UNIVERSITAT OBERTA DE CATALUNYA

UNIVERSITAT AUTÓNOMA DE BARCELONA

UNIVERSITAT ROVIRA I VIRGILI

MÁSTER UNIVERSITARIO EN SEGURIDAD DE LAS TECNOLOGÍAS DE LA INFORMACIÓN Y DE LAS COMUNICACIONES

TRABAJO FIN DE MÁSTER

ANCERT: APLICACIÓN DE TÉCNICAS DE MACHINE LEARNING A LA SEGURIDAD

FRANCISCO JAVIER MERCHÁN MACÍAS

Entrega: junio 2018

UNIVERSITAT OBERTA DE CATALUNYA

UNIVERSITAT AUTÓNOMA DE BARCELONA

UNIVERSITAT ROVIRA I VIRGILI

MÁSTER UNIVERSITARIO EN SEGURIDAD DE LAS TECNOLOGÍAS DE LA INFORMACIÓN Y DE LAS COMUNICACIONES

TRABAJO FIN DE MÁSTER

Entrega: Junio 2018

ANCERT: APLICACIÓN DE TÉCNICAS DE MACHINE LEARNING A LA SEGURIDAD

Palabras clave: *Machine Learning, Ciberseguridad, análisis, algoritmos, C5.0, Red neuronal.*

TUTOR: VICTOR GARCÍA FONT
Fdo.:

TUTOR COLABORADOR:
ENRIC HERNÁNDEZ JIMÉNEZ
Fdo.:

AUTOR: FRANCISCO JAVIER MERCHÁN MACÍAS

Fdo.:

MISTIC
TFM: APLICACIÓN DE TÉCNICAS DE MACHINE LEARNING A LA SEGURIDAD
2018 – Francisco Javier Merchán Macías

Resumen

En un mundo en el que hasta ahora hemos basado nuestras decisiones empresariales y personales en experiencias y estadísticas de pequeños conjuntos de información, hemos pasado a un mundo en el que disponemos de múltiples conjuntos de datos, de gran tamaño en el que necesitamos la ayuda de algoritmos que nos ayuden a clasificar correctamente esta información. Estos algoritmos son los llamados algoritmos de inteligencia artificial y aprendizaje automático.

La seguridad informática está aprovechando la potencia de estos algoritmos para la predicción de problemas de seguridad en los sistemas. En este trabajo vamos a desarrollar los algoritmos de predicción C5.0 y redes neuronales aplicados a la seguridad informática.

Abstract

In a world where we have so far based our business and personal decisions on experiences and statistics from small sets of information, we have moved on to a world where we have multiple, large data sets, in which we need the help of algorithms to help us classify this information correctly. These algorithms are the so-called algorithms of artificial intelligence and automatic learning.

Computer security is harnessing the power of these algorithms to predict system security problems. In this work we will develop the C5.0 prediction algorithms and neural networks applied to computer security.

Agradecimientos

Marta, mi media naranja y la razón por la que respiro.

Mi madre, por tantas cosas, que no puedo enumerar... ¡que puedo decir de mi madre que no sea bueno!.

Mi tío Benito, que me alegra cada día que puedo disfrutar de su compañía.

Mi hermano Manolito, que aunque su nombre sea un diminutivo, es muy grande.

Elisa, Aitor, Alma, Alonso, Tere, Mamen y Hugo, siempre cerca.

Paulino, Lola y Alicia, me aportáis mucho y mi mundo sería muy diferente sin vosotros.

Por supuesto, a los que no están y que no podemos ni queremos olvidar.

A mis amig@s, por estar siempre ahí.

Spot, Koki, Kiko y Kokitos de Céspedes, Nimbo y Seda de Malvasia y Claus De Rimacán, que me alegran y me han alegrado mis días.

Y finalmente, a compañeros, personal docente y UOC, por todos los conocimientos y calidad humana que nos aportáis.

Gracias a tod@s.

UGVrZSwgwqF0ZSBxdWllcm8h

CED21E005528FC271D4C76DA4E8A6C30F78D2799

Índice general

Índice de figuras

Índice de tablas

Capítulo 1 Introducción

Hace unos años, las decisiones empresariales y técnicas se realizaban según la experiencia del director o técnico encargado del proyecto o tarea a realizar. En materia de seguridad informática, el técnico encargado de administrar y securizar un sistema, aplicaba su experiencia y conocimientos al tráfico disponible en sus sistemas. Hoy en día, ese análisis sería insuficiente ya que nuestra experiencia puede fallar.

Hoy en día podemos almacenar muchos datos que hace sólo unos años se descartaban por motivos tales como la falta de espacio, falta de capacidad de cómputo para analizarlos, etc. Hoy en día, gracias al Big Data, podemos almacenar multitud de datos y mediante técnicas de machine learning, podemos analizar esos datos para obtener conclusiones basadas en datos y no basadas en la experiencia, obteniendo modelos y algoritmos que aprenden y evolucionan teniendo en cuenta el histórico de nuestros datos y los datos nuevos que se van almacenando, realizando una determinada acción cada día mejor.

Este proyecto lo enfocaremos en el uso de estas técnicas de machine learning, para realizar un estudio y comparativa de las capacidades predictivas de los algoritmos elegidos al usar un Data Set conocido, obtenido de un sistema de detección de intrusiones. De esta forma cada algoritmo mostrará un nivel de predicción al analizar el Data Set, para determinar si existen o no intrusiones y ataques.

Objetivos

Los objetivos que buscamos conseguir durante la realización del proyecto se pueden dividir en:

Objetivos teóricos:

- Estudiar el funcionamiento de algoritmos de aprendizaje automático.
- Estudiar el análisis de datos del Data Set a analizar.

Objetivos prácticos:

- Realizar un preprocesado del Data Set a analizar. En este apartado podremos comprobar la gran importancia de hacer un buen preprocesado de características para mejorar la capacidad de predicción de los algoritmos.
- Realizar un análisis comparativo de las capacidades de predicción de los algoritmos seleccionados. Para ello debemos realizar un entrenamiento de los algoritmos y su posterior fase de test para pasar a reconocer patrones finalmente.

Metodología

Para alcanzar los objetivos anteriores se van a definir varias fases, las cuales completarán paso a paso los objetivos planteados.

En la primera fase realizaremos un estudio de los algoritmos de machine learning tales como C4.5 y redes neuronales y realizaremos una comparativa teórica de ambos algoritmos.

En una segunda fase, se aplicarán dichos algoritmos a un Data Set específico de tráfico de red recogido de un Sistema de Detección de Intrusiones y deberemos medir y comparar las capacidades predictivas de los algoritmos, realizando una comparativa entre ellos.

Finalmente, se obtendrán unas conclusiones sobre el desarrollo realizado en las dos fases anteriores.

Tareas y planificación

Hemos definido las tareas a realizar, las relaciones, el orden y la estimación de tiempos mediante el uso de un diagrama de Gantt. A priori, no hemos adaptado la planificación a las entregas de las PECs ya que debido a que este estudio puede llevar asociado un gran tiempo de análisis de características y patrones y a su vez mucho tiempo de entrenamiento de los algoritmos, no se puede fijar fielmente los tiempos al tiempo de entregas de las PECs.

La planificación se realizará de la siguiente manera:

- Se realizará una búsqueda de información de los algoritmos de machine learning o aprendizaje automático C4.5 y redes neuronales. Si se considera oportuno, se podrían añadir, modificar o eliminar algoritmos.
- Una vez realizado un estado del arte de los algoritmos seleccionados, se procederá a hacer una pequeña comparativa teórica de dichos algoritmos y se plantearán unas conclusiones al respecto.
- Tras realizar el estado del arte de los algoritmos, se realizará una pequeña prueba de concepto con los algoritmos seleccionados utilizando un data set que contienen diferentes tipos de amenaza y ataques. Este Data Set deberá ser analizado para detectar patrones de características que puedan dar mejores resultados a la hora de realizar el entrenamiento de nuestros algoritmos.
- A continuación, el Data Set deberá ser analizados por cada uno de los algoritmos para detectar el grado de predicción conseguido.
- Finalmente, se realizará una pequeña comparativa con los resultados obtenidos.

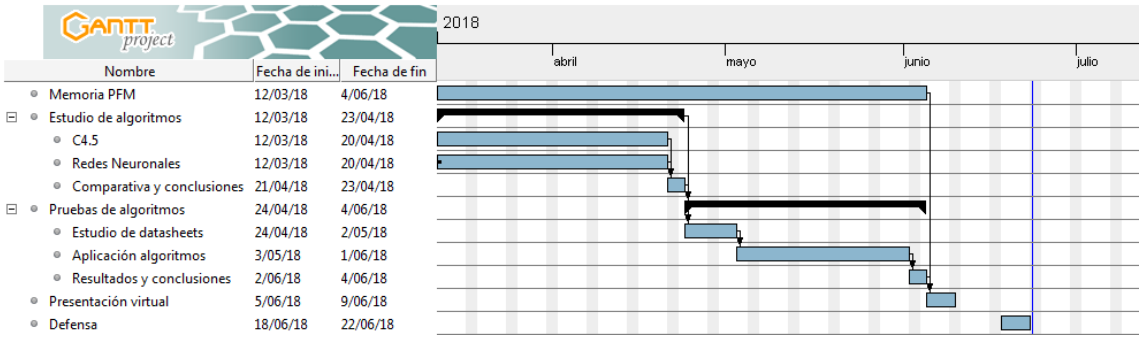


Ilustración 1. Diagrama de Gantt de la planificación del proyecto fin de máster.

Software y hardware utilizado

El software previsto[1] que vamos a utilizar para realizar este proyecto será el siguiente:

- Microsoft Windows 10.
- Debian Linux 9.3.
- Microsoft Office 365
    - Microsoft Word 2016.
    - Microsoft PowerPoint 2016.
    - Microsoft Excel 2016.
- Gantt Project(«GanttProject: free desktop project management app», s. f.)
- MATLAB Student R2017b («MATLAB Student R2017b - MathWorks España», s. f.). Se intentará usar la alternativa libre, R 3.4.3.
- R 3.4.3 («R: The R Project for Statistical Computing», s. f.).
- Weka 3.8 («Weka 3 - Data Mining with Open Source Machine Learning Software in Java», s. f.).
- KNIME 3.5.2 («KNIME Analytics Platform & SDK | KNIME», s. f.).
- 7Zip 18.01 («7-Zip», s. f.).
- Google Chrome («Chrome para ordenadores», s. f.).
- Adobe Acrobat Reader DC.
- Zotero 5.0 («Zotero | Your personal research assistant», s. f.).
- Gimp 2.8.22 («GIMP Descargas, tutoriales y foros. Alternativa a Photoshop gratis y libre», s. f.).

El hardware previsto a utilizar es:

- Ordenador portátil con conexión a Internet.

[1] El software podrá variar ya que ha sido una previsión de software antes de realizar el estudio. Por lo que es posible que la lista de Software que se utilice finalmente sea modificada.

Costes de realización del proyecto

Se ha intentado usar software de fuentes abiertas con licencias gratuitas para uso de estudiantes, a excepción de MATLAB que tiene un coste de 69,00€ y Windows 10 PRO, que venía incluido como licencia en el ordenador portátil utilizado, valorado en 599,00€. La licencia de Microsoft Office 365 viene incluida en la matrícula de alumno de la UOC, por lo que no consideramos coste para este proyecto. El resto de software es gratuito.

El coste total del proyecto sería:

- Ordenador portátil: 599,00€
- MATLAB: 69,00€
- Energía eléctrica: 20,00€
- Tiempo empleado: 230 horas estimadas aproximadamente a 5,00€ hora: 1150,00€

TOTAL: 1838,00€.

Estructura del documento

En el **Capítulo 1** realizaremos una pequeña introducción del TFM, enumeraremos los objetivos, metodología y planificación del trabajo.

En el **Capítulo 2**, se hará una breve introducción al Machine Learning y algunos algoritmos relevantes para llegar a describir a continuación en los siguientes capítulos los algoritmos seleccionados.

En el **Capítulo 3**, haremos un estudio del algoritmo C4.5, sin profundizar demasiado en su base matemática, ya que no es el objetivo de este trabajo.

En el **Capítulo 4**, haremos un estudio de las redes neuronales, sin profundizar demasiado en su base matemática, ya que no es el objetivo de este trabajo.

En el **Capítulo 5**, tenemos como objetivo realizar un pequeño estudio de un conjunto de datos (Data Set) utilizando el Algoritmo C4.5 y Redes neuronales, a priori, utilizando como software de análisis R.

En el **Capítulo 6**, expondremos los resultados obtenidos en el Capítulo 5.

Finalmente, en el **Capítulo 7**, lo dedicaremos a tratar las conclusiones de este TFM y su posible desarrollo futuro en otros posibles trabajos que puedan aprovechar el conocimiento aportado en este TFM.

Capítulo 2 Introducción al Machine Learning y los algoritmos

Introducción

El concepto de Machine Learning o aprendizaje de las máquinas no es un concepto nuevo, aunque sí que se trata de unos de los temas de más actualidad gracias al impulso que le han dado las grandes compañías tecnológicas como Google, Microsoft o Facebook. Actualmente disponemos de grandes conjuntos de datos que hace tan solo unos años no teníamos ni siquiera la capacidad de almacenar, y mucho menos de analizar, debido a los altos costes del almacenamiento y de la capacidad de cómputo, pero hoy en día, estos datos se pueden almacenar y analizar de manera automática, ya que tanto el precio del almacenamiento como de la capacidad de cómputo es muy barata. Debido a esto, el Machine Learning es una tecnología en pleno auge.

Para empresas como Google, Microsoft, Facebook, Telefónica, etc., les supone unos costes mínimos en infraestructura de computación y almacenamiento en relación con el beneficio obtenido con el análisis de los datos, e incluso en alguna ocasión, almacenan datos que hoy en día no saben si serán útiles en el futuro, pero que posiblemente pasados varios años, podrían ser útiles y ventajosos en las compañías.

En el sector de la banca o el de los seguros han sido los más atrevidos en utilizarlo para la mejora de sus procesos, y se están llevando a cabo proyectos muy ambiciosos. Las PYMES españolas siguen en su mayoría sin apostar por el Machine Learning, a pesar de la cantidad de datos valiosos que poseen a día de hoy, porque existe mucho desconocimiento respecto a cómo puede potenciar los negocios (Raona, 2017).

Se ha visto, que a medida que se aprende cómo funciona el cerebro humano, también se aprenden nuevas técnicas que se aplican a las redes neuronales y a su algoritmia, por lo que esta tecnología es bioinspirada. Es cierto que hay una cierta especialización en una serie de neuronas, y este conocimiento intrínseco de los datos es la nueva técnica de funcionamiento llamada Deep Learning. La diferencia entre Machine Learning y Deep Learning es que el Deep Learning tiene más flexibilidad, ya que tiene grupos de neuronas más especializadas que resolverían problemas concretos. Estos grupos de neuronas podrían resolver problemas más complejos. En ciberseguridad, el Machine Learning y el Deep Learning se aplican en detección y clasificación de spam, malware, botnets, fraude en tarjetas de crédito, ciberterrorismo en redes sociales, reconocimiento del habla y lenguaje natural, sentimientos, etc. («ElevenPaths Talks 3: Inteligencia Artificial y Machine Learning - YouTube», 2017).

El mundo del Machine Learning es muy interesante ya que, junto a otras tecnologías como Big Data, IoT, etc. abre una gran cantidad de aplicaciones que están cambiando múltiples áreas de nuestra vida, entre ellas, la ciberseguridad. Ya no es el futuro, es el presente, ya que se utiliza en multitud de sistemas y problemas. En este TFM vamos a tocar superficialmente esta tecnología, ya que es muy amplia, pero estamos seguros de que cautivaremos al lector para seguir profundizando aún más en los beneficios que nos brinda el Machine Learning y le abriremos la mente hacia nuevas aplicaciones. Nunca dejé de aprender, es maravilloso.

¿Por qué debo usar Machine Learning?

Existen varios motivos por los que deberíamos usar el Machine Learning a todos los niveles. Enumeramos algunos motivos:

1. El volumen de datos que podemos manejar hoy en día es muy grande y cada día será mayor, llegando al punto de no poderlos manejar con las herramientas tradicionales.
2. El coste del almacenamiento y la potencia de computación ha bajado considerablemente, aumentando de forma significativa la capacidad de almacenamiento y la potencia de computación, tanto en el hogar como a nivel empresarial.
3. Los beneficios resultantes al analizar grandes conjuntos de datos. Por ejemplo, hoy en día las compañías ya no toman las decisiones empresariales basadas solamente en la experiencia acumulada, sino que van un paso más allá, y se convierten en empresas Data-Driven («Data-driven science», 2016; «Data science», 2018), las cuales toman las decisiones empresariales guiadas por el análisis de datos y por ello, los algoritmos analizados en este TFM, entre otros muchos, son básicos en estas tareas.
4. Soluciones de inteligencia artificial se pueden aplicar y realizar de forma relativamente sencilla, implementándolas en múltiples dispositivos y máquinas, en los cuales podríamos implementar un algoritmo de clasificación tipo C4.5 o una red neuronal correctamente entrenada y validada para realizar diversas tareas.

Hoy en día, gracias al Machine Learning, podemos llegar a analizar datos a gran escala y con distintos algoritmos, detectando patrones y modelos en un período muy corto de tiempo, identificando oportunidades rentables y evitando riesgos que podrían ser imperceptibles a la experiencia y "ojo" humano. Tal es la potencia de esta tecnología, que personas tales como Chema Alonso («Chema Alonso», 2018), CDO de Telefónica, afirman en sus conferencias que alguien como él, ya usa en su vida diaria aplicaciones basadas en el Machine Learning, como Google Maps («Google Maps», 2018), para elegir la mejor ruta de su casa al trabajo o viceversa, a pesar de conocer perfectamente su ciudad, pero no puede conocer en todo momento todos los problemas de tráfico,

meteorológicos, etc., que puedan afectar a la ruta más corta en tiempo y kilómetros en ese mismo momento.

Tipos de aprendizajes en Machine Learning

El Machine Learning parte de unos datos de entrada y una serie de características facilitadas por el analista de datos para aplicar al modelo. De esta forma, se puede entrenar a la máquina para que establezca una conexión a partir de los datos de una persona (sexo, edad, nivel económico, etc.) para determinar si esa persona tiene un riesgo alto para concederle o no un crédito bancario solicitado. Con los datos solicitados podemos crear un modelo gracias al Machine Learning para identificar estos perfiles y predecir comportamientos. El campo que abarca el Machine Learning es muy amplio y debemos diferenciar sus tipologías, las cuales pueden ser:

Aprendizaje supervisado.

Es una técnica para deducir una función a partir de datos de entrenamiento. Los datos de entrenamiento consisten en pares de objetos, por un lado, los datos de entrada y por otro lado el resultado deseado. De esta forma, podemos enseñar a la máquina que, con una serie de características, un cliente es de tipo A y que el que tiene otra serie de características no es de tipo A. De esta forma, la máquina genera una estructura de información que llega a esos resultados, siendo una especie de caja negra en la que no sabemos cómo hace la máquina para llegar a ese resultado o al menos es bastante complicado seguir esas reglas creadas.

Dentro del aprendizaje supervisado, se puede dividir en (Macías Macías, Peguero Chamizo, & García Orellana, 2016):

- Clasificación: la salida del sistema debe ser asociada a una de entre un conjunto discreto de clases C_k con k=1, 2, …, e.
- Regresión: cuando la salida del sistema representa a los valores de una variable continua.

En ambos casos para resolver el problema desde el punto de vista matemático, tanto el patrón de entrada como el de salida, vienen definidos por un vector numérico.

Aprendizaje no supervisado.

Es una técnica en que los datos no han sido etiquetados previamente y solo se dispone de datos de entrada. Por tanto, la máquina debe de ser capaz de encontrar la estructura existente de datos. Este tipo de aprendizaje es muy útil para reducir la dimensionalidad de los datos reduciendo la pérdida de información. Un ejemplo es la segmentación de clientes, en la que, a partir de toda una serie de características, el Machine Learning es

capaz de encontrar un número de grupos con características similares definido por el analista de datos humano. Mediante la reducción de la dimensionalidad a dos o tres componentes, podemos representar gráficamente estos grupos para poder visualizarlos y tener así una mejor comprensión.

Aprendizaje por refuerzo.

Este tipo de aprendizaje funciona en base a premios, a través del ensayo y error. Los datos de entrada se obtienen a través del feedback o retroalimentación del entorno. Si la máquina no lo hace bien se le da un premio igual a 0 o negativo. En cambio, si toma una acción acertada se le dan premios con valor positivo para que finalmente encuentre una buena solución. De esta forma conseguimos que el sistema aprenda de forma inteligente cuando toma decisiones acertadas, mejorando los procesos de decisión. El sistema será más capaz de tomar buenas decisiones si se realiza un entrenamiento adecuado para trasladar la perspectiva subjetiva humana a un proceso que mejorará continuamente su aprendizaje y su efectividad predictiva.

Capítulo 3 Algoritmo C4.5/C5.0

El algoritmo C4.5 es un algoritmo usado para generar un árbol de decisión desarrollado por Ross Quinian en 1993. El algoritmo C4.5 es una extensión del algoritmo ID3 desarrollado anteriormente con Quinian en 1986. Los árboles de decisión generados por C4.5 pueden ser usados para clasificar, y por esta razón, suele ser referido como un clasificador estadístico («C4.5», 2018).

C4.5 construye árboles de decisión desde un grupo de datos de entrenamiento de la misma forma que lo hace ID3, usando un concepto de entropía de información («C4.5», 2018).

Algunas mejoras del algoritmo C4.5 con respecto a su antecesor ID3 son («C4.5», 2018, p. 5):

- Manejo de ambos atributos continuos y discretos - A fin de manejar atributos continuos, C4.5 crea un umbral y luego se divide la lista en aquellos cuyo valor de atributo es superior al umbral y los que son menores o iguales a él.
- Manejo de atributos con costos diferentes.
- Podando árboles después de la creación - C4.5 se remonta a través del árbol una vez que ha sido creado e intenta eliminar las ramas que no ayudan, reemplazándolos con los nodos de hoja.

Como mejoras al algoritmo C4.5, disponemos del algoritmo C5.0, el cual aporta muchas mejoras con respecto al C4.5. A modo de ejemplo, podemos enumerar algunas mejoras:

- Velocidad - C5.0 es significativamente más rápido que el C4.5 (varios órdenes de magnitud)
- El uso de memoria - C5.0 es más eficiente que el C4.5

- Árboles de decisión más pequeños - C5.0 obtiene resultados similares a C4.5 con árboles de decisión mucho más pequeños.
- Soporte para boosting - Boosting mejora los árboles y les da una mayor precisión.
- Ponderación - C5.0 le permite ponderar los distintos casos y tipos de errores de clasificación.
- Winnowing - una opción automática de C5.0 consiste en aplicar un algoritmo de clasificación (algoritmo Winnow) a los atributos para eliminar aquellos que sean de poca ayuda.

Capítulo 4 Redes neuronales

Introducción

Las redes neuronales artificiales (en adelante las nombraremos como redes neuronales) están inspiradas, en el funcionamiento que tiene el cerebro de cualquier animal. El cerebro humano está compuesto por millones de neuronas interconectadas que producen una salida tras la introducción de múltiples entradas, es decir, y a modo de ejemplo didáctico, simulando que los sentidos y el conocimiento son neuronas de entrada y nuestra reacción es una neurona de salida, si una persona detecta con su piel un cambio de las condiciones del tiempo, ve con sus ojos nubes negras y conoce que está en un mes de lluvias, probablemente su salida será que lloverá, pero posiblemente si esta misma persona detecta el mismo cambio en las condiciones del tiempo, no ve ninguna nube y está en un mes de lluvias, su salida será que no lloverá. Una red neuronal basa su comportamiento en los pesos asignados a cada entrada de una neurona y una función de salto o activación que provocará una salida, normalmente en números reales. La conexión de múltiples neuronas provocará que podamos enseñar, entrenar y validar una red neuronal para que realice una tarea que le asignemos.

Los sistemas de redes neuronales intentan resolver los problemas como lo podría resolver el cerebro humano, sin necesidad de programar complejas reglas que en muchos casos serían muy complejas de escribir en cualquier lenguaje de programación y que este algoritmo crea automáticamente. Por este motivo, tras entrenar una red neuronal, no podemos saber que reglas ha creado para llegar a la salida deseada, de ahí que se llame algoritmo de caja negra. Actualmente un sistema de redes neuronales utiliza un rango de miles a unos pocos millones de unidades neuronales («Red neuronal artificial», 2018), pero esa magnitud es muy pequeña y mucho menos compleja que un cerebro humano, llegando quizás a la potencia de cálculo de un gusano.

El aumento de la potencia de computación de los sistemas actuales ha ayudado a las redes neuronales, como a tantas otras tecnologías, a desarrollarse aún más, y es por todo esto que el uso de esta tecnología se aplica a múltiples problemas.

Historia

Realizaremos un pequeño viaje a lo largo del tiempo para ver la evolución de las redes neuronales desde al año 1913 a la actualidad (Jaomun, 200d. C.a, 200d. C.b), dándonos una visión de partida y hasta donde estamos llegando.

- 1913: S. Rusell plantea una máquina hidráulica basada en las redes neuronales, sin plantear matemáticamente estos modelos.
- 1936: Alan Turing, es el precursor del estudio de la asimilación de la computación al comportamiento humano.
- 1943: Warren McCulloch (neuro físico) y Walter Pitts (matemático), proponen el primer modelo matemático de red neuronal sin capacidad de aprendizaje.
- 1949: Doanld Hebb es el precursor en la explicación del proceso de aprendizaje de las neuronas, desarrolló un algoritmo de aprendizaje denominado aprendizaje Hebbiano, cuya idea principal es que el aprendizaje se produce ante cambios que hacen que una neurona se active.
- 1950:
    - Karl Lashley tras diversas pruebas, determina que la información no se almacena en el cerebro de forma centralizada, sino que se distribuye por todo el cerebro.
    - Como nota curiosa sin relación directa con las redes neuronales, Alan Turing crea en 1950 su famoso test, "test de Turing", en su ensayo "Computing Machinery and Intelligence, en el que introduce la siguiente cuestión: "¿Pueden pensar las máquinas?" («Computing machinery and intelligence», 2018; «Test de Turing», 2018; «Un ordenador logra superar por primera vez el test de Turing», 2014).
- 1954: Marvin Minsky y Dean Edmons obtienen los primeros resultados con una máquina con 40 neuronas la cual incorpora el modelo de McCulloch y Pitts junto al aprendizaje Hebbiano.
- 1956:
    - El congreso de Dartmouth es el primer congreso científico sobre redes neuronales.
    - Albert Uttley realiza unos trabajos sobre su modelo Informon.
- 1957: Frank Roseblatt, basándose en los trabajos de Uttley, desarrolla el Perceptrón, siendo uno de los modelos más conocidos a lo largo de la historia.
- 1959: Frank Roseblatt, describe como aprende la red Perceptrón en su libro "Principios de Neurodinámica".
- 1960: Bernard Wifroof y Marcian Hoff desarrollan el modelo Adaline (Adaptative Linear Elements), el cual se utilizó durante varias décadas para eliminar los ecos en las líneas telefónicas.

- 1969: Marvin Minsky y Seymour Papert, plantean las limitaciones de las capacidades de Percetrón, demostrándolo matemáticamente en su libro "Perceptrons".
- 1974: Paul Werbos, desarrolla el algoritmo de aprendizaje de propagación hacia atrás (Backpropagation).
- 1977: Stephen Grossberg es capaz de crear una arquitectura de red que simula la memoria a largo y corto plazo planteando la teoría de resonancia adaptada.
- 1982 al 1985: John Hopfield, describe un sistema (modelo Hopfield) basado en sistemas olfativos en su libro "Computación neuronal de decisiones en problemas de optimización".
- 1986: David Rumelhart y G. Hinton redescubren y desarrollan el algoritmo de propagración hacia atrás (Back propagration).
- A partir de aquí se publican múltiples estudios sobre redes neuronales.

Definición de una red neuronal

Existen muchas definiciones de una red neuronal, pero por norma general, una red neural artificial es un modelo de computación inspirado en la estructura de redes neurales del cerebro de los animales. En modelos simplificados del cerebro, consiste en un gran número de dispositivos básicos de computación (neuronas) que son conectadas unos con otros en una red compleja de comunicaciones, a través del cual, el cerebro es capaz de realizar cálculos muy complejos. Las redes neuronales artificiales son construcciones formales de computación que se modelan a partir de este paradigma de computación. El aprendizaje con redes neuronales fue propuesto a mediados del siglo XX. Produce un paradigma de aprendizaje efectivo y recientemente se ha demostrado que logra un desempeño de vanguardia en varias tareas de aprendizaje (Shalev-Shwartz & Ben-David, 2014).

Una red neuronal puede describirse como un gráfico dirigido cuyos nodos corresponden a las neuronas y los bordes corresponden a los enlaces entre ellas. Cada neurona recibe como entrada una suma ponderada de las salidas de las neuronas conectadas a sus bordes de entrada.

En el contexto del aprendizaje, podemos diseñar una clase de hipótesis consistente en predictores de redes neuronales, donde todas las hipótesis comparten la estructura gráfica subyacente de la red y mueren en las pesas sobre los bordes.

El uso de redes neuronales permite implementar funciones no lineales de muchas variables mediante la superposición de funciones no lineales de una única variable denominada unidades ocultas. Una red neuronal con al menos dos capas de procesamiento con funciones de activación de las capas intermedias no lineales, permiten implementar cualquier mapping entre dos subespacios siempre que el número de neuronas de la capa oculta sea suficientemente alta (Macías Macías et al., 2016).

Al usar una red neuronal, conlleva un buen escalado con la dimensionalidad, es decir, el aumento en el número de parámetros adaptivos no aumenta mucho con el número de parámetros de entrada y depende prácticamente de la dificultad del problema a resolver. La etapa de reconocimiento es rápida, pero el modelo no es lineal. La etapa de aprendizaje es un proceso de optimización no lineal generalmente lento y con mínimos locales. Se suelen utilizar algoritmos iterativos de minimización del error cometido por la red neuronal.

Elementos básicos de una red neuronal

Una red neuronal se compone de neuronas interconectadas y situadas en al menos tres capas. Los datos entran en la red neuronal por la "capa de entrada", pasando a través de ella a la "capa oculta" que puede tener varias capas y finalmente salen de la red neuronal por la "capa de salida".

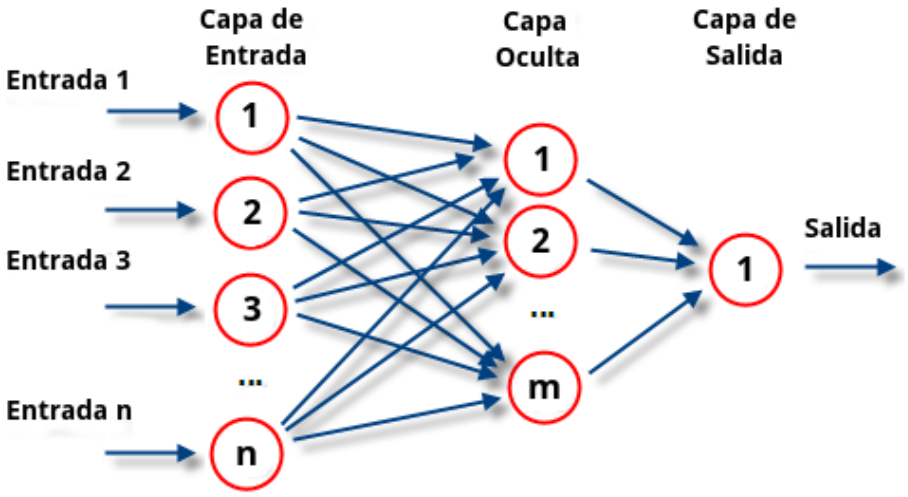


Ilustración 2. Elementos básicos de una red neuronal (Gengiskanhg, 2004).

Función de entrada

La neurona trata a muchos valores de entrada como si fueran uno solo, esto recibe el nombre de entrada global, por lo tanto, ahora nos enfrentamos al problema de cómo se pueden combinar estas simples entradas (ini_1, ini_2, ini_3) dentro de la entrada global, gini. Esto se logra a través de la función de entrada, la cual se calcula a partir del vector de entrada (Neuronales, 2012a). La función de entrada se puede describir como:

$$input = (ini\ 1 * wi\ 1) * (ini\ 2 * wi\ 2) * \cdots (ini\ n * wi\ n)$$

Donde:

* representa al operador apropiado (por ejemplo: máximo, suma, producto, etc.)
n representa al número de entradas a la neurona ini y wi al peso.

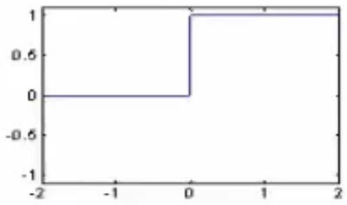
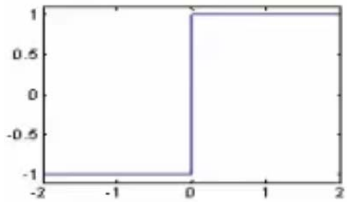
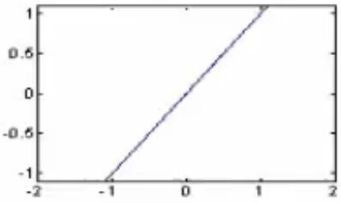
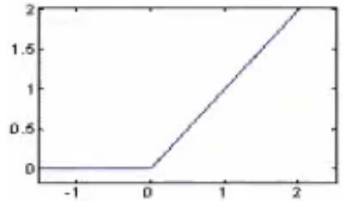
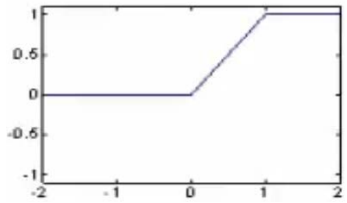
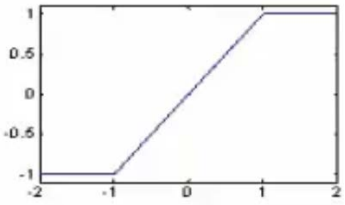
Los valores de entrada se multiplican por los pesos anteriormente ingresados a la neurona. Estos pesos van cambiando a medida que influencian los valores de entrada, por lo que se permiten variar el valor de las entradas según el peso asignado.

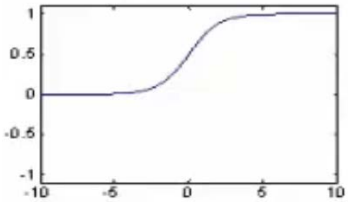
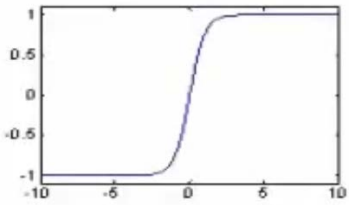
Función de activación

Las neuronas artificiales pueden tener dos estados de activación, al igual que las neuronas biológicas, o más estados, dentro de los valores de un conjunto determinado.

La función de activación calcula el estado de actividad de una neurona, transformando la entrada global (menos el umbral, θi) en un valor o estado de activación, cuyo rango normalmente va de [0, 1] o de [-1, 1]. Esto es así porque una neurona puede estar totalmente inactiva, estando en los valores [0 ó -1] o totalmente activa, estando en el valor [1]. La función de activación, es una función de la entrada global (Gini) menos el umbral (θi) (Neuronales, 2012a). Existen diferentes funciones de activación, que pasamos mostrar algunas a continuación (Hackeando Tec, 2015):

Tabla 1. Distintas funciones de activación.

Escalón	Escalón Simétrico	Lineal
 $f(n)=\begin{cases}1 & n\geq 0\\ 0 & n<0\end{cases}$	 $f(n)=\begin{cases}+1 & n\geq 0\\ -1 & n<0\end{cases}$	 $f(n)=n$
Lineal Positiva	Lineal Saturada	Lineal Saturada Simétrica
 $f(n)=\begin{cases}n & n\geq 0\\ 0 & n<0\end{cases}$	 $f(n)=\begin{cases}0 & n<0\\ n & 0\leq n\leq 1\\ 1 & n>1\end{cases}$	 $f(n)=\begin{cases}-1 & n<-1\\ n & -1\leq n\leq 1\\ 1 & n>1\end{cases}$

Sigmoide Logarítmica	Sigmoide Tangente Hiperbólica	
 $f(n) = \frac{1}{1+e^{-n}}$	 $f(n) = \frac{e^{n} - e^{-n}}{e^{n} + e^{-n}}$	

Función de salida

El último componente del sistema neuronal es la función de salida. El valor que resulta de esta función es la salida de la neurona i (outi). El valor que se le transfiere a las neuronas vinculadas es la función de salida, por lo que, si el valor de la función de activación está por debajo del umbral determinado, no se producirá ninguna salida y por lo tanto no se le pasará ningún valor a la neurona siguiente. Normalmente no se permite cualquier valor como entrada para una neurona, por lo tanto, los valores de salida están comprendidos en el rango de [0, 1] o [-1, 1]. También pueden ser valores binarios tales como {0, 1} o {-1, 1} (Neuronales, 2012a).

Ventajas que ofrecen las redes neuronales

Las redes neuronales presentan un gran número de características semejantes a las del cerebro, como, por ejemplo, la capacidad de aprender de la experiencia, de generalizar según los casos anteriores, de abstraer características esenciales a partir de entradas que representan información irrelevante, etc. Por ello, ofrecen numerosas ventajas que hacen que esta tecnología se esté aplicando en múltiples áreas.

Algunas de sus ventajas son (Neuronales, 2012b):

- Aprendizaje adaptativo: la capacidad de aprender a realizar tareas basadas en un entrenamiento o en una experiencia inicial.
- Auto organización: una red neuronal puede crear su propia organización o representación de la información que recibe mediante una fase de aprendizaje.
- Tolerancia a fallos: algunas capacidades de la red se pueden retener, aunque la red sufra una destrucción o degradación parcial de su estructura.
- Operación en tiempo real: la capacidad de computación de una red puede realizarse en paralelo, para ello se diseñan máquinas con hardware que permita realizar cálculos en paralelo para aprovechar esta capacidad. Aquí podemos ver que el abaratamiento del hardware actual ha propiciado a que la capacidad de cómputo de una red sea mucho mayor.

- Fácil inserción dentro de la tecnología existente: se pueden fabricar chips especializados para redes neuronales que mejoran su capacidad en ciertas tareas facilitando también la integración modular en los sistemas existentes.

Aplicaciones de las redes neuronales

Las redes neuronales son apropiadas para aplicaciones en las que no se dispone de un modelo identificable que pueda ser programado, pero se dispone de un conjunto básico de ejemplos de entrada, previamente clasificados o no. Las redes neuronales son muy robustas tanto al ruido como a la disfunción de elementos concretos y son fácilmente paralelizables («Red neuronal artificial», 2018).

La aplicación de las técnicas de machine learning en general y de las redes neuronales en particular es muy variada. Podemos encontrarla en infinidad de ejemplos, cada cual más variado de nuestra vida diaria. Algunos ejemplos podrían ser:

- Reconocimiento de caracteres manuscritos.
- Clasificación de texturas en imágenes capturadas por satélite o drones. Un ejemplo útil sería la clasificación de tres tipos de cultivos de una imagen tomada desde un satélite como puede ser SIGPAC. En este caso clasificamos la imagen como viñedos, olivar y tierras de labor.

Ilustración 3. Imagen generada por Francisco Javier Merchán Macías, en el Máster Universitario en Investigación en Ingeniería y Arquitectura, especialidad TIC utilizando una red neuronal implementada con lenguaje R.

- Detección de comportamiento de software malicioso en redes de datos. En este caso podemos detectar flujos de paquetes o paquetes que puedan corresponder a malware conocido o no, realizar actuaciones para mitigar dichos comportamientos y alertar a los responsables de la red.

- Análisis del tráfico por carretera, ferrocarril y aéreo, obteniendo rutas más cortas y eficientes evitando congestión y ahorrando costes.
- Etc.

Capítulo 5 Solución desarrollada

Introducción

Data set a utilizar

Vamos a utilizar el Data Set de UCI KDD Archive, referente a "detección de intrusión" llamado "KDD CUP 1999 Data" («KDD Cup 1999 Data», 1999). Este Data Set fue utilizado para el tercer concurso internacional de herramientas para el descubrimiento de conocimientos y la minería de datos, que se llevó a cabo en conjunto con KDD-99 La quinta conferencia internacional sobre descubrimiento de conocimientos y minería de datos. La tarea de la competición fue construir un detector de intrusiones de red, un modelo predictivo capaz de distinguir entre conexiones "malas", llamadas intrusiones o ataques, y "buenas", o conexiones normales. Esta base de datos contiene un Data Set a auditar, que incluye una amplia variedad de intrusiones simuladas en un entorno de red militar.

¿Qué contiene el Data Set?

El Data Set contiene los siguientes ficheros:

Tabla 2. Ficheros que contienen el Data Set KDD CUP 1999 Data.

Fichero	Descripción
task_description	(Archivo de información): Esta es la descripción de la tarea original dada a los participantes de la competición.
kddcup.names	(Archivo de datos): Lista de características.
kddcup.data.gz	(Archivo de datos): Conjunto completo de datos (18 MB; 743 MB sin comprimir).
kddcup.data_10_percent.gz[2]	(Archivo de datos): Un subconjunto con el 10% de los datos (2.1 MB; 75 MB sin comprimir).
kddcup.newtestdata_10_percent_unlabeled.gz	(Archivo de datos): (1.4 MB; 45 MB sin comprimir).
kddcup.testdata.unlabeled.gz	(Archivo de datos): (11.2 MB; 430 MB sin comprimir).
kddcup.testdata.unlabeled_10_percent.gz	(Archivo de datos): (1.4 MB; 45 MB sin comprimir).
corrected.gz	(Archivo de datos): Datos de prueba con etiquetas corregidas.
training_attack _types	(Archivo de datos): Lista con los tipos de intrusión.

[2] Este será el Data Set que utilizaremos en nuestras pruebas.

typo-correction.txt	(Archivo de datos): Breve nota sobre un error tipográfico en el conjunto de datos que ha sido corregido (26/06/2007).

Aprendizaje de detectores de intrusión

El software para detectar intrusiones en la red protege una red de computadoras de usuarios no autorizados, incluyendo quizás personas con información privilegiada. La tarea de aprendizaje del detector de intrusiones consiste en construir un modelo predictivo (es decir, un clasificador) capaz de distinguir entre conexiones "malas", denominadas intrusiones o ataques y "buenas", denominadas conexiones normales. El programa de evaluación de detección de intrusos, DARPA, de 1998 fue preparado y administrado por los Laboratorios Lincoln del MIT. El objetivo fue estudiar y evaluar la investigación en la detección de intrusos. Se proporcionó un conjunto estándar de datos para auditar, que incluye una amplia variedad de intrusiones simuladas en un entorno de red militar. El concurso de detección de intrusos KDD de 1999 utiliza una versión de este Data Set.

Lincoln Labs creó un entorno para adquirir nueve semanas de datos de volcado TCP en bruto para una red de área local (LAN) que simula una LAN típica de la Fuerza Aérea de Estados Unidos. Operaron la LAN como si fuera un verdadero entorno de la Fuerza Aérea, pero la disponía muestras de múltiples ataques.

Los datos de entrenamiento en bruto eran de unos cuatro gigabytes de datos de volcado TCP binario comprimido de siete semanas de tráfico de red. Esto se procesó en aproximadamente unos cinco millones de registros de conexión. Del mismo modo, las dos semanas de datos de prueba arrojaron alrededor de dos millones de registros de conexión.

Una conexión es una secuencia de paquetes TCP que comienzan y terminan en momentos bien definidos, entre los cuales los datos fluyen hacia y desde una dirección IP de origen a una dirección IP de destino bajo un protocolo bien definido. Cada conexión está etiquetada como normal o como un ataque específico. Cada registro de conexión consta de aproximadamente unos 100 bytes.

Los ataques se dividen en cuatro categorías principales:

- DOS: denegación de servicio, por ejemplo: SYN Flood.
- R2L: acceso no autorizado desde una máquina remota, por ejemplo: adivinar la contraseña.
- U2R: acceso no autorizado a privilegios de superusuario (root), por ejemplo, varios ataques de "buffer overflow".
- Sondeo: vigilancia y otras pruebas, por ejemplo: exploración de puertos.

Es importante tener en cuenta que los datos de test no provienen de la misma distribución de probabilidad que los datos de entrenamiento, e incluyen tipos de ataques específicos que no están en los datos de entrenamiento. Esto hace que la tarea sea más realista. Algunos expertos en intrusión creen que la mayoría de los nuevos ataques son variantes

de ataques conocidos y la "firma" de ataques conocidos puede ser suficiente para detectar nuevas variantes. Los Data Sets contienen un total de 24 tipos de ataque de entrenamiento, con 14 tipos adicionales solo en los datos de prueba.

Características derivadas

Existen características de alto nivel que ayudan a distinguir las conexiones normales de los ataques (Stolfo, Fan, Lee, Prodromidis, & Chan, 1999). Hay varias categorías de características derivadas.

Las características del mismo "host" examinan solo las conexiones en los últimos dos segundos que tienen el mismo host destino que la conexión actual, y calcula estadísticas relacionadas con el comportamiento del protocolo, el servicio, etc.

Las características similares del "mismo servicio" examinan solo las conexiones en los dos segundos que tienen el mismo servicio que la conexión actual.

Las características del "mismo host" y "mismo servicio" se denominan en conjunto función de tráfico basadas en el tiempo de los registros de conexión.

Algunos ataques de sondeo escanean los hosts (o puertos) utilizando un intervalo de tiempo mucho mayor que dos segundos, por ejemplo, una vez por minuto. Por lo tanto, los registros de conexión también fueron ordenados por el host de destino, y las características se construyeron utilizando una ventana de 100 conexiones al mismo host en lugar de una ventana de tiempo. Esto produce un conjunto de las llamadas características de tráfico basadas en host.

A diferencia de la mayoría de los ataques DOS e indagación, no parece haber patrones secuenciales que sean frecuentes en los registros de ataques R2L y U2R. Esto se debe a que los ataques DOS e indagación implican muchas conexiones con algunos hosts en un periodo de tiempo muy corto, pero los ataques R2L y U2R están incrustados en las porciones de datos de los paquetes, y normalmente implican solo una conexión única.

Algoritmos útiles para extraer porciones de datos no estructurados de paquetes automáticamente son un pregunta de investigación abierta. Usó el conocimiento del dominio para agregar características que busquen un comportamiento sospechoso en las porciones de datos, como el número de intentos fallidos de inicio de sesión. Estas características se llaman características de "contenido" (Stolfo et al., 1999).

Una lista completa del conjunto de características definidas para los registros de conexión se proporciona en las siguientes tres tablas a continuación. El esquema de datos del Data Set del concurso está disponible en formato legible por la máquina.

Tabla 3. Características básicas de conexiones TCP individuales.

Nombre de la característica	Descripción	Tipo

duration	Longitud (número de segundos) de la conexión	Continuo
protocol_type	Tipo de protocolo, ej: TCP, UDP, etc.	Discreto
service	Servicio de red en el destino, ej: http, telnet, etc.	Discreto
src_bytes	Número de bytes de datos desde el origen al destino.	Continuo
dst_bytes	Número de bytes de datos desde el destino al origen.	Continuo
flag	Estado normal o error de la conexión.	Discreto
land	1 si la conexión es desde/al mismo host/puerto; 0 si no lo es.	Discreto
wrong_fragment	Número de fragmentos "malos".	Continuo
urgent	Número de paquetes urgentes.	Continuo

Tabla 4. Características de contenido dentro de una conexión sugerida por el conocimiento del dominio.

Nombre de la característica	Descripción	Tipo
hot	Número de indicadores "hot"	Continuo
num_failed_logins	Número de intentos de login fallidos.	Continuo
logged_in	1 si loguea correctamente, 0 si no loguea correctamente.	Discreto
num_compromised	Número de condiciones "comprometidas".	Continuo
root_shell	1 si se obtiene una Shell root: 0 si no la obtiene.	Discreto
su_attempted	1 si se intentó ejecutar el comando "su root"; 0 si no se ejecutó.	Discreto
num_root	Número de accesos "root".	Continuo
num_file_creations	Número de operaciones de creación de ficheros.	Continuo
num_shells	Número de consolas Shell.	Continuo
num_access_files	Número de operaciones de control de acceso a ficheros.	Continuo
num_outbound_cmds	Número de comandos de salida en una sesión FTP.	Continuo
is_hot_login	1 si el inicio de sesión pertenece a la lista "hot"; 0 si no.	Discreto
is_guest_login	1 si el inicio de sesión es un inicio de sesión de "invitado"; 0 si no.	Discreto

Tabla 5. Características de tráfico calculadas utilizando una ventana de tiempo de dos segundos.

Nombre de la característica	Descripción	Tipo
count	Número de conexiones al mismo host que la conexión actual en los últimos dos segundos.	Continuo
	Nota: Las siguientes características se refieren a las mismas conexiones del host.	
serror_rate	% de conexiones que tienen errores "SYN".	Continuo
rerror_rate	% de conexiones que tienen errores "REJ".	Continuo
same_srv_rate	% de conexiones al mismo servicio.	Continuo

diff_srv_rate	% de conexiones a servicios diferentes	Continuo
srv_count	Número de conexiones al mismo servicio que las conexión actual en los últimos dos segundos.	Continuo
	Nota: Las siguientes funciones se refieren a conexiones del mismo servicio.	
srv_serror_rate	% de conexiones que tienen errores "SYN".	Continuo
srv_rerror_rate	% de conexiones que tienen errores "REF".	Continuo
srv_diff_host_rate	% de conexiones a diferentes host.	Continuo

Pruebas realizadas

Se han hecho distintas pruebas con el Data Set, y debido a que R utiliza un solo CORE de procesador al realizar el entrenamiento y test, los cálculos se pueden eternizar con el volumen de datos que tenemos disponibles para un portátil medio.

Tras múltiples decisiones, se ha decidido cambiar el uso del algoritmo C4.5 por el C5.0 y añadir el algoritmo SVM como mejora. Esto es debido a que debería haber usado una variante llamada J4.8(Hothorn, 2018) del paquete RWeka de R, para simular el algoritmo C4.5 y sabiendo que el C5.0 es la nueva versión del C4.5, finalmente se ha cambiado en las pruebas, además de que se especificó en el Plan de Trabajo, cuando se dijo que "*Si se considera oportuno, se podrían añadir, modificar o eliminar algoritmos.*".

Hemos decidido partir del Data Set "kddcup.data_10_percent.gz", que corresponde al 10% del conjunto de datos completo y desde este, realizaremos distintas modificaciones para hacer las pruebas. Dicho Data Set, corresponde a distintos tipos de paquetes de red a los que se les ha etiquetado por distintos tipos de ataque o paquetes normales. En este caso, este fichero contiene una muestra de 494.139 paquetes, diferenciados de la siguiente manera:

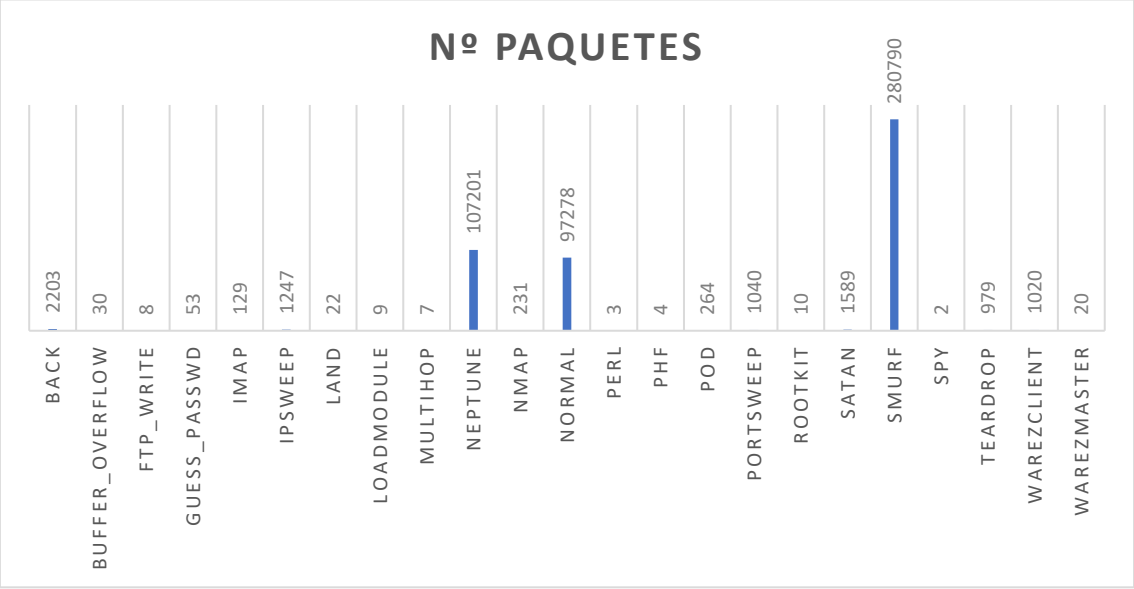


Ilustración 4. Paquetes totales del Data Set utilizado. Correspondientes al 10% del conjunto de datos total.

Primera prueba de clasificación. Data Set de 8752 muestras

Para esta prueba, hemos eliminado del Data Set del 10% de la Ilustración 5, los tres grupos de paquetes más numerosos (neptune -107201 paquetes-, normal -97278 paquetes- y smurf -280790 paquetes-) y hemos dejado 12 paquetes de Imap y 21 de Land, obteniendo un total de 8752 paquetes o muestras distribuidas de la siguiente manera:

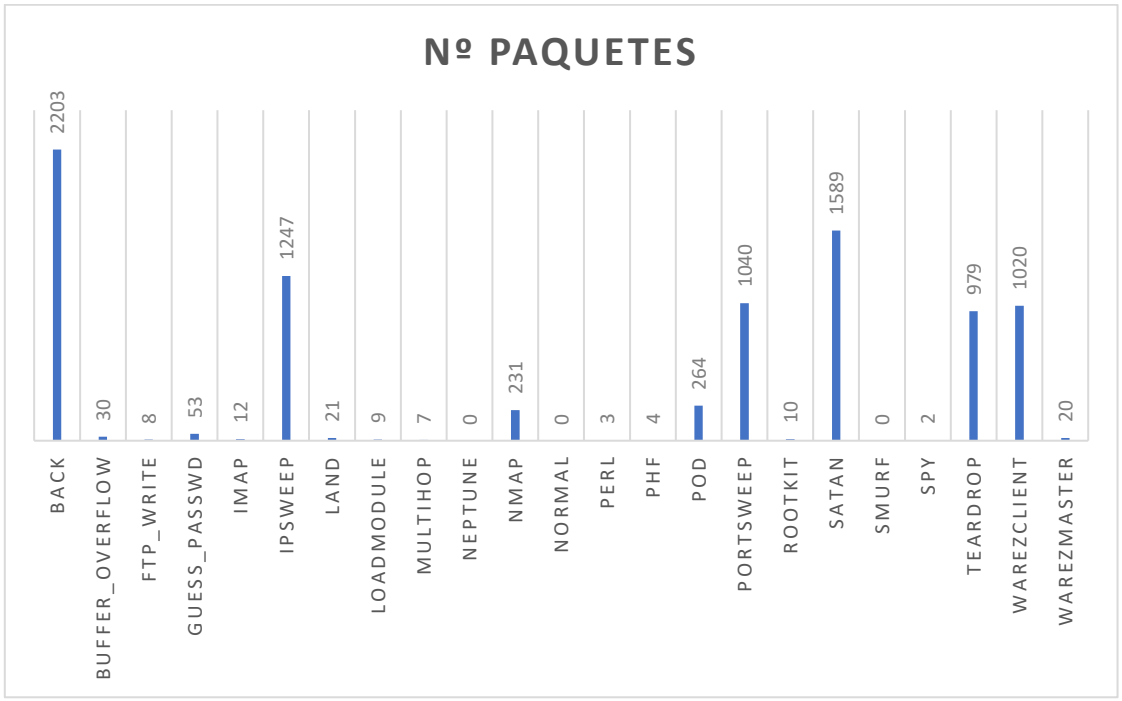


Ilustración 5. Detalle del número de paquetes utilizados.

```
kddcup.dat… 8752 obs. of 42 variables
Functions
modelos     Large function (826.5 Kb)
```

Ilustración 6. Datos cargados en R-Studio.

Disponemos de un Data Set de 8752 muestras distribuidas, como vemos en la Ilustración 6, de las cuales, aproximadamente el 75% se usarán para entrenamiento y el 25% restante, para el test. Tras ejecutar los algoritmos, obtenemos los siguientes resultados.

```
** Red neuronal: nnet
                Real
Asignado         back. buffer_overflow. ftp_write. guess_passwd. imap. ipsweep. land. loadmodule. multihop. nmap.
  back.           548                4          1             0     1        0     0           0         0     0
  guess_passwd.     0                3          0            12     0        1     0           1         0     5
  ipsweep.          0                0          1             0     2      285     4           0         0    24
  nmap.             0                0          0             0     0       10     0           0         1     0
  portsweep.        0                0          0             0     0       14     0           0         0    24
  satan.            0                0          0             0     0        0     1           0         0     0
  teardrop.         0                0          0             0     0        0     0           0         0     2
  warezclient.      2                0          0             1     0        1     0           1         0     2
                Real
Asignado         perl. phf. pod. portsweep. rootkit. satan. spy. teardrop. warezclient. warezmaster.
  back.              0    0   18          0        0      0    0         0           33            0
  guess_passwd.      0    1    0          0        1      0    0         0            2            5
  ipsweep.           0    0    0          1        0      0    0         0            0            0
  nmap.              0    0    0          5        1      0    0         1            3            0
  portsweep.         0    0    0        232        0      1    0         0            1            0
  satan.             0    0    0         21        0    395    0        11            0            0
  teardrop.          0    0    0          1        0      1    0       232            0            0
  warezclient.       0    0   48          0        0      0    0         0          216            0

Correctamente clasificados: 25.36 %
```

Ilustración 7. Prueba con Red Neuronal y 8752 muestras (de las cuales, 2181 muestras para test).

Obtenemos una clasificación de correctos del 25,36% con la red neuronal. A continuación, podemos ver los resultados del algoritmo C5.0.

```
** Árbol de clasificación C5.0
                Real
Asignado         back. buffer_overflow. ftp_write. guess_passwd. imap. ipsweep. land. loadmodule. multihop.
  back.            550                0          0             0     0        1     0           0         0
  buffer_overflow.   0                6          1             0     0        0     0           0         0
  ftp_write.         0                0          0             0     0        2     0           0         0
  guess_passwd.      0                0          0            13     0        0     0           0         0
  imap.              0                0          0             0     3        0     0           0         0
  ipsweep.           0                0          0             0     0      308     0           0         0
  land.              0                0          0             0     0        0     5           0         0
  loadmodule.        0                1          0             0     0        0     0           1         0
  multihop.          0                0          1             0     0        0     0           0         1
  nmap.              0                0          0             0     0        0     0           0         0
  perl.              0                0          0             0     0        0     0           0         0
  phf.               0                0          0             0     0        0     0           0         0
  pod.               0                0          0             0     0        0     0           0         0
  portsweep.         0                0          0             0     0        0     0           0         0
  rootkit.           0                0          0             0     0        0     0           0         0
  satan.             0                0          0             0     0        0     0           0         0
  spy.               0                0          0             0     0        0     0           0         0
  teardrop.          0                0          0             0     0        0     0           0         0
  warezclient.       0                0          0             0     0        0     0           0         0
  warezmaster.       0                0          0             0     0        0     0           1         0
                Real
Asignado         nmap. perl. phf. pod. portsweep. rootkit. satan. spy. teardrop. warezclient. warezmaster.
  back.              0     0    0    0          0        0      0    0         0            0            0
  buffer_overflow.   0     0    0    0          0        1      0    0         0            0            0
  ftp_write.         0     0    0    0          0        0      0    0         0            0            0
  guess_passwd.      0     0    0    0          0        0      0    0         0            0            0
  imap.              0     0    0    0          0        0      0    0         0            0            0
  ipsweep.           0     0    0    0          1        0      1    0         0            0            0
  land.              0     0    0    0          0        0      0    0         0            0            0
  loadmodule.        0     0    0    0          0        0      0    0         0            0            0
  multihop.          0     0    0    0          0        0      0    0         0            0            1
  nmap.             57     0    0    0          0        0      0    0         0            0            0
  perl.              0     0    0    0          0        0      0    0         0            0            0
  phf.               0     0    1    0          0        0      0    0         0            0            0
  pod.               0     0    0   66          0        0      0    0         0            0            0
  portsweep.         0     0    0    0        258        1      1    0         0            0            0
  rootkit.           0     0    0    0          0        0      0    0         0            0            0
  satan.             0     0    0    0          1        0    395    0         0            0            0
  spy.               0     0    0    0          0        0      0    0         0            0            0
  teardrop.          0     0    0    0          0        0      0    0       244            0            0
  warezclient.       0     0    0    0          0        0      0    0         0          255            0
  warezmaster.       0     0    0    0          0        0      0    0         0            0            4

Correctamente clasificados: 99.36 %
```

Ilustración 8. Prueba con algoritmo C5.0 y 8752 muestras (de las cuales, 2179 muestras para test).

A continuación, podemos ver el resultado del algoritmo "Máquina de vectores de soporte" o SVM (Support Vector Machines, en inglés). SVM es un conjunto de algoritmos de aprendizaje supervisado desarrollados por Vladimir Vapnik y su equipo en los laboratorios AT&T.

Estos métodos están propiamente relacionados con problemas de clasificación y regresión. Dado un conjunto de ejemplos de entrenamiento (de muestras) podemos etiquetar las clases y entrenar una SVM para construir un modelo que prediga la clase de una nueva muestra. Intuitivamente, una SVM es un modelo que representa a los puntos de muestra en el espacio, separando las clases a 2 espacios lo más amplios posibles mediante un hiperplano de separación definido como el vector entre los 2 puntos, de las 2 clases, más cercanos al que se llama vector soporte. Cuando las nuevas muestras se ponen en correspondencia con dicho modelo, en función de los espacios a los que pertenezcan, pueden ser clasificadas a una o la otra clase («Máquinas de vectores de soporte», 2018).

Más formalmente, una SVM construye un hiperplano o conjunto de hiperplanos en un espacio de dimensionalidad muy alta (o incluso infinita) que puede ser utilizado en problemas de clasificación o regresión. Una buena separación entre las clases permitirá una clasificación correcta («Máquinas de vectores de soporte», 2018).

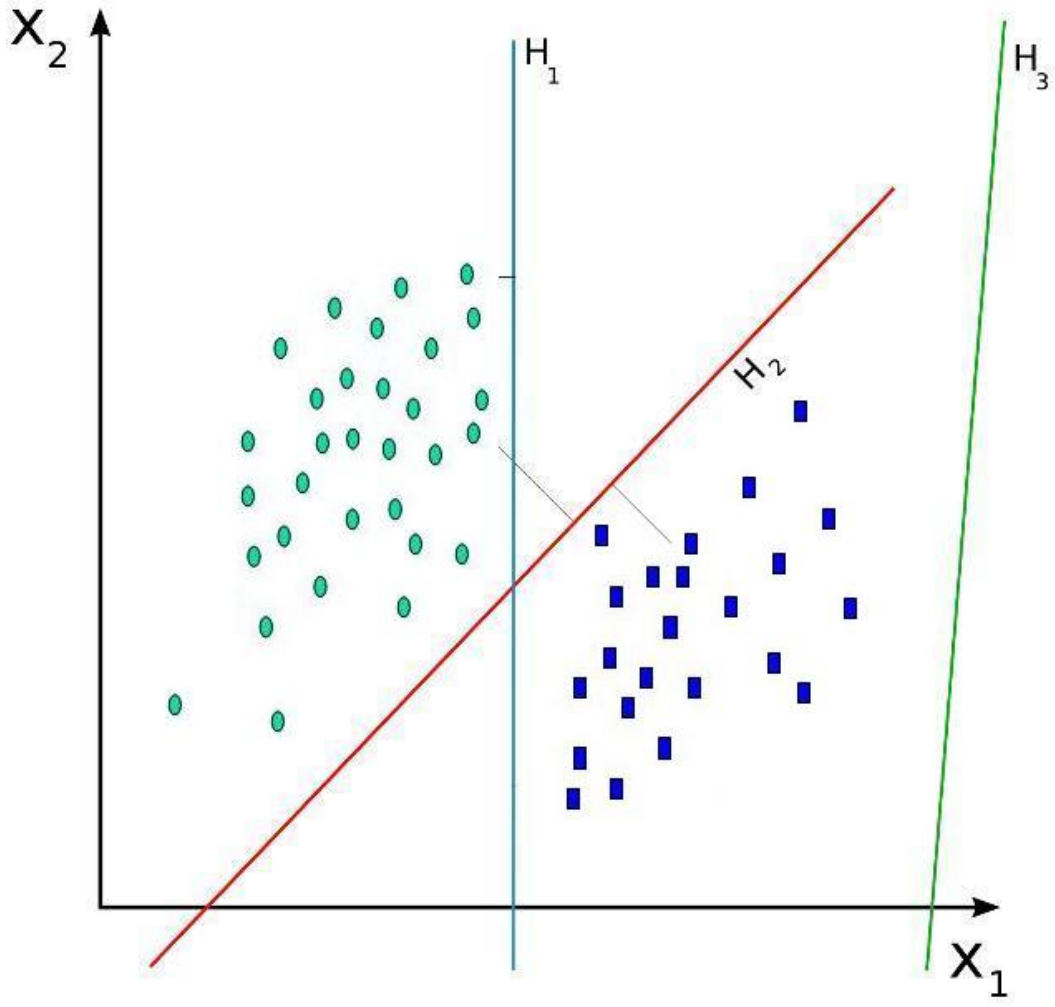


Ilustración 9. Los hiperplanos posibles en SVM pueden ser infinitos. Visión en 2D.

A continuación, podemos ver el resultado del algoritmo SVM, el cual añadiremos al final de cada una de las pruebas a modo de mejora en el trabajo.

```
** SVM
                Real
Asignado         back. buffer_overflow. ftp_write. guess_passwd. imap. ipsweep. land. loadmodule. multihop.
  back.            535               0          0             0     0        0     0           0         0
  buffer_overflow.   0               1          0             0     0        0     0           0         0
  ftp_write.         0               0          0             0     0        0     0           0         0
  guess_passwd.      0               0          0            12     0        0     0           0         0
  imap.              0               0          0             0     2        0     0           0         0
  ipsweep.           0               0          0             0     0      306     0           0         0
  land.              0               0          0             0     0        0     4           0         0
  loadmodule.        0               0          0             0     0        0     0           0         0
  multihop.          0               0          0             0     0        0     0           0         0
  nmap.              0               0          0             0     0        4     0           0         0
  perl.              0               0          0             0     0        0     0           0         0
  phf.               0               0          0             0     0        0     0           0         0
  pod.               0               0          0             0     0        0     0           0         0
  portsweep.         0               0          0             0     0        0     0           0         0
  rootkit.           0               0          0             0     0        0     0           0         0
  satan.             0               0          0             0     0        0     0           0         0
  spy.               0               0          0             0     0        0     0           0         0
  teardrop.          0               0          0             0     0        0     0           0         0
  warezclient.      15               6          2             1     1        1     1           2         1
  warezmaster.       0               0          0             0     0        0     0           0         0
                Real
Asignado         nmap. perl. phf. pod. portsweep. rootkit. satan. spy. teardrop. warezclient. warezmaster.
  back.              0     0     0    0         0        0      0    0         0            0           0
  buffer_overflow.   0     0     0    0         0        0      0    0         0            0           0
  ftp_write.         0     0     0    0         0        0      0    0         0            0           0
  guess_passwd.      0     0     0    0         0        0      0    0         0            0           0
  imap.              0     0     0    0         0        0      0    0         0            0           0
  ipsweep.          20     0     0    0         3        0      1    0         0            0           0
  land.              0     0     0    0         0        0      0    0         0            0           0
  loadmodule.        0     0     0    0         0        0      0    0         0            0           0
  multihop.          0     0     0    0         0        0      0    0         0            0           0
  nmap.             26     0     0    0         1        0      0    0         0            0           0
  perl.              0     0     0    0         0        0      0    0         0            0           0
  phf.               0     0     0    0         0        0      0    0         0            0           0
  pod.               0     0     0   66         0        0      0    0         0            0           0
  portsweep.         3     0     0    0       217        0     25    0         0            0           0
  rootkit.           0     0     0    0         0        0      0    0         0            0           0
  satan.             0     0     0    0        14        0    365    0         0            1           0
  spy.               0     0     0    0         0        0      0    0         0            0           0
  teardrop.          0     0     0    0         0        0      0    0       242            0           0
  warezclient.       8     0     1    0        25        2      6    0         2          254           5
  warezmaster.       0     0     0    0         0        0      0    0         0            0           0

Correctamente clasificados: 93.08 %
```

Ilustración 10. Prueba con Máquina Vector Soporte (SVM) y 8752 muestras (de las cuales, 2181 muestras para test).

En este Data Set, la red neuronal ha fallado muchísimo, ya que muchos ataques no han sido identificados en ningún momento, llegando a detectar únicamente paquetes de los tipos back, guess_passwd, ipsweep, nmap, portsweep, satan, teardrop y warezclient. La tasa paquetes correctamente clasificados ha sido del 25,36%, siendo muy inferior a cualquier clasificador de árbol de decisión como es C5.0 con un 99,36% o como en este caso, hemos incluido también, SVM (Máquina Vector Soporte) con un 93,08%. Hemos podido comprobar que el algoritmo C5.0 ha obtenido una excelente tasa de clasificación al igual que el algoritmo SVM, siendo este último unos puntos peor que C5.0. En este caso C5.0 es el mejor

Segunda prueba de clasificación. Data Set de 35021 muestras

Para esta prueba, hemos eliminado una parte de las muestras del Data Set de la Ilustración 5 que contiene el 10% del Data Set total. Hemos reducido el número de muestras en Normal (dejando 7218), Smurf (dejando 10790) y Neptune (dejando 8201), obteniendo el siguiente Data Set con 35021 muestras:

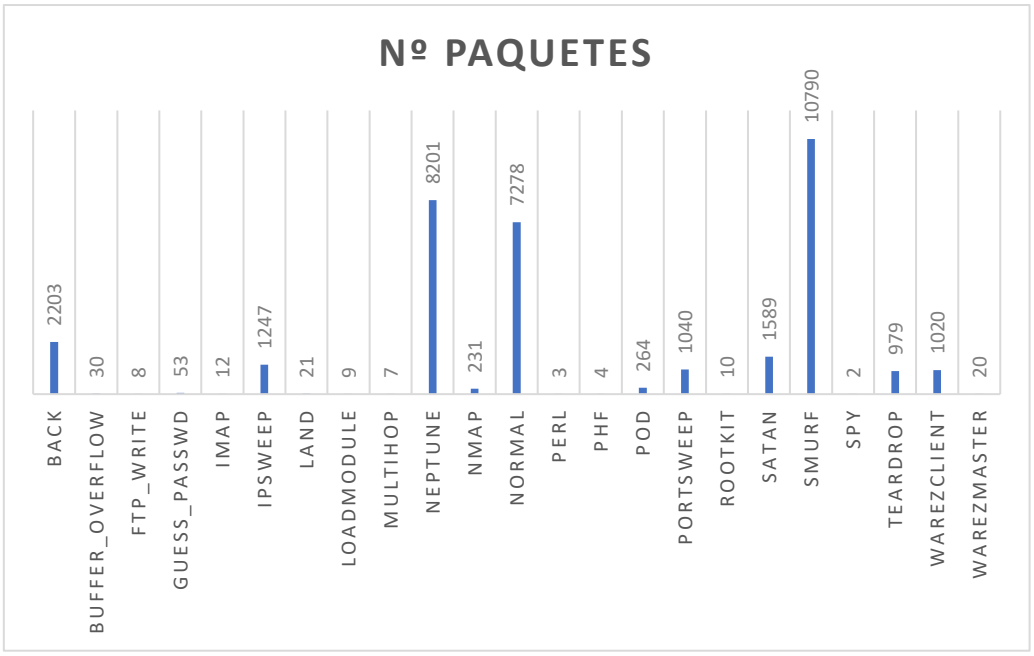


Ilustración 11. Detalle del número de paquetes utilizados.

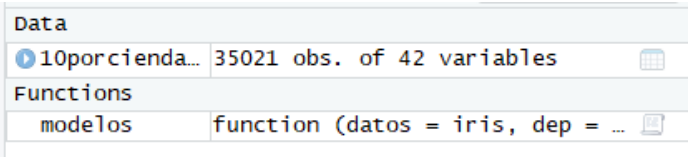


Ilustración 12. Datos cargados en R-Studio.

```
** Red neuronal: nnet
                Real
Asignado        back. buffer_overflow. ftp_write. guess_passwd. imap. ipsweep. land.
  back.           544               0          0             0     0        0     0
  ipsweep.          0               0          0             0     0      286     0
  land.             0               0          0             0     0        2     4
  normal.           6               7          2            13     2        2     0
  portsweep.        0               0          0             0     1        0     1
  smurf.            0               0          0             0     0       20     0
  warezclient.      0               0          0             0     0        1     0
                Real
Asignado        loadmodule. multihop. neptune. nmap. normal. perl. phf. pod.
  back.                 0         0        0     0       6     0    0    0
  ipsweep.              0         0        0    25       9     0    0    0
  land.                 0         0        0     0       0     0    0    0
  normal.               2         1        1     5    1705     0    1   11
  portsweep.            0         0        0    27      47     0    0    0
  smurf.                0         0     2049     0      13     0    0    0
  warezclient.          0         0        0     0      39     0    0   55
                Real
Asignado        portsweep. rootkit. satan. smurf. spy. teardrop. warezclient.
  back.                 0        0      0      0    0         0            0
  ipsweep.              0        0      0      0    0         0            2
  land.                 0        0      0      0    0         0            0
  normal.               0        2      1      9    0         0           72
  portsweep.          217        0      5      0    0        12            0
  smurf.               43        0    391   2688    0       232            0
  warezclient.          0        0      0      0    0         0          181
                Real
Asignado        warezmaster.
  back.                   0
  ipsweep.                0
  land.                   0
  normal.                 5
  portsweep.              0
  smurf.                  0
  warezclient.            0

Correctamente clasificados: 6.61 %
```

Ilustración 13. Resultados clasificación Red Neuronal con 8747 paquetes para test.

Como siempre, el conjunto de datos se divide en aproximadamente un 75% para entrenamiento (26274 paquetes) y el restante 25% para test (8747 paquetes). Podemos comprobar los resultados pésimos que hemos obtenido usando la red neuronal con este Data Set. Podemos ver que, en esta nueva prueba, la red neuronal ha caído aún más en

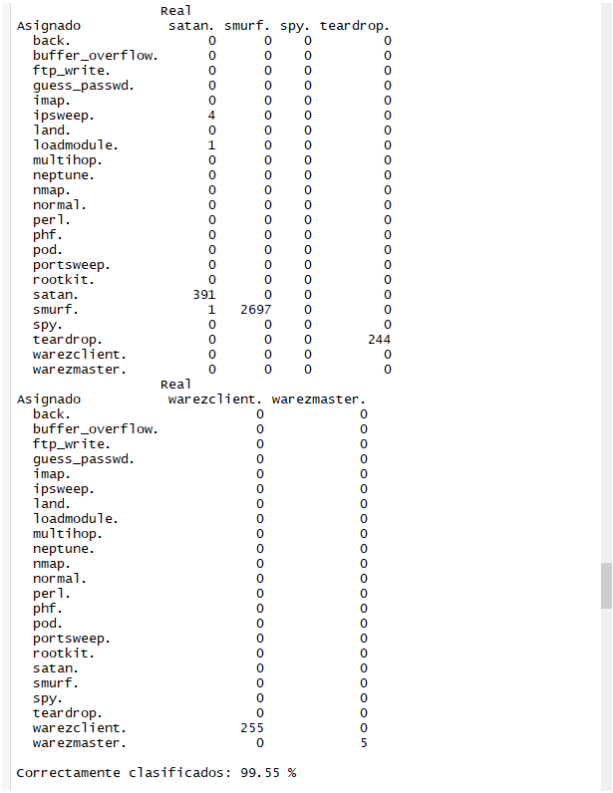
la tasa que en la primera prueba, llegando al 6,61%. Al obtener este resultado, hemos vuelto a reiniciar esta prueba en otro equipo para ver si obtenemos un resultado diferente, ya que prácticamente no ha acertado nada. Es posible que la red neuronal haya llegado a un nivel de sobre aprendizaje muy elevado.

A continuación, podemos ver los resultados de los algoritmos C5.0 y SVM, los cuales son muchísimo mejores que la red neuronal, como podemos comprobar a continuación.

```
** Árbol de clasificación C5.0
                    Real
Asignado             back. buffer_overflow.
  back.                550                0
  buffer_overflow.       0                6
  ftp_write.             0                0
  guess_passwd.          0                0
  imap.                  0                0
  ipsweep.               0                0
  land.                  0                0
  loadmodule.            0                0
  multihop.              0                1
  neptune.               0                0
  nmap.                  0                0
  normal.                0                0
  perl.                  0                0
  phf.                   0                0
  pod.                   0                0
  portsweep.             0                0
  rootkit.               0                0
  satan.                 0                0
  smurf.                 0                0
  spy.                   0                0
  teardrop.              0                0
  warezclient.           0                0
  warezmaster.           0                0
                    Real
Asignado             ftp_write. guess_passwd.
  back.                       0             0
  buffer_overflow.            0             0
  ftp_write.                  0             0
  guess_passwd.               0            13
  imap.                       0             0
  ipsweep.                    0             0
  land.                       0             0
  loadmodule.                 0             0
  multihop.                   1             0
  neptune.                    0             0
  nmap.                       0             0
  normal.                     1             0
  perl.                       0             0
  phf.                        0             0
  pod.                        0             0
  portsweep.                  0             0
  rootkit.                    0             0
  satan.                      0             0
  smurf.                      0             0
  spy.                        0             0
  teardrop.                   0             0
  warezclient.                0             0
  warezmaster.                0             0
```

```
                  Real
Asignado           imap. ipsweep. land.
  back.                0         0     0
  buffer_overflow.     0         0     0
  ftp_write.           0         0     0
  guess_passwd.        0         0     0
  imap.                3         0     0
  ipsweep.             0       304     0
  land.                0         0     3
  loadmodule.          0         0     0
  multihop.            0         0     0
  neptune.             0         0     1
  nmap.                0         0     0
  normal.              0         3     1
  perl.                0         0     0
  phf.                 0         0     0
  pod.                 0         0     0
  portsweep.           0         0     0
  rootkit.             0         4     0
  satan.               0         0     0
  smurf.               0         0     0
  spy.                 0         0     0
  teardrop.            0         0     0
  warezclient.         0         0     0
  warezmaster.         0         0     0
                  Real
Asignado           loadmodule. multihop.
  back.                      0         0
  buffer_overflow.           0         0
  ftp_write.                 0         0
  guess_passwd.              0         0
  imap.                      0         0
  ipsweep.                   0         0
  land.                      0         0
  loadmodule.                0         0
  multihop.                  2         0
  neptune.                   0         0
  nmap.                      0         0
  normal.                    0         0
  perl.                      0         0
  phf.                       0         0
  pod.                       0         0
  portsweep.                 0         0
  rootkit.                   0         0
  satan.                     0         0
  smurf.                     0         0
  spy.                       0         0
  teardrop.                  0         0
  warezclient.               0         0
  warezmaster.               0         1
```

```
                  Real
Asignado           neptune. nmap. normal. perl.
  back.                   0     0       1     0
  buffer_overflow.        0     0       1     0
  ftp_write.              0     0       0     0
  guess_passwd.           0     0       0     0
  imap.                   0     0       0     0
  ipsweep.                0     2       0     0
  land.                   0     0       0     0
  loadmodule.             0     0       0     0
  multihop.               0     0       0     0
  neptune.             2050     0       0     0
  nmap.                   0    55       1     0
  normal.                 0     0    1808     0
  perl.                   0     0       0     0
  phf.                    0     0       0     0
  pod.                    0     0       1     0
  portsweep.              0     0       0     0
  rootkit.                0     0       0     0
  satan.                  0     0       0     0
  smurf.                  0     0       3     0
  spy.                    0     0       2     0
  teardrop.               0     0       0     0
  warezclient.            0     0       2     0
  warezmaster.            0     0       0     0
                  Real
Asignado           phf. pod. portsweep. rootkit.
  back.               0    0          0        0
  buffer_overflow.    0    0          0        0
  ftp_write.          0    0          0        1
  guess_passwd.       0    0          0        0
  imap.               0    0          0        0
  ipsweep.            0    0          0        0
  land.               0    0          0        0
  loadmodule.         0    0          0        0
  multihop.           0    0          0        0
  neptune.            0    0          1        0
  nmap.               0    0          0        0
  normal.             0    0          0        0
  perl.               0    0          0        0
  phf.                1    0          0        0
  pod.                0   66          0        0
  portsweep.          0    0        256        0
  rootkit.            0    0          1        1
  satan.              0    0          2        0
  smurf.              0    0          0        0
  spy.                0    0          0        0
  teardrop.           0    0          0        0
  warezclient.        0    0          0        0
  warezmaster.        0    0          0        0
```

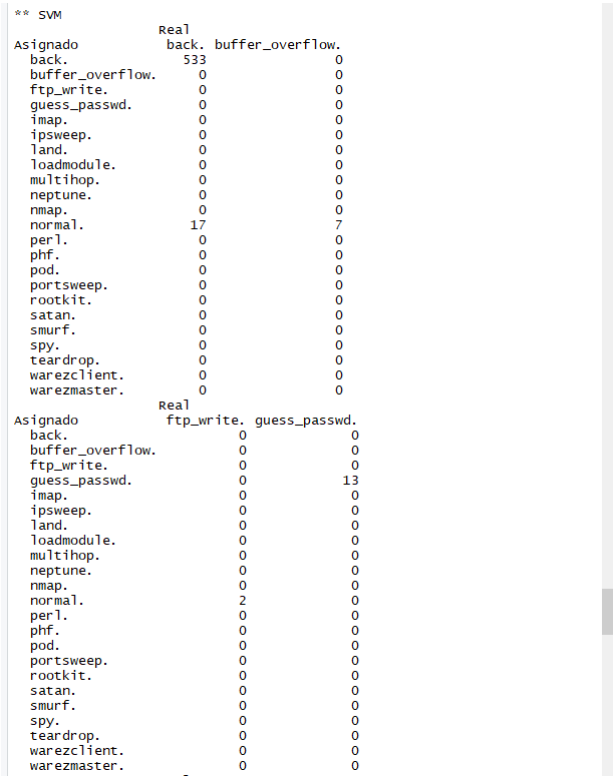


```
                    Real
Asignado             satan. smurf. spy. teardrop.
  back.                   0      0    0         0
  buffer_overflow.        0      0    0         0
  ftp_write.              0      0    0         0
  guess_passwd.           0      0    0         0
  imap.                   0      0    0         0
  ipsweep.                4      0    0         0
  land.                   0      0    0         0
  loadmodule.             1      0    0         0
  multihop.               0      0    0         0
  neptune.                0      0    0         0
  nmap.                   0      0    0         0
  normal.                 0      0    0         0
  perl.                   0      0    0         0
  phf.                    0      0    0         0
  pod.                    0      0    0         0
  portsweep.              0      0    0         0
  rootkit.                0      0    0         0
  satan.                391      0    0         0
  smurf.                  1   2697    0         0
  spy.                    0      0    0         0
  teardrop.               0      0    0       244
  warezclient.            0      0    0         0
  warezmaster.            0      0    0         0
                    Real
Asignado             warezclient. warezmaster.
  back.                         0            0
  buffer_overflow.              0            0
  ftp_write.                    0            0
  guess_passwd.                 0            0
  imap.                         0            0
  ipsweep.                      0            0
  land.                         0            0
  loadmodule.                   0            0
  multihop.                     0            0
  neptune.                      0            0
  nmap.                         0            0
  normal.                       0            0
  perl.                         0            0
  phf.                          0            0
  pod.                          0            0
  portsweep.                    0            0
  rootkit.                      0            0
  satan.                        0            0
  smurf.                        0            0
  spy.                          0            0
  teardrop.                     0            0
  warezclient.                255            0
  warezmaster.                  0            5

Correctamente clasificados: 99.55 %
```

Ilustración 14. Resultados del algoritmo C5.0, con 8747 paquetes para test.

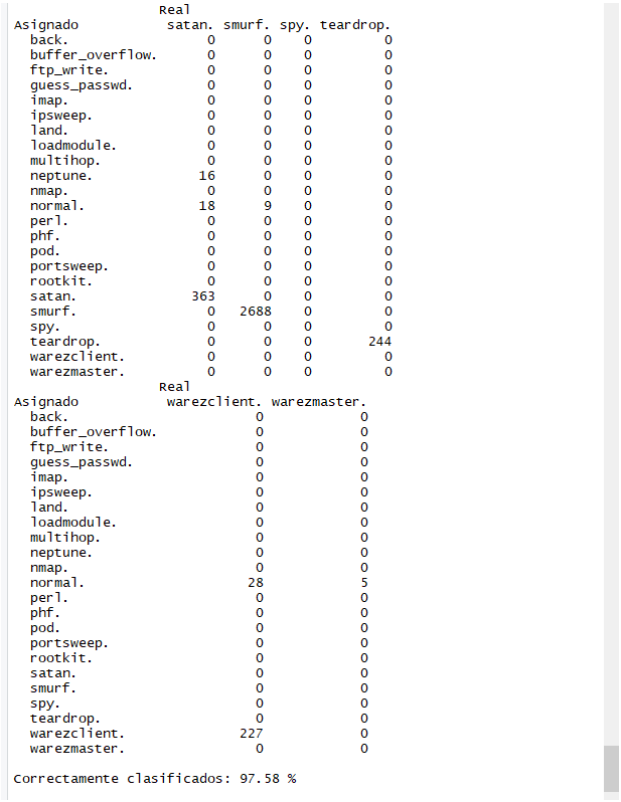
Vemos que C5.0 ha arrojado un resultado de un 99,55% de paquetes correctamente clasificados, por lo que podemos decir que se trata de un método muy válido para este tipo de tareas. Se han usado 26274 paquetes para entrenamiento y 8747 paquetes para test. A continuación, vemos los resultados arrojados por SVM.



```
** SVM
                    Real
Asignado             back. buffer_overflow.
  back.                533                0
  buffer_overflow.       0                0
  ftp_write.             0                0
  guess_passwd.          0                0
  imap.                  0                0
  ipsweep.               0                0
  land.                  0                0
  loadmodule.            0                0
  multihop.              0                0
  neptune.               0                0
  nmap.                  0                0
  normal.               17                7
  perl.                  0                0
  phf.                   0                0
  pod.                   0                0
  portsweep.             0                0
  rootkit.               0                0
  satan.                 0                0
  smurf.                 0                0
  spy.                   0                0
  teardrop.              0                0
  warezclient.           0                0
  warezmaster.           0                0
                    Real
Asignado             ftp_write. guess_passwd.
  back.                       0             0
  buffer_overflow.            0             0
  ftp_write.                  0             0
  guess_passwd.               0            13
  imap.                       0             0
  ipsweep.                    0             0
  land.                       0             0
  loadmodule.                 0             0
  multihop.                   0             0
  neptune.                    0             0
  nmap.                       0             0
  normal.                     2             0
  perl.                       0             0
  phf.                        0             0
  pod.                        0             0
  portsweep.                  0             0
  rootkit.                    0             0
  satan.                      0             0
  smurf.                      0             0
  spy.                        0             0
  teardrop.                   0             0
  warezclient.                0             0
  warezmaster.                0             0
```

```
                  Real
Asignado           imap. ipsweep. land.
  back.                0         0     0
  buffer_overflow.     0         0     0
  ftp_write.           0         0     0
  guess_passwd.        0         0     0
  imap.                2         0     0
  ipsweep.             0       299     1
  land.                0         0     4
  loadmodule.          0         0     0
  multihop.            0         0     0
  neptune.             0         0     0
  nmap.                0         9     0
  normal.              1         3     0
  perl.                0         0     0
  phf.                 0         0     0
  pod.                 0         0     0
  portsweep.           0         0     0
  rootkit.             0         0     0
  satan.               0         0     0
  smurf.               0         0     0
  spy.                 0         0     0
  teardrop.            0         0     0
  warezclient.         0         0     0
  warezmaster.         0         0     0
                  Real
Asignado           loadmodule. multihop.
  back.                    0          0
  buffer_overflow.         0          0
  ftp_write.               0          0
  guess_passwd.            0          0
  imap.                    0          0
  ipsweep.                 0          0
  land.                    0          0
  loadmodule.              0          0
  multihop.                0          0
  neptune.                 0          0
  nmap.                    0          0
  normal.                  2          1
  perl.                    0          0
  phf.                     0          0
  pod.                     0          0
  portsweep.               0          0
  rootkit.                 0          0
  satan.                   0          0
  smurf.                   0          0
  spy.                     0          0
  teardrop.                0          0
  warezclient.             0          0
  warezmaster.             0          0
```

```
                  Real
Asignado           neptune. nmap. normal. perl.
  back.                   0     0       0     0
  buffer_overflow.        0     0       0     0
  ftp_write.              0     0       0     0
  guess_passwd.           0     0       0     0
  imap.                   0     0       0     0
  ipsweep.                0    14       0     0
  land.                   1     0       1     0
  loadmodule.             0     0       0     0
  multihop.               0     0       0     0
  neptune.             2039     0       0     0
  nmap.                   0    26       0     0
  normal.                 8    12    1817     0
  perl.                   0     0       0     0
  phf.                    0     0       0     0
  pod.                    0     0       0     0
  portsweep.              0     5       0     0
  rootkit.                0     0       0     0
  satan.                  2     0       0     0
  smurf.                  0     0       0     0
  spy.                    0     0       0     0
  teardrop.               0     0       1     0
  warezclient.            0     0       0     0
  warezmaster.            0     0       0     0
                  Real
Asignado           phf. pod. portsweep. rootkit.
  back.               0    0          0        0
  buffer_overflow.    0    0          0        0
  ftp_write.          0    0          0        0
  guess_passwd.       0    0          0        0
  imap.               0    0          0        0
  ipsweep.            0    0          1        0
  land.               0    0          0        0
  loadmodule.         0    0          0        0
  multihop.           0    0          0        0
  neptune.            0    0          9        0
  nmap.               0    0          0        0
  normal.             1    1         24        2
  perl.               0    0          0        0
  phf.                0    0          0        0
  pod.                0   65          0        0
  portsweep.          0    0        215        0
  rootkit.            0    0          0        0
  satan.              0    0         11        0
  smurf.              0    0          0        0
  spy.                0    0          0        0
  teardrop.           0    0          0        0
  warezclient.        0    0          0        0
  warezmaster.        0    0          0        0
```



```
                    Real
Asignado             satan. smurf. spy. teardrop.
  back.                   0      0    0         0
  buffer_overflow.        0      0    0         0
  ftp_write.              0      0    0         0
  guess_passwd.           0      0    0         0
  imap.                   0      0    0         0
  ipsweep.                0      0    0         0
  land.                   0      0    0         0
  loadmodule.             0      0    0         0
  multihop.               0      0    0         0
  neptune.               16      0    0         0
  nmap.                   0      0    0         0
  normal.                18      9    0         0
  perl.                   0      0    0         0
  phf.                    0      0    0         0
  pod.                    0      0    0         0
  portsweep.              0      0    0         0
  rootkit.                0      0    0         0
  satan.                363      0    0         0
  smurf.                  0   2688    0         0
  spy.                    0      0    0         0
  teardrop.               0      0    0       244
  warezclient.            0      0    0         0
  warezmaster.            0      0    0         0
                    Real
Asignado             warezclient. warezmaster.
  back.                         0            0
  buffer_overflow.              0            0
  ftp_write.                    0            0
  guess_passwd.                 0            0
  imap.                         0            0
  ipsweep.                      0            0
  land.                         0            0
  loadmodule.                   0            0
  multihop.                     0            0
  neptune.                      0            0
  nmap.                         0            0
  normal.                      28            5
  perl.                         0            0
  phf.                          0            0
  pod.                          0            0
  portsweep.                    0            0
  rootkit.                      0            0
  satan.                        0            0
  smurf.                        0            0
  spy.                          0            0
  teardrop.                     0            0
  warezclient.                227            0
  warezmaster.                  0            0

Correctamente clasificados: 97.58 %
```

Ilustración 15. Resultados del algoritmo SVM con 8747 paquetes para test.

Finalmente vemos que con un 97,58% de paquetes correctamente clasificados, SVM es el segundo método que mejor ha clasificado en esta prueba. Podemos observar que el algoritmo ha seleccionado también 26274 paquetes para entrenamiento y 8747 paquetes para test.

Tercera prueba de clasificación. Data Set de 35021 muestras, eliminadas columnas protocol_type, service y flag y 25% muestras para aprendizaje

Hemos procedido a realizar la prueba eliminando las columnas “protocol_type”, “service” y “flag” con tipos de datos de texto, reduciendo el número de columnas a 39. También se reduce al 25% la cantidad de paquetes para entrenamiento, siendo el 75% para test. Obtenemos los siguientes resultados.

```
> modelos(datos=`10porciendatosConCabeceraSinColumnasDatos`,dep=39,p=0.25)
Loading required package: lattice
Loading required package: ggplot2
** Red neuronal: nnet
          Real
Asignado     back. buffer_overflow. ftp_write. guess_passwd. imap. ipsweep. land. loadmodule. multihop. neptune. nmap. normal. perl. phf. pod. portsweep. rootkit.
  ipsweep.       0               0          1             0     1      920    15           0         0       15   156      99     0    0    0          9         2
  neptune.       0               0          0             0     6        5     0           0         0     6131     1       0     0    0    0         15         0
  normal.        0               0          0             0     0        6     0           0         0        0     0     375     0    0    0          0         0
  portsweep.     0               0          0             0     0        0     0           0         0        4     0       5     0    0    0        691         0
  satan.         0               0          0             0     0        0     0           0         0        0     0       1     0    0    0         64         1
  smurf.      1652              22          5            39     2        4     0           6         5        0    16    4977     2    3  198          1         4
  teardrop.      0               0          0             0     0        0     0           0         0        0     0       1     0    0    0          0         0
          Real
Asignado     satan. smurf. spy. teardrop. warezclient. warezmaster.
  ipsweep.        0      0    0         2           5            0
  neptune.        4      0    0         0           0            0
  normal.         6      0    0         0          16            0
  portsweep.    131      0    0        75           0            0
  satan.       1039      0    0         5           0            0
  smurf.          8   8092    1         1         744           15
  teardrop.       3      0    0       651           0            0

Correctamente clasificados: 0.02 %
```

Ilustración 16. Resultado de la Red Neuronal con 25% de muestras para entrenamiento y 3 columnas menos.

Obtenemos un resultado muy malo para la red neuronal. Teniendo una tasa de clasificados correctos del 0,02%.

A continuación, vemos el resultado del algoritmo C5.0.

```
** Árbol de clasificación C5.0
                Real
Asignado         back. buffer_overflow. ftp_write. guess_passwd. imap. ipsweep. land. loadmodule. multihop. neptune. nmap. normal. perl. phf. pod. portsweep.
  back.          1647               3          1             1     1        0     0           0         0        0    0       2     0    0    0          0
  buffer_overflow.  0              17          1             0     0        3     0           4         2        0    0       1     0    0    0          0
  ftp_write.        0               0          1             0     0        0     0           0         0        0    0       0     0    3    0          0
  guess_passwd.     0               0          0            37     0        0     0           0         0        0    0       0     0    0    0          0
  imap.             0               0          0             0     6        0     0           0         0        0    0       1     0    0    0          0
  ipsweep.          0               0          0             0     0      925     0           0         0        0    0       0     0    0    0          6
  land.             0               0          0             0     0        0    15           0         0        0    0       0     0    0    0          0
  loadmodule.       0               0          0             0     0        0     0           0         0        0    0       0     0    0    0          0
  multihop.         0               0          0             0     0        0     0           0         0        0    0       0     0    0    0          0
  neptune.          0               0          0             0     1        0     0           0         0     6150    0       1     0    0    0          0
  nmap.             0               0          0             0     0        4     0           0         1        0  169      12     0    0    0          1
  normal.           5               2          2             1     1        3     0           2         2        0    4    5420     2    0    0          1
  perl.             0               0          0             0     0        0     0           0         0        0    0       0     0    0    0          0
  phf.              0               0          0             0     0        0     0           0         0        0    0       0     0    0    0          0
  pod.              0               0          0             0     0        0     0           0         0        0    0       0     0    0  194          0
  portsweep.        0               0          0             0     0        0     0           0         0        0    0       2     0    0    0        758
  rootkit.          0               0          0             0     0        0     0           0         0        0    0       0     0    0    0          0
  satan.            0               0          0             0     0        0     0           0         0        0    0       1     0    0    0         14
  smurf.            0               0          0             0     0        0     0           0         0        0    0       0     0    0    0          0
  spy.              0               0          0             0     0        0     0           0         0        0    0       0     0    0    0          0
  teardrop.         0               0          0             0     0        0     0           0         0        0    0       0     0    0    0          0
  warezclient.      0               0          1             0     0        0     0           0         0        0    0      18     0    0    0          0
  warezmaster.      0               0          0             0     0        0     0           0         0        0    0       0     0    0    0          0
                Real
Asignado         rootkit. satan. smurf. spy. teardrop. warezclient. warezmaster.
  back.                 1      0      0    0         0           1            0
  buffer_overflow.      1      0      0    0         0           0            0
  ftp_write.            0      0      0    0         0           0            0
  guess_passwd.         0      0      0    0         0           0            0
  imap.                 0      0      0    0         0           0            0
  ipsweep.              0      4      0    0         0           0            0
  land.                 0      0      0    0         0           0            0
  loadmodule.           0      0      0    0         0           0            0
  multihop.             1      0      0    0         0           0            2
  neptune.              0      0      0    0         0           0            0
  nmap.                 1      0      0    0         0           0            0
  normal.               3      7      3    1         0          12            1
  perl.                 0      0      0    0         0           0            0
  phf.                  0      0      0    0         0           0            0
  pod.                  0      0      0    0         0           0            0
  portsweep.            0      0      0    0         0           0            0
  rootkit.              0      0      0    0         0           0            0
  satan.                0   1180      0    0         0           1            0
  smurf.                0      0   8089    0         0           0            0
  spy.                  0      0      0    0         0           0            0
  teardrop.             0      0      0    0       734           0            0
  warezclient.          0      0      0    0         0         751            1
  warezmaster.          0      0      0    0         0           0           11

Correctamente clasificados: 99.41 %
```

Ilustración 17. Resultado del algoritmo C5.0 con 25% de muestras para entrenamiento y 3 columnas menos.

Volvemos a comprobar que el algoritmo C5.0 funciona muy bien incluso reduciendo la cantidad de muestras de aprendizaje al 25% de los paquetes. C5.0 dispone de una tasa de aciertos del 99,41%. Pasamos a ver el resultado del algoritmo SVM.

```
** SVM
                Real
Asignado         back. buffer_overflow. ftp_write. guess_passwd. imap. ipsweep. land. loadmodule. multihop. neptune. nmap. normal. perl. phf. pod. portsweep.
  back.           1593               0          0             0     0        0     0           0         0        0     0       0     0     0    0          0
  buffer_overflow.   0               0          0             0     0        0     0           0         0        0     0       0     0     0    0          0
  ftp_write.         0               0          0             0     0        0     0           0         0        0     0       0     0     0    0          0
  guess_passwd.      0               0          0            29     0        0     0           0         0        0     0       0     0     0    0          0
  imap.              0               0          0             0     0        0     0           0         0        0     0       0     0     0    0          0
  ipsweep.           0               0          0             0     0      792     2           0         0        0    27       0     0     0    0          2
  land.              0               0          0             0     0        0    12           0         0        1     0       0     0     0    0          0
  loadmodule.        0               0          0             0     0        0     0           0         0        0     0       0     0     0    0          0
  multihop.          0               0          0             0     0        0     0           0         0        0     0       0     0     0    0          0
  neptune.           0               0          0             0     0        0     0           0         0     6087     0       0     0     0    0         32
  nmap.              0               0          0             0     0        3     0           0         0        0    63       0     0     0    0          1
  normal.           59              22          6            10     9      140     1           6         5       62    82    5457     2     3   10        122
  perl.              0               0          0             0     0        0     0           0         0        0     0       0     0     0    0          0
  phf.               0               0          0             0     0        0     0           0         0        0     0       0     0     0    0          0
  pod.               0               0          0             0     0        0     0           0         0        0     0       0     0     0  188          0
  portsweep.         0               0          0             0     0        0     0           0         0        0     1       0     0     0    0        612
  rootkit.           0               0          0             0     0        0     0           0         0        0     0       0     0     0    0          0
  satan.             0               0          0             0     0        0     0           0         0        0     0       0     0     0    0         11
  smurf.             0               0          0             0     0        0     0           0         0        0     0       0     0     0    0          0
  spy.               0               0          0             0     0        0     0           0         0        0     0       0     0     0    0          0
  teardrop.          0               0          0             0     0        0     0           0         0        0     0       1     0     0    0          0
  warezclient.       0               0          0             0     0        0     0           0         0        0     0       0     0     0    0          0
  warezmaster.       0               0          0             0     0        0     0           0         0        0     0       0     0     0    0          0
                Real
Asignado         rootkit. satan. smurf. spy. teardrop. warezclient. warezmaster.
  back.                 0      0      0    0         0           0            0
  buffer_overflow.      0      0      0    0         0           0            0
  ftp_write.            0      0      0    0         0           0            0
  guess_passwd.         0      0      0    0         0           0            0
  imap.                 0      0      0    0         0           0            0
  ipsweep.              0      0      0    0         0           0            0
  land.                 0      0      0    0         0           0            0
  loadmodule.           0      0      0    0         0           0            0
  multihop.             0      0      0    0         0           0            0
  neptune.              0     49      0    0         0           0            0
  nmap.                 0      0      0    0         0           0            0
  normal.               6    116     51    1       129         304           15
  perl.                 0      0      0    0         0           0            0
  phf.                  0      0      0    0         0           0            0
  pod.                  0      0      0    0         0           0            0
  portsweep.            0      3      0    0         0           0            0
  rootkit.              0      0      0    0         0           0            0
  satan.                0   1023      0    0         0           0            0
  smurf.                0      0   8041    0         0           0            0
  spy.                  0      0      0    0         0           0            0
  teardrop.             1      0      0    0       605           0            0
  warezclient.          0      0      0    0         0         461            0
  warezmaster.          0      0      0    0         0           0            0

Correctamente clasificados: 95.07 %
```

Ilustración 18. Resultado del algoritmo SVM con 25% de muestras para entrenamiento y 3 columnas menos.

Finalmente podemos ver que el algoritmo SVM se comporta de forma muy correcta, aportando una tasa de clasificados correctamente del 95,07%.

Cuarta prueba de clasificación. Data Set de 35021 muestras, eliminadas columnas protocol_type, service y flag y 50% muestras para aprendizaje

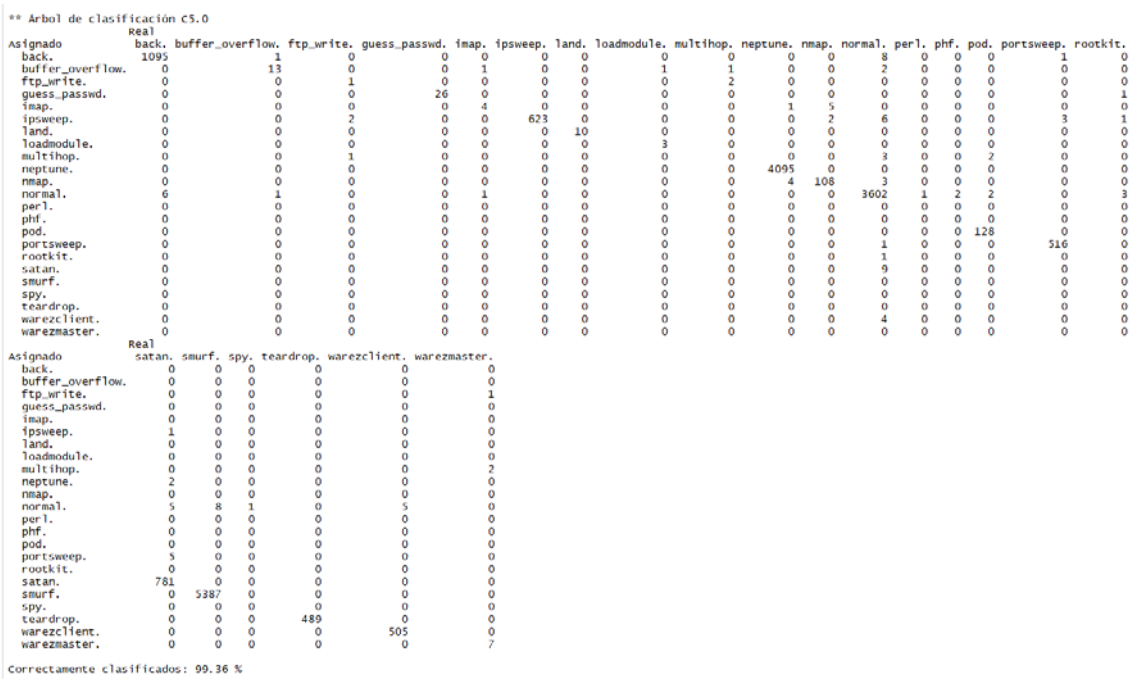
Pasamos a realizar nuestra última prueba en la que aumentamos las muestras de entrenamiento al 50% y se eliminan las columnas “protocol_type”, “service” y “flag”. Con respecto a la red neuronal disponemos de los siguientes resultados.

```
> modelos(datos=`10porciendatosConCabeceraSinColumnasDatos`,dep=39)
Loading required package: lattice
Loading required package: ggplot2
** Red neuronal: nnet
                Real
Asignado         back. buffer_overflow. ftp_write. guess_passwd. imap. ipsweep. land. loadmodule. multihop. neptune. nmap. normal. perl. phf. pod. portsweep. rootkit.
  back.           1094               0          0             0     0        0     0           0         0        0     1      44     0     0   45          0        0
  ipsweep.           0               0          0             0     3      616     0           0         0        0    65      57     0     0    0          0        0
  neptune.           0               0          0             0     0        0     0           0         0     4083     0       0     0     0    0         26        0
  nmap.              0               0          0             0     0        0     0           0         0        0     0       1     0     0    0          2        0
  normal.            7              15          4            26     2        7     7           4         3        3     3    3530     1     2   83          3        4
  portsweep.         0               0          0             0     0        0     2           0         0        4    45       1     0     0    0        479        0
  satan.             0               0          0             0     0        0     1           0         0       10     0       0     0     0    0          8        0
  smurf.             0               0          0             0     0        0     0           0         0        0     0       1     0     0    0          0        0
  teardrop.          0               0          0             0     1        0     0           0         0        0     0       4     0     0    0          2        1
  warezclient.       0               0          0             0     0        0     0           0         0        0     1       1     0     0    4          0        0
                Real
Asignado         satan. smurf. spy. teardrop. warezclient. warezmaster.
  back.             0      0    0         0         283            0
  ipsweep.          0      2    0         5           3            0
  neptune.         58      0    0         2           0            0
  nmap.             1      0    0         0           0            0
  normal.          10      6    1         2         224           10
  portsweep.       78      0    0         0           0            0
  satan.          644      0    0         0           0            0
  smurf.            2   5380    0         0           0            0
  teardrop.         1      7    0       480           0            0
  warezclient.      0      0    0         0           0            0

Correctamente clasificados: 6.27 %
```

Ilustración 19. Resultado de la red neuronal con 50% de muestras para entrenamiento y 3 columnas menos.

Podemos ver que el resultado arrojado por la red neuronal es del 6,27%, la cual no consigue mejorar los resultados para el Data Set de 35021 muestras. Pasamos a ver el resultado del algoritmo C5.0.



** Arbol de clasificación C5.0

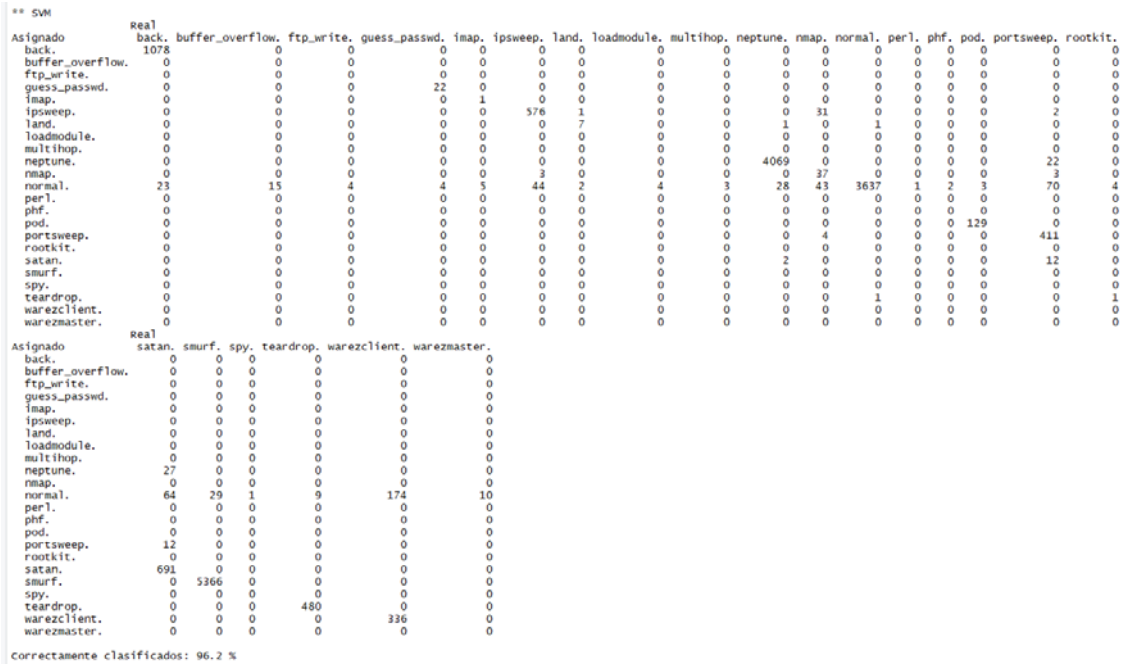
Asignado	Real back.	buffer_overflow.	ftp_write.	guess_passwd.	imap.	ipsweep.	land.	loadmodule.	multihop.	neptune.	nmap.	normal.	perl.	phf.	pod.	portsweep.	rootkit.
back.	1095	1	0	0	0	0	0	0	0	0	0	8	0	0	0	1	0
buffer_overflow.	0	13	0	0	1	0	0	1	1	0	0	2	0	0	0	0	0
ftp_write.	0	0	1	0	0	0	0	0	2	0	0	0	0	0	0	0	0
guess_passwd.	0	0	0	26	0	0	0	0	0	0	0	0	0	0	0	0	1
imap.	0	0	0	0	4	0	0	0	0	1	5	0	0	0	0	0	0
ipsweep.	0	0	2	0	0	623	0	0	0	0	2	6	0	0	0	3	1
land.	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0
loadmodule.	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0
multihop.	0	0	1	0	0	0	0	0	0	0	0	3	0	0	2	0	0
neptune.	0	0	0	0	0	0	0	0	0	4095	0	0	0	0	0	0	0
nmap.	0	0	0	0	0	0	0	0	0	4	108	3	0	0	0	0	0
normal.	6	1	0	0	1	0	0	0	0	0	0	3602	1	2	2	0	3
perl.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
phf.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
pod.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	128	0	0
portsweep.	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	516	0
rootkit.	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
satan.	0	0	0	0	0	0	0	0	0	0	0	9	0	0	0	0	0
smurf.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
spy.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
teardrop.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
warezclient.	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0
warezmaster.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Asignado	Real satan.	smurf.	spy.	teardrop.	warezclient.	warezmaster.
back.	0	0	0	0	0	0
buffer_overflow.	0	0	0	0	0	0
ftp_write.	0	0	0	0	0	1
guess_passwd.	0	0	0	0	0	0
imap.	0	0	0	0	0	0
ipsweep.	1	0	0	0	0	0
land.	0	0	0	0	0	0
loadmodule.	0	0	0	0	0	0
multihop.	0	0	0	0	0	2
neptune.	2	0	0	0	0	0
nmap.	0	0	0	0	0	0
normal.	5	8	1	0	5	0
perl.	0	0	0	0	0	0
phf.	0	0	0	0	0	0
pod.	0	0	0	0	0	0
portsweep.	5	0	0	0	0	0
rootkit.	0	0	0	0	0	0
satan.	781	0	0	0	0	0
smurf.	0	5387	0	0	0	0
spy.	0	0	0	0	0	0
teardrop.	0	0	0	489	0	0
warezclient.	0	0	0	0	505	0
warezmaster.	0	0	0	0	0	7

Correctamente clasificados: 99.36 %

Ilustración 20. Resultado del algoritmo C5.0 con 50% de muestras para entrenamiento y 3 columnas menos.

Podemos ver que el algoritmo C5.0 se mantiene constante en el 99,36%, obteniendo un resultado excelente en la clasificación de ataques de redes de datos. Finalmente mostramos el resultado obtenido con el algoritmo SVM.



** SVM

Asignado	Real back.	buffer_overflow.	ftp_write.	guess_passwd.	imap.	ipsweep.	land.	loadmodule.	multihop.	neptune.	nmap.	normal.	perl.	phf.	pod.	portsweep.	rootkit.
back.	1078	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
buffer_overflow.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ftp_write.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
guess_passwd.	0	0	0	22	0	0	0	0	0	0	0	0	0	0	0	0	0
imap.	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
ipsweep.	0	0	0	0	0	576	1	0	0	0	31	0	0	0	0	2	0
land.	0	0	0	0	0	0	7	0	0	1	0	1	0	0	0	0	0
loadmodule.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
multihop.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
neptune.	0	0	0	0	0	0	0	0	0	4069	0	0	0	0	0	22	0
nmap.	0	0	0	0	0	3	0	0	0	0	37	0	0	0	0	3	0
normal.	23	15	4	4	5	44	2	4	3	28	43	3637	1	2	3	70	4
perl.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
phf.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
pod.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	129	0	0
portsweep.	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	411	0
rootkit.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
satan.	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	12	0
smurf.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
spy.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
teardrop.	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1
warezclient.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
warezmaster.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Asignado	Real satan.	smurf.	spy.	teardrop.	warezclient.	warezmaster.
back.	0	0	0	0	0	0
buffer_overflow.	0	0	0	0	0	0
ftp_write.	0	0	0	0	0	0
guess_passwd.	0	0	0	0	0	0
imap.	0	0	0	0	0	0
ipsweep.	0	0	0	0	0	0
land.	0	0	0	0	0	0
loadmodule.	0	0	0	0	0	0
multihop.	0	0	0	0	0	0
neptune.	27	0	0	0	0	0
nmap.	0	0	0	0	0	0
normal.	64	29	1	9	174	10
perl.	0	0	0	0	0	0
phf.	0	0	0	0	0	0
pod.	0	0	0	0	0	0
portsweep.	12	0	0	0	0	0
rootkit.	0	0	0	0	0	0
satan.	691	0	0	0	0	0
smurf.	0	5366	0	0	0	0
spy.	0	0	0	0	0	0
teardrop.	0	0	0	480	0	0
warezclient.	0	0	0	0	336	0
warezmaster.	0	0	0	0	0	0

Correctamente clasificados: 96.2 %

Ilustración 21. Resultado del algoritmo SVM con 50% de muestras para entrenamiento y 3 columnas menos.

Finalmente obtenemos un resultado del 96,20% de clasificación correcta con el algoritmo SVM, siendo la segunda mejor opción para la clasificación de ataques en redes de datos.

Capítulo 6 Resultados

Podemos mostrar una tabla resumen con los resultados obtenidos.

Tabla 6. Porcentajes de aciertos y números de paquetes clasificados en cada prueba.

Número de prueba	Número de paquetes	Red Neuronal	C5.0	SVM
Primera	8752	25,36%	99,36%	93,08%
Segunda	35021	6,61%	99,55%	97,58%
Tercera	35021	0,02%	99,41%	95,07%
Cuarta	35021	6,27%	99,36%	96,20%

Nos sorprende muy negativamente el comportamiento de la red neuronal con este tipo de datos. Como hemos descrito en la Ilustración 4, podemos afirmar que el uso de redes neuronales para la clasificación de imágenes y vídeos funciona muy bien, pero en el caso de paquetes de datos, parece ser que funcionan mejor los algoritmos de árbol de decisiones como C5.0 y SVM. Se ha notado que la red neuronal funciona algo mejor si dispone de más muestras de entrenamiento, pero no mejora significativamente comparado con los algoritmos de clasificación C5.0 y SVM, los cuales consiguen un alto grado de aciertos en la clasificación con muy pocas muestras de entrenamiento.

Capítulo 7 Conclusión y trabajo futuro

Conclusión

Con el presente trabajo hemos querido mostrar una aplicación práctica del uso de algoritmos de Machine Learning aplicados a la detección de amenazas de tráfico de una red. Se ha conseguido un alto grado de detección con los algoritmos C5.0 y SVM. Al inicio de este trabajo, se pensó que la clasificación de amenazas en una red de datos usando redes neuronales iba a conseguir unos resultados muchísimo mejores, pero en nuestras pruebas, arrojan unos resultados muy malos para este propósito, comparado con los otros algoritmos analizados.

A priori, se pensó usar únicamente los algoritmos C4.5 y redes neuronales, pero a medida que comenzábamos las pruebas prácticas vinos que existía una versión más actual del algoritmo C4.5, por lo que pensamos que sería mejor idea usarlo, por ello, hemos usado el algoritmo C5.0 y redes neuronales. Como mejora, hemos añadido el algoritmo SVM (máquina de vector soporte) que complementa muy bien el estudio de amenazas en redes de datos.

La planificación del proyecto ha sido bastante fiel respecto a los hitos planteados, aunque la parte más práctica del trabajo ha retrasado ligeramente algún hito debido al tiempo de cómputo necesario para obtener los resultados de los algoritmos. Es por ello, que no se ha podido ajustar al 100% la planificación con respecto al proyecto real, aun así, se ha cumplido bastante bien. Se ha podido mejorar el trabajo, añadiendo un algoritmo más a los planteados y mejorando la versión de uno de ellos.

Trabajo futuro

El uso de técnicas y algoritmos de Machine Learning es muy beneficioso para el tratamiento de grandes cantidades de información, es por ello que los posibles trabajos futuros son muy variados. A modo de ejemplo:

Se podrían implementar más algoritmos para poder verificar que tipos de algoritmos funcionan mejor para la detección de amenazas en el tráfico de red e investigar aún más la posible mejora de redes neuronales aplicadas a este tipo de tráfico, estudiando el comportamiento más interno de la red neuronal, como, por ejemplo, la ampliación de las capas ocultas y el número de neuronas en dichas capas para ver si así mejora la clasificación.

Por otro lado, se podría implementar uno o varios algoritmos previamente entrenados o con un entrenamiento continuo a un posible prototipo de equipo de red, como, por ejemplo, una red definida por software (SDN), para que monitorice el tráfico de red y evite ataques a una organización.

El presente y futuro de los algoritmos y técnicas de Machine Learning es muy variado, por lo que hoy en día se pueden aplicar a múltiples áreas del conocimiento con grandes resultados.

Capítulo 8 Glosario

Paradigma
: Según la RAE, es una teoría o conjunto de teorías cuyo núcleo central se acepta sin cuestionar y que suministra la base y modelo para resolver problemas y avanzar en el conocimiento. 21

TFM
: Trabajo fin de máster. 3, 14

Capítulo 9 Anexo. Código fuente utilizado para las pruebas

A continuación mostramos el código fuente utilizado para la realización de las pruebas con los diferentes algoritmos («R - Comparativa de modelos de clasificación», s. f.). Dicho código fuente permite el cálculo de más algoritmos, aunque en este trabajo nos hemos centrado en Redes Neuronales, C5.0 y SVM.

Para poder utilizar esta función de R, debemos cargar nuestro Data Set y aplicar los siguientes parámetros:

- datos: data frame con las variables. Valor por defeco: iris
- dep: valor numérico que indica el número de la variable respuesta (dependiente). Valor por defecto 5 (variable Species de los datos iris)
- p: es la proporción usada para el entrenamiento (el resto se usa para probarlo). Valor por defecto p = 0.75 (75% de los datos usados para entrenamiento)
- ad: 1 si se realizarán los análisis discriminantes lineal y cuadrático, 0 si se omiten. Valor por defecto ad = 1

Tabla 7. Código fuente en R utilizado para el cálculo de los algoritmos (Red Neuronal, C5.0 y SVM).

```
modelos <- function(datos = iris, dep = 5, p = 0.75, ad=1) {

  # Creación del subgrupo de entrenamiento y de prueba
  library(caret)
  datos.indices <- createDataPartition(datos[,dep],p = p, list=F)
  datos.train <- datos[datos.indices,]

  if(p != 1) {
    datos.test <- datos[-datos.indices,]
  } else {
    datos.test <- datos.train
  }
  # Fórmula
  f <- as.formula(paste(names(datos[dep]),"~."))


  # Redes neuronales
  library(nnet)
  parametros <- train(f, data=datos.train, method="nnet", trace=F)
  size <- parametros$bestTune$size
  decay <- parametros$bestTune$decay
  modelo.nnet <- nnet(f,trace=F, data=datos.train, size=size,
decay=decay)
  pred.nnet <- predict(modelo.nnet, datos.test, type="class")
  # Matriz de confusión
  mc <- table(pred.nnet,datos.test[,dep], dnn =
c("Asignado","Real"))
  # Ordenar tabla alfabéticamente
  # Por algún motivo a veces sale desordenada
  mc <- mc[order(rownames(mc)),order(colnames(mc))]
```

```
cat("** Red neuronal: nnet\n")
print(mc)
# Aciertos en %
aciertos1 <- sum(diag(mc)) / sum(mc) * 100
cat("\nCorrectamente clasificados:",round(aciertos1,2),"%\n\n\n")


# C5.0
library(C50)
modelo.c50 <- C5.0(f, data=datos.train)
pred <- predict(modelo.c50, datos.test, type="class")
# Matriz de confusión
mc <- table(pred,datos.test[,dep], dnn = c("Asignado","Real"))
# Ordenar tabla alfabéticamente
# Por algún motivo a veces sale desordenada
mc <- mc[order(rownames(mc)),order(colnames(mc))]
cat("** Árbol de clasificación C5.0\n")
print(mc)
# Aciertos en %
aciertos2 <- sum(diag(mc)) / sum(mc) * 100
cat("\nCorrectamente clasificados:",round(aciertos2,2),"%\n\n\n")


# SVM
library(e1071)
modelo.svm <- svm(f, data=datos.train)
pred <- predict(modelo.svm, datos.test, type="class")
# Matriz de confusión
mc <- table(pred,datos.test[,dep], dnn = c("Asignado","Real"))
# Ordenar tabla alfabéticamente
# Por algún motivo a veces sale desordenada
mc <- mc[order(rownames(mc)),order(colnames(mc))]
cat("** SVM\n")
print(mc)
# Aciertos en %
aciertos3 <- sum(diag(mc)) / sum(mc) * 100
cat("\nCorrectamente clasificados:",round(aciertos3,2),"%\n\n\n")


# Bootstrapped Aggregation: Bagging
library(ipred)
modelo.ba <- ipred::bagging(f, data=datos.train)
pred <- predict(modelo.ba, datos.test, type="class")
# Matriz de confusión
mc <- table(pred,datos.test[,dep], dnn = c("Asignado","Real"))
# Ordenar tabla alfabéticamente
# Por algún motivo a veces sale desordenada
mc <- mc[order(rownames(mc)),order(colnames(mc))]
cat("** Bootstrapped Aggregation: bagging\n")
print(mc)
# Aciertos en %
aciertos4 <- sum(diag(mc)) / sum(mc) * 100
cat("\nCorrectamente clasificados:",round(aciertos4,2),"%\n\n\n")


# Random Forest
library(randomForest)
modelo.rf <- randomForest(f, data=datos.train)
pred <- predict(modelo.rf, datos.test, type="class")
# Matriz de confusión
mc <- table(pred, datos.test[,dep], dnn = c("Asignado","Real"))
```

```
  # Ordenar tabla alfabéticamente
  # Por algún motivo a veces sale desordenada
  mc <- mc[order(rownames(mc)),order(colnames(mc))]
  cat("Random Forest\n")
  print(mc)
  # Aciertos en %
  aciertos5 <- sum(diag(mc)) / sum(mc) * 100
  cat("\nCorrectamente clasificados:",round(aciertos5,2),"%\n\n\n")


  # AdaBoost - Adaptative Boosting
  library(adabag)
  modelo.ad <- boosting(f, data=datos.train)
  pred <- predict(modelo.ad, datos.test, type="class")
  # Matriz de confusión
  mc <- table(pred$class, datos.test[,dep], dnn =
c("Asignado","Real"))
  # Ordenar tabla alfabéticamente
  # Por algún motivo a veces sale desordenada
  mc <- mc[order(rownames(mc)),order(colnames(mc))]
  cat("** Adaptative Boosting: boosting\n")
  print(mc)
  # Aciertos en %
  aciertos6 <- sum(diag(mc)) / sum(mc) * 100
  cat("\nCorrectamente clasificados:",round(aciertos6,2),"%\n\n\n")


  # Árboles de clasificación: rpart
  library(rpart)
  modelo.rp <- rpart(f, data=datos.train)
  pred <- predict(modelo.rp, datos.test, type="class")
  # Matriz de confusión
  mc <- table(pred, datos.test[,dep], dnn = c("Asignado","Real"))
  # Ordenar tabla alfabéticamente
  # Por algún motivo a veces sale desordenada
  mc <- mc[order(rownames(mc)),order(colnames(mc))]
  cat("** Árboles de clasificación: rpart\n")
  print(mc)
  # Aciertos en %
  aciertos7 <- sum(diag(mc)) / sum(mc) * 100
  cat("\nCorrectamente clasificados:",round(aciertos7,2),"%\n\n\n")


  # Clasificador bayesiano ingenuo (Naive Bayes classifier):
naiveBayes

  modelo.nb <- naiveBayes(f, data=datos.train)
  pred <- predict(modelo.nb, datos.test, type="class")
  # Matriz de confusión
  mc <- table(pred, datos.test[,dep], dnn = c("Asignado","Real"))
  # Ordenar tabla alfabéticamente
  # Por algún motivo a veces sale desordenada
  mc <- mc[order(rownames(mc)),order(colnames(mc))]
  cat("** Clasificador bayesiano ingenuo: naiveBayes\n")
  print(mc)
  # Aciertos en %
  aciertos10 <- sum(diag(mc)) / sum(mc) * 100
  cat("\nCorrectamente clasificados:",round(aciertos10,2),"%\n\n\n")

  if(ad != 0) {
    library(MASS)
```

```
    # Análisis disciminane lineal
    modelo.lda <- rpart(f, data=datos.train)
    pred <- predict(modelo.lda, datos.test, type="class")
    # Matriz de confusión
    mc <- table(pred, datos.test[,dep], dnn = c("Asignado","Real"))
    # Ordenar tabla alfabéticamente
    # Por algún motivo a veces sale desordenada
    mc <- mc[order(rownames(mc)),order(colnames(mc))]
    cat("** Análisis disciminane lineal: lda\n")
    print(mc)
    # Aciertos en %
    aciertos8 <- sum(diag(mc)) / sum(mc) * 100
    cat("\nCorrectamente
clasificados:",round(aciertos8,2),"%\n\n\n")


     # Análisis disciminane cuadrático
    modelo.qda <- rpart(f, data=datos.train)
    pred <- predict(modelo.qda, datos.test, type="class")
    # Matriz de confusión
    mc <- table(pred, datos.test[,dep], dnn = c("Asignado","Real"))
    # Ordenar tabla alfabéticamente
    # Por algún motivo a veces sale desordenada
    mc <- mc[order(rownames(mc)),order(colnames(mc))]
    cat("** Análisis disciminane cuadrático: qda\n")
    print(mc)
    # Aciertos en %
    aciertos9 <- sum(diag(mc)) / sum(mc) * 100
    cat("\nCorrectamente
clasificados:",round(aciertos9,2),"%\n\n\n")
  }

  resumen <-
c(aciertos1,aciertos2,aciertos3,aciertos4,aciertos5,aciertos6,aciert
os7,aciertos10)
  names(resumen) <-
c("nnet","C5.0","svm","bagging","randomForest","boosting","rpart","n
aiveBayes")
  mod.list <- list(nnet=modelo.nnet, C5.0=modelo.c50,
svm=modelo.svm, bagging=modelo.ba, randomForest=modelo.rf,
boosting=modelo.ad, rpart=modelo.rp,naiveBayes=modelo.nb)
  if(ad != 0) {
    resumen <- c(resumen,aciertos8,aciertos9)
    names(resumen) <- c(names(resumen[1:8]),"lda","qda")
    mod.list <- list(nnet=modelo.nnet, C5.0=modelo.c50,
svm=modelo.svm, bagging=modelo.ba, randomForest=modelo.rf,
boosting=modelo.ad, rpart=modelo.rp, naiveBayes=modelo.nb,
lda=modelo.lda, qda=modelo.qda)
  }
  resumen <- sort(resumen,decreasing=T)
  resumen <- as.matrix(resumen,nc=1)
  colnames(resumen) <- "% correctos"
  cat("Resumen:\n")

  print(resumen)

  return(invisible(mod.list))
}
```

Capítulo 10 Bibliografía

7-Zip. (s. f.). Recuperado 12 de marzo de 2018, a partir de https://www.7-zip.org/

C4.5. (2018, abril 9). En *Wikipedia, la enciclopedia libre*. Recuperado a partir de https://es.wikipedia.orghttps://es.wikipedia.org/w/index.php?title=C4.5&oldid=106915652

Chema Alonso. (2018, abril 4). En *Wikipedia, la enciclopedia libre*. Recuperado a partir de https://es.wikipedia.org/w/index.php?title=Chema_Alonso&oldid=106715052

Chrome para ordenadores. (s. f.). Recuperado 12 de marzo de 2018, a partir de https://www.google.es/chrome/index.html

Computing machinery and intelligence. (2018, marzo 2). En *Wikipedia, la enciclopedia libre*. Recuperado a partir de https://es.wikipedia.org/w/index.php?title=Computing_machinery_and_intelligence&oldid=105943944

Data science. (2018, mayo 10). En *Wikipedia*. Recuperado a partir de https://en.wikipedia.org/w/index.php?title=Data_science&oldid=840567330

Data-driven science. (2016, diciembre 16). En *Wikipedia*. Recuperado a partir de https://en.wikipedia.org/w/index.php?title=Data-driven_science&oldid=755120624

ElevenPaths Talks 3: Inteligencia Artificial y Machine Learning - YouTube. (2017, agosto 25). Recuperado 12 de mayo de 2018, a partir de https://www.youtube.com/watch?v=MH4812-Iavo

GanttProject: free desktop project management app. (s. f.). Recuperado 12 de marzo de 2018, a partir de http://www.ganttproject.biz/

Gengiskanhg. (2004). *Red Neuronal Artificial de tipo es:Perceptrón simple con n neuronas de entrada, m neuronas en su capa oculta y una neurona de salida.* Recuperado a partir de https://commons.wikimedia.org/wiki/File:RedNeuronalArtificial.png?uselang=es

GIMP Descargas, tutoriales y foros. Alternativa a Photoshop gratis y libre. (s. f.). Recuperado 12 de marzo de 2018, a partir de http://www.gimp.org.es/

Google Maps. (2018, abril 2). En *Wikipedia, la enciclopedia libre*. Recuperado a partir de https://es.wikipedia.org/w/index.php?title=Google_Maps&oldid=106667118

Hackeando Tec. (2015). *Redes Neuronales - 2.4 Funciones de Activación - Hackeando Tec*. Recuperado a partir de https://www.youtube.com/watch?v=aaVRKzYIIO8

Hothorn, T. (2018). CRAN Task View: Machine Learning & Statistical Learning. Recuperado a partir de https://CRAN.R-project.org/view=MachineLearning

Jaomun. (200d. C.a, 12:00). Redes Neuronales: Un poco de historia (I). Recuperado 8 de mayo de 2018, a partir de https://www.rankia.com/blog/redes-neuronales/1408704-redes-neuronales-poco-historia-i

Jaomun. (200d. C.b, 22:00). Redes Neuronales: Un poco de historia (II). Recuperado 8 de mayo de 2018, a partir de https://www.rankia.com/blog/redes-neuronales/1408730-redes-neuronales-poco-historia-ii

KDD Cup 1999 Data. (1999, octubre 28). Recuperado 12 de mayo de 2018, a partir de http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

KNIME Analytics Platform & SDK | KNIME. (s. f.). Recuperado 12 de marzo de 2018, a partir de https://www.knime.com/downloads

Macías Macías, M., Peguero Chamizo, J. C., & García Orellana, C. J. (2016). Apuntes de la asignatura Iniciación a la investigación en arquitecturas paralelas para el tratamiento de imágenes del Máster Universitario en Investigación en Ingeniería y Arquitectura, especialidad TIC por la Universidad de Extremadura. Recuperado a partir de https://www.unex.es/conoce-la-uex/centros/cum/titulaciones/info/asignaturas?id=1518

Máquinas de vectores de soporte. (2018, mayo 23). En *Wikipedia, la enciclopedia libre*. Recuperado a partir de https://es.wikipedia.org/w/index.php?title=M%C3%A1quinas_de_vectores_de_soporte&oldid=108054208

MATLAB Student R2017b - MathWorks España. (s. f.). Recuperado 12 de marzo de 2018, a partir de https://es.mathworks.com/store/link/products/student/new?s_tid=ac_buy_sv_cta

Neuronales, R. (2012a, noviembre 4). Redes Neuronales: Elementos Básicos que Componen una Red Neuronal. Recuperado 13 de mayo de 2018, a partir de http://redesneuronales-uba.blogspot.com.es/2012/11/elementos-basicos-que-componen-una-red.html

Neuronales, R. (2012b, noviembre 4). Redes Neuronales: Ventajas que ofrecen las Redes Neuronales. Recuperado 13 de mayo de 2018, a partir de http://redesneuronales-uba.blogspot.com.es/2012/11/ventajas-que-ofrecen-las-redes.html

R - Comparativa de modelos de clasificación. (s. f.). Recuperado 2 de junio de 2018, a partir de https://rpro.wikispaces.com/Comparativa+de+modelos+de+clasificaci%C3%B3n

R: The R Project for Statistical Computing. (s. f.). Recuperado a partir de https://www.r-project.org/

Raona. (2017, marzo). *Machine Learning Whitepaper*. Technology. Recuperado a partir de https://www.slideshare.net/raona/machine-learning-whitepaper

Red neuronal artificial. (2018, mayo 1). En *Wikipedia, la enciclopedia libre*. Recuperado a partir de https://es.wikipedia.org/w/index.php?title=Red_neuronal_artificial&oldid=107483201

Shalev-Shwartz, S., & Ben-David, S. (2014). *Understanding Machine Learning: From Theory to Algorithms*. Cambridge: Cambridge University Press. https://doi.org/10.1017/CBO9781107298019

Stolfo, S., Fan, W., Lee, W., Prodromidis, A., & Chan, P. (1999). Cost-Based modeling and evaluation for data mining with application to fraud and intrusion detection: Results from the JAM project.

Test de Turing. (2018, abril 22). En *Wikipedia, la enciclopedia libre*. Recuperado a partir de https://es.wikipedia.org/w/index.php?title=Test_de_Turing&oldid=107214111

Un ordenador logra superar por primera vez el test de Turing. (2014, junio 9). Recuperado 8 de mayo de 2018, a partir de http://www.elmundo.es/ciencia/2014/06/09/539589ee268e3e096c8b4584.html

Weka 3 - Data Mining with Open Source Machine Learning Software in Java. (s. f.). Recuperado 12 de marzo de 2018, a partir de https://www.cs.waikato.ac.nz/~ml/weka/downloading.html

Zotero | Your personal research assistant. (s. f.). Recuperado 12 de marzo de 2018, a partir de https://www.zotero.org/