALWARE”.

Durante la creación del conjunto de datos se observó que existen varias familias que a lo largo de su historia han empleado diferentes mecanismos de generación de DGA. En estos casos se han generado los mil nombres de dominio para cada una de estas versiones. Este es el motivo por el cual se ha obtenido un conjunto de datos de 32 000 nombres de dominio en lugar de los 27 000 que hubiera sido de esperar de las 27 familias consideradas. En concreto, las familias con diferentes tipos de DGA son *Fobber*, *Kraken*, *Locky*, *Murofet* y *Ranbyus*.

Conjunto	**# dominios**	**Seleccionados**	**Etiqueta**
Alexa 1M	1 000 000	32 000	CLEAN
Bader extendido	3 198 304	32 000	MALWARE

Tabla 4.3: Conjuntos de datos.

Tal y como se muestra en la tabla 4.3, con el fin de que el conjunto de datos final esté balanceado, no solo entre las diferentes familias de DGAs sino entre dominios maliciosos y dominios legítimos, del millón de nombres de dominio recogidos a partir de Alexa se han seleccionado los primeros 32 000 nombres de la lista para formar el conjunto de datos final. El motivo por el que no se extrajo una muestra aleatoria, a diferencia de los nombres de dominio generados con DGA, es que la lista de Alexa guarda un orden de relevancia según el número de visitas y, por tanto, presumiblemente tendrán un aspecto más representativo de lo que debería ser un dominio legítimo que otros dominios situados hacia el final de esa misma lista.

Este conjunto de datos, siguiendo las buenas prácticas en materia de investigación [77], se ha publicado en Zenodo [79]. Su acceso es libre y gratuito, por lo que puede ser empleado por cualquier investigador para validar los resultados de esta tesis doctoral y para cualquier otro fin.

El conjunto de datos está formado por los siguientes ficheros:

- "alexa-32k.txt": fichero compuesto por 32 000 nombres nombres de dominio legítimos correspondiente a los primeros 32 000 elementos de la lista "Alexa's Top 1 million". Son por tanto los 32 000 nombres de dominio más empleados en Internet de forma global.
- "dga-32k.txt": 32 000 nombres de dominio generados mediante diferentes tipos de DGAs, de forma balanceada (misma cantidad de nombres de dominio por cada algoritmo).

El contenido de estos ficheros es únicamente los nombres del dominio, sin ningún otro tipo de información o dato extraído del mismo, separados por saltos de línea.

Capítulo 5

Detección de dominios generados algorítmicamente usando aprendizaje no profundo

Este capítulo explica el clasificador empleando la técnica *Random Forest* y los *N-Grams* enmascarados propuestos en esta tesis. Se inicia con la descripción del proceso de ingeniería de características y extracción de las mismas, para luego pasar a detallar el diseño de la solución, experimentos realizados y conclusiones. Finalmente se analiza la interpretabilidad del modelo y se hace referencia a las herramientas empleadas y publicadas.

5.1. Extracción de características

Dado que el nombre de dominio es una secuencia de cadenas de caracteres separadas por el carácter punto ('.'), es posible considerarla como una única cadena o, por el contrario, separarla en cada uno de las partes que lo componen (TLD, dominio, subdominios y nombre del dispositivo; véase explicación de la sección 2.1), extrayendo las características de cada uno de ellos por separado. Por ejemplo, tal y como se menciona en la sección 3 al detallar el estado del arte, Da Luz emplea una serie de características léxicas que son extraídas considerando el nombre de dominio como una única cadena de texto [68].

Tal y como se menciona en el capítulo 4, esta tesis se centra en técnicas de detección basadas únicamente en características léxicas del nombre del dominio, debido a la volatilidad que tienen otras características usadas en la literatura. Esta volatilidad impide el entrenamiento con DGAs de malware que no se encuentre operativo en el momento de la recolección del conjunto de datos. Por este motivo se han descartado todas las características propuestas por Da Luz que no pueden ser extraídas del nombre de dominio, mostrándose las resultantes en la tabla 5.1.

Las características de la F1 a la F12 representan información estadística (media, varianza y desviación estándar) de los *N-Grams* de tamaño 1 a 4 extraídos de los nombres de dominio en bruto. Estas características proporcionan resultados relevantes cuando el nombre del dominio es suficientemente largo como para que estas resulten representativas. Por el contrario, pueden no resultar de mucha utilidad en nombres de dominio cortos, ya que se pueden extraer de ellos pocos *N-Grams*.

ID	Descripción
F1 – 3	Media, varianza y desviación estándar (*1-gram*).
F4 – 6	Media, varianza y desviación estándar (*2-gram*).
F7 – 9	Media, varianza y desviación estándar (*3-gram*).
F10 – 12	Media, varianza y desviación estándar (*4-gram*).
F13 – 14	Entropía de Shannon de los dominios de segundo y tercer nivel.
F15	Cantidad de caracteres diferentes.
F16	Proporción de dígitos respecto a la longitud total.
F17	Proporción de consonantes con respecto a la longitud total.
F18	Proporción de consonantes con respecto a vocales.

Tabla 5.1: Características léxicas empleadas por Da Luz en [68].

Las características F13 y F14 se obtienen de calcular la entropía de Shannon [80] a partir de los dominios de segundo y tercer nivel. Este factor es usado para medir el nivel de aleatoriedad en la cadena de texto del nombre del dominio, excluyendo el TLD. Dado que proporciona una medida de aleatoriedad, se obtienen valores más altos para aquellos nombres de dominio que son generados mediante DGA que para aquellos creados de forma manual, ya que estos suelen construirse a partir de términos más sencillos de recordar y escribir por seres humanos.

Finalmente, las características de la F15 a la F18 dan una medida del equilibrio entre vocales, consonantes, dígitos y la longitud total del nombre de dominio. Estos valores deberían ser muy diferentes en los nombres generados con DGAs con respecto al lenguaje natural que suele ser empleado en los dominios legítimos. En general, todas las características descritas anteriormente son características estadísticas que buscan medir el nivel de aleatoriedad del nombre del dominio como manera de identificar nombres de dominio generados con DGA.

5.2. *N-Grams* enmascarados

En esta tesis se propone emplear, además de las características léxicas ya utilizadas por Da Luz [68], nuevas características basadas en la ocurrencia de determinados *N-Grams*. Recuérdese que un *N-Gram* es el resultado de dividir una secuencia en todas las posibles sub-secuencias de longitud N y es una técnica ampliamente usada en diferentes campos de la ciencia. En la tabla 5.2 se observa un ejemplo de la división del nombre de dominio "google.com" en sus *2-Grams* (remarcados en negrita). En este mismo ejemplo, la característica del *2-Gram* "ww" (resaltado en color rojo) tendría un valor de dos, ya que aparece dos veces al descomponer la secuencia completa en *2-Gram*, mientras que otras características como el *2-Gram* "go" (resaltado en color verde) tendrían el valor uno, ya que se da una sola vez.

Sin embargo, el uso de las ocurrencias de los *N-Grams* como característica tiene una gran desventaja en secuencias con una gran variedad de posibles valores, como son un nombre de dominio, ya que la cantidad de características necesarias para representar todos y cada uno de los *N-Grams* crece exponencialmente cuanto mayor es el valor N. Por ejemplo, el empleo de la ocurrencia de *3-Grams* como característica en una secuencia donde cada elemento pertenece a un alfabeto de 36 posibles valores, tiene como resultado

w	**w**	w	.	g	o	o	g	l	e	.	c	o	m
w	**w**	**w**	.	g	o	o	g	l	e	.	c	o	m
w	w	**w**	**.**	g	o	o	g	l	e	.	c	o	m
w	w	w	**.**	**g**	o	o	g	l	e	.	c	o	m
w	w	w	.	**g**	**o**	o	g	l	e	.	c	o	m
w	w	w	.	g	**o**	**o**	g	l	e	.	c	o	m
w	w	w	.	g	o	**o**	**g**	l	e	.	c	o	m
w	w	w	.	g	o	o	**g**	**l**	e	.	c	o	m
w	w	w	.	g	o	o	g	**l**	**e**	.	c	o	m
w	w	w	.	g	o	o	g	l	**e**	**.**	c	o	m
w	w	w	.	g	o	o	g	l	e	**.**	**c**	o	m
w	w	w	.	g	o	o	g	l	e	.	**c**	**o**	m
w	w	w	.	g	o	o	g	l	e	.	c	**o**	**m**

Tabla 5.2: Ejemplo de *2-Grams*.

que es necesario incorporar 36^3 características. Del mismo modo, si con el mismo alfabeto se quisieran tener en cuenta los *4-Grams*, implicaría el uso de 36^4 nuevas características. Por tanto, la complejidad computacional hace muy poco eficiente, o incluso imposible, la creación de un modelo que empleara esas características, motivo por el cual gran parte de los autores emplean únicamente información estadística extraída de los *N-Grams* (media, varianza y desviación estándar), pero no los *N-Grams* en sí mismos.

Para evitar esta limitación, esta tesis propone el uso de *N-Grams* enmascarados (en inglés, *Masked N-Grams*). Empleando esta aproximación, cada uno de los caracteres del nombre del dominio es substituido por un símbolo que representa el tipo de carácter. Las consonantes se substituyen por símbolos 'c', las vocales por símbolos 'v', los dígitos por símbolos 'n' y cualquier otro tipo de carácter por el símbolo 's'. Por ejemplo, el sitio web "www.my-website.com" será enmascarado como "cccscscvccvcvscvc". Esta aproximación reduce el tamaño del alfabeto usado a tan solo 4 elementos, y por tanto reduce la cantidad de posibles combinaciones de los *N-Grams* a 4^N, siendo N el tamaño del *N-Gram*.

Esta transformación del nombre de dominio a su equivalente enmascarado para la extracción de los *N-Grams* tiene como consecuencia cierta pérdida de información con respecto al original. *N-Grams* como "car", "bus" o "yet" estarán representados por la misma característica, correspondiente al *N-Gram* enmascarado "cvc". Sin embargo, esta aproximación aporta mucha más información que el uso de los valores estadísticos extraídos de los *N-Grams*.

Así pues, para la realización de los experimentos se combinan las características léxicas propuestas por Da Luz, obtenidas a partir de los nombres de dominio sin modificar, y las características extraídas los *N-Grams* enmascarados, obtenidos a partir de los nombres de dominio enmascarados, con el fin de evaluar la representatividad de las nuevas características comparadas con las ya empleadas por otros autores.

5.3. Descripción del modelo

Las técnicas basadas en árboles de decisión [40] han sido ampliamente usadas en la industria de la ciberseguridad [81, 82, 83] debido a su buen rendimiento. Una vez que el

árbol es construido, se trata de un modelo que es capaz de clasificar a una gran velocidad, por lo que encaja bien en muchos campos de aplicación de la industria de la ciberseguridad, donde la velocidad en la respuesta a la clasificación es determinante, ya que la respuesta a una amenaza debe producirse lo antes posible.

Una de estas técnicas es conocida como *Random Forest* [41] cuyos fundamentos teóricos son tratados en las secciones 2.3.1 y 2.3.2. Este tipo de modelos proporciona una muy buena generalización, lo cual es muy deseable para la identificación de amenazas que, si bien no se encontraban en el conjunto de entrenamiento, son similares a otras ya analizadas con anterioridad.

Por este motivo se elige *Random Forest* como modelo de aprendizaje no profundo. Además, los modelos basados en árboles funcionan bien tanto con valores numéricos como con valores literales, por lo que no se requiere ningún tipo de normalización o adaptación del conjunto de datos. Por último, es un modelo que ya ha sido usado con anterioridad en problemas similares, para los que ha proporcionado buenos resultados [68].

Finalmente se usa el método de selección de características Boruta [43], que se basa en un modelo *Random Forest* y cuyos fundamentos son tratados en la sección 2.3.3. Empleando este método se calcula la importancia de cada una de las características para obtener la clasificación final y a partir de este dato se reduce la dimensionalidad del problema con una mínima pérdida de exactitud.

En este trabajo se usa la implementación de *Random Forest* de la librería Caret [84], disponible para el lenguaje de programación R [44], que proporciona una capa de abstracción y facilita el uso de otras conocidas librerías disponibles en R. En concreto, se usa el método *rf*, que es una abstracción que emplea la librería *randomForest* de R [85]. Por defecto esta implementación emplea un conjunto de 500 árboles y elije en cada bifurcación, de entre las características elegidas aleatoriamente, aquella que produce mejores valores en su curva ROC (*Receiver Operating Characteristic*).

5.4. Experimentos

Para la evaluación de esta aproximación se diseñan varios experimentos que se detallan a continuación:

Experimento 1. Se emplea el método de clasificación *Random Forest* implementado en *Caret* con sus hiperparámetros por defecto. Se mide la precisión, sensibilidad y exactitud al emplear *N-Grams* de tamaño desde 1 a 4 de forma separada.

Experimento 2. En este experimento se emplean todas las características disponibles (incluyendo los *N-Grams* de diferentes tamaños) de forma conjunta y se aplica el método de Boruta para establecer cuáles son las características más importantes para la clasificación. Esta técnica también permite comparar las características basadas en *N-Grams* con el resto de características ya empleadas por otros autores.

Experimento 3. Finalmente se repite el primer experimento con un conjunto de características reducido, elegido a partir de los resultados del segundo experimento. Los resultados obtenidos se comparan con el mejor rendimiento obtenido en el primer experimento.

En cuanto al conjunto de datos, tras combinar los conjuntos de datos "CLEAN" y "MALWARE" descritos en el capítulo 4, este se divide en dos partes: un 60 % de los nombres de dominio elegidos aleatoriamente forman el conjunto de entrenamiento, mientras que el resto de ellos forman el conjunto de test. Para el entrenamiento se usa un mecanismo de validación cruzada de tamaño 10.

5.5. Discusión de resultados

Primer experimento. El primero de los experimentos muestra unos buenos resultados en su clasificación, especialmente en el experimento en el que se emplean las características extraídas de los *N-Grams* de tamaño 2, en el que se alcanza una exactitud del 98.91 % con una tasa de falsos positivos alrededor del 0.16 % (nombres de dominio legítimos clasificados como generados por DGA) en el conjunto de datos de validación. En este experimento no se emplea el conjunto de datos de test debido a que se reserva para evaluar los resultados finales del modelo.

Un bajo ratio de falsos positivos es importante para ser aplicable a la industria de la seguridad, especialmente para sistemas que realicen un bloqueo automático, ya que de lo contrario ciertos nombres de dominio legítimos pueden ser bloqueados y, por tanto, sus servicios no serían accesibles debido a este error en la clasificación.

En cuanto al rendimiento, el tiempo de pruebas y de entrenamiento son mayores cuanto mayor es el tamaño de los *N-Grams* empleados, ya que se emplea una mayor cantidad de características. De forma general se puede pensar que el rendimiento debe decrecer y aumentar la exactitud en la clasificación al incrementar la cantidad de características. Sin embargo, los resultados muestran que la mejor exactitud se obtiene empleando las características extraídas de los *N-Grams* de tamaño 1 y 2. Ese resultado muestra lo que es una tendencia en el tema del aprendizaje automático: *más características no significa necesariamente mejor funcionamiento*.

La tabla 5.3 muestra los resultados de este primer experimento. En dicha tabla el tiempo de entrenamiento está representado en horas, mientras que el tiempo de pruebas está expresado en segundos. También se ha destacado en la tabla el mejor resultado obtenido.

	1-grams	**2-grams**	**3-grams**	**4-grams**
Características	22	**34**	82	274
T. entrenamiento (h)	0.65 h	**1.21 h**	4.21 h	47.74 h
T. pruebas (s)	1 s	**1 s**	2 s	4 s
Exactitud	0.9890	**0.9891**	0.9879	0.9873
Kappa	0.9780	**0.9783**	0.9758	0.9747
Sensibilidad	0.9896	**0.9859**	0.9857	0.9848
Especificidad	0.9884	**0.9924**	0.9901	0.9899

Tabla 5.3: Resultados de clasificación con *Random Forest* (primer experimento).

Segundo experimento. Con este experimento se pretende identificar las características más relevantes con respecto a la clasificación. Las características se combinan en un nuevo conjunto de datos que contienen tanto las características léxicas y estadísticas como las características de ocurrencia de *N-Grams* para los diferentes tamaños, a excepción de

las basadas en *4-Grams*, debido al gran tiempo de entrenamiento que requiere y que, a priori, no proporciona muy buenos resultados según el experimento anterior comparado con *N-Grams* de tamaño inferior.

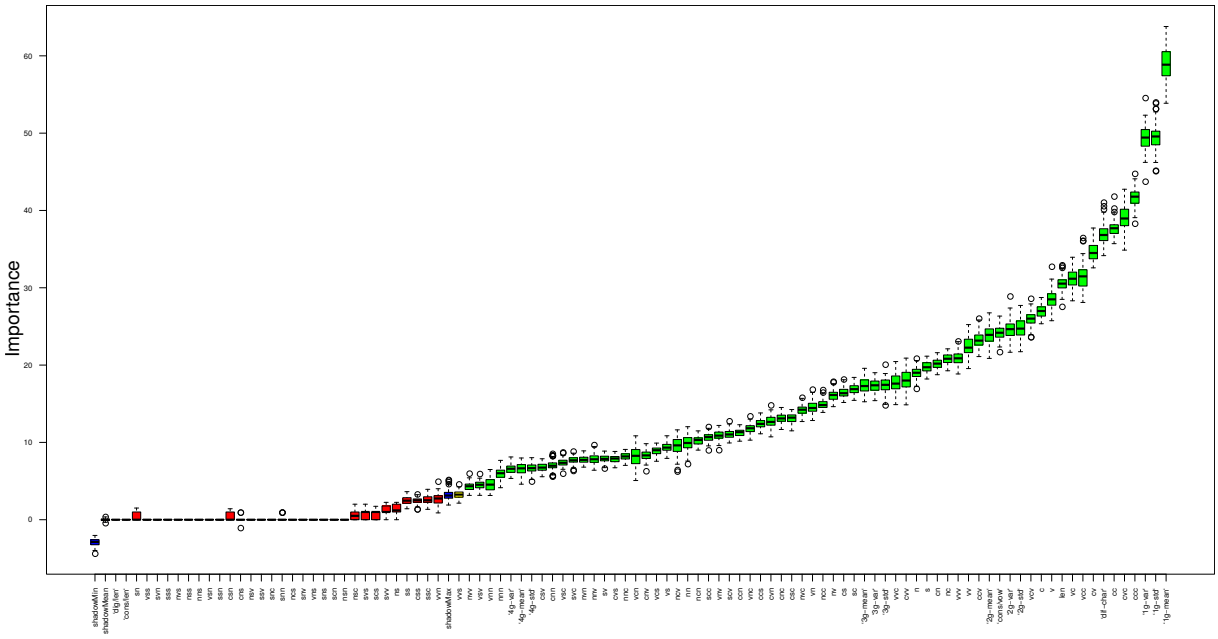


Figura 5.1: Gráfico de los resultados de Boruta.

En la figura 5.1 están representados los resultados del método de Boruta en un diagrama *boxplot*. En ella se muestra que la importancia de las características crece de una forma aproximadamente lineal hasta que llega a las 10 características con mayor importancia. Tras ese punto la importancia aumenta de forma exponencial. La tabla 5.5 muestra las 15 características más relevantes según Boruta. De entre ellas, las tres características con mayor importancia son la media, varianza y desviación estándar calculada a partir de los *1-Grams*. Sin embargo, 9 de las 15 características más importantes corresponden a características de ocurrencia de *N-Grams* enmascarados, que han sido resaltadas en dicha tabla.

Como era de esperar, la exactitud del modelo se reduce al disminuir la cantidad de características empleadas, aunque esta reducción es únicamente de un 0.5 %, mientras que el tiempo de entrenamiento disminuye de forma notable. En concreto, el tiempo de entrenamiento se reduce en hasta 7 veces al reducir las características empleadas de 22 a las 5 más relevantes. Esta mejora en el rendimiento es muy deseable, especialmente en entornos reales y es interesante por dos motivos: primero, una mejora en el tiempo de entrenamiento permite entrenar al modelo con una cantidad más amplia de datos; y segundo, extraer menos características mejora el tiempo de respuesta de la clasificación, permitiendo respuestas cercanas al tiempo real, lo cual es de gran importancia en entornos de prevención de intrusiones, donde se desea una respuesta activa ante la detección como puede ser un bloqueo automático.

Tercer experimento. Finalmente, en el tercer experimento se repite el entrenamiento del modelo *Random Forest* empleando las características más importantes según la información recogida del segundo experimento. En concreto, se repite el experimento para las 10, 15, 20 y 25 características más importantes. Para comparar estos resultados con los obtenidos en el primer experimento, se tienen en cuenta sus mejores resultados, usando *2-grams* enmascarados.

La tabla 5.4 recoge los resultados de este experimento. Los resultados muestran que el modelo *Random Forest* entrenado únicamente con las 15 características más importantes (según Boruta) tiene una exactitud muy cercana a los mejores resultados obtenidos en el primer experimento. Sin embargo, el tiempo de entrenamiento empleando únicamente 15 características es prácticamente la mitad. En base a estos datos se concluye que el uso de *N-Grams* enmascarados aporta información muy relevante para una correcta clasificación de dominios generados con DGA.

	2-grams	**Top 25**	**Top 20**	**Top 15**	**Top 10**
Características	**34**	25	20	**15**	10
T. entrenamiento (h)	**1.21 h**	1.18 h	0.82 h	**0.63 h**	0.44 h
Exactitud	**0.9891**	0.9865	0.9865	**0.9873**	0.9832
Kappa	**0.9783**	0.9730	0.9730	**0.9747**	0.9664
Sensibilidad	**0.9859**	0.9851	0.9848	**0.9859**	0.9827
Especificidad	**0.9924**	0.9880	0.9883	**0.9888**	0.9837

Tabla 5.4: Resultados de clasificación con *Random Forest* (tercer experimento).

Media (1-gram)	Varianza (1-gram)	Desviación estándar (1-gram)
Desviación estándar (2-gram)	Caracteres diferentes	Longitud
'ccc'	**'cvc'**	**'vcc'**
'vcv'	**'cv'**	**'vc'**
'cc'	**'c'**	**'v'**

Tabla 5.5: Listado de las 15 características más relevantes según Boruta. Los *N-Grams* enmascarados están resaltados en verde.

5.6. Interpretabilidad del modelo

Con el fin de analizar el proceso de clasificación establecido por el modelo *Random Forest* se ha extraído de forma detallada la votación de cada uno de los 500 árboles generados por el modelo para los elementos del conjunto de datos de test y se ha identificado el árbol cuya votación coincide con la mayoría de las votaciones del resto de árboles en un mayor porcentaje (99 %). Al ser su votación coincidente con la de la mayoría, por tanto, también lo es con la clasificación proporcionada por la totalidad del modelo. La figura 5.2 muestra la estructura de dicho árbol. Como se puede observar, el árbol presenta una gran cantidad de nodos (más de 6 000) y de profundidad, lo cual dificulta su visualización y por tanto su análisis más detallado.

En la figura 5.3 se muestra el mismo árbol que en la figura 5.2 pero limitando su profundidad a tres niveles con el fin de facilitar su visualización. Las hojas que muestran la decisión final fueron creadas de forma artificial para esta simplificación en base a la probabilidad de que se diera esa clasificación, pero no representa la clasificación real del árbol puesto que este considera otros muchos parámetros que no son mostrados en este árbol simplificado. Aún así, esta simplificación es de utilidad para observar qué tipo de decisiones toman cada uno de los árboles con respecto al nombre del dominio, ya que el árbol escogido coincidía en su votación con la mayoría del resto de árboles en un 99 % de las ocasiones, como ya se ha mencionado.

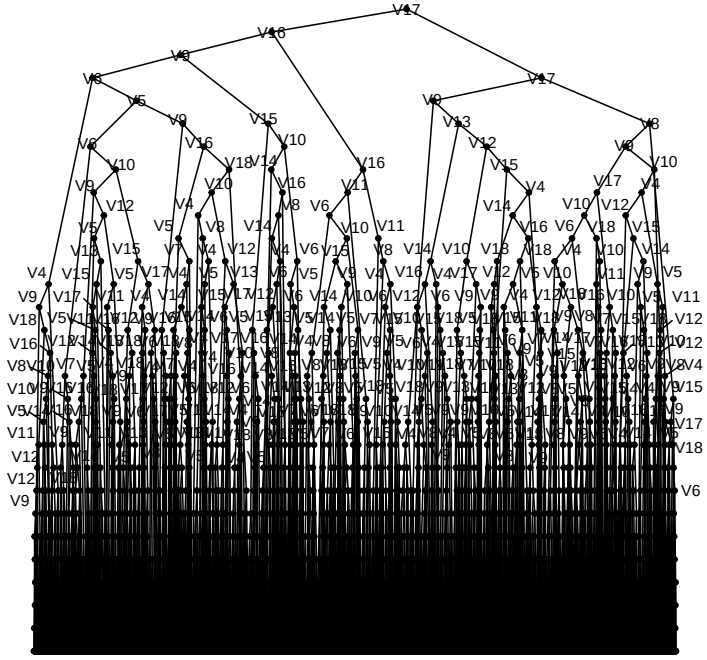


Figura 5.2: Árbol cuyo voto coincide en más ocasiones con el de la mayoría.

En este árbol de decisión simplificado se observa cómo el modelo ha establecido diferentes comprobaciones sobre la cantidad de consonantes presentes en el nombre del dominio y su relación con los caracteres cercanos, concretamente con las vocales. Estas comprobaciones se encuentran en ocasiones ponderadas a la longitud del nombre del dominio y, en menor medida, a la variabilidad en el tipo de carácter empleado. Este análisis refleja lo que se puede esperar de forma intuitiva, ya que los nombres de dominio que contengan una gran cantidad de consonantes o combinaciones inusuales entre vocales y consonantes son presumiblemente más difíciles de pronunciar y, por tanto, más sospechosos de haber sido generados de manera algorítmica.

Para ilustrar el funcionamiento del modelo de una forma empírica se elige del conjunto de datos el nombre de dominio legítimo "facebook.com" y el dominio malicioso "wxhyqqrbouru.pw" para realizar un ejemplo de clasificación. Facebook es un portal web conocido por albergar una de las redes sociales más utilizadas en el mundo. Por su parte, "wxhyqqrbouru.pw" es un dominio generado por el DGA del malware *Tiny Banker*, también conocido como *Timba*. Para este ejemplo se consideran únicamente las 15 características más relevantes para el problema de clasificación.

	Media 1-Gram	**Std 1-Gram**	**Var 1-Gram**	**Std 2-Gram**
facebook.com	1.33	0.71	0.50	0.00
wxhyqqrbouru.pw	1.36	0.50	0.25	0.00

Tabla 5.6: Características estadísticas de los *N-Grams* del nombre de dominio.

En primer lugar, la tabla 5.6 muestra los valores de media, varianza (Var) y desviación estándar (Std) de los *N-Grams* de tamaño 1 y 2 extraídos del nombre del dominio. Aquellos nombres de dominio en los que cada carácter se usa una única vez resultan en una menor varianza y desviación estándar que otros nombres de dominio en los que los caracteres se

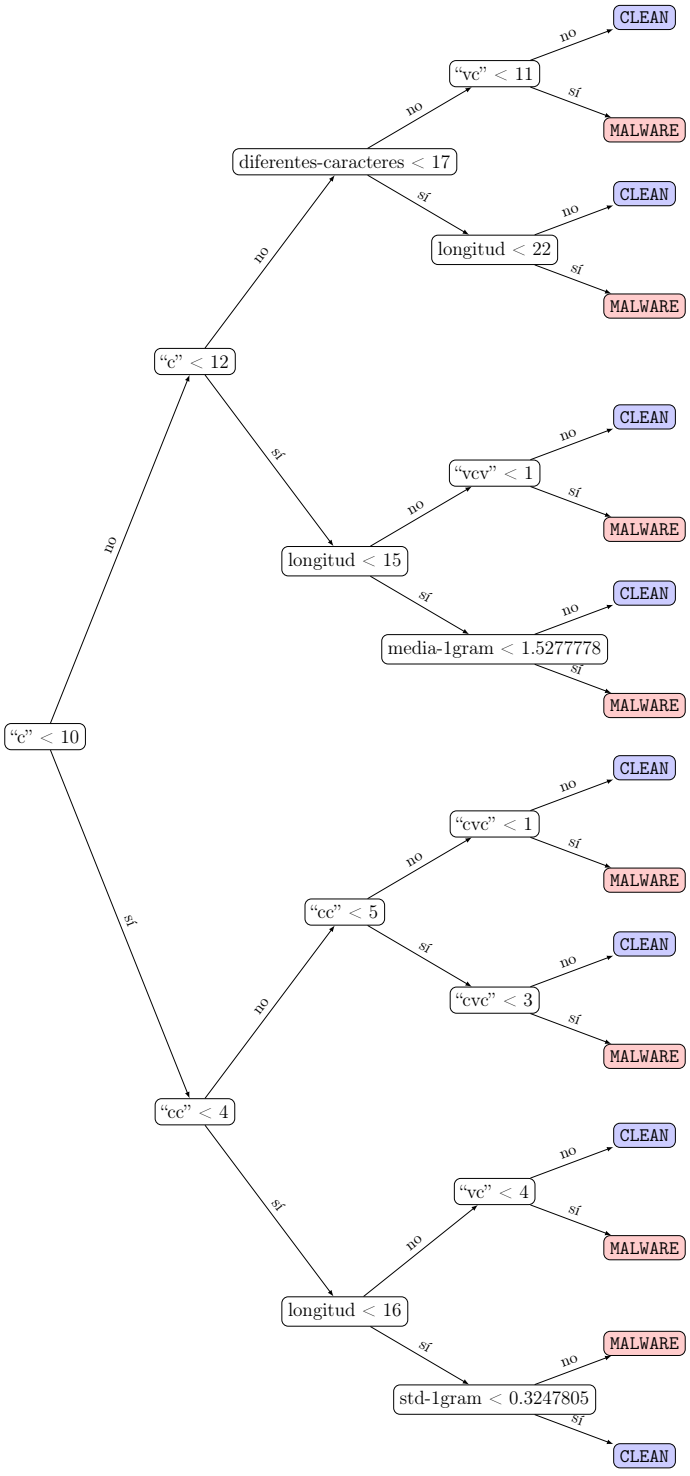


Figura 5.3: Árbol de decisión (simplificado).

repiten. Estas características, por tanto, proporcionan información sobre la aleatoriedad del nombre de dominio.

En segundo lugar, la tabla 5.7 muestra otra información no relacionada con los *N-Grams*. En concreto, la cantidad de caracteres diferentes (sin incluir caracteres repetidos) y la longitud total del nombre de dominio. Estas características son extraídas del nombre del dominio y, junto con las anteriores, son un subconjunto de las características previamente utilizadas por Da Luz y otros autores [63, 64, 65, 16, 66].

	#Caracteres diferentes	Longitud
facebook.com	9	12
wxhyqqrbouru.pw	11	15

Tabla 5.7: Características basadas en otro tipo de estadísticas sobre el nombre del dominio.

Dominio	Dominio enmascarado	ccc	cvc	cc	cv	vcc	vc	v	c	vcv
facebook.com	cvcvcvvc.cvc	0	3	0	4	0	4	5	6	2
wxhyqqrbouru.pw	ccccccccvvcv.cc	6	0	8	2	0	1	3	11	1

Tabla 5.8: Características basadas en los *N-Grams* del nombre de dominio enmascarado.

Finalmente se genera el nombre de dominio enmascarado y se extraen los 9 *N-Grams* que proporcionan una buena capacidad de clasificación. Por cada uno de los *N-Grams* se cuentan las veces que aparecen en el nombre de dominio enmascarado, tal y como puede apreciarse en la tabla 5.8. Por ejemplo, dado el nombre de dominio "facebook.com" (enmascarado como "cvcvcvvc.cvc"), la característica correspondiente al *N-Gram* "ccc" es cero, dado que no se da la circunstancia de que aparezcan tres consonantes seguidas en el nombre de dominio original. Sin embargo, para el nombre de dominio "wxhyqqrbouru.pw" (enmascarado como "ccccccccvvcv.cc"), esa misma característica tiene un valor de 6, ya que el nombre de dominio contiene hasta seis *3-Grams* compuestos únicamente por consonantes. Esta característica muestra para este ejemplo la diferencia entre un dominio creado manualmente y un dominio generado por un DGA, puesto que el uso de demasiadas consonantes de forma simultánea y repetido en demasiadas ocasiones está alejado de lo que se espera del lenguaje natural y de lo que es un nombre de dominio creado de forma manual por un ser humano [86].

De entre los falsos negativos (dominios maliciosos no detectados como tales), la tabla 5.9 muestra un ejemplo de 10 de ellos extraídos de forma aleatoria. Varios de estos dominios han sido generados a partir de palabras en lenguaje humano y no de forma completamente aleatoria. Por ello su forma y aspecto son muy similares, o incluso idénticos, a los generados de forma manual, a no ser que se emplee una excesiva longitud para aumentar el espectro de posibles combinaciones de palabras y ello delate su condición de malicioso.

Por otro lado, dominios como "ocdocgoklu.com" no pertenecen a esta categoría pero así mismo resultaron ser clasificados como dominios legítimos. Esto se debe a la inherente probabilidad que al generar una gran cantidad de nombres de forma aleatoria una mínima cantidad de ellos tengan una distribución de vocales y consonantes que lo haga parecer un dominio legítimo.

jowovy.info	ocdocgoklu.com	verysugar.net
roazmec.cc	roazmec.cc	ltitgreqelrqrip.com
pupycop.com	beginfather.net	rockstep.net
	iosyajvaxk.com	

Tabla 5.9: Ejemplo de falsos negativos con *Random Forest*.

En cuanto a los falsos positivos, en la lista de "Alexa's Top 1000" se han detectado 27 nombres de dominio como falsos positivos que se listan en la tabla 5.10. Algunos de

ellos son nombres de servicios tan populares como “netflix.com” o “airbnb.com”. Muchos de ellos son nombres de dominio compuestos de varias palabras en los que se emplean siglas compuestas únicamente por consonantes. Esto ocasiona que su ratio de consonantes sea mayor debido a la unión entre palabras que finalizan por consonantes y palabras que comienzan por consonantes, así como las propias siglas compuestas únicamente por consonantes. Por último, el análisis lleva a observar que existen consonantes como ‘h’ e ‘y’ que en determinados idiomas pueden no sonar o tener un sonido vocálico, pero que sin embargo el modelo propuesto los evalúa como caracteres consonantes.

netflix.com	onclickads.net	pornhub.com	craigslist.org
wellsfargo.com	hdfcbank.com	bestbuy.com	adplxmd.com
airbnb.com	hotstar.com	zippyshare.com	thewatchseries.to
bookmyshow.com	streamcloud.eu	conservativetribune.com	makemytrip.com
sciencedirect.com	wattpad.com	buzzfil.net	marktplaats.nl
rappler.com	inspsearch.com	nydailynews.com	curapelanatureza.com.br
myfreecams.com	cbssports.com	ampclicks.com	

Tabla 5.10: Falsos positivos con *Random Forest* en “Alexa’s Top 1000”.

En la tabla 5.11 se observa cómo, al introducir diferentes cambios en el dominio “netflix.com”, los dominios resultantes son clasificados como “CLEAN” por el modelo. Solo con introducir una vocal en cualquier lugar o eliminar una de las consonantes que formaban el *3-Gram* “tfl” el modelo cambia su clasificación inicial, lo cual refuerza la interpretación mencionada con anterioridad respecto a los falsos positivos detectados. En este caso el dominio “netflix.com” está formado por dos palabras: “net” (red) y “flix” (derivada de *flicks*, películas). Ambas son palabras cortas que comienzan y terminan por letras consonantes, y que por tanto tienen mucha presencia de consonantes. Además al combinarlas se genera una sucesión de varias consonantes consecutivas en el punto en el que ambas palabras se unen. Del mismo modo, la tabla 5.11 muestra cómo al cambiar la consonante ‘y’ por la vocal ‘i’ en el dominio “myfreecams.com” (y por tanto convertirse en “mifreecams.com”), su clasificación pasa de ser incorrecta a ser correcta.

Dominio	**Clasificación**	**Votos malware**	**ccc**	**cvc**	**vcc**	**vcv**	**cv**	**vc**	**cc**	**c**	**v**
netflix.com	MALWARE	488	1	3	1	0	3	3	2	7	3
netflixa.com	CLEAN	12	1	3	1	1	4	3	2	7	4
netoflix.com	CLEAN	0	0	4	1	1	4	4	1	7	4
neflix.com	CLEAN	63	0	3	1	0	3	3	1	6	3
myfreecams.com	MALWARE	360	2	2	1	1	3	3	4	9	4
mifreecams.com	CLEAN	5	0	3	2	1	4	4	2	8	5

Tabla 5.11: Variaciones con respecto a nombres de dominio que produjeron falsos positivos.

En cuanto a la velocidad de la clasificación, este modelo es capaz de clasificar el conjunto de datos completo (compuesto de 64 000 nombres de dominio) en 2.596 segundos. Para tomar esta medida no es necesario dividir el conjunto de datos de ningún modo ya que el objetivo es medir la velocidad en la clasificación y no su exactitud.

5.7. Herramientas desarrolladas

Todo el código empleado para la construcción de los modelos resultantes de este trabajo se ha publicado de forma abierta en GitHub [18]. Este repositorio contiene el código en lenguaje Python y R empleado para la extracción de características y para la implementación del clasificador *Random Forest* descrito con anterioridad.

Además de publicar todo el código que implementa el clasificador, con el fin de facilitar la reproducción de los resultados, se ha generado una imagen Docker que está públicamente accesible y que puede ser utilizada para reproducir los experimentos de esta tesis doctoral [18].

Capítulo 6

Detección de dominios generados algorítmicamente usando aprendizaje profundo

Este capítulo explica la propuesta de esta tesis sobre un clasificador para la detección de dominios generados algorítmicamente empleando una red neuronal recurrente LSTM con un diseño óptimo. Se inicia con la descripción de la codificación de los caracteres para luego detallar el diseño de la solución, los experimentos realizados y las conclusiones obtenidas. Finalmente analiza la interpretabilidad del modelo y referencia las herramientas empleadas y publicadas.

6.1. Codificación de características

Uno de los objetivos de esta tesis es evaluar si el complejo proceso de ingeniería de características (extracción, normalización y ajuste), necesario cuando se emplean otro tipo de técnicas de aprendizaje automático, puede ser reducido o evitado por medio del uso de aprendizaje profundo. En una arquitectura de aprendizaje profundo cada una de las capas transforma la entrada en una representación más abstracta. Un ejemplo clásico es el problema de reconocimiento facial, en el que las neuronas de las primeras capas se especializan en detectar líneas horizontales y verticales. La siguiente capa usa las líneas detectadas por la capa anterior para detectar formas simples como cuadrados o círculos. Las siguientes capas se especializan en la detección de otras formas más complejas como puede ser un ojo o una nariz. Finalmente, la última capa detecta la presencia de una cara en base a la información proporcionada por las anteriores capas [87]. Debido a esta característica, las redes neuronales son particularmente buenas manejando datos no estructurados, por lo que han sido utilizadas con mucho éxito en campos como el procesamiento de imágenes o el procesamiento de lenguaje natural, entre otros [49], ya que la red aprende de forma automática las características más importantes directamente de los datos de entrada como parte del entrenamiento. El uso de redes neuronales profundas simplifica el proceso de ingeniería de características, ya que deja de ser necesario extraer las características del dato de forma manual.

En el campo de las ciencias de datos, las cadenas de texto son generalmente representadas por un vector con valores numéricos entre 0 y 255 que representan un carácter, tal y como establece el *American Standard Code for Information Interchange* (ASCII) [88]. Sin

embargo, esta representación no funciona bien en la mayor parte de modelos de aprendizaje automático, ya que crea una similitud entre caracteres con códigos ASCII contiguos que en realidad no existe, ya que la secuencia de caracteres ASCII no guarda ningún tipo de continuidad entre ellos. Por ejemplo, dadas las cadenas de texto "AAAAA", "BCDFG" y "UUUUU", un modelo que usa el código ASCII para representar los caracteres probablemente concluya que las cadenas "AAAAA" y "BCDFG" son más similares entre ellas que "UUUUU", ya que 'A' y 'B' están representados por un valor numérico contiguo en ASCII, mientras que un ser humano probablemente diría que "AAAAA" y "UUUUU" son más similares entre ellas porque ambas cadenas están compuestas de una secuencia de vocales.

Para resolver este tipo de problemas se emplea el concepto de *embedding*. El *embedding* es una técnica que consiste en reducir un vector de gran dimensionalidad en otro vector de menor dimensionalidad que preserva la información más relevante del dato inicial. La mayoría de los modelos basados en redes neuronales incorporan de forma optativa el uso de *embeddings* en su diseño. Cuando esta capa se incorpora en el diseño se ve afectada por el aprendizaje del mismo modo que se ve afectada cualquier capa del modelo. Como consecuencia, el aprendizaje durante el proceso de entrenamiento encuentra la representación óptima de ese dato para resolver el problema [49].

A pesar de que gran parte de los autores citados en el capítulo 3 hacen uso de *embeddings* al construir sus modelos, bajo ciertas circunstancias se ha demostrado que el uso de representaciones estáticas también puede llegar a producir muy buenos resultados. *Word2vec*, publicado por Tomas Mikolov et al. [89], es una representación que enlaza cada palabra que se puede encontrar en el campo del lenguaje natural con un vector de cientos de características que representan su significado y el contexto de la palabra. Un ejemplo clásico de lo bien que esta aproximación representa el significado de las palabras es que si se obtiene la diferencia entre los vectores de características de la palabra "hombre" y "mujer" y se le suma al vector de características de la palabra "rey", el resultado será el vector de características de la palabra "reina". Por este motivo, esta representación es ampliamente utilizada en el campo de las ciencias de la computación para resolver los problemas de procesamiento de lenguaje natural.

El uso de una representación estática como la proporcionada por *Word2vec* implica que no es necesario el uso de una capa de *embeddings*. Esto reduce el número de parámetros entrenables del modelo y por tanto su complejidad.

Sin embargo, los FQDNs no están necesariamente compuesto por palabras, por lo que no es posible usar estrategias como *Word2vec*. Por este motivo, y con el fin de reducir el número de parámetros entrenables al mínimo posible, en esta tesis se utiliza una aproximación llamada *One-Hot Encoding* [49]. La técnica de codificación *One-Hot Encoding* es ampliamente usada para representar características categóricas en problemas de clasificación. Según esta técnica, cada carácter del FQDN está representado por un vector de 38 elementos (tamaño del alfabeto inglés junto con los símbolos '+', '-' y '.') donde la primera posición del vector representaría al carácter 'a', la segunda posición al carácter 'b', y así sucesivamente. Un vector que tuviera el valor 1 en la primera posición y 0 en el resto representaría al carácter 'a'. Siguiendo esta aproximación, no son válidos aquellos vectores que tienen más de una posición con valor 1, ya que no es posible que un vector represente a dos caracteres diferentes al mismo tiempo. La tabla 6.1 muestra algunos ejemplos de este tipo de codificación.

Para finalizar, modelos como LSTM requieren que todos los datos de los lotes del

foobar	**a**	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**	**k**	**l**	**m**	**n**	**o**	**...**
f	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	...
o	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	...
o	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	...
b	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	...
a	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...
r	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...

Tabla 6.1: Ejemplos de codificación *One-Hot Encoding*.

conjunto de entrenamiento tengan la misma longitud. Según el RFC de FQDN [31], su longitud está limitada a 255 caracteres y cada una de las partes que lo forman está limitado a 63 caracteres. Sin embargo, en la práctica la longitud promedio de los FQDN es muy inferior, motivo por el cual se adopta la decisión de diseño de tomar la longitud máxima de todos los FQDN del conjunto de datos y extender el resto de FQDNs del conjunto mediante el uso de un carácter nulo representado por un vector nulo de características, es decir, con un valor 0 en todas sus posiciones.

6.2. Descripción del modelo

Las redes neuronales recurrentes, descritas con detalle en la sección 2.3.5, fueron diseñadas para solucionar el problema del procesamiento de secuencias por una red neuronal. En las redes neuronales no recurrentes, la salida de la red se obtiene en función de los datos de entrada de la red, independientemente de los datos anteriores y posteriores que haya recibido dicha red. Aunque este comportamiento resulta deseable para un gran número de aplicaciones, impide el correcto procesamiento de secuencias en las que el resultado de la red debe estar en función tanto del dato de entrada como de los datos de entrada anteriores. Debido a ello, este tipo de redes se han mostrado efectivas para solucionar problemas asociados a todo tipo de secuencias y series temporales [90, 91, 92, 93] incluyendo el procesamiento de lenguaje natural [94, 95, 96] y la traducción automática de idioma [97], tomándose en estos casos la entrada de datos como una secuencia de palabras o sonidos a ser procesada por la red.

Por su parte, la sección 6.1 detalla como un FQDN puede ser interpretado como una cadena de texto y por tanto como una secuencia de caracteres, por lo que es un tipo de problema de clasificación de secuencias que se asemeja al tipo de problemas para los cuales las redes neuronales recurrentes fueron diseñadas. Por este motivo se elige el uso de una red neuronal recurrente LSTM como modelo de aprendizaje automático, que además es un modelo que ya ha sido usado con éxito anteriormente para la resolución de problemas de este tipo [69].

La figura 6.1 muestra el modelo de aprendizaje profundo diseñado para la detección de nombres de dominios generados por DGA, que está formado por dos capas. En primer lugar y como fase previa, los datos de entrada se codifican adecuadamente empleando el método *One-Hot Encoding* descrito en la sección 6.1. La primera capa de la red contiene una célula LSTM, mientras que la segunda capa es una capa densa (*Dense*) que combina todos los elementos del vector de salida de la capa LSTM para obtener el valor final de la clasificación.

Este diseño es similar al propuesto por Woodbridge et al. [69] pero modificado para reducir la cantidad de parámetros entrenables y hacer más eficiente el modelo. Esto se consigue por medio de la eliminación de la capa de *embeddings* en favor de una codificación estática *One-Hot Encoding* y de la reducción del tamaño de la célula LSTM con respecto a la célula de tamaño 128 empleada en [69].

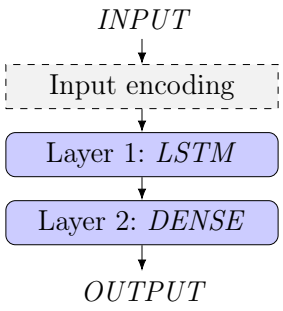


Figura 6.1: Diseño para detección de DGA con LSTM.

En este trabajo se usa la implementación de LSTM de la librería Keras [47], disponible para el lenguaje de programación Python. Esta librería proporciona una capa de abstracción y facilita el uso del ecosistema *Tensorflow*, que implementa los modelos de aprendizaje automático utilizados [98].

6.3. Experimentos

Para la evaluación de esta aproximación se diseñan varios experimentos consistentes en modificar ciertos hiperparámetros del modelo del siguiente modo:

Tamaño de la célula LSTM. Se emplean diferentes tamaños de célula LSTM para todos los experimentos. En concreto, se emplean tamaños de 5, 15, 25 y 50 unidades. El tamaño de la célula tiene un impacto significativo ya que determina el tamaño de la red en cuanto a parámetros entrenables. Redes más pequeñas pueden ser entrenadas con un conjunto de datos de entrenamiento menor, de forma mucho más rápida y con una mejor generalización. Por otro lado, redes de mayor tamaño son capaces de resolver problemas de clasificación más complejos de forma más precisa, pero por contra requieren el uso de conjuntos de datos de entrenamiento más grandes para entrenar adecuadamente su mayor cantidad de parámetros entrenables [99]. Debido a esto no se consideran tamaños de célula superiores a 50, puesto que pruebas preliminares mostraron que el conjunto de datos de entrenamiento empleado en este trabajo no era suficientemente grande como para entrenar redes mayores.

Funciones de activación. Se prueban las diferentes opciones de funciones de activación que permite Keras [100], librería empleada para la construcción de las redes neuronales en esta tesis. La implementación de LSTM de la librería Keras permite parametrizar dos tipos de funciones de activación. Una de ellas consiste en la función de activación de las puertas, pero es excluida de este estudio ya que las pruebas preliminares indicaron que otro tipo de funciones de activación distintas a la función por defecto provoca que el modelo tenga una respuesta inconsistente. Por este motivo se decide mantener la función de activación por defecto para todos los experimentos (en concreto, la función sigmoide). Por otro lado, otro parámetro

controla la función de activación empleada en el procesamiento del dato de entrada, que recuérdese es usada posteriormente para obtener el estado de la célula y su salida. En este caso se consideran únicamente el uso de las funciones de activación de tangente hiperbólica *htan* y *softsign*. Otras funciones de activación se han excluido del estudio final ya que, al igual que con el anterior hiperparámetro, producen que el modelo responda de forma inestable.

Tamaño del lote de datos. Se emplean dos alternativas de tamaño de lote para el proceso de entrenamiento: de 10 y de 100 elementos. Durante este proceso, el conjunto de datos se procesa por la red neuronal y se obtiene un resultado que es comparado con el resultado esperado para dicha entrada y a partir de esta información se ejecuta el mecanismo de propagación hacia atrás y tiene lugar el ajuste de los pesos (o parámetros entrenables) en los que se basa el aprendizaje. El tamaño del lote, que determina cuántas entradas son proporcionadas a la red antes de ejecutar el mecanismo de propagación hacia atrás, tiene ciertas repercusiones en cuanto a la capacidad y estabilidad del aprendizaje que se mencionan en la sección 2.3.4, y por tanto son un hiperparámetro relevante en la construcción del modelo.

Otros hiperparámetros del modelo permanecen estáticos por decisiones de diseño. Por ejemplo, se establece un límite máximo de 1 000 épocas durante el entrenamiento, que se consideran suficientes para que el aprendizaje tenga lugar. Además, dado que la mejor exactitud no tiene por qué obtenerse necesariamente en la última de las épocas, se establece un mecanismo para almacenar la configuración de la red que proporciona los mejores resultados sin importar en qué época del proceso de entrenamiento se encuentre. Por otro lado, se establece una paciencia de 100 épocas, lo cual quiere decir que el entrenamiento se termina si los resultados no mejoran durante 100 épocas consecutivas, a pesar de no haber alcanzado las 1 000 épocas establecidas como máximo. Esta configuración permite detener el entrenamiento en un punto en el que la red pueda haber ya encontrado la mejor configuración posible dados unos determinados valores de inicialización de sus pesos.

Con el fin de evitar que una inicialización afortunada o desafortunada de los valores de los pesos de la red neuronal no permita un análisis adecuado de los resultados, se establece que cada experimento debe repetirse al menos en 10 ocasiones, con lo que la inicialización de dichos valores es diferente cada vez y resulta más sencillo hacer una interpretación de los resultados.

La verificación de este modelo se realiza mediante una estrategia *Hold-Out* [101]. En primer lugar se separa el conjunto de datos de forma aleatoria en dos conjuntos de datos menores de igual tamaño. Uno de ellos es empleado para la comprobación final de los resultados y recibe el nombre de *conjunto de datos de test*. Por otro lado, la otra mitad del conjunto de datos recibe el nombre de *conjunto de datos de trabajo* y es a su vez dividido de forma aleatoria en otros dos subconjuntos de datos. El primero de ellos representa el 80 % del tamaño del conjunto de trabajo y se emplea para entrenar el modelo, por lo que recibe el nombre de *conjunto de datos de entrenamiento*. Finalmente, el 20 % restante es el *conjunto de datos de validación* y se emplea para la selección de aquella configuración de la red que proporcione mejores resultados de clasificación. Esta aproximación permite la detección de un sobreentrenamiento en caso de que este se produzca, ya que los resultados se prueban empleando un conjunto de datos que no se usa ni en el entrenamiento ni en la elección del modelo, lo que permite comprobar que el modelo sigue proporcionando una buena clasificación (es decir, que generaliza de forma adecuada).

6.4. Discusión de resultados

Tamaño de lote	Cantidad de unidades			
	5	15	25	50
10	0.9595	0.9655	0.9648	0.5951
100	0.9592	0.9655	0.9669	0.9678

Tabla 6.2: Mejor exactitud usando la función de activación tangente hiperbólica.

En primer lugar se analiza cómo el empleo de diferentes tamaños de lote impactan en el rendimiento de la red. La tabla 6.2 muestra que el tamaño de lote no impacta de forma significativa en redes de menor tamaño, mientras que su relevancia se vuelve significativa al usar redes de un mayor tamaño. El empleo de tamaños de lote menores significa que los pesos de la red son actualizados con más frecuencia, en función del error obtenido por un subconjunto pequeño de los datos de entrenamiento. Por el contrario, un mayor tamaño de lote implica una actualización más suave de los pesos de la red, ya que cada actualización está calculada en base a los errores para un subconjunto de datos de entrenamiento mayor. En concreto se observa que para redes de tamaño 50 y tamaño de lote de 10 elementos la exactitud de la red cae de forma muy significativa. La figura 6.2 muestra cómo el uso de un tamaño de lote menor impacta en el aprendizaje, haciendo que la exactitud alcanzada mejore o empeore sustancialmente en cada época. Esta falta de estabilidad al usar tamaños de lote pequeños con redes LSTM de mayor tamaño implica que existe una alta probabilidad de que el experimento no obtenga buenos resultados de clasificación. Este hecho se observa en la figura 6.3, donde el ratio de experimentos satisfactorios (aquellos que alcanzan una exactitud superior al 90 %) decrece al aumentar el tamaño de la red, llegando a alcanzar un valor de cero para un tamaño de red 50.

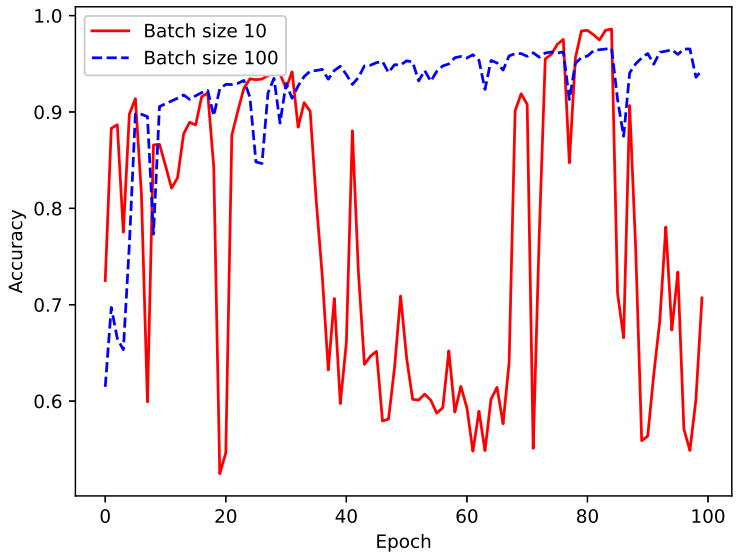


Figura 6.2: Evolución de la exactitud durante el entrenamiento para LSTM de tamaño 25. Tamaño de lote 10 (rojo) y 100 (azul).

La sección 6.2 menciona cómo la mayoría de funciones de activación soportadas por Keras resultaron en un comportamiento inestable de la red, excepto en el caso de las

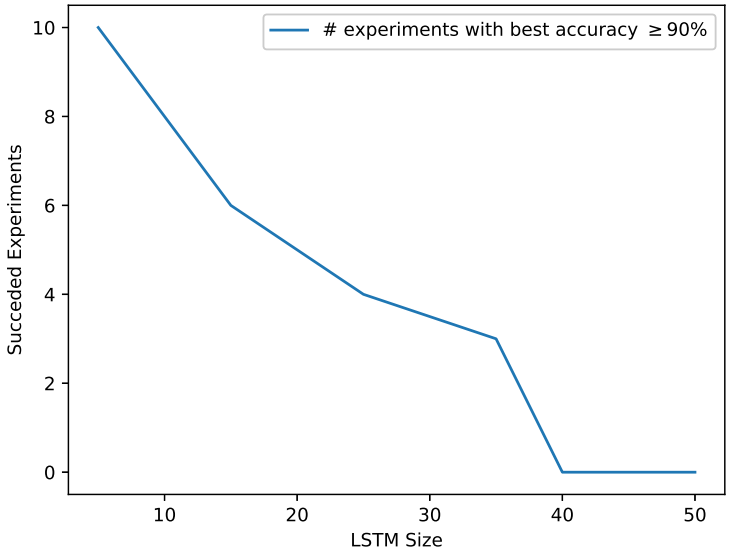


Figura 6.3: Cantidad de experimentos con exactitud por encima de 90 %.

funciones tangente hiperbólica y *softsign*. Para evaluar los resultados de la red empleando cada una de estas funciones se emplea la misma configuración de red (mismo tamaño de la célula LSTM y mismo tamaño de lote) y se obtiene el mejor valor de exactitud de las 10 repeticiones del experimento. Para la obtención de dicho dato de exactitud se emplea el conjunto de datos de validación.

Función de activación	**Cantidad de unidades**			
	5	**15**	**25**	**50**
htan	0.9592	0.9655	0.9669	0.9678
softsign	0.9594	0.9652	0.9652	0.9681

Tabla 6.3: Mejor exactitud empleando tamaño de lote 100.

La tabla 6.3 muestra los resultados de estas pruebas, en los que se aprecia que el uso de una u otra función de activación no impacta significativamente en la exactitud de la mayoría de las configuraciones de red. Sin embargo, la función de activación *softsign* mejora la exactitud del modelo con respecto a la función de activación tangente hiperbólica, especialmente para aquellas redes de mayor tamaño. En base a estos resultados, se decide emplear *softsign* como función de activación para el resto de los experimentos.

Al analizar los resultados teniendo en cuenta todos los tamaños de célula incluidos en los experimentos, se observa también que la exactitud aumenta al aumentar el tamaño de la misma. Una red con un tamaño de célula de 50 unidades proporciona mejores resultados en el conjunto de validación que el resto de redes de menor tamaño. Sin embargo, la diferencia de exactitud entre redes con células de tamaño 50 y de tamaño 25 es muy leve, a pesar que el tamaño de la red y la cantidad de pesos a entrenar es mucho mayor.

Aunque la mejor exactitud de los experimentos realizados con una configuración dada mejora al aumentar el tamaño de la red, la exactitud promedio de todas las repeticiones del experimento desciende, en especial al emplear tamaños de lote pequeños. Este comportamiento está motivado porque el uso de lotes de pequeño tamaño aumenta las

probabilidades de producir grandes cambios en los pesos de la red y esto hace empeorar la exactitud en promedio cuando la tasa de aprendizaje es demasiado grande [102].

La configuración que proporciona los mejores resultados en cuanto a exactitud en la clasificación es la formada por la célula LSTM de tamaño 50, con la función de activación *softsign* y un tamaño de lote de 10 elementos. Esta configuración se ha verificado empleando el conjunto de test, obteniéndose la matriz de confusión representada en la tabla 6.4, en la que las filas muestran la clase real de las muestras, y las columnas representan la predicción realizada por el modelo.

	Estimación	
	"MALWARE"	**"CLEAN"**
Clase "MALWARE"	15731	391
Clase "CLEAN"	369	15501

Tabla 6.4: Matriz de confusión.

Este modelo tiene una exactitud de 0.9762 y un área bajo la curva (AUC) de 0.9955 en el conjunto de datos de test. La métrica AUC permite medir el rendimiento de la aplicación, ya que se basa en el área bajo la curva que representa el número de falsos positivos combinado con el número de falsos negativos. Dado que el valor del AUC es cercano a uno, se considera que el modelo propuesto proporciona buenos resultados para el problema de clasificación estudiado.

En concreto, la TPR es de 97.57 %, lo cual representa la tasa de nombres de dominio generados mediante DGA que son clasificados como tales por el modelo. Por otro lado, la FPR es del 2.33 %, lo cual representa a los nombres de dominio legítimos que son erróneamente clasificados como generados por un DGA. Repitiendo este ejercicio únicamente con los 1 000 primeros dominios de la lista de Alexa, la FPR se reduce hasta el 1.7 % y hasta el 0 % cuando se consideran únicamente los 100 primeros dominios de esa misma lista. Estos resultados resultan muy positivos, ya que las peticiones DNS reales que se encuentran en Internet deben corresponder de forma mayoritaria a las primeras posiciones de la lista de Alexa, por ser esta la lista de dominios más visitados de Internet. Por este motivo la FPR calculada a partir del conjunto de datos puede ser muy inferior si es calculada de forma ponderada con respecto a la cantidad de peticiones de cada uno de los nombre de dominio de la lista.

Por último, se calcula el coeficiente de correlación de Matthews (MCC) como indicador de la calidad de la clasificación específico para clasificaciones entre dos clases, obteniéndose una puntuación de 0.9525. Este valor muestra una correlación cercana a lo que el indicador llama un ajuste perfecto [103]. La tabla 6.5 muestra el resto de métricas calculadas, incluyendo exhaustividad (*recall*) y *Kappa* entre otras.

Exactitud	**AUC**	**TPR**	**FPR**	**Exhaustividad**	**F1-score**	**Kappa**	**MCC**
0.9762	0.9955	0.9757	0.0233	0.9762	0.9762	0.9524	0.9525

Tabla 6.5: Métricas del modelo propuesto.

En cuanto a la comparación de resultados con los obtenidos por otros autores, Woodbridge et al. [69] emplean una red LSTM y un conjunto de datos de cerca de 2 millones de

nombres de dominio, obteniendo un 98 % de acierto en la detección de nombres de dominios generados mediante DGA y una FPR de 0.1 %, con un AUC de 0.9993. Su propuesta es posteriormente reevaluada por Catania et al. [71], obteniéndose una TPR del 94 % y una FPR del 3 %. El modelo de una red 1D-CNN propuesto por Catania et al. obtiene un TPR del 97 % y un FPR del 0.7 %. En la tabla 6.6 se encuentran las métricas de cada uno de los trabajos mencionados en el capítulo 3.

Trabajo	Método	Exactitud	TPR	FPR	AUC	Precisión	Exhaustividad
Woodbridge et al. [69]	LSTM	–	0.9800	0.0010	0.9993	0.9942	0.9937
Tran et al. [70]	LSTM.MI	–	–	–	–	0.9842	0.9842
Catania et al. [71]	CNN	–	0.9700	0.0070	–	–	–
Liu et al. [73]	RCNN	0.9236	–	–	0.9539	0.9236	0.8955
Yang et al. [74]	HDNN	0.9773	–	–	–	0.9793	0.9751
Selvi et al. [104]	RF	0.9873	0.9859	–	–	0.9859	0.9888
Selvi et al. [105]	LSTM	0.9762	0.9757	0.0233	0.9955	0.9770	0.9762

Tabla 6.6: Métricas de cada uno de los trabajos relacionados.

Cabe destacar que los datos mostrados en la tabla 6.6 no han sido obtenidos de reproducir los experimentos de cada uno de los autores con el mismo conjunto de datos sino que recoge las métricas compartidas en sus respectivas publicaciones. Los motivos de tomar esta aproximación son varios. En primer lugar, no todos los autores han publicado suficiente información como para reproducir sus experimentos de forma precisa. En muchos casos su conjunto de datos no ha sido publicado y en otros no se comparten detalles sobre la implementación, por lo que hubiera sido necesario tomar ciertas decisiones de diseño en cuanto a los hiperparámetros de cada modelo que muy probablemente hubieran impactado en el resultado, haciendo la comparación poco efectiva. En segundo lugar, el estudio realizado en este capítulo se ha hecho bajo el requisito de ser un modelo que pueda trabajar con el mismo conjunto de datos con el que se entrena y evalúa el modelo propuesto en el capítulo 5. Este requisito es debido a que uno de los objetivos de esta tesis es determinar si la aproximación mediante aprendizaje profundo puede ser utilizada para evitar el costoso proceso de ingeniería de características y aún así obtener unos resultados cercanos o incluso mejores. Por este motivo, el modelo propuesto está específicamente diseñado para ser funcional con conjuntos de datos de tamaño reducido, algo que no ocurre en el resto de trabajos propuestos por otros autores. Como consecuencia, el uso de un conjunto de datos reducido en los modelos de otros autores presumiblemente no proporcionará tan buenos resultados como empleando los conjuntos de datos para los que fueron diseñados.

Según la información proporcionada por Catania et al. [71], la propuesta de red LSTM de Woodbridge et al. [69] presenta peores resultados que la propuesta de red LSTM simplificada presentada en esta tesis para su uso con un conjunto de datos reducido. Así mismo, la propuesta aquí planteada consigue una TPR similar a la obtenida por la red 1D-CNN propuesta por Catania et al. Sin embargo, la FPR obtenida por Catania et al. es mucho mejor que la obtenida aquí. Esto se debe a que el umbral de la frontera de decisión del modelo de Catania et al. está cambiado a 0.9 en lugar del valor por defecto de 0.5. Esta configuración hace que el modelo clasifique un nombre de dominio como generado por DGA solo si la certeza de dicha clasificación es superior al 90 %, lo cual tiene como consecuencia una menor FPR y una mayor tasa de verdaderos negativos, pero a su vez una menor TPR y una mayor tasa de falsos negativos. Este tipo de aproximación tiene sentido en escenarios en los que la petición de resolución del nombre de dominio fuera

bloqueada automáticamente, ya que es preferible no bloquear nunca un dominio legítimo incluso si para ello algunos dominios maliciosos no son bloqueados. Una aproximación contraria es preferible en otro tipo de escenarios en los que el modelo es usado para una detección fuera de línea, donde es mejor identificar todos los dominios maliciosos incluso si ello implica que se clasifiquen como maliciosos algunos nombres de dominio legítimos, ya que ellos no son en ningún caso bloqueados de forma automática. Por este motivo, en esta propuesta se decide mantener dicho umbral en su configuración por defecto, para que sea fácilmente aplicable a ambos escenarios, si bien éste es un parámetro que se puede ajustar posteriormente en función del escenario concreto al que se vaya a aplicar el modelo.

Otra de las grandes diferencias es debida al conjunto de datos empleado. Catania et al. emplean un conjunto de datos desbalanceado compuesto de un millón de nombres de dominio legítimos y cerca de dos millones de nombres de dominios generados por DGA. Esto conduce a una TPR mayor, ya que la clase que representa una clasificación positiva es mucho mayor en el conjunto de entrenamiento. Por el contrario, el conjunto de datos empleado en los experimentos de esta tesis está balanceado y es mucho menor (64 000 nombres de dominio), debido a las restricciones ya mencionadas con anterioridad.

6.5. Interpretabilidad del modelo

El análisis de modelos basados en redes neuronales como LSTM no resulta tan sencillo como el análisis de un modelo basado en *Random Forest* u otros tipos de modelo basado en árboles. Por este motivo, para el análisis de los resultados con la aproximación mediante aprendizaje profundo se han empleado técnicas de XAI, explicadas en más detalle en la sección 2.3.6. En el caso de este trabajo se realiza el análisis mediante un método independiente al modelo que puede ser empleado de forma posterior al entrenamiento como es LIME [62].

Para ilustrar el funcionamiento del modelo de una forma empírica se repite el mismo análisis que el realizado para *Random Forest*. Se selecciona el dominio legítimo "facebook.com" y el dominio malicioso "wxhyqqrbouru.pw" del conjunto de datos para realizar un ejemplo de clasificación. Para una mejor comprensión del análisis conviene destacar que, dado que las células LSTM pueden aprender relaciones entre un carácter y los caracteres anteriores en cualquier posición, la puntuación y color mostrados por la interfaz de LIME no aplican únicamente a un carácter en sí mismo, sino a una relación indeterminada entre ese carácter y los que fueron procesados con anterioridad.

De este modo, la figura 6.4 muestra el resultado de aplicar LIME para el nombre de dominio "facebook.com", en el que se observa que la mayor parte de sus caracteres muestran una tendencia a la clasificación como "CLEAN" (en color azul), mientras que uno de los caracteres 'o' y el carácter 'f' muestran una leve tendencia a la clasificación como "MALWARE" (en color naranja), si bien esta es claramente minoritaria. La intensidad de cada uno de los colores mostrada en la parte derecha de la figura representa la importancia que ese carácter tiene en el resultado de la clasificación. Esta importancia también se ve representada numéricamente en la parte izquierda de la figura, mostrándose los caracteres de forma ordenada según su importancia para la clasificación. Por el contrario, la figura 6.5 muestra cómo prácticamente la totalidad de los caracteres del dominio "wxhyqqrbouru.pw" tienen una clara tendencia a la clasificación como "MALWARE". Extendiendo este análisis, la tabla 6.7 muestra cómo el peso de los caracteres en la clasificación cambia al substituir ciertos caracteres. Esta tabla emplea el mismo código de colores que LIME y marca en

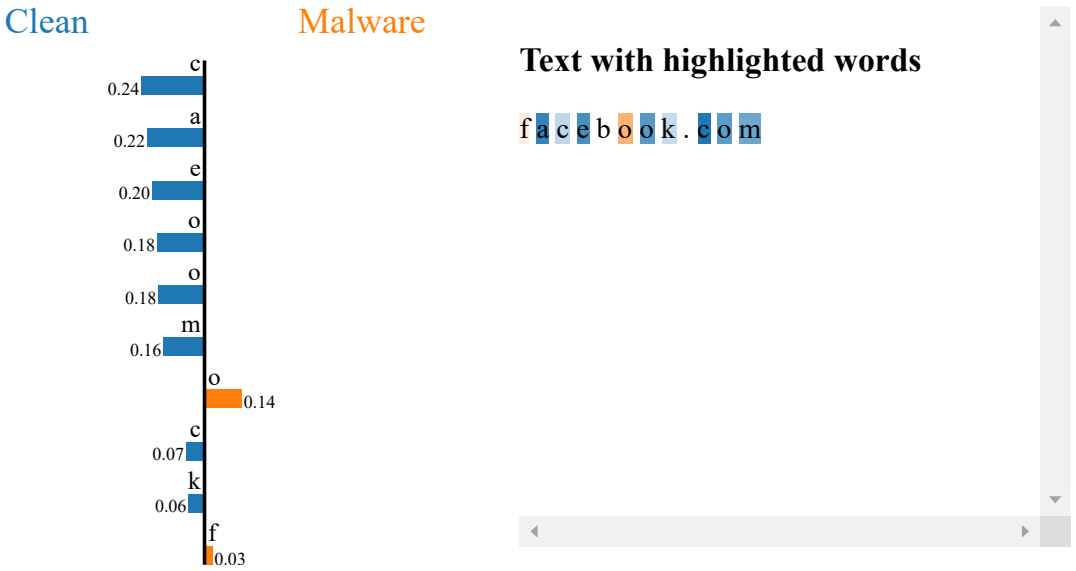


Figura 6.4: Resultado de LIME para “facebook.com”.

Figura 6.5: Resultado de LIME para “wxhyqqrbouru.pw”.

negrita los caracteres modificados con respecto a la línea superior, así como la clasificación que cada una de las líneas recibe del modelo. En ella se puede observar cómo aumenta la cantidad de caracteres con un peso hacia la clasificación como “CLEAN” al introducir vocales que interrumpen la larga secuencia de consonantes, si bien esta clasificación no llega a producirse para el dominio completo. Otras modificaciones posteriores como el

cambio del dominio ".pw" por ".es" o el cambio de la subcadena "bouru" por "boura", sí que finalmente producen un cambio en la clasificación, haciendo que el nombre de dominio resultante pase a ser clasificado como "CLEAN".

w	x	h	y	q	q	r	b	o	u	r	u	.	p	w	MALWARE
w	**a**	h	**i**	q	**u**	r	b	o	u	r	u	.	p	w	MALWARE
w	a	h	i	q	u	**e**	b	o	u	r	u	.	p	w	MALWARE
w	a	h	i	q	u	e	b	o	u	r	**a**	.	p	w	CLEAN
w	a	h	i	q	u	e	b	o	u	r	**u**	.	**e**	**s**	CLEAN

Tabla 6.7: Modificaciones de "wxhyqqrbouru.pw".

nosegrain.net	onlletgodftxsels.ru	dzrecwimln.com
blaspsacerpotest.com	cdpsad.com	earnestinelongstaff.net
facenine.net	chiefdinner.net	tosxxoa.com
	persitretinere.com	

Tabla 6.8: Ejemplo de falsos negativos con LSTM.

Con respecto a los falsos negativos, la tabla 6.8 muestra un ejemplo de 10 de ellos extraídos de forma aleatoria. La mayor parte de ellos, como "chiefdinner.net" o "facenine.net", han sido generados mediante un DGA basado en diccionario. Al igual que sucede con los falsos positivos generados por la aproximación mediante *Random Forest*, estos falsos positivos son generados a partir de palabras en lenguaje humano y no de una forma completamente aleatoria, por lo que tienen una forma visualmente similar a un dominio legítimo y por lo tanto su detección resulta más problemática que con dominios de otros tipos de DGAs.

doubleclick.net	slideshare.net
slickdeals.net	adplxmd.com
secureserver.net	themeforest.net
trackingclick.net	daikynguyenvn.com
prjcq.com	bookmyshow.com
seesaa.net	inquirer.net
uploaded.net	torcache.net
youjizz.com	fanfiction.net
commentcamarche.net	

Tabla 6.9: Falsos positivos en "Alexa's Top 1000" con LSTM.

En cuanto a los falsos positivos, la tabla 6.9 muestra los 17 nombres de dominios de la lista de "Alexa's Top 1000" detectados como tales, mientras que en la lista de "Alexa's Top 100" se producen cero falsos positivos.

El análisis con LIME de la figura 6.6, muestra como mayoritariamente los caracteres que forman el nombre de dominio "doubleclick.net" tienden a ser favorables a su clasificación como "CLEAN" pero el uso del TLD "net" cambia esta tendencia y hacer que sea clasificado como "MALWARE". Lejos de ser un hecho puntual, gran parte de los falsos positivos de la tabla 6.9 emplean el dominio ".net", por lo que el uso de este TLD

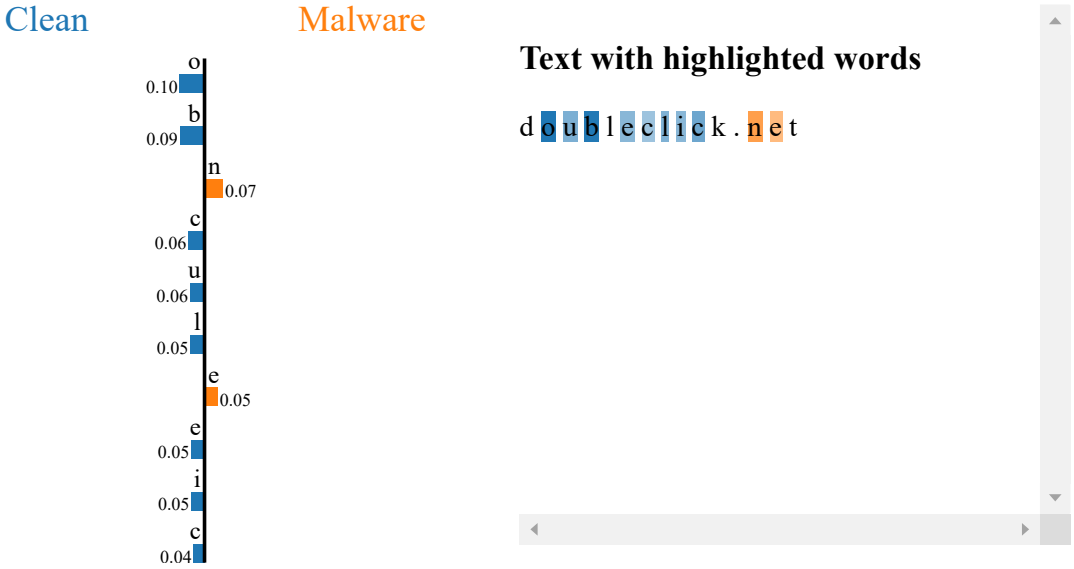


Figura 6.6: Resultado de LIME para “doubleclick.net”.

puede resultar una característica importante para la clasificación de estos dominios como “MALWARE”. Esto es debido al amplio uso que el malware hace de este TLD, ya que en el conjunto de datos hay 7 110 dominios maliciosos “.net” frente a tan solo 1 772 dominios legítimos de ese mismo TLD. En concreto, 12 de los 17 falsos positivos emplean dominios con TLD “.net” y todos ellos pasan a ser clasificados como legítimos al cambiar su dominio de “.net” a “.com”.

De entre los restantes 5 falsos positivos, “adplxmd.com” y “prjcq.com” (servicios de publicidad on-line) están formados por siglas y por tanto tienen un aspecto similar a un dominio generado algorítmicamente, por lo que es lógico que se produzca dicho error de clasificación al evaluarse únicamente el propio nombre de dominio. Por otro lado, dominios como “bookmyshow.com”, “daikynguyenvn.com” o “youjizz.com” muestran secuencias de caracteres que son identificados con más alta probabilidad de existir en nombres de dominio generados mediante DGA, como la secuencia “zz”, “kyng” o “oo”. En este sentido, la repetición del mismo carácter de forma consecutiva parece ser una característica que el modelo emplea para detectar un nombre de dominio como generado por un DGA. No obstante, dicha característica no resulta ser una condición suficiente, ya que LIME no identifica esta misma secuencia como relevante asociada a la clasificación como “MALWARE” en ciertos dominios que también la usan, como se puede apreciar en la figura 6.7, en la que se muestra el resultado de LIME para “google.com”.

A pesar de la ayuda de las técnicas de inteligencia artificial explicable, el uso de redes neuronales recurrentes dificulta la visualización de la explicación, ya que la importancia en la clasificación de un carácter concreto no depende únicamente de dicho carácter. Además del propio carácter, su importancia depende de su posición en el nombre del dominio y del resto de caracteres que fueron procesados previamente. Este comportamiento se ilustra en la tabla 6.10, que tiene el mismo formato de la tabla 6.7. En esa tabla se muestra el resultado de LIME al evaluar el dominio “bookmyshow.com” añadiendo carácter a carácter

Figura 6.7: Resultado de LIME para “google.com”.

para observar los cambios en los resultados. Cada una de las entradas resultantes de ir añadiendo estos caracteres uno a uno se representa como una línea en la tabla, omitiendo aquellos pasos que no producen ningún cambio representativo. En esta tabla se observa como la secuencia “oo” no es tomada por el modelo como característica de un nombre de dominio malicioso hasta que al añadir sucesivos caracteres esta importancia aumenta, llegando a cambiar la clasificación del nombre del dominio.

b	o	o										CLEAN
b	o	o	k									CLEAN
b	o	o	k	m	y	s	h	o				CLEAN
b	o	o	k	m	y	s	h	o	w			MALWARE
b	o	o	k	m	y	s	h	o	w	.	c	MALWARE

Tabla 6.10: Resultado de LIME para “bookmyshow.com”, carácter a carácter.

En cuanto a la velocidad de la clasificación, este modelo es capaz de clasificar el conjunto de datos completo, compuesto de 64 000 nombres de dominio, en 59 segundos. Para tomar esta medida no es necesario dividir el conjunto de datos de ningún modo, ya que el objetivo es medir la velocidad en la clasificación y no su exactitud.

6.6. Herramientas desarrolladas

Todo el código empleado para la construcción de los modelos descritos en este capítulo ha sido publicado de forma abierta en GitHub [19]. Este repositorio contiene diferentes *Jupyter Notebooks* (código en Python) que emplean librerías como *sklearn*, *keras* y *tensorflow* para implementar el clasificador LSTM.

Además de publicar todo el código que implementa el clasificador, con el fin de facilitar la reproducción de los resultados, se ha generado una imagen Docker que está públicamente accesible y que puede ser utilizada para reproducir los experimentos descritos [19].

Capítulo 7

Conclusiones y trabajo futuro

Este capítulo contiene las conclusiones obtenidas de esta tesis doctoral. En él se analizan las fortalezas y debilidades de cada una de las propuestas, y en qué situaciones resulta más ventajosa cada una de ellas. A continuación se proponen diversas opciones para implementar un servidor DNS que empleara las técnicas descritas en esta tesis doctoral, y se sugieren una serie de trabajos futuros. Finalmente se enumeran las publicaciones relevantes derivadas de esta tesis doctoral.

7.1. Conclusiones

Emplear una aproximación de detección basada en el tráfico de red permite detectar el malware al establecer comunicación e interactuar con otros elementos presentes en Internet. Con el fin de dificultar la detección los desarrolladores de malware han ido progresivamente abandonando la estrategia de conectar directamente a direcciones IP fijas de Internet o a nombres de dominio únicos. En su lugar han adoptado estrategias que emplean un nombre de dominio generado algorítmicamente para determinar a qué dirección de Internet deben conectarse.

En esta tesis se han explorado diferentes técnicas de detección de este tipo de algoritmos mediante aprendizaje automático, empleando como única fuente de características para los modelos el texto del nombre de dominio. A partir de este dato se han empleado y se han extendido diferentes técnicas descritas en la literatura, adaptándolas a las necesidades específicas de este problema y a las limitaciones impuestas como parte de los objetivos de este trabajo.

En concreto, se han estudiado dos aproximaciones diferentes. La primera de ellas emplea una técnica de aprendizaje no profundo como *Random Forest*. La extracción de características para este modelo se ha realizado tras un proceso de ingeniería de características manual en la que se propone utilizar una serie de características empleadas con anterioridad en la literatura, en combinación con una nueva característica llamada *N-Grams* enmascarados. Esta nueva característica preserva una mayor cantidad de información sobre el nombre de dominio que meramente la información estadística de sus *N-Grams*.

Por otro lado, también se ha empleado otra aproximación mediante aprendizaje profundo, en concreto redes neuronales recurrentes LSTM. En este caso, la extracción automática de características permite evitar cualquier ingeniería de características específica,

más allá de la elección de la estrategia de codificación de los caracteres que forman el nombre del dominio, que en este caso ha sido *One Hot Encoding*.

Los resultados obtenidos de los experimentos realizados permiten concluir que a pesar de que la aproximación mediante Random Forest obtiene mejores resultados de clasificación en el conjunto de datos de test (no usado durante el entrenamiento), es posible diseñar un modelo que obtiene resultados de clasificación muy similares sin la necesidad de realizar el complejo y costoso proceso de ingeniería de características.

Evitar el proceso de ingeniería de características resulta fundamental en muchos campos de aplicación, especialmente cuando es un proceso que debe repetirse de forma frecuente. Esto puede ocurrir en escenarios adversarios como los descritos en [106]. Los desarrolladores de malware pueden además modificar la manera en la que los dominios son generados, de tal modo que las características extraídas por un proceso previo de ingeniería de características dejan de ser relevantes y la tarea de extracción no resulte provechosa. En esos escenarios, las redes LSTM pueden ser entrenadas de nuevo añadiendo la nueva familia de nombres de dominio y aprender de forma automática las características necesarias para seguir produciendo buenos resultados de clasificación.

Además, los resultados obtenidos mediante la aproximación de aprendizaje profundo obtiene una menor cantidad de falsos positivos en la lista de los 1 000 dominios más utilizados de Internet. Esto implica que es probable que los resultados de detección de la red LSTM en un entorno real sean mejores de forma global, ya que estos dominios suelen ser prevalentes en tráfico real al ser los de mayor utilización en Internet.

Las técnicas propuestas en este trabajo tienen como principal fortaleza que han proporcionado buenos resultados de clasificación (97-98 % de acierto en el conjunto de datos de test) empleando únicamente el nombre de dominio como dato original a partir del cual se extraen las características. Este factor presenta una gran ventaja con respecto a otras soluciones en las que es necesario disponer de más información relativa al tráfico DNS completo o la información de registro del dominio, ya que es posible entrenar a los modelos con DGAs que no se encuentran en uso en la actualidad, ampliando la diversidad del conjunto de datos.

Además, las técnicas empleadas han sido capaces de completar el aprendizaje empleando un conjunto de datos mucho más reducido que los empleados por otras técnicas propuestas en la literatura. Por ello el entrenamiento de estos modelos es posible extrayendo un menor número de muestras de DGA por cada familia de algoritmos detectado.

Por contra, las soluciones encontradas tienen debilidades que hacen que no exista una técnica que sea claramente mejor que la otra. En el caso de la aproximación mediante *Random Forest*, el entrenamiento es muy eficiente y proporciona los mejores resultados de entre los experimentos realizados. Además es la técnica que da una respuesta de clasificación más rápida y ofrece las mejores capacidades de explicabilidad. Por el contrario, tiene una mayor cantidad de falsos positivos entre el "Alexa's Top 1000" que, al tratarse de los dominios más empleados en Internet, puede producir el bloqueo de dominios legítimos usados frecuentemente. Este modelo también tiene la debilidad de ser más estático, ya que se basa en una extracción de características diseñada manualmente que puede ser abusada en entornos adversarios en la que los desarrolladores de DGA modifican sus algoritmos para evadir la detección de este modelo.

Por su parte, la aproximación mediante LSTM no resulta ser tan eficiente como la aproximación mediante *Random Forest*. Por la naturaleza de las redes neuronales, el entrenamiento ha sido ejecutado en repetidas ocasiones con una inicialización diferente de

los parámetros entrenables de la red para obtener la configuración más óptima posible, por lo que su tiempo de entrenamiento es mucho mayor que el de *Random Forest*. Aunque *Random Forest* tiene mejor exactitud en el conjunto de datos de test, el modelo LSTM obtiene una menor cantidad de falsos positivos en los dominios del "Alexa's Top 1000". Como principal fortaleza, este modelo cuenta con la extracción de características automática, lo que permite una mayor flexibilidad en entornos adversarios. Aunque los desarrolladores de malware crearan nuevos DGAs que no fueran detectados por el modelo, un reentrenamiento incluyendo los nuevos nombres de dominio en el conjunto de entrenamiento bastaría para aprender a detectar el nuevo algoritmo. Por tanto, este proceso no necesita la intervención humana que requiere *Random Forest* para extraer nuevas características que permitan la detección. Por último, una de las principales debilidades de este modelo es la baja explicabilidad que dificulta detectar qué características en concreto han sido extraídas y empleadas por el modelo incluso empleando técnicas de inteligencia artificial explicable.

En cuanto a su rendimiento, el clasificador LSTM procesa 64 000 nombres de dominio en 59 segundos mientras que el clasificador *Random Forest* procesa esa misma cantidad de nombres de dominio en 2.596 segundos, es decir, el clasificador *Random Forest* es casi 23 veces más rápido que el clasificador LSTM. Este resultado es esperable ya que en general los modelos basados en árboles de decisión son más eficientes que los modelos basados en redes neuronales recurrentes. Esto a priori convierte al clasificador *Random Forest* en un modelo más adecuado para grandes flujos de tráfico que deban ser clasificados.

No obstante, si se asume que en el tráfico de red existe una pequeña cantidad de dominios muy empleados que deben ser clasificados con mucha frecuencia, el clasificador LSTM puede beneficiarse en gran medida de una arquitectura como la mostrada en la figura 7.1, donde se usa una memoria caché que guarda los resultados de los últimos dominios clasificados y que por tanto permita una rápida respuesta ante dominios usados con gran frecuencia, reduciendo estas diferencias de rendimiento.

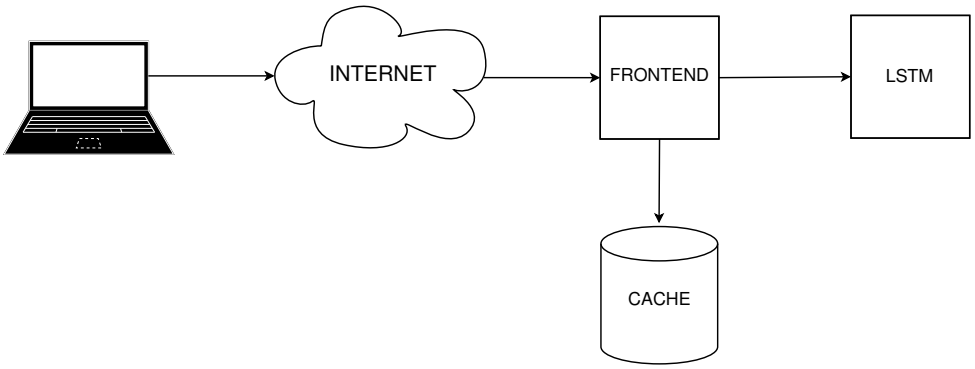


Figura 7.1: Clasificador LSTM con caché.

Además de estos factores, los resultados del clasificador *Random Forest* son más sencillos de explicar que los del clasificador LSTM. Este factor puede ser de gran relevancia para su puesta en funcionamiento en un sistema real, ya que la explicabilidad de los resultados proporciona a los usuarios confianza en la clasificación y facilita la depuración de falsos positivos en el caso de que el titular de un dominio legítimo notifique que su dominio está siendo clasificado incorrectamente como malicioso.

También es posible combinar ambos modelos de forma que el servicio resultante aproveche las ventajas y mitigue las desventajas de cada uno de los modelos. La figura 7.2

muestra una arquitectura en la que una primera capa de software llamada *frontend* recibe la petición a clasificar. Este *frontend* busca en la memoria caché si dicha petición ha sido resuelta con anterioridad, en cuyo caso responde con el resultado, lo cual es muy eficiente ya que se evita tener que volver a realizar la clasificación. En el caso que la información no se encuentre en la memoria caché, el *frontend* solicita la clasificación al modelo *Random Forest*. Este modelo proporciona una rápida respuesta y una fácil explicabilidad en el caso de que se bloquee un dominio legítimo de forma accidental. De esta manera se justifica el motivo del bloqueo y permite ajustar el funcionamiento del modelo. Este resultado se guarda en la memoria caché para futuras peticiones. Posteriormente, tras haber respondido con el resultado de la clasificación, se ejecuta fuera de línea la misma petición de clasificación en el modelo LSTM. A pesar de que esta clasificación tiene un tiempo de respuesta muy superior, no impacta al tiempo de respuesta del servicio por tratarse de una petición fuera de línea. En el caso que el clasificador LSTM determine que el dominio clasificado no es malicioso, se actualiza el valor en la memoria caché para que en sucesivas peticiones el dominio no sea clasificado como malicioso de nuevo. Esta arquitectura combina lo mejor de ambos sistemas. Por un lado el clasificador *Random Forest* proporciona una rápida respuesta y facilita su explicabilidad en el caso de producirse falsos positivos. Por otro lado, el clasificador LSTM verifica fuera de línea los resultados del clasificador *Random Forest* para aprovechar su mejor capacidad para no producir falsos positivos. Finalmente, los nombres de dominio clasificados como maliciosos por LSTM y como legítimos por *Random Forest* pueden ser reportados a analistas humanos para su verificación. De esta manera se puede detectar la presencia de ataques adversarios que requieran modificaciones y actualizaciones en alguno de los clasificadores.

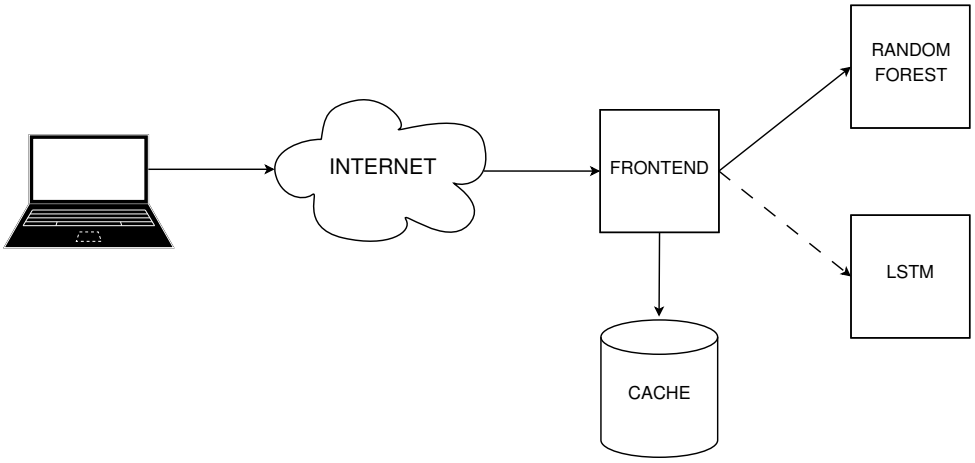


Figura 7.2: Diseño de un clasificador combinado.

7.2. Trabajo futuro

En primer lugar, es interesante implementar todo este trabajo de investigación y desplegarlo en un entorno real y accesible desde Internet en el cual los usuarios y profesionales de la seguridad puedan solicitar la clasificación de determinados dominios. Este servicio puede ofrecer una interfaz DNS configurada para resolver correctamente los nombres que sean detectados como legítimos. También es interesante ofrecer una interfaz REST API para proporcionar más información, como la puntuación que otorga cada uno de los clasificadores con el fin de facilitar las actividades de investigación.

Se considera un futuro trabajo de interés el estudio en detalle del *embedding* empleado en la aproximación mediante LSTM, ya que la técnica utilizada en este trabajo (*One-Hot Encoding*) es mejorable. Para este propósito debe estudiarse la posibilidad de emplear una codificación estática que no esté influenciada por el entrenamiento. Otra posibilidad es emplear una capa adicional que aprenda a codificar cada carácter, si bien esta capa incrementaría la complejidad de la red neuronal y requeriría el uso de un mayor conjunto de datos de entrenamiento.

Otra idea de interés a estudiar es el uso de células GRU en lugar de LSTM, con el fin de determinar si emplear un tipo de técnica diferente que requiere entrenar menos pesos mejora los resultados o mantiene los buenos resultados reduciendo el conjunto de datos de entrenamiento.

Por último, se puede estudiar el uso de redes neuronales recurrentes modificadas para facilitar su explicabilidad.

7.3. Publicaciones relevantes derivadas de esta tesis

Como resultado de la investigación realizada se han publicado los siguientes artículos:

- "*Detection of Algorithmically Generated Malicious Domain Names using Masked N-Grams*". *Expert Systems with Applications* 124 (2019): 156–163 [104]. Clasificado **21/137** (cuartil **Q1**) en la categoría "*Computer Science, Artificial Intelligence*", con factor de impacto **5.452** (2019). Este artículo recoge las contribuciones detalladas en el capítulo 4 y el capítulo 5.
- "*Toward Optimal LSTM Neural Networks for Detecting Algorithmically Generated Domain Names*". *IEEE Access* 9 (2021): 126446-126456 [105]. Clasificado **65/162** (cuartil **Q2**) en la categoría "*Computer Science, Information Systems*", con factor de impacto **3.367** (2020). Este artículo recoge las contribuciones detalladas en el capítulo 6.
- Además de estos dos artículos, actualmente se está preparando un nuevo artículo para las contribuciones detalladas en el capítulo 5 y el capítulo 6 relacionadas con la explicabilidad de los modelos.

Bibliografía

[1] Página web de 29A Labs. [Online; http://vxheaven.org/29a/], 1995. Accedido el 2 de octubre de 2016.

[2] Panda Security: Los virus más famosos de la historia - Viernes 13. [Online; https://www.pandasecurity.com/es/mediacenter/malware/virus-viernes-13/], 2015. Accedido el 18 de diciembre de 2020.

[3] Panda Security: Detalles técnicos del malware Barrotes. [Online; https://www.pandasecurity.com/es/security-info/979/information/Barrotes/], 2020. Accedido el 18 de diciembre de 2020.

[4] Europol. EU Serious and Organised Crime Threat Assessment. techreport, European Union Agency for Law Enforcement Cooperation (Europol), 2013.

[5] AV Test. Malware Statistics & Trends Report. [Online; https://www.av-test.org/en/statistics/malware/], 2020. Accedido el 13 de abril de 2020.

[6] Xabier Ugarte-Pedrero, Mariano Graziano y Davide Balzarotti. A Close Look at a Daily Dataset of Malware Samples. *ACM Trans. Priv. Secur.*, 22(1), Enero 2019.

[7] Kaspersky Lab. Kaspersky Security Bulletin 2014. [Online; http://securelist.com/files/2014/12/Kaspersky-Security-Bulletin-2014-EN.pdf], Diciembre 2014.

[8] Michael Sikorski y Andrew Honig. *Practical malware analysis: the hands-on guide to dissecting malicious software*. O'Reilly, 2012.

[9] Thomas Roccia. Malware packers use tricks to avoid analysis, detection. [Online; https://www.mcafee.com/blogs/enterprise/malware-packers-use-tricks-avoid-analysis-detection/], 2017. Accedido el 7 de octubre de 2021.

[10] Amir Afianian, Salman Niksefat, Babak Sadeghiyan y David Baptiste. Malware dynamic analysis evasion techniques: A survey. *ACM Computing Surveys (CSUR)*, 52(6):1–28, 2019.

[11] Richard Bejtlich. *The Tao of network security monitoring: beyond intrusion detection*. Pearson Education, 2004.

[12] Jarkko Oikarinen y Darren Reed. Internet Relay Chat Protocol. Technical report, Internet Engineering Task Force, 1993. Disponible en https://tools.ietf.org/html/rfc1459.

[13] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Luca Invernizzi, Michalis Kallitsis et al. Understanding the Mirai Botnet. En *26th USENIX security symposium USENIX Security 17*, páginas 1093–1110, 2017.

[14] Cisco Umbrella (anteriormente OpenDNS). The Role of DNS in Botnet Command and Control. [Online; https://es.slideshare.net/courtlandsmith/open-dns-securitywhitepaperdnsroleinbotnets], 2020. Accedido el 26 de diciembre de 2020.

[15] Phillip Porras, Hassen Saïdi y Vinod Yegneswaran. A Foray into Conficker's Logic and Rendezvous Points. En *Proceedings of the 2nd USENIX Conference on Large-scale Exploits and Emergent Threats: Botnets, Spyware, Worms, and More*, LEET'09, Berkeley, CA, USA, 2009. USENIX Association.

[16] Manos Antonakakis, Roberto Perdisci, Yacin Nadji, Nikolaos Vasiloglou, Saeed Abu-Nimeh, Wenke Lee y David Dagon. From Throw-Away Traffic to Bots: Detecting the Rise of DGA-Based Malware. En Tadayoshi Kohno, editor, *Proceedings of the 21th USENIX Security Symposium, Bellevue, WA, USA, August 8-10, 2012*, páginas 491–506. USENIX Association, 2012.

[17] Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador García, Sergio Gil-López, Daniel Molina y Richard Benjamins. Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion*, 58:82–115, 2020.

[18] Jose Selvi. Docker image for the paper: Jose Selvi, Ricardo J Rodríguez y Emilio Soria-Olivas (2019). Detection of algorithmically generated malicious domain names using masked N-grams. [Online; https://github.com/jselvi/docker-r-masked-ngrams], 2019. Accedido el 8 de agosto de 2021.

[19] Jose Selvi. Docker image for the paper: Jose Selvi, Ricardo J Rodríguez y Emilio Soria-Olivas (2021). Toward Optimal LSTM Neural Networks for Detecting Algorithmically Generated Domain Names. [Online; https://github.com/jselvi/docker-lstm-dga], 2021. Accedido el 8 de agosto de 2021.

[20] Cisco Networks. Página web de Snort. [Online; https://www.snort.org], 2021. Accedido el 30 de abril de 2021.

[21] Emerging Threats. Trojan Rules (Open License). [Online; https://rules.emergingthreats.net/open/snort-2.9.0/rules/emerging-trojan.rules], 2021. Accedido el 30 de abril de 2021.

[22] Marc Norton. Optimizing pattern matching for intrusion detection. *Sourcefire, Inc., Columbia, MD*, 2004.

[23] Timothy Shimeall y Jonathan Spring. *Introduction to information security: a strategic-based approach*. Newnes, 2013.

[24] ENISE. Encrypted Traffic Analysis. *Sourcefire, Inc., Columbia, MD*, 2021. Accedido el 30 de abril de 2021.

[25] Roy Arends, Rob Austein, Matt Larson, Dan Massey y Scott Rose. DNS Security Introduction and Requirements. Technical report, Internet Engineering Task Force, 2005. Disponible en https://tools.ietf.org/html/rfc4033.

[26] Guy Bruneau. DNS Sinkhole. *SANS Institute Reading Room*, 2010.

[27] Aditya K Sood y Sherali Zeadally. A taxonomy of domain-generation algorithms. *IEEE Security & Privacy*, 14(4):46–53, 2016.

[28] John Viega. Practical random number generation in software. En *Proceedings of the 19th Annual Computer Security Applications Conference*, páginas 129–140. IEEE, 2003.

[29] Johannes Bader. Some results of my DGA reversing efforts. [Online; https://github.com/baderj/domain_generation_algorithms], 2016. Accedido el 2 de octubre de 2016.

[30] Daniel Plohmann, Khaled Yakdan, Michael Klatt, Johannes Bader y Elmar Gerhards-Padilla. A Comprehensive Measurement Study of Domain Generating Malware. En *25th USENIX Security Symposium USENIX Security 16)*, páginas 263–278, 2016.

[31] P. Mockapetris. RFC 1035: Domain Names - Implementation and Specification. Technical report, Internet Engineering Task Force, Noviembre 1987. Disponible en http://www.rfc-editor.org/rfc/rfc1035.txt.

[32] Sreerama K Murthy. Automatic construction of decision trees from data: A multi-disciplinary survey. *Data mining and knowledge discovery*, 2(4):345–389, 1998.

[33] Claude Sammut y Geoffrey I Webb. *Encyclopedia of machine learning*. Springer Science & Business Media, 2011.

[34] Leo Breiman, Jerome H Friedman, Richard A Olshen y Charles J Stone. *Classification and regression trees*. Routledge, 2017.

[35] Gordon V Kass. An exploratory technique for investigating large quantities of categorical data. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 29(2):119–127, 1980.

[36] Earl B Hunt, Janet Marin y Philip J Stone. *Experiments in induction*. Academic Press, New York, 1966.

[37] J Ross Quinlan. Learning efficient classification procedures and their application to chess end games. En *Machine learning*, páginas 463–482. Springer, 1983.

[38] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[39] J Ross Quinlan. *Programs for machine learning*. Kluwer Academic Publishers, 1993.

[40] L. Breiman, J. H. Friedman, R. A. Olshen y C. J. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984.

[41] Trevor Hastie, Robert Tibshirani y Jerome Friedman. *The Elements of Statistical Learning: Data mining, Inference, and Prediction*. Springer, 9ª edición, 2017.

[42] Daniel T Larose y Chantal D Larose. *Discovering knowledge in data: an introduction to data mining*, volumen 4. John Wiley & Sons, 2014.

[43] Miron B. Kursa y Witold R. Rudnicki. Feature Selection with the Boruta Package. *Journal of Statistical Software*, 36(11):1–13, 2010.

[44] R Core Team. R Language Definition. [Online; https://cran.r-project.org/doc/manuals/r-release/R-lang.html], 2017.

[45] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[46] P. J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. Tesis Doctoral, Harvard University, 1974.

[47] François Chollet et al. Keras. https://keras.io, 2015.

[48] David E Rumelhart, Geoffrey E Hinton y Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[49] Ian Goodfellow, Yoshua Bengio y Aaron Courville. *Deep learning*. MIT press, 2016.

[50] Andrej Karpathy. The Unreasonable Effectiveness of Recurrent Neural Networks. [Online; https://karpathy.github.io/2015/05/21/rnn-effectiveness/], 2015. Accedido el 7 de octubre de 2021.

[51] Sepp Hochreiter y Jürgen Schmidhuber. Long Short-Term Memory. *Neural computation*, 9(8):1735–1780, 1997.

[52] Xiangang Li y Xihong Wu. Constructing long short-term memory based deep recurrent neural networks for large vocabulary speech recognition. En *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, páginas 4520–4524. IEEE, 2015.

[53] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk y Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[54] David Gunning, Mark Stefik, Jaesik Choi, Timothy Miller, Simone Stumpf y Guang-Zhong Yang. XAI—Explainable artificial intelligence. *Science Robotics*, 4(37), 2019.

[55] Houtao Deng. Interpreting tree ensembles with intrees. *International Journal of Data Science and Analytics*, 7(4):277–287, 2019.

[56] Lidia Auret y Chris Aldrich. Interpretation of nonlinear relationships between process variables by use of random forests. *Minerals Engineering*, 35:27–42, 2012.

[57] Gabriele Tolomei, Fabrizio Silvestri, Andrew Haines y Mounia Lalmas. Interpretable predictions of tree-based ensembles via actionable feature tweaking. En *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, páginas 465–474, 2017.

[58] Leila Arras, Grégoire Montavon, Klaus-Robert Müller y Wojciech Samek. Explaining recurrent neural network predictions in sentiment analysis. *arXiv preprint arXiv:1706.07206*, 2017.

[59] Andrej Karpathy, Justin Johnson y Li Fei-Fei. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*, 2015.

[60] Edward Choi, Mohammad Taha Bahadori, Joshua A Kulas, Andy Schuetz, Walter F Stewart y Jimeng Sun. Retain: An interpretable predictive model for healthcare using reverse time attention mechanism. *arXiv preprint arXiv:1608.05745*, 2016.

[61] Scott Wisdom, Thomas Powers, James Pitton y Les Atlas. Interpretable recurrent neural networks using sequential sparse recovery. *arXiv preprint arXiv:1611.07252*, 2016.

[62] Marco Tulio Ribeiro, Sameer Singh y Carlos Guestrin. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. En *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, páginas 1135–1144, 2016.

[63] Leyla Bilge, Engin Kirda, Christopher Kruegel y Marco Balduzzi. EXPOSURE: Finding Malicious Domains Using Passive DNS Analysis. En *Proceedings of the Network and Distributed System Security Symposium, (NDSS 2011)*, 2011.

[64] Leyla Bilge, Sevil Sen, Davide Balzarotti, Engin Kirda y Christopher Kruegel. Exposure: A Passive DNS Analysis Service to Detect and Report Malicious Domains. *ACM Trans. Inf. Syst. Secur.*, 16(4):14:1–14:28, Abril 2014.

[65] Ian H. Witten, Eibe Frank y Mark A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3ª edición, 2011.

[66] Stefano Schiavoni, Federico Maggi, Lorenzo Cavallaro y Stefano Zanero. Phoenix: DGA-Based Botnet Tracking and Intelligence. En *Proceedings of the 11th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, páginas 192–211, Cham, 2014. Springer International Publishing.

[67] Alexa, an Amazon Company. Alexa Top Sites. [Online; http://www.alexa.com/topsites], 2020. Accedido el 27 de abril de 2020.

[68] Pedro Marques da Luz. Botnet Detection Using Passive DNS. Tesis de master, Department of Computing Science, Radboud University Nijmegen, 2013.

[69] Jonathan Woodbridge, Hyrum S. Anderson, Anjum Ahuja y Daniel Grant. Predicting Domain Generation Algorithms with Long Short-Term Memory Networks. *CoRR*, abs/1611.00791, 2016.

[70] Duc Tran, Hieu Mac, Van Tong, Hai Anh Tran y Linh Giang Nguyen. A LSTM based framework for handling multiclass imbalance in DGA botnet detection. *Neurocomputing*, 275:2401–2413, 2018.

[71] Carlos Catania, Sebastian García y Pablo Torres. Deep Convolutional Neural Networks for DGA Detection. En *Argentine Congress of Computer Science*, páginas 327–340. Springer, 2018.

[72] Siwei Lai, Liheng Xu, Kang Liu y Jun Zhao. Recurrent convolutional neural networks for text classification. En *Proceedings of the AAAI Conference on Artificial Intelligence*, volumen 29, 2015.

[73] Zhanghui Liu, Yudong Zhang, Yuzhong Chen, Xinwen Fan y Chen Dong. Detection of Algorithmically Generated Domain Names Using the Recurrent Convolutional Neural Network with Spatial Pyramid Pooling. *Entropy*, 22(9):1058, 2020.

[74] Luhui Yang, Guangjie Liu, Yuewei Dai, Jinwei Wang y Jiangtao Zhai. Detecting Stealthy Domain Generation Algorithms Using Heterogeneous Deep Neural Network Framework. *IEEE Access*, 8:82876–82889, 2020.

[75] Sandro Skansi. *Introduction to Deep Learning: From Logical Calculus to Artificial Intelligence*. Springer, Cham, 2018.

[76] VirusTotal. Statistics - VirusTotal. [Online; https://www.virustotal.com/en/statistics/], 2020. Accedido el 10 de abril de 2020.

[77] C. Rossow, C. J. Dietrich, C. Grier, C. Kreibich, V. Paxson, N. Pohlmann, H. Bos y M. v. Steen. Prudent Practices for Designing Malware Experiments: Status Quo and Outlook. En *2012 IEEE Symposium on Security and Privacy*, páginas 65–79, Mayo 2012.

[78] Alexa 1 Million. [Online; http://s3.amazonaws.com/alexa-static/top-1m.csv.zip], 2016. Accedido el 2 de octubre de 2016.

[79] Jose Selvi, Ricardo J Rodríguez y Emilio Soria-Olivas. Dataset from Alexa Top Sites and Johannes Bader's repository. [Online; https://github.com/jselvi/phd/tree/master/dga], 2018. Accedido el 27 de abril de 2020.

[80] C. E. Shannon. A Mathematical Theory of Communication. *The Bell System Technical Journal*, 27(4):623–656, Octubre 1948.

[81] Jeff Markey. Using Decision Tree Analysis for Intrusion Detection: A How-To Guide. Technical report, SANS Institute, Junio 2011.

[82] Sumeet Dua y Xian Du. *Data Mining and Machine Learning in Cybersecurity*. Auerbach Publications, Boston, MA, USA, 1ª edición, 2011.

[83] Ekta Gandotra, Divya Bansal y Sanjeev Sofat. Malware analysis and classification: A survey. *Journal of Information Security*, 5(2):56–64, Abril 2014.

[84] Max Kuhn. Building Predictive Models in R Using the caret Package. *Journal of Statistical Software*, 28(5):1–26, 2008.

[85] Andy Liaw y Matthew Wiener. Classification and Regression by randomForest. *R News*, 2/3:18–22, 2002.

[86] Fast & Fresh. Choosing a domain name to be heard but not seen. Technical report, Association Française pour le Nommage Internet en Coopération, 2018. Disponible en https://www.afnic.fr/wp-media/uploads/2021/01/Choosing_a_domain_name_to_be_heard_and_not_seen_complete_study.pdf.

[87] Honglak Lee, Roger Grosse, Rajesh Ranganath y Andrew Y Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. En *Proceedings of the 26th annual international conference on machine learning*, páginas 609–616, 2009.

[88] Saul Gorn, Robert W Bemer y Julien Green. American standard code for information interchange. *Communications of the ACM*, 6(8):422–426, 1963.

[89] Tomas Mikolov, Kai Chen, Greg Corrado y Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. En Yoshua Bengio y Yann LeCun, editors, *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, 2013.

[90] Jürgen Schmidhuber, Daan Wierstra y Faustino J Gomez. Evolino: Hybrid neuroevolution/optimal linear search for sequence prediction. En *Proceedings of the 19th International Joint Conferenceon Artificial Intelligence (IJCAI)*, 2005.

[91] Gábor Petneházi. Recurrent neural networks for time series forecasting. *arXiv preprint arXiv:1901.00069*, 2019.

[92] Recurrent neural networks for time series forecasting: Current status and future directions, author = Hewamalage, Hansika and Bergmeir, Christoph and Bandara, Kasun. *International Journal of Forecasting*, 37(1):388–427, 2021.

[93] Pankaj Malhotra, Lovekesh Vig, Gautam Shroff y Puneet Agarwal. Long short term memory networks for anomaly detection in time series. En *Proceedings*, volumen 89, páginas 89–94, 2015.

[94] Alex Graves y Jürgen Schmidhuber. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural networks*, 18(5-6):602–610, 2005.

[95] Santiago Fernández, Alex Graves y Jürgen Schmidhuber. An application of recurrent neural networks to discriminative keyword spotting. En *International Conference on Artificial Neural Networks*, páginas 220–229. Springer, 2007.

[96] Alex Graves, Abdel-rahman Mohamed y Geoffrey Hinton. Speech Recognition with Deep Recurrent Neural Networks. En *2013 IEEE international conference on acoustics, speech and signal processing*, páginas 6645–6649. IEEE, 2013.

[97] Ilya Sutskever, Oriol Vinyals y Quoc V Le. Sequence to sequence learning with neural networks. En *Advances in neural information processing systems*, páginas 3104–3112, 2014.

[98] TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015. Software available from tensorflow.org.

[99] Steve Lawrence, C Lee Giles y Ah Chung Tsoi. What size neural network gives optimal generalization? Convergence properties of backpropagation. Technical report, Institute for Advanced Computer Studies, University of Maryland, 1996.

[100] Keras. Keras Documentation: Use of activations. [Online; https://keras.io/activations/], 2020. Accedido el 27 de abril de 2020.

[101] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2020.

[102] Samuel L Smith, Pieter-Jan Kindermans, Chris Ying y Quoc V Le. Don't decay the learning rate, increase the batch size. *arXiv preprint arXiv:1711.00489*, 2017.

[103] Davide Chicco y Giuseppe Jurman. The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. *BMC genomics*, 21(1):1–13, 2020.

[104] Jose Selvi, Ricardo J Rodríguez y Emilio Soria-Olivas. Detection of Algorithmically Generated Malicious Domain Names using Masked N-Grams. *Expert Systems with Applications*, 124:156–163, 2019.

[105] Jose Selvi, Ricardo J Rodríguez y Emilio Soria-Olivas. Toward optimal lstm neural networks for detecting algorithmically generated domain names. *IEEE Access*, 9:126446–126456, 2021.

[106] Jan Spooren, Davy Preuveneers, Lieven Desmet, Peter Janssen y Wouter Joosen. Detection of algorithmically generated domain names used by botnets: a dual arms race. En *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, páginas 1916–1923, 2019.