Universitat Oberta de Catalunya
uoc.edu

Detector predictivo de conexiones fraudulentas

David Martin Tinaquero
Máster en Seguridad de las TIC
Aplicación de técnicas de *Machine Learning* a la Seguridad

Enric Hernández
Víctor García Font

4 de Junio de 2018

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Detector predictivo de conexiones fraudulentas*
Nombre del autor:	*David Martin Tinaquero*
Nombre del consultor/a:	*Enric Hernández*
Nombre del PRA:	*Víctor García Font*
Fecha de entrega (mm/aaaa):	03/2018
Titulación::	*Máster en Seguridad de las TIC*
Área del Trabajo Final:	*Aplicación de técnicas de Machine Learning a la Seguridad*
Idioma del trabajo:	*Español*
Palabras clave	*Machine Learning Seguridad*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados i conclusiones del trabajo.*

La finalidad principal de este trabajo es el diseño e implementación de un detector predictivo de conexiones fraudulentas eficiente, que cumpla con las necesidades manifestadas por el cliente (Ancert). Para llevar a cabo el proyecto, primero se han estudiado a fondo las necesidades reales a cubrir. Después se han obtenido tramas de red suficientes y de calidad para cubrir con las fases de modelado del clasificador predictivo.

Posteriormente, se han investigado sistemas para estructurar los ficheros, que faciliten su exploración y tratamiento masivo. Una vez que los datos han quedado estructurados en la BD NoSQL Cassandra, se ha hecho una investigación de los diferentes métodos de *Machine Learning* i sus ámbitos de aplicación, escogiendo el que se ha estimado más idóneo para el proyecto.

Una vez decido que el diseño del detector se hará mediante redes neuronales profundas (*Deep Learning*) con Python y las librerías TensorFlow, se ha empezado con el modelado del algoritmo predictivo, que incluye las etapas de diseño, entrenamiento, validación, prueba y guardado del modelo para su reutilización posterior. La precisión obtenida con los datos de entrenamiento ha sido de ~99% y ~92% con los datos de test. Todo ello, con un porcentaje ínfimo (~0,05%) de falsos positivos, relevante para evitar la no disponibilidad del servicio a los usuarios lícitos del sistema.

El detector predictivo se ha utilizado para clasificar un conjunto de tramas de red sin etiquetar y realizar su posterior inserción en la BD Cassandra. El tiempo total empleado para completar clasificación de ~300.000 tramas e insertarlas en la BD ha sido inferior a 3’ en todas las pruebas realizadas.

Abstract (in English, 250 words or less):

The main purpose of this work is the design and implementation of an efficient fraudulent predictive detector that meets the needs expressed by the client (Ancert). To carry out the project, the real needs to be covered have been studied in depth. After that, sufficient and quality network frames have been obtained to cover the modeling phases of the predictive classifier.

Subsequently, systems have been investigated to structure the files, which facilitate their exploration and massive treatment. Once the data has been structured in the Cassandra NoSQL DB, an investigation has been made of the different methods of Machine Learning and its application areas, choosing the one that has been considered most suitable for the project.

Once I have decided that the design of the detector will be made through deep neural networks (Deep Learning) with Python and the TensorFlow libraries, we have started with the predictive algorithm modeling, which includes the stages of design, training, validation, testing and saving of the model for its subsequent reuse. The accuracy obtained with the training data was ~ 99% and ~ 92% with the test data. All this, with a negligible percentage (~ 0.05%) of false positives, relevant to avoid the non-availability of the service to the legal users of the system.

The predictive detector has been used to classify a set of unlabeled network frames and perform their subsequent insertion into the Cassandra database. The total time spent to complete the classification of ~ 300,000 frames and insert them into the database has been less than 3 'in all tests performed.

Índice

Lista de figuras

Lista de tablas

1. Introducción

1.1 Contexto y justificación del Trabajo

Este proyecto nace de la necesidad que manifiesta Ancert, compañía con más de 300 profesionales especializados en el ámbito de la Informática, las Telecomunicaciones, el Derecho y la Empresa, y que es el resultado del esfuerzo decidido por parte del Consejo General del Notariado por hacer frente a las imparables demandas de la sociedad.

Mediante el desarrollo de aplicaciones electrónicas, seguras, rápidas y eficaces, conectan a los más de 3.000 notarios distribuidos por todo el territorio nacional con todo tipo de organismos, sean públicos o privados, con el objetivo final de facilitar el asesoramiento al ciudadano en la autorización de documentos públicos y la manera de proceder a su posterior inscripción, tramitación y/o remisión a terceros organismos en su relación con las diferentes administraciones públicas (nacionales, autonómicas o locales).

Por lo tanto, se trata de una red muy grande por la que viaja mucha información sensible. Aunque dicha información viaja por una red VPN (red RENO), la parte de la red WAN, está expuesta a algunos peligros que se encuentran latentes en Internet. Es por ello, que surge la necesidad de disponer de un sistema predictivo capaz de distinguir entre conexiones intrusivas (“malas”) y lícitas (“buenas”) de manera eficiente y fiable, con el objetivo de aplicar las medidas que sean necesarias para que el servicio no se vea afectado.

1.2 Objetivos del Trabajo

El objetivo de este trabajo es el diseño e implementación de un algoritmo predictivo capaz de clasificar las conexiones que circulan por la red en las categorías “buena” y “mala” a partir, únicamente, de los datos de sus tramas respectivas. Dicho sistema, debe tener un porcentaje de acierto muy elevado, que maximice la detección de las conexiones fraudulentas y minimice los falsos positivos, conexiones clasificadas como intrusivas que en realidad son lícitas, para evitar el repudio de conexiones de clientes autorizados.

El sistema a desarrollar deber ser autónomo y con capacidad de aprendizaje tal, que le permita detectar a partir del conocimiento adquirido previamente, intentos de intrusión de tipologías nuevas que se produzcan a posteriori.

Debido al gran número de conexiones de la red, dicho sistema debe ser escalable, de manera que pueda clasificar de forma eficiente grandes volúmenes de tramas de conexión. No puede ocurrir que el clasificador entorpezca la red, de manera que el servicio se vuelva deficiente para los usuarios de Ancert.

Teniendo en cuenta todo lo anterior, el trabajo tendrá los siguientes objetivos:

1. **Recolecta de datos.** Hay que conseguir que el volumen de datos sea grande y de buena calidad, con el objetivo de que nuestro algoritmo pueda aprender más y mejor.

2. **Preparación de los datos.** Dar a los datos crudos una estructura tabular que facilite su exploración.

3. **Elección del modelo.** Analizar los diferentes modelos de ML (*Machine Learning*) y elegir el que mejor se ajuste a las necesidades del proyecto.

4. **Entrenamiento del algoritmo.** Seleccionar los datos que se van a utilizar para entrenar el modelo y realizar el entrenamiento del mismo.

5. **Evaluación de los resultados.** Comprobar el modelo creado contra el juego de datos de evaluación, que contiene entradas que el modelo desconoce. Verificar la precisión del modelo entrenado.

6. **Ajuste de los parámetros.** Si en la fase de evaluación el algoritmo no tiene la precisión mínima deseada, hay que realizar cambios en la configuración de los parámetros y volver a la fase de entrenamiento.

7. **Predicción.** Realizar predicciones de las conexiones sin etiquetar.

1.3 Enfoque y método seguido

La metodología a seguir para dar lugar a la realización de este proyecto es, analizar los requisitos necesarios a partir de la petición de Ancert. Una vez identificados los requisitos, se ha elaborado un plan de trabajo con todas las fases e hitos necesarios para llevarlo a cabo.

En primer lugar, se investigan y analizan los diferentes tipos de algoritmos de ML, para determinar cuál es el más adecuado para cumplir con el objetivo del proyecto. Posteriormente, se analizan las herramientas de software (BD, IDE, librerías, etc) para decidir las que son más idóneas para utilizar el algoritmo seleccionado en el paso previo y dar la estructura adecuada a los datos. A continuación, se diseñará e implementará el algoritmo. Finalmente, una vez ajustado el algoritmo, se realizarán las predicciones sobre los datos ciegos o sin etiquetar.

El tipo de metodología de investigación es la más apropiada en el campo de IA (Inteligencia Artificial), en especial cuando se trata de ML, debido a que los cambios en esta rama se realizan con mucha frecuencia. Por ello, es importante investigar en las fuentes de información más actualizadas, que nos ayuden a crear un sistema mejor.

1.4 Planificación del Trabajo

Con la finalidad de cumplir con todos los objetivos del proyecto, se ha elaborado un plan de trabajo donde quedan representadas todas las tareas, actividades e hitos a realizar para llevar a cabo el proyecto.

Las tareas se han ajustado a los 5 hitos principales, ya que el sexto es la defensa del TFM, y el proyecto, memoria final i presentación deben entregarse en el cuarto y quinto hito respectivamente.

En la tabla siguiente se muestra la planificación del proyecto:

Tareas y actividades	Duración	Fecha inicio	Fecha fin	Hitos
1.Análisis y planificación de los objetivos del TFM	**20 días**	**21/02/18**	**12/03/18**	
1.1.Análisis de los requisitos indicados por Ancert	3 días	21/02/18	23/02/18	
1.2.Investigación web de la información necesaria	6 días	24/02/18	01/03/18	
1.3.Aprendizaje de la herramienta para elaborar el plan de trabajo	2 días	02/03/18	03/03/18	
1.4.Elaboración del plan de trabajo	5 días	04/03/18	08/03/18	
1.5.Redacción de la parte proporcional de la memoria	4 días	09/03/18	12/03/18	**PEC 1 – Plan de trabajo**
2.Recolección y preparación de los datos	**28 días**	**13/03/18**	**09/04/18**	
2.1.Colecta de datos	2 días	13/03/18	14/03/18	
2.2.Análisis de herramientas para estructurar datos	5 días	15/03/18	19/03/18	
2.3.Aprendizaje de instalación y uso de la herramienta seleccionada	4 días	20/03/18	23/03/18	
2.4.Diseño e implementación del modelo para importar los datos	5 días	24/03/18	28/03/18	
2.5.Importación de los datos	3 días	29/03/18	31/03/18	
2.6.Exploración de los datos	4 días	01/04/18	04/04/18	
2.7.Redacción de la parte proporcional de la memoria	5 días	05/04/18	09/04/18	**PEC 2 – Entrega parcial**
3.Elección del modelo ML y preparación del entorno de trabajo	**28 días**	**10/04/18**	**07/05/18**	
3.1.Investigación sobre los modelos ML que existen	7 días	10/04/18	16/04/18	
3.2.Elección y justificación del modelo ML seleccionado	3 días	17/04/18	19/04/18	
3.3.Análisis del software necesario para implementar el algoritmo con el modelo elegido	6 días	20/04/18	25/04/18	
3.4.Aprendizaje de instalación del software seleccionado en el paso anterior	6 días	26/04/18	01/05/18	
3.5.Redacción de la parte proporcional de la memoria	6 días	02/05/18	07/05/18	**PEC 3 – Entrega parcial**
4.Entrenamiento del algoritmo ML	**12 días**	**08/05/18**	**19/05/18**	
4.1.Diseño e implementación del algoritmo	5 días	08/05/18	12/05/18	
4.2.Selección de los datos que se van a usar para entrenar	2 días	13/05/18	14/05/18	
4.3.Entrenamiento del algoritmo	3 días	15/05/18	17/05/18	
4.4.Redaccción de la parte proporcional de la memoria	2 días	18/05/18	19/05/18	
5.Evaluación del algoritmo ML	**4 días**	**20/05/18**	**23/05/18**	
5.1.Análisis de resultados obtenidos	2 días	20/05/18	21/05/18	
5.2.Redacción de la parte proporcional de la memoria	2 días	22/05/18	23/05/18	
6.Configuración de los parámetros	**6 días**	**24/05/18**	**29/05/18**	
6.1.Ajuste de parámetros	4 días	24/05/18	27/05/18	
6.2.Redaccción de la parte proporcional de la memoria	2 días	28/05/18	29/05/18	
7.Predicción de los datos	**6 días**	**30/05/18**	**04/06/18**	
7.1.Predicción de los datos ciegos	2 días	30/05/18	31/05/18	
7.2.Análisis de los resultados obtenidos	2 días	01/06/18	02/06/18	
7.3.Redacción de la parte proporcional de la memoria	2 días	03/06/18	04/06/18	**PEC 4 –**

				Memoria final
8.Elaboración de la presentación	**7 días**	**05/06/18**	**11/06/18**	
8.1.Preperación del entorno de trabajo	1 día	05/06/18	05/06/18	
8.2.Grabación de la presentación	6 días	06/06/18	11/06/18	**PEC 5 - Entrega**
9.Defensa de TFM	**5 días**	**18/06/18**	**22/06/18**	
9.1.Revisar el correo electrónico periódicamente y dar respuesta a las preguntas planteadas	5 días	18/06/18	22/06/18	

Tabla 1. Planificación del trabajo

1.5 Breve sumario de productos obtenidos

En este proyecto se obtendrá un sistema capaz de importar tramas de un gran volumen de conexiones en formato de texto plano y convertirlas a formato estructurado. Una vez estructuradas, un algoritmo predictivo basado en ML clasificará las conexiones en “buenas” o “malas”, con un porcentaje de acierto elevado. A continuación se detallan los productos obtenidos al finalizar el trabajo:

- Scripts del importador de tramas en formato crudo a base de datos (BD).
- Ficheros de datos de la BD con las tramas importadas.
- Código fuente del algoritmo predictivo de tipo clasificador.
- Memoria final con el detalle del proyecto.
- Vídeo de presentación del proyecto.

1.6 Breve descripción de los otros capítulos de la memoria

En los capítulos siguientes se empieza con un análisis de diferentes tecnologías para estructurar datos crudos, tales como bases de datos relacionales (RDBMS) y NoSQL. Se justifica la elección del tipo de base de datos seleccionada a partir de los resultados del análisis previo.

Se detallan los pasos seguidos para la instalación y configuración del sistema de BD. A continuación, se crea el modelo y los scripts necesarios para importar los datos y, posteriormente se importan a la BD. También se realiza una exploración superficial de los datos desde la BD.

Se realiza una breve introducción a los conceptos de IA y *Machine Learning* (ML), y se comparan algunos de sus tipos más relevantes. Se lleva a cabo un estudio de algunos sistemas ML *Deep Learning* que existen hoy en día y se justifica la elección del modelo escogido para elaborar el trabajo.

Se describe el diseño e implementación del algoritmo predictivo, así como los pasos seguidos para llevar a cabo su entrenamiento. Se muestran los datos resultantes de la evaluación realizada y los ajustes llevados a cabo para mejorar su precisión y rendimiento. Finalmente, se analizan los resultados obtenidos en la fase de predicción.

La memoria termina con las conclusiones del trabajo, el glosario de términos, la bibliografía y los anexos.

2. Recolección y preparación de los datos

2.1. Colecta de datos

Para diseñar un algoritmo capaz de predecir o clasificar datos a partir de un vector de características que describen la muestra, es imprescindible disponer de datos que sean lo más realistas posibles. Es importante que la cantidad de datos sea abundante y de calidad, es decir, que contenga la mayor diversidad posible, de manera que el modelo represente una aproximación fiel de los fenómenos que se modelan. Partimos de la base de que es prácticamente imposible que los datos de entrenamiento contengan todos los posibles vectores de características de entrada.

Los datos pueden ser reales, tales como capturas de *logs*, o sintéticos, generados mediante algoritmos o scripts diseñados para tal fin. Para este proyecto se utilizan los juegos de datos que fueron utilizados en la Tercera Competición Internacional de Descubrimiento del Conocimiento y Herramientas de Minería de Datos, que se realizó en conjunto con la KDD-99, Quinta Conferencia Internacional sobre Descubrimiento de Conocimiento y Minería de Datos.

El Programa de Evaluación de Detección de Intrusos DARPA de 1998 fue preparado y gestionado por el MIT Lincoln Labs. El objetivo fue estudiar y evaluar la investigación de detección de intrusos. Se proporcionó un juego estándar de datos para ser auditado, el cual incluye una amplia variedad de intrusiones simuladas en un entorno de red militar. El concurso de detección de intrusos KDD de 1999 usó una versión de este juego de datos.

Lincoln Labs creó un entorno para adquirir nueve semanas de datos de volcado de TCP sin formato para una red de área local (LAN) que simula una LAN de la Fuerza Aérea típica de EE.UU. Operaron en la LAN como si fuera un verdadero entorno de la Fuerza Aérea, pero la inundaron de múltiples ataques.

Los datos de entrenamiento sin procesar ocupan aproximadamente cuatro gigabytes de datos de volcado TCP binario comprimido a partir de siete semanas de tráfico de red. Son casi cinco millones de registros de conexiones. Del mismo modo, las dos semanas de datos de prueba son alrededor de dos millones de registros de conexiones.

2.2. Ficheros del juego de datos

El juego de datos consta de los siguientes ficheros:

- **kddcup.data**. Contiene 4.898.431 registros con 42 campos cada uno. El último campo de cada registro contiene la etiqueta con uno de los 22 tipos de ataque (ver Tabla 3) o el texto “normal.”

- **kddcup.data_10_percent**. Contiene un subconjunto del ~10% (494.091) del fichero anterior.

- **kddcup.newtestdata_10_percent_unlabeled**. Contiene un subconjunto del ~10% (311.079) del fichero siguiente.

- **kddcup.testdata_unlabeled**. Contiene 2.984.154 registros con 41 campos cada uno. Los registros tienen un campo menos que los dos primeros ficheros porque carecen del campo de la etiqueta.

- **kddcup.testdata_10_percent_unlabeled**. Contiene un subconjunto del ~10% (311.029) del fichero anterior.

- **corrected**. Es idéntico al fichero anterior pero con el campo adicional de la etiqueta. Este fichero es ideal para validar la precisión del algoritmo.

Es importante tener en cuenta que los datos de prueba no provienen de la misma distribución de probabilidad que los datos de entrenamiento, e incluyen tipos de ataque específicos que no están en los ficheros de datos de entrenamiento. Los conjuntos de datos contienen un total de 22 tipos de ataque en los datos de entrenamiento y 14 tipos adicionales solo en los datos de prueba.

2.3. Exploración del contenido y estructura

Una vez en posesión de los datos que se van a usar para entrenar y validar el algoritmo predictivo, el primer ejercicio a realizar es un examen visual de su contenido mediante el uso de un editor de texto. Para este proyecto se utiliza Notepad++ y Vim, pero pueden usarse otros editores, como por ejemplo, Nano o Gedit.

El objetivo de la exploración inicial es ver cómo están separados los registros y los campos entre ellos, así como observar el tipo de dato de cada campo. Se observa que los campos están delimitados por coma (“,”) y los registros a su vez, separados por salto de línea y retorno de carro. Es un formato de fichero muy común, que viene bien para la mayoría de sistemas de importación de datos.

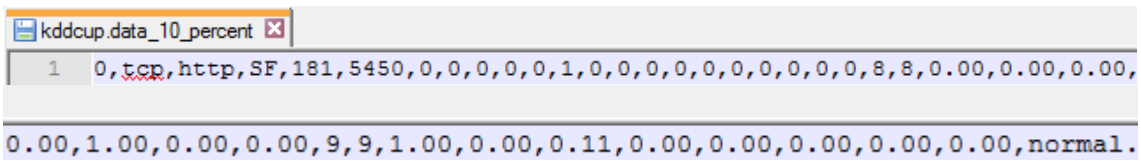


Fig. 1. Captura de un registro del fichero kddcup.data_10_percent

En la tabla siguiente se muestran los campos en el orden que están en cada registro del fichero, el tipo de dato y una breve descripción de cada uno:

Nombre	Tipo	Descripción
duration	continuos	Longitud en número de segundos de la conexión.
protocol_type	symbolic	Tipo de protocolo. Ej.: TCP, UDP, etc.
service	symbolic	Servicio de red en el destino. Ej.: HTTP, Telnet, etc.
flag	symbolic	Estado normal o de error de la conexión.
src_bytes	continuos	Número de bytes de datos desde el origen al destino.
dst_bytes	continuos	Número de bytes de datos desde el destino al origen.
land	symbolic	1 si la conexión es desde/a el mismo host/puerto; 0 en caso contrario.
wrong_fragment	continous	Número de fragmentos erróneos.
urgent	continous	Número de paquetes urgentes.
hot	continous	Número de indicadores "*hot*".
num_failed_logins	continous	Número de intentos de inicio de sesión fallidos.
logged_in	symbolic	1 si ha iniciado sesión correctamente; 0 en caso contrario.
num_compromised	continous	Número de condiciones comprometidas.
root_shell	continous	1 si se ha obtenido shell de root; 0 en caso contrario.
su_attempted	continous	1 si el comando "su root" se ha intentado; 0 en caso contrario.
num_root	continous	Número de accesos de root.
num_file_creations	continous	Número de operaciones de creación de fichero.
num_shells	continous	Número de *prompts shells*.
num_access_files	continous	Número de operaciones de control de acceso a ficheros.
num_outbound_cmds	continous	Número de comandos salientes en una sesión de FTP.
is_host_login	symbolic	1 si el inicio de sesión pertenece a la lista "*hot*"; 0 en caso contrario.
is_guest_login	symbolic	1 si el inicio de sesión pertenece a invitado; 0 en caso contrario.
count	continous	Número de conexiones al mismo host como la actual en los últimos 2 segundos.
srv_count	continous	Número de conexiones al mismo servicio como la conexión actual in los últimos 2 segundos.
serror_rate	continous	% de conexiones que tienen errores "SYN".
srv_serror_rate	continous	% de conexiones que tienen errores "SYN".
rerror_rate	continous	% de conexiones que tienen errores "REJ".
srv_rerror_rate	continous	% de conexiones que tienen errores "REJ".
same_srv_rate	continous	% de conexiones al mismo servicio.
diff_srv_rate	continous	% de conexiones a diferente servicio.
srv_diff_host_rate	continous	% de conexiones a diferentes hosts.
dst_host_count	continous	Número de conexiones al mismo host de destino.
dst_host_srv_count	continous	Número de conexiones al mismo servicio.
dst_host_same_srv_rate	continous	% de conexiones al mismo servicio del host de destino.
dst_host_diff_srv_rate	continous	% de conexiones a diferente servicio del host de destino.
dst_host_same_src_port_rate	continous	% de conexiones al mismo puerto del host de destino que el puerto de origen.
dst_host_srv_diff_host_rate	continous	% de conexiones a diferente puerto del host de destino que el puerto de origen.
dst_host_serror_rate	continous	% de conexiones que tienen errores "SYN".
dst_host_srv_serror_rate	continous	% de conexiones que tienen errores "SYN".
dst_host_rerror_rate	continous	% de conexiones que tienen errores "REJ".
dst_host_srv_rerror_rate	continous	% de conexiones que tienen errores "REJ".
label (solo en los ficheros etiquetados)	symbolic	"normal." si la conexión es normal o uno de los valores de la Tabla 3 si está etiquetada como intrusiva.

Tabla 2. Estructura de los ficheros de datos

Los ataques posibles están repartidos entre las cuatro categorías siguientes:

- **DOS**. Denegación de servicio. Ej.: inundación SYN.
- **R2L**. Acceso no autorizado desde una máquina remota. Ej.: adivinar contraseña.
- **U2R**. Acceso no autorizado a privilegios de súper usuario local (*root*). Ej.: ataque de *buffer overflow*.
- ***Probing*** **(sondeo)**. Vigilancia y otras pruebas. Ej.: escaneo de puertos.

La tabla siguiente contiene los 22 tipos de ataque que se encuentran en los ficheros de datos de entrenamiento y las categorías a las que pertenecen.

Tipo de ataque	Categoría
back	DOS
buffer_overflow	U2R
ftp_write	R2L
guess_passwd	R2L
imap	R2L
ipsweep	PROBE
land	DOS
loadmodule	U2R
multihop	R2L
neptune	DOS
nmap	PROBE
perl	U2R
phf	R2L
pod	DOS
portsweep	PROBE
rootkit	U2R
satan	PROBE
smurf	DOS
spy	R2L
teardrop	DOS
warezclient	R2L
warezmaster	R2L

Tabla 3. Tipos de ataques y categorías

2.4. Estructuración de los datos

Una vez realizada la exploración inicial, el siguiente paso es convertir los ficheros de datos a una estructura tabular que permita realizar un análisis más exhaustivo. Los sistemas más utilizados para este fin son las bases de datos (BD). Para un análisis puntual, con un objetivo fijo acotado a unos datos muy concretos y un volumen bajo-medio, se pueden utilizar herramientas menos complejas, como hojas de cálculo (Ej.: Excel).

Debido a que el objetivo del proyecto es obtener un producto capaz de cubrir con las necesidades reales de Ancert, la alternativa más acertada es utilizar un sistema de BD.

Existen dos tipos de BD: relacionales (RDBMS) y NoSQL. En el apartado siguiente realizamos una comparativa entre ambos sistemas para ver cuál es el que se mejor adapta a las necesidades del proyecto.

2.5. Comparativa SQL vs NoSQL

Las bases de datos relacionales son el modelo estándar de toda la vida y también el más utilizado en el mundo tecnológico. Constan de un lenguaje de peticiones estructuradas bastante robusto pero muy poco flexible. SQL se compone de bases de datos, compuestas por tablas que poseen filas con campos estructurados.

Pros

- Mayor soporte y herramientas, debido a que lleva más tiempo en el mercado.

- Es una tecnología ampliamente conocida y los perfiles que lo conocen son, mayoritariamente, económicos.

- Las bases de datos relacionales llevan una clase de "cabecera" que permite transacciones entre tablas. Esto significa que si hay un error durante la petición a cualquier nivel de la operación, se devuelve al punto inicial sin comprometer los datos que fueron utilizados durante el proceso.

- Los datos deben cumplir con el tipo definido en su estructura.

Contras

- Poca o nula flexibilidad. Todos los objetos ingresados deben tener los mismos campos y estar correctamente validados.

- Rendimiento y recursos. Necesita más procesamiento como más compleja sea la base de datos y sus relaciones, debido a la atomicidad.

- Escalabilidad reducida. Una BD relacional requiere un aumento de recursos de hardware que, generalmente, son bastante costosos para escalar su rendimiento.

Algunas BD relacionales conocidas son:

- Microsoft SQL Server
- MySQL
- Oracle
- PostgreSQL
- SQlite

Las BD NoSQL son un modelo nuevo que se ha puesto muy de moda entre desarrolladores *Full Stack*, porque no requiere un alto conocimiento académico de BD y su curva de aprendizaje y practicidad lo hacen bastante atractivo para proyectos rápidos. NoSQL se compone, generalmente, de BD compuestas a su vez por colecciones que poseen documentos. También hay otras tecnologías NoSQL que poseen columnas y estructuras diferentes.

Pros

- Su naturaleza descentralizada permite una alta escalabilidad. NoSQL es muy utilizada de una amplia forma en aplicaciones con Big Data.

- Más abierta y flexible a diferente tipos de datos.

- Se ejecutan en máquinas con pocos recursos, ya que no requieren de apenas computación, por lo que se pueden montar en máquinas de un coste reducido.

- Escalabilidad horizontal. Son capaces de crecer en número de máquinas en vez de en cantidad de recursos de hardware en una sola máquina. La mejora de rendimiento se consigue añadiendo más nodos, con la única operación de indicar al sistema cuáles son los nodos que están disponibles.

- Pueden manejar gran cantidad de datos, debido a que utiliza una estructura distribuida, en muchos casos mediante tablas hash.

Contras

- La integridad de los datos se ve comprometida por el poco soporte a transacciones, esto la hace más rápida pero menos segura al ejecutar consultas.

- No hay una estandarización y diferentes compañías poseen su propia solución.

- Muchas veces son poco compatibles con RDBMS.

- Suelen tener pocas herramientas de monitorización y casi todo su mantenimiento se realiza por consola.

Algunas BD NoSQL conocidas son:

- Apache Cassandra
- CouchDB
- MongoDB
- Redis

2.6. Elección del sistema de BD

Después de ver en el apartado anterior las características, ventajas y desventajas de cada uno de los dos sistemas de BD, se decide que el más idóneo para este proyecto es NoSQL. Algunos motivos por los que se ha tomado esta decisión son:

- Por el volumen de tráfico de conexiones que se generan en la red del cliente, se necesita una escalabilidad importante, que permita un crecimiento horizontal, añadiendo más servidores.
- Pueden ejecutarse en máquinas con pocos recursos, cosa que reduce el coste del proyecto considerablemente.
- Descentralización. Al soportar estructuras distribuidas, se pueden añadir servidores en varios puntos de la red del cliente.
- Optimización de consultas en BD para grandes cantidades de datos.
- Se necesita alto rendimiento y disponibilidad para gestionar grandes volúmenes de datos por encima de la atomicidad, consistencia, aislamiento y durabilidad (ACID).

La tabla siguiente ofrece una breve comparación entre las funcionalidades de las BD relacionales y las NoSQL. Cabe señalar que la tabla muestra una comparación a nivel de BD, no sobre los diversos sistemas de gestión de BD que implementan ambos modelos. Estos sistemas proporcionan sus propias técnicas patentadas para superar los problemas y deficiencias encontradas en el sistema, además de intentar mejorar significativamente el rendimiento y fiabilidad.

Característica	RDBMS	NoSQL
Rendimiento	Bajo	Alto
Confiabilidad	Buena	Pobre
Disponibilidad	Buena	Buena
Consistencia	Buena	Pobre
Almacenamiento de datos	De mediano a grande	Optimizado para datos enormes
Escalabilidad	Alta (pero más caro)	Alta

Tabla 4. Características de RDBMS y NoSQL

2.7. Tipos de bases de datos NoSQL

Dependiendo de la forma en la que almacenan la información, se pueden encontrar distintos tipos de BD NoSQL. A continuación se realiza un breve resumen de los más utilizados.

2.7.1. Clave-valor

Es el modelo de BD NoSQL más sencillo en cuanto a funcionalidad. En este tipo de sistema, cada elemento está identificado por una clave única, lo que permite la recuperación de la información de forma muy rápida, información que habitualmente está almacenada como un objeto binario (BLOB). Se caracterizan por ser muy eficientes, tanto para las lecturas como para las escrituras.

En este tipo de BD clave-valor, se utiliza una tabla hash en la que una clave única apunta a un elemento. Las claves pueden ser originadas por grupos de clave lógicos, requiriendo solamente estas claves para ser únicas dentro de su propio grupo. Esto permite tener claves idénticas en diferentes grupos lógicos.

Algunas implementaciones de las BD de clave-valor proporcionan mecanismos de almacenamiento en la caché, lo que mejora en gran medida su rendimiento.

Uno de los mayores defectos en esta forma de BD es la falta de consistencia a nivel de la BD. Ésta puede ser añadida por los desarrolladores mediante código propio, aunque esto suponga más esfuerzo y tiempo.

La base de datos NoSQL de clave-valor más famosa es DynamoDB de Amazon.

2.7.2. Orientadas a documentos

Las BD orientadas a documentos son similares a las BD de clave-valor, porque no tienen un esquema y se basan en un modelo de clave-valor. Ambos carecen de coherencia en el nivel de BD, lo que hace posible que las aplicaciones proporcionen más fiabilidad.

La diferencia más significativa es que en las BD orientadas a documentos los valores (documentos) proporcionan codificación XML, JSON o BSON (JSON codificado binario) para los datos almacenados.

La aplicación de base de datos más popular orientada a documentos es MongoDB.

2.7.3. Orientadas a columnas

Los datos se almacenan en columnas, en lugar de almacenarse en filas. Está compuesta por una o más familias de columnas que se agrupan de forma lógica en determinadas columnas en la base de datos. Una clave se utiliza para identificar y señalar a un número de columnas de la BD. Cada columna contiene filas de nombres o tuplas, y valores, ordenados y separados por comas.

Las BD de columnas tienen acceso rápido de lectura y escritura a los datos almacenados. En un almacén de columnas, las filas que corresponden a una sola columna se almacenan como una sola entrada de disco, lo cual facilita el acceso durante las operaciones de lectura y escritura.

El sistema NoSQL orientado a columnas más popular es Apache Cassandra, a pesar de que está considerado como un sistema híbrido compuesto por clave-valor y orientado a columnas.

2.7.4. Orientadas a grafo

En este tipo de BD, la información se representa como nodos y sus relaciones son aristas del mismo, de manera que se puede hacer uso de la teoría de grafos para recorrerla. Para sacar el máximo rendimiento a este tipo de BD, su estructura debe estar totalmente normalizada, de forma que cada tabla tenga una sola columna y cada relación dos.

Este tipo de BD ofrece una navegación más eficiente entre relaciones que en un modelo relacional.

El sistema más popular de este tipo Neo4j.

2.7.5. Orientadas a objetos

En este tipo de BD, la información se almacena como objetos en el paradigma homónimo. Las BD orientadas a objetos difieren de la serialización de los mismos en que deben proveer concurrencia, capacidad de recuperación y algún mecanismo para consultar sobre estos objetos.

El sistema más popular es DB4O.

2.8. Apache Cassandra

Después de ver los tipos de BD NoSQL más comunes y los sistemas de gestión más populares de cada uno de ellos, se ha decidido que para este proyecto se utilizará Apache Cassandra. Es un sistema de BD de código abierto diseñado para obtener escalabilidad y alta disponibilidad sin comprometer el rendimiento general del sistema.

Cuenta con factores como la escalabilidad lineal y la tolerancia a fallos en hardware básico o infraestructura en la nube, lo cual la convierten en la plataforma perfecta para datos de misión crítica. El respaldo de Cassandra para replicar en múltiples centros de datos es el mejor de su categoría, lo cual asegura una menor latencia para los usuarios, mejorando los niveles de rendimiento general.

Es una BD que admite la replicación local y de centros de datos múltiples para redundancia, *failover* y recuperación ante desastres.

Algunas de sus características son:

- **Disponibilidad**. Es reconocida por su fiabilidad hasta tal punto que es usada por más de 1.000 empresas a nivel mundial, dentro de las cuales destacan Instagram, GitHub y Netflix.

- **Tolerancia a fallos**. Cuenta con una amplia tolerancia a fallos al permitir que todos los datos alojados en ella tengan capacidad de ser replicados de forma automática en otros nodos, garantizando tanto la integridad como disponibilidad de estos. Cuando algún nodo presenta algún tipo de error, este puede ser reemplazado sin necesidad de apagar la BD, lo cual no afecta la operatividad y productividad de los usuarios.

- **Rendimiento**. Su alto rendimiento, mediante el cual se logra ir a un nivel más alto de las plataformas NoSQL ya conocidas, tanto a nivel de gestión como de aplicaciones.

- **Descentralizado**. Gracias a su administración descentralizada cada nodo involucrado en la BD es único, evitando así fallos masivos.

- **Escalable**. En este punto Cassandra es una de las mejores opciones ya que, según sus estadísticas, tiene la capacidad de soportar hasta más de 2.000 nodos, más de 400 TB de datos y alrededor de 1 billón o más de solicitudes al día, lo cual asegura su amplio alcance.

- **Durable**. Tiene la capacidad de permitir que los datos críticos siempre resten disponibles, independientemente del tipo de fallo que se presente, con esto se garantiza la óptima y correcta gestión de cada dato alojado.

- **Opciones de control**. Dispone de diversas opciones de administración, como la replicación sincrónica o asincrónica. En caso de usar la opción de gestión asincrónica, dispone de una BD que soporta funciones como *Hinted Handoff* y *Read Repair* para ampliar sus capacidades de uso.

- **Lenguaje CQL (Cassandra *Query Language*)**. Es un lenguaje de consultas muy parecido al SQL, lo cual evita invertir tiempo de aprendizaje en un lenguaje específico, como en otros sistemas. Ej: MongoDB.

El producto que se entrega junto con esta memoria contiene el documento "**Descarga e instalación de Apache Cassandra.pdf**" con las instrucciones para descargar e instalar Apache Cassandra 3.11.2 en un sistema operativo (SO) CentOS 7.

2.9. Importación de los datos

De los 6 ficheros de la KDD CUP, solo se van a utilizar los tres que tienen los datos etiquetados, ya que solo con ellos, es posible validar el porcentaje de acierto de las predicciones del algoritmo.

Antes de importar los datos a la BD, es necesario añadir un identificador único que sirva de clave primaria, ya que en Cassandra es obligatorio tener al menos uno por tabla. En el caso de no añadir un identificador único para cada registro, habría que seleccionar otro campo de los 42 posibles de los ficheros de datos y, como no hay ninguno que contenga un valor único para cada registro, no se importarían todas las filas, debido a que cada vez que encontrara una fila que el campo declarado como clave primaria tuviera un valor repetido, no lo añadiría por ya existir en la tabla.

También se ha añadido un campo con el nombre "label_class" para poder agrupar todos los tipos de ataques diferentes con el texto "ATTACK". Los registros de las conexiones normales quedan agrupadas con el texto "NORMAL". Este paso facilitará mucho el trabajo cuando se trabaje con el algoritmo de aprendizaje, ya que el objetivo es obtener un clasificador predictivo binario, es decir, que sea capaz de clasificar de manera predictiva entre 2 valores posibles. En este caso, los valores serán: NORMAL o ATTACK.

2.10. Distribución de los datos

Una vez que los ficheros están cargados en la BD, es el momento de realizar un análisis más profundo de los datos que contienen. Se empieza por la tabla "train_data", ya que es la que tiene más registros y es la que usará posteriormente para entrenar el algoritmo.

En primer lugar, se obtiene la cantidad de registros que hay etiquetados como conexiones intrusivas (label_class: ATTACK) y los etiquetados como conexiones normales (label_class: NORMAL). En la tabla siguiente se observa que el porcentaje de conexiones intrusivas es del 80,14% frente al 19,86% de conexiones etiquetadas como lícitas:

label_class	Cantidad	% vs Total
ATTACK	3.925.650	80,14 %
NORMAL	972.781	19,86 %
Total	4.898.431	

Tabla 5. Registros de la tabla train_data agrupados por label_class

Se continúa con el análisis sobre la misma tabla, pero ahora la agrupación se realiza por el campo “label”, que contiene el literal con el tipo de ataque específico. Estos datos se comparan más adelante con la tabla “validation_data”, para comprobar si la distribución de los datos es parecida entre ambas, ya que el algoritmo se validará con dicha tabla. Es ideal que la tabla de validación sea un subconjunto que represente de la forma más fiel posible la tabla de entrenamiento.

La tabla siguiente muestra la distribución de los datos agrupados por la etiqueta “label”:

label	Cantidad	% vs Total
back.	2.203	0,04497%
buffer_overflow.	30	0,00061%
ftp_write.	8	0,00016%
guess_passwd.	53	0,00108%
imap.	12	0,00024%
ipsweep.	12.481	0,25480%
land.	21	0,00043%
loadmodule.	9	0,00018%
multihop.	7	0,00014%
neptune.	1.072.017	21,88491%
nmap.	2.316	0,04728%
perl.	3	0,00006%
phf.	4	0,00008%
pod.	264	0,00539%
portsweep.	10.413	0,21258%
rootkit.	10	0,00020%
satan.	15.892	0,32443%
smurf.	2.807.886	57,32215%
spy.	2	0,00004%
teardrop.	979	0,01999%
warezclient.	1.020	0,02082%
warezmaster.	20	0,00041%
normal.	972.781	19,85903%
Total	4.898.431	

Tabla 6. Registros de la tabla train_data agrupados por label

A continuación, se analiza la tabla “validation_data” de la misma forma que se ha hecho con la tabla “train_data”. En la siguiente tabla puede observarse que la distribución de los datos por el campo “label_class” es casi idéntica a la de la tabla “train_data”:

label_class	Cantidad	% vs Total
ATTACK	396.743	80,31 %
NORMAL	97.278	19,69 %
Total	494.021	

Tabla 7. Registros de la tabla validation_data agrupados por label_class

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda

cqlsh:intrusion_detector> SELECT label_class, count(*) FROM validation_data GROUP BY label_class ;

 label_class | count
-------------+--------
      ATTACK | 396743
      NORMAL |  97278

(2 rows)

Warnings :
Aggregation query used without partition key
```

Fig. 2. Registros de validation_data agrupados por label_class (Cassandra)

Cuando se agrupa por el campo “label”, salvo en algunos casos de muy pocos registros, la distribución también es similar a la de la tabla “train_data”. Pueden observarse los registros agrupados por el campo “label” en la siguiente tabla:

label	Cantidad	% vs Total
back.	2.203	0,44593%
buffer_overflow.	30	0,00607%
ftp_write.	8	0,00162%
guess_passwd.	53	0,01073%
imap.	12	0,00243%
ipsweep.	1.247	0,25242%
land.	21	0,00425%
loadmodule.	9	0,00182%
multihop.	7	0,00142%
neptune.	107.201	21,69968%
nmap.	231	0,04676%
perl.	3	0,00061%
phf.	4	0,00081%
pod.	264	0,05344%
portsweep.	1.040	0,21052%
rootkit.	10	0,00202%
satan.	1.589	0,32165%
smurf.	280.790	56,83766%
spy.	2	0,00040%
teardrop.	979	0,19817%
warezclient.	1.020	0,20647%
warezmaster.	20	0,00405%
normal.	97.278	19,69107%
Total	494.021	

Tabla 8. Registros de la tabla validation_data agrupados por label

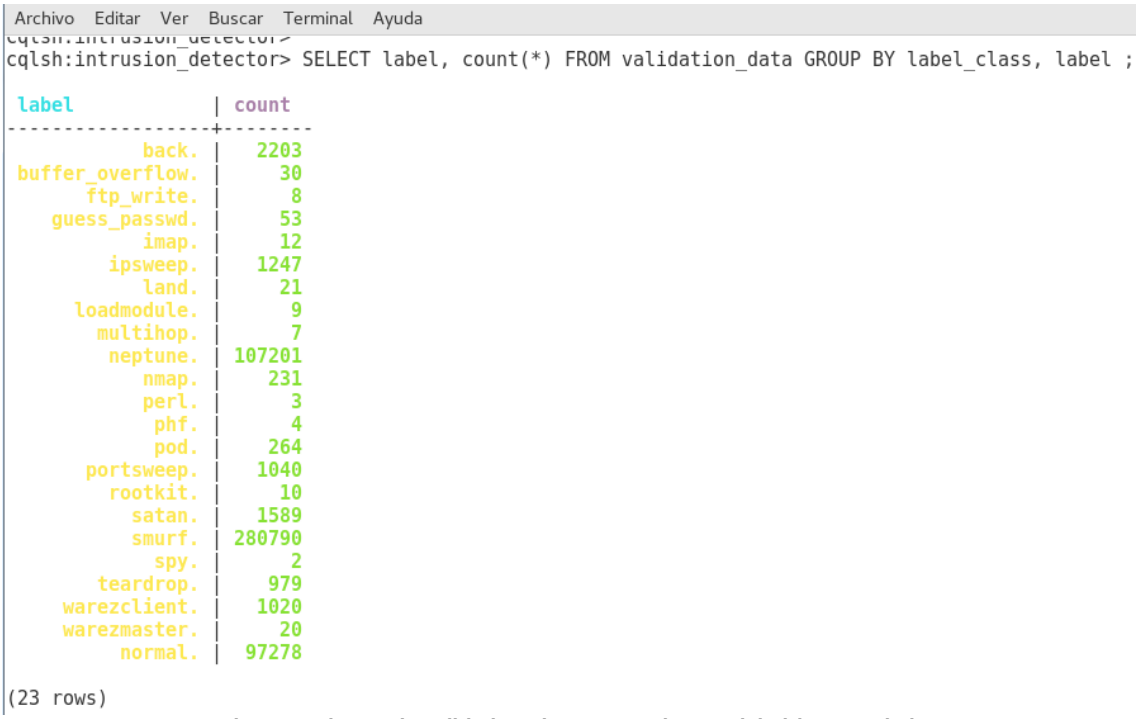


Fig. 3. Registros de validation_data agrupados por label (Cassandra)

Una vez que se ha comprobado que la distribución de los datos en las dos tablas, según los campos “label_class” y “label”, es casi idéntica, se puede decir que la tabla “validation_data” es una representación de ~10% de la tabla “train_data” y, por lo tanto, se podrá utilizar para validar el porcentaje de acierto del algoritmo que se creará en este proyecto.

Se finaliza con los análisis de las tablas repitiendo los realizados con las tablas precedentes, pero ahora con la tabla “test_data”. Se observa que en esta tabla el porcentaje de conexiones normales sobre intrusivas es el mismo que en las otras dos tablas, tal y como se muestra en la tabla siguiente:

label_class	Cantidad	% vs Total
ATTACK	250.436	80,52 %
NORMAL	60.593	19,48 %
Total	311.029	

Tabla 9. Registros de la tabla test_data agrupados por label_class

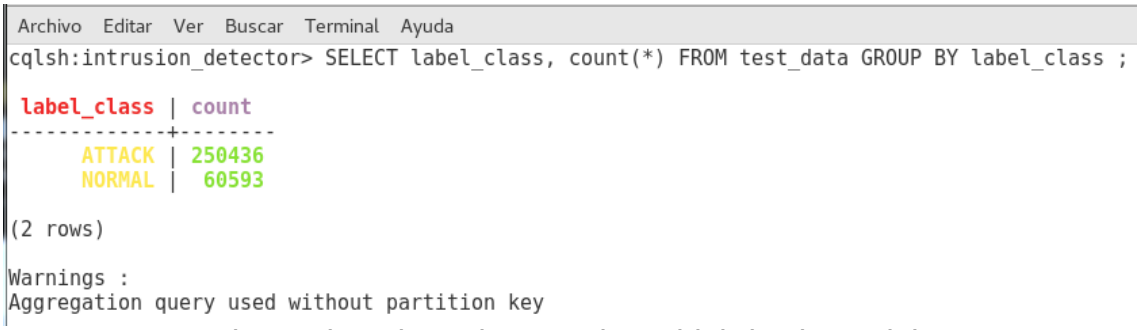


Fig. 4. Registros de test_data agrupados por label_class (Cassandra)

En cambio, cuando se realiza el análisis agrupando los registros por el campo “label”, se observa que hay 17 tipos de ataque que no están presentes en las dos tablas anteriores (en color rojo en la Tabla 10). El hecho de que la tabla de test contenga tipos de ataques que no están en las tablas de entrenamiento y validación servirá para comprobar si el algoritmo de predicción generaliza bien y, por lo tanto, no está sobreentrenado. Se explica lo que significa esto más adelante en esta memoria.

label	Cantidad	% vs Total
apache2.	794	0,25528%
back.	1.098	0,35302%
buffer_overflow.	22	0,00707%
ftp_write.	3	0,00096%
guess_passwd.	4.367	1,40405%
httptunnel.	158	0,05080%
imap.	1	0,00032%
ipsweep.	306	0,09838%
land.	9	0,00289%
loadmodule.	2	0,00064%
mailbomb.	5.000	1,60757%
mscan.	1.053	0,33855%
multihop.	18	0,00579%
named.	17	0,00547%
neptune.	58.001	18,64810%
nmap.	84	0,02701%
perl.	2	0,00064%
phf.	2	0,00064%
pod.	87	0,02797%
portsweep.	354	0,11382%
processtable.	759	0,24403%
ps.	16	0,00514%
rootkit.	13	0,00418%
saint.	736	0,23663%
satan.	1.633	0,52503%
sendmail.	17	0,00547%
smurf.	164.091	52,75746%
snmpgetattack.	7.741	2,48884%
snmpguess.	2.406	0,77356%
sqlattack.	2	0,00064%
teardrop.	12	0,00386%
updstorm.	2	0,00064%
warezmaster.	1.602	0,51506%
worm.	2	0,00064%
xlock.	9	0,00289%
xsnoop.	4	0,00129%
xterm.	13	0,00418%
normal.	60.593	19,48146%
Total	311.029	

Tabla 10. Registros de la tabla test_data agrupados por label

3. Inteligencia Artificial, *Machine Learning* y *Deep Learning*.

En los apartados siguientes se realiza una introducción conceptual de Inteligencia Artificial (IA), *Machine Learning* (ML) y *Deep Learning* (DL). A continuación, se listan los tipos de algoritmos de ML más utilizados hoy en día y sus usos más comunes. Posteriormente, se profundiza en el *Deep Learning*, que es la rama a la que pertenece el tipo de algoritmo que se va a utilizar para desarrollar el detector predictivo en este proyecto.

3.1. Definición y origen de IA

La Inteligencia Artificial (IA) es una de las ramas de la Informática, con fuertes raíces en otras áreas como la lógica y las ciencias cognitivas. Existen muchas definiciones de lo que es la IA, sin embargo, todas ellas coinciden en la necesidad de validar el trabajo mediante programas.

Es la simulación de procesos de inteligencia humana por parte de máquinas, especialmente sistemas informáticos. Estos procesos incluyen el aprendizaje (la adquisición de información y reglas para el uso de la información), el razonamiento (usando las reglas para llegar a conclusiones aproximadas o definitivas) y la autocorrección. Las aplicaciones particulares de la IA incluyen sistemas expertos, reconocimiento de voz y visión artificial.

El término IA fue acuñado por John McCarthy, un informático estadounidense, en 1956 durante la Conferencia de Dartmouth, en la que participaron los que más tarde han sido los investigadores principales del área. Para la preparación de la reunión, J. McCarthy, M. Minsky, N. Rochester y C. E. Shannon redactaron una propuesta en la que aparece por primera vez el término “inteligencia artificial”. Parece ser que este nombre se dio a instancias de J. McCarthy.

El documento define el problema de la inteligencia artificial como aquel de construir una máquina que se comporte de manera que si el mismo comportamiento lo realizara un ser humano, este sería llamado inteligente.

Existen, sin embargo, otras definiciones que no se basan en el comportamiento humano. Son las cuatro siguientes.

1. Actuar como las personas. Esta es la definición de McCarthy, donde el modelo a seguir para la evaluación de los programas corresponde al comportamiento humano. El llamado Test de Turing (1950) también utiliza este punto de vista. El sistema Eliza, un *bot* (programa software) conversacional es un ejemplo de ello.

2. Razonar como las personas. Lo importante es cómo se realiza el razonamiento y no el resultado de este razonamiento. La propuesta aquí es

desarrollar sistemas que razonen del mismo modo que las personas. La ciencia cognitiva utiliza este punto de vista.

3. Razonar racionalmente. En este caso, la definición también se focaliza en el razonamiento, pero aquí se parte de la premisa de que existe una forma racional de razonar. La lógica permite la formalización del razonamiento y se utiliza para este objetivo.

4. Actuar racionalmente. De nuevo el objetivo son los resultados, pero ahora evaluados de forma objetiva. Por ejemplo, el objetivo de un programa en un juego como el ajedrez será ganar. Para cumplir este objetivo es indiferente la forma de calcular el resultado.

Hoy en día, es un término general que abarca todo, desde la automatización de procesos robóticos hasta la robótica actual. Ha ganado prominencia recientemente debido, en parte, a los grandes volúmenes de datos, o al aumento de velocidad, tamaño y variedad de datos que las empresas están recopilando. La IA puede realizar tareas tales como identificar patrones en los datos de manera más eficiente que los seres humanos, lo que permite a las empresas obtener más información sobre sus datos.

La idea fundamental en que se basa la Inteligencia Artificial es en conseguir que una computadora resuelva un problema complejo como lo haría un humano. En ocasiones, esos problemas “complejos” para una persona no lo son tanto. Por ejemplo, para una persona, resulta muy sencillo:

- Identificar un animal en una foto.
- Descifrar un texto borroso, o en el que falta alguna letra.
- Identificar un sonido.
- Priorizar tareas.
- Conducir un coche.
- Jugar a un juego y ganar (incluso ganar al campeón mundial de ajedrez, aunque no sea buen ejemplo del algo “muy sencillo” para cualquier persona).
- Hacer algo creativo como escribir un poema o resumir una idea con un dibujo.

3.2. Tareas que resuelven las personas

Para poder emular el proceso, una computadora debe sumar a sus capacidades en cuanto a potencia de cálculo, velocidad de procesamiento y capacidad de almacenamiento de datos, otras nuevas que permitan imitar el razonamiento de la mente humana de forma funcional. Así, necesita poder:

- **Captar información del entorno: Percepción.** Los humanos, nos comunicamos con nuestro entorno por medio de nuestros sentidos. Hoy en día existen multitud de sensores de todo tipo que pueden realizar esta función, recogiendo información del entorno y enviándola para su

procesamiento a la computadora. Incluso pueden superar a los “sentidos humanos”, ya que no están sometidos a los límites de nuestra biología.

- **Comprender el lenguaje natural (*Natural Language Processing*).** Interpretar el lenguaje hablado y escrito. Comprender el significado de una frase, entender distintos acentos. Ésta es una tarea difícil, ya que el significado de una frase puede variar mucho según su contexto.

- **Representar el conocimiento.** La IA capaz de percibir personas, objetos, conceptos, palabras, símbolos matemáticos etc., necesita poder representar ese conocimiento en su “cerebro artificial”.

- **Razonar.** Ser capaz de conectar todo ese conocimiento, datos y conceptos, para poder resolver problemas usando la lógica. Por ejemplo, una máquina de ajedrez detecta los movimientos de las fichas sobre el tablero, y aplicando las reglas del ajedrez a los datos que ha recogido, decide la mejor jugada.

- **Planificar y desplazarse.** Para parecerse realmente a un humano, no basta con pensar como un humano. Nuestra IA debe ser capaz de moverse en un mundo tridimensional, eligiendo la ruta óptima. Esto es lo que hacen ya lo vehículos autónomos, pero deben hacerlo bien, porque en este caso, los errores cuestan vidas.

3.3. Factores que han ayudado a impulsar la IA

A lo largo de las últimas décadas aquello que en un principio parecía una utopía ha ido haciéndose realidad. Son varios los factores que han ayudado a impulsar el desarrollo de la IA.

Uno de los factores que más ha contribuido al avance de la IA, además de la inversión de las grandes tecnológicas en I+D, ha sido la ley de Moore. La ley de Moore no es una ley en el sentido científico, sino más bien una observación. En 1965 Gordon Moore predijo el incremento continuado de la complejidad de los circuitos integrados, (medida por el número de transistores contenidos en un chip de computador), al tiempo que se reducía su coste.

Esto permitió a la entonces naciente industria de semiconductores crear el microprocesador (en 1971) y otros circuitos integrados que en principio se aplicaban a las computadoras, pero hoy en día podemos encontrar en cualquier dispositivo (móviles, televisores, vehículos) o incluso en seres vivos (como los chips de identificación implantados en animales). Gracias a esto, las aplicaciones de Inteligencia Artificial forman hoy en día parte de nuestra vida cotidiana.

Otro de los factores que ha impulsado en gran medida el desarrollo de la Inteligencia Artificial (IA) han sido las tecnologías Big Data. En 2012, Google dio la campanada cuando demostró ser capaz de identificar la imagen de un gato en una foto con un 75% de precisión. Para lograrlo, utilizó redes

neuronales a las que entrenó con un corpus de 10 millones de vídeos de YouTube. Evidentemente, esto no hubiera sido posible sin recurrir al Big Data.

3.4. Tipos de IA

En función de los objetivos finales, las IA se clasifican en inteligencia artificial débil o estrecha e inteligencia artificial fuerte.

3.4.1. Inteligencia artificial débil

Se considera que los ordenadores únicamente pueden simular que razonan, y únicamente pueden actuar de forma inteligente. Las partidarios de la inteligencia artificial débil consideran que no será nunca posible construir ordenadores conscientes, y que un programa es una simulación de un proceso cognitivo pero no un proceso cognitivo en sí mismo.

Se caracteriza por estar “especializada” en una tarea concreta, tal como ganar a un juego concreto. Algunos ejemplos de inteligencia artificial débil son el IBM Deep Blue, que ganó en 1996 al gran maestro de ajedrez Gary Kasparov, o DeepMind’s AlphaGO, creada por Google en 2016, y que venció al jugador profesional de Go surcoreano Lee Sedol.

Los asistentes digitales como Siri y Cortana también son ejemplos de este tipo. Pueden dar la predicción meteorológica o recomendar una ruta alternativa para ir al trabajo, pero no leer mails y borrar los que no son importantes. No pueden ir más allá de aquello para lo que originalmente fueron programados.

3.4.2. Inteligencia artificial fuerte

En este caso se considera que un ordenador puede tener una mente y unos estados mentales, y que, por tanto, un día será posible construir uno con todas las capacidades de la mente humana. Este ordenador será capaz de razonar, imaginar, etc. Podría organizar el correo, reuniones, ganar al ajedrez y escribir un libro, entre otras. Puede ser inteligente, empática, adaptativa, etc. Aquí es donde entran el uso que se hace de algoritmos y aprendizaje guiado con el *Machine Learning* y el *Deep Learning*.

3.5. Definición de *Machine Learning*

Machine Learning o Aprendizaje Automático es una disciplina científica del ámbito de la Inteligencia Artificial que crea sistemas que aprenden automáticamente. Aprender en este contexto quiere decir identificar patrones complejos en millones de datos. La máquina que realmente aprende es un algoritmo que revisa los datos y es capaz de predecir comportamientos futuros.

Automáticamente, también en este contexto, implica que estos sistemas se mejoran de forma autónoma con el tiempo, sin intervención humana.

Los sistemas ML trabajan sobre grandes volúmenes de datos, identifican patrones de comportamiento y, basándose en ellos, son capaces de predecir comportamientos futuros. De esa forma son capaces de identificar a una persona por su cara, comprender un discurso, distinguir un objeto en una imagen, hacer traducciones y muchas otras cosas más. Es la herramienta más potente en el kit de IA para los negocios.

Por ello, las grandes empresas tecnológicas como Amazon, Baidu, Google, IBM, Microsoft y otras más, ofrecen sus propias plataformas de "*ML for business*".

3.6. ¿Cómo aprenden las máquinas?

El principal objetivo de todo aprendiz es desarrollar la capacidad de generalizar y asociar. Cuando traducimos esto a una máquina o computadora, significa que éstas deberían poder desempeñarse con precisión y exactitud, tanto en tareas familiares, como en actividades nuevas o imprevistas.

El aprendizaje automático se produce por medio de algoritmos. Un algoritmo no es más que una serie de pasos ordenados que se dan para realizar una tarea. El objetivo del ML es crear un modelo que nos permita resolver una tarea dada. Luego se entrena el modelo usando gran cantidad de datos. El modelo aprende de estos datos y es capaz de hacer predicciones.

Según la tarea que se quiera realizar, será más adecuado trabajar con un algoritmo u otro. Los modelos que obtenemos dependen del tipo de algoritmo elegido. Así, podemos trabajar con modelos geométricos, modelos probabilísticos o modelos lógicos. Por ejemplo, uno de los modelos lógicos más conocidos es el basado en el algoritmo árbol de decisión.

3.7. Tipos de algoritmos de ML

Los algoritmos de ML se dividen, principalmente en tres grandes categorías: *supervised learning* (aprendizaje supervisado), *unsupervised learning* (aprendizaje no supervisado) y *reinforcement learning* (aprendizaje por refuerzo). A continuación, detallamos las diferencias entre los tres tipos.

Para este proyecto en concreto, tal y como se ha escrito encima de estas líneas, debido a que se dispone de juegos de datos etiquetados y sin etiquetar, se ha tomado la decisión de utilizar solo los primeros y, por lo tanto, se diseñará un algoritmo de tipo supervisado.

Fig. 5. Tipos de algoritmos ML

3.7.1. *Supervised learning* (aprendizaje supervisado)

Es una técnica para deducir una función a partir de los datos de entrenamiento. Los datos de entrenamiento consisten en pares de objetos (normalmente vectores): un componente del par son los datos de entrada y el otro, los resultados deseados. En otras palabras, se puede enseñar a la máquina que con una serie de características un cliente es del tipo “A” y que el que tiene otras características no es el del tipo “A”. En el aprendizaje supervisado por tanto, el *data scientist* introduce los datos de entrada y salida y el ML se encarga de encontrar el patrón en la estructura interna de la información.

Son problemas que ya se han resuelto pero que seguirán surgiendo en un futuro. La idea es que las computadoras aprendan de una multitud de ejemplos, y a partir de ahí puedan hacer el resto de cálculos necesarios para que no haya que volver a ingresar más información.

Ejemplos: reconocimiento de voz, detección de spam, reconocimiento de escritura, entre otros.

3.7.2. *Unsupervised learning* (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 maquina debe de ser capaz de encontrar la estructura existente en los datos. Es muy útil para reducir la dimensionalidad de los datos reduciendo la pérdida de información o para el “*clustering*”.

En esta categoría lo que sucede es que al algoritmo se le despoja de cualquier etiqueta, de modo que no cuenta con ninguna indicación previa. En cambio, se le provee de una enorme cantidad de datos con las características propias de un objeto (aspectos o partes que conforman a un avión o a un coche, por ejemplo), para que pueda determinar qué es, a partir de la información recopilada.

Ejemplos: detectar morfología en oraciones, clasificar información, etc.

3.7.3. *Reinforcement learning* (aprendizaje por refuerzo)

En este caso particular, la base del aprendizaje es el refuerzo. La máquina es capaz de aprender con base a pruebas y errores en un número de diversas situaciones. Aunque conoce los resultados desde el principio, no sabe cuáles son las mejores decisiones para llegar a obtenerlos. Lo que sucede es que el algoritmo progresivamente va asociando los patrones de éxito, para repetirlos una y otra vez hasta perfeccionarlos y volverse infalible.

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" negativo. En cambio, si toma una acción acertada se le dan "premios" con valor positivo, para al final acabar encontrando una "buena" solución.

De esta forma se consigue 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 adecuando para trasladar la perspectiva subjetiva humana a un proceso que mejorará continuamente su aprendizaje y su efectividad predictiva.

Ejemplos: navegación de un vehículo en automático, toma de decisiones, etc.

3.8. Clasificación y regresión

La clasificación y regresión son conceptos del tipo de algoritmos supervisados. Un sistema de clasificación predice una categoría ("SI" o "NO", "A" o "B" o "C", "HOMBRE" o "MUJER", etc), mientras que un sistema de regresión predice un número ([1, 2, 3, …], [0.0, 0.25, 0.5, …], etc).

Un ejemplo de clasificación puede ser un sistema que clasifica los mails en spam o legítimos. Otro ejemplo cásico de este tipo es el de predicción de abandono, por ejemplo, en una empresa de telecomunicaciones. El objetivo en este caso es detectar los patrones de comportamiento de los clientes, que se utilizarán para predecir si se van a la competencia.

Por otro lado, la regresión predice un número, como cuál será el precio de un artículo, o el importe de deuda que una empresa de gestión de cobro va a recuperar en el mes de julio.

Para este proyecto, se necesita utilizar un sistema de clasificación, ya que el objetivo del mismo es clasificar las conexiones en intrusivas o lícitas, representadas por los literales "ATTACK" y "NORMAL", respectivamente. A este tipo de clasificación con tan solo dos clases, también se la denomina clasificación binomial.

3.9. Algoritmos supervisados de clasificación

A continuación se realiza una breve descripción de los principales algoritmos supervisados más utilizados para resolver problemas de clasificación.

3.9.1. Algoritmos de regresión logística

Los algoritmos de regresión logística modelan la relación entre distintas variables (*features*) utilizando una medida de error que se intentará minimizar en un proceso iterativo para poder realizar predicciones “lo más acertadas posible”. Se utilizan mucho en el análisis estadístico.

La idea es ser capaces de asignar una categoría u otra dadas unas características de entrada. Para ello se apoya en una función logística como la función “*sigmoide*”, que toma como entrada cualquier número real y devuelve un número real comprendido entre 0 y 1, y que se puede interpretar como una probabilidad.

La regresión logística tiene múltiples aplicaciones, como por ejemplo la valoración de riesgos, la clasificación de tumores o la detección de SPAM.

3.9.2. Algoritmos basados en instancia

Son modelos de aprendizaje para problemas de decisión con instancias o ejemplos (muestras) de datos de entrenamiento que son importantes o requeridos por el modelo. Aprendizaje basado en memoria en el que se crea un modelo a partir de una base de datos y se agregan nuevos datos comparando su similitud con las muestras ya existentes para encontrar “la mejor pareja” y hacer la predicción. Los algoritmos basados en instancia más usados son:

- K-Nearest Neighbor (kNN)
- Self-Organizing Map

3.9.3. Algoritmos de árboles de decisión

Modelan la toma de decisión basado en los valores actuales (reales) de los atributos que tienen los datos. Se utilizan sobre todo para clasificación de información, bifurcando y modelando los posibles caminos tomados y su probabilidad de ocurrencia para mejorar su precisión. Una vez armados, los arboles de decisión ejecutan muy rápido para obtener resultados. Los algoritmos de árbol de decisión más usados son:

- CART (*Classification And Regression Trees*)
- *Random Forest*

3.9.4. Algoritmos bayesianos

Son algoritmos que utilizan explícitamente el Teorema de Bayes de probabilidad para problemas de clasificación y regresión. Los más utilizados son:

- Naive Bayes
- Gaussian Naive Bayes
- Multinomial Naive Bayes
- Bayesian Network

3.9.5. Algoritmos SVM (*Suport Vector Machine*)

La idea del algoritmo es ser capaces de encontrar, con los datos de entrenamiento, un hiperplano que maximiza la distancia a las diferentes clases, lo que es conocido como el "margen máximo". Una vez hallado este hiperplano puede usarse para clasificar nuevos puntos. SVM tiene múltiples aplicaciones, por ejemplo para reconocimiento de imágenes, clasificación de texto o aplicaciones en el área de la biotecnología.

SVM permite utilizar las llamadas funciones *kernel* (no lineales). Estas funciones resuelven el problema de clasificación trasladando los datos a un espacio donde el hiperplano solución es lineal y, por tanto, más sencillo de obtener. Una vez conseguido, la solución se transforma, de nuevo, al espacio original.

3.9.6. Algoritmos de redes neuronales

Son algoritmos inspirados en las funciones biológicas de las redes neuronales. Se suelen utilizar para problemas de clasificación y regresión pero realmente tienen un gran potencial para resolver multitud de problemáticas. Son muy buenos para detectar patrones. Las redes neuronales artificiales requieren mucha capacidad de procesamiento y memoria y estuvieron muy limitadas por la tecnología del pasado hasta estos últimos años, en los que resurgieron con mucha fuerza dando lugar al *Deep Learning*. Las redes neuronales básicas y clásicas son:

- Perceptron
- Back-propagation
- Hopfield Network

3.9.7. Algoritmos de *Deep learning*

Son la evolución de las redes neuronales artificiales que aprovechan el abaratamiento de la tecnología y la mayor capacidad de ejecución, memoria y disco para explotar gran cantidad de datos en enormes redes neuronales

interconectadas en diversas capas que pueden ejecutar en paralelo para realizar cálculos. Los algoritmos más populares de *Deep Learning* son:

- *Convolutional Neural Networks*

3.10. *Deep Learning*

El *Deep Learning* (DL) o aprendizaje profundo es un subcampo específico del *Machine Learning*, una nueva forma de aprender representaciones a partir de datos que pone énfasis en aprender sucesivas "capas" de representaciones cada vez más significativas.

Lleva a cabo el proceso de *Machine Learning* usando una red neuronal artificial que se compone de un número de niveles jerárquicos. En el nivel inicial de la jerarquía la red aprende algo simple y luego envía esta información al siguiente nivel. El siguiente nivel toma esta información sencilla, la combina, compone una información algo un poco más compleja, y se lo pasa al tercer nivel, y así sucesivamente.

El *Deep Learning* permite a modelos computacionales que están compuestos de múltiples capas de procesamiento aprender representaciones de datos con múltiples niveles de abstracción. Estas representaciones en capas se aprenden a través de modelos llamados "redes neuronales", estructurados en capas apiladas una detrás de la otra.

En realidad, lo que utiliza DL es algo llamado red neuronal artificial (ANN por sus siglas en inglés), que es una red inspirada en redes neuronales biológicas que se utilizan para estimar o aproximar funciones que pueden depender de una gran cantidad de entradas que generalmente son desconocidas.

Aunque el aprendizaje profundo es un subcampo bastante antiguo del aprendizaje automático, solo alcanzó prominencia a principios de esta década. En los pocos años transcurridos desde entonces, ha logrado grandes cosas, tales como:

- Clasificación de imágenes a nivel casi humano.
- Reconocimiento de voz a nivel casi humano.
- Transcripción de escritura a mano a nivel humano.
- Mejora de la traducción automática.
- Mejora de la conversión de texto a voz.
- Asistentes digitales como Google Now o Amazon Alexa.
- Conducción autónoma a nivel humano.

- Mejora de la orientación de anuncios, tal como la utilizan Google, Baidu y Bing.

- Mejora de los resultados de búsqueda en la web.

- Responder preguntas de lenguaje natural.

- Detección, predicción y prevención de amenazas sofisticadas en tiempo real en el campo de la ciberseguridad.

- Identificación en textos de sentimientos positivos y negativos, temas y palabras clave.

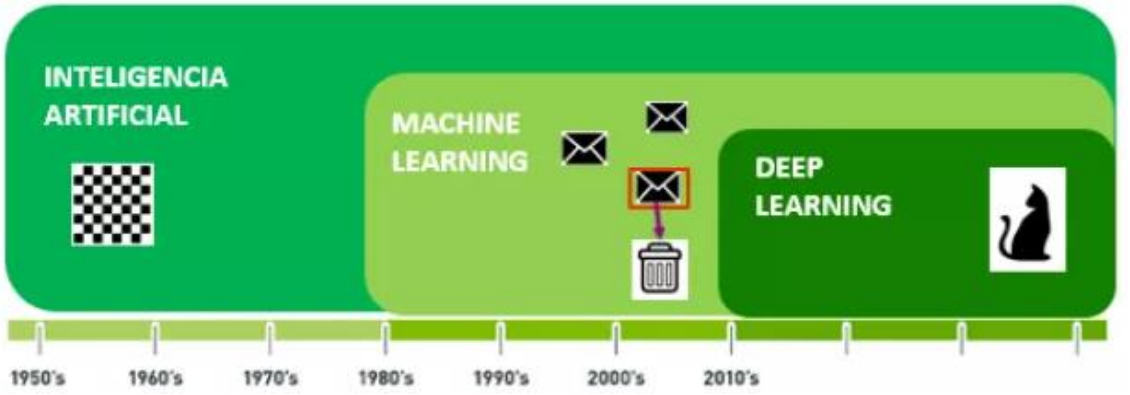


Fig. 6. De la IA al *Deep Learning* pasando por *Machine Learning*

4. Herramientas de trabajo

En los apartados siguientes se describen algunas herramientas (lenguajes de programación, IDE y librerías) que se utilizan hoy en día para desarrollar algoritmos de *Deep Learning*. Se justifica las que se han escogido para llevar elaborar este proyecto.

4.1. Lenguajes de programación

A continuación se describen brevemente cuatro de los lenguajes de programación más populares en *Machine Learning*.

4.1.1. Python

Es un lenguaje de programación versátil multiplataforma y multiparadigma que se destaca por su código legible y limpio. La licencia de código abierto permite su utilización en distintos contextos sin la necesidad de abonar por ello y se emplea en plataformas de alto tráfico como Google, YouTube o Facebook. Su objetivo es la automatización de procesos para ahorrar tanto complicaciones como tiempo, los dos pilares en cualquier tarea profesional. Dichos procesos se reducirán en pocas líneas de código que pueden insertarse en una variedad de plataformas y sistemas operativos.

Phyton es ideal para trabajar con grandes volúmenes de datos porque favorece su extracción y procesamiento, siendo el elegido por las empresas de Big Data. A nivel científico, posee una amplia biblioteca de recursos con especial énfasis en las matemáticas para aspirantes a programadores en áreas especializadas. También es útil para crear videojuegos gracias a su dinamismo y simplicidad, aunque tratándose de un lenguaje de programación interpretado, es más lento que Java, C++ o C#. Cuenta con una comunidad de usuarios muy activos que comparten constantemente sus conocimientos y recursos en línea.

Fig. 7. Logo de Python

4.1.2. R

R es un lenguaje que combina el paradigma funcional y orientado a objetos, en este sentido es muy versátil y extensible mediante los paquetes disponibles en CRAN - *Comprehensive* R *Archive Network*.

.

R es un lenguaje interpretado (como Java) y no compilado (como C, C++, Fortran, Pascal, etc.), lo cual significa que los comandos escritos en el teclado son ejecutados directamente sin necesidad de construir ejecutables.

Cuando se ejecuta un comando en R este manipula objetos que están en la memoria volátil del computador, no se utilizan archivos o almacenamiento secundario. Este hecho puede ser una de las ventajas ya que ofrece un buen desempeño para ciertas tareas, pero para la manipulación de grandes volúmenes de datos puede ser una desventaja. Se utilizan archivos de entrada y salida para consumir datos y para generar los resultados. El origen de los datos puede ser local o en línea, es decir, los datos y las estructuras que se utilizan para manipularlos deben entrar en la memoria del computador para poder trabajar con R.

Los resultados se pueden visualizar directamente en la pantalla, se pueden guardar en objetos o se pueden almacenar en archivos de la computadora.

Las funciones disponibles están guardadas en una paquetes (librerías) disponibles desde el mismo momento en que se ejecuta la línea de comandos o se pueden cargar a la memoria al momento de ejecutar algún programa.

Fig. 8. Logo de R

4.1.3. Octave

Es un lenguaje de alto nivel destinado para el cálculo numérico. Provee una interfaz sencilla, orientada a la línea de comandos (consola), que permite la resolución de problemas numéricos, lineales y no lineales, además permite la ejecución de scripts y puede ser usado como lenguaje orientado al procesamiento por lotes.

Octave nació alrededor del año 1988, y fue concebido originalmente para ser usado en un curso de diseño de reactores químicos para los alumnos de Ingeniería Química de la Universidad de Texas y la Universidad de Wisconsin-Madison.

Posee una gran cantidad de herramientas que permiten resolver problemas de álgebra lineal, cálculo de raíces de ecuaciones no lineales, integración de funciones ordinarias, manipulación de polinomios, integración de ecuaciones diferenciales ordinarias y ecuaciones diferenciales algebraicas. Sus funciones también se pueden extender mediante funciones definidas por el usuario escritas en el lenguaje propio de Octave o usando módulos

dinámicamente cargados escritos en lenguajes como C, C++ y Fortran entre otros.

Fig. 9. Logo de Octave

4.1.4. Matlab

Es un lenguaje de alto desempeño diseñado para realizar cálculos técnicos. MATLAB integra el cálculo, la visualización y la programación en un ambiente fácil de utilizar donde los problemas y las soluciones se expresan en una notación matemática.

MATLAB es un sistema interactivo cuyo elemento básico de datos es el arreglo que no requiere de dimensionamiento previo. Esto permite resolver muchos problemas computacionales, específicamente aquellos que involucren vectores y matrices, en un tiempo mucho menor al requerido para escribir un programa en un lenguaje escalar no interactivo tal como C o Fortran.

Fig. 10. Logo de Matlab

4.2. Métricas para la elección del lenguaje correcto

A continuación se listan las métricas para decidir sobre el lenguaje de programación correcto entre los anteriores:

- **Curva de aprendizaje**. R es más funcional, mientras que Python está más orientado a objetos. Por lo tanto, si tiene más experiencia en la programación orientada a objetos, aprender Python es más fácil que R y viceversa. Matlab y Octave son más como escribir ecuaciones matemáticas que una vez más es muy fácil de aprender. Entonces, los cuatro lenguajes están bastantes igualados.

- **Velocidad**. R se construyó como un lenguaje estadístico y, por lo tanto, tiene incorporado más soporte estadístico y de análisis de datos, donde Python depende de los paquetes. Debido a estos paquetes incorporados, R es ligeramente más rápido en tareas estadísticas relacionadas.

- **Soporte de la comunidad**. Octave es menos popular que otros lenguajes de programación, por tanto, tiene un apoyo de la comunidad ligeramente menor que R, Python y Matlab. En esta métrica, Octave pierde en comparación con los otros tres.

- **Coste económico**. Matlab es un software propietario que necesita una licencia para su uso, mientras que los otros tres son software gratuito (de código abierto) y no tienen coste alguno para su uso. Aquí es donde Matlab pierde un poco en comparación con los otros tres.

- **Compatibilidad con DNN (*Deep Neural Net*) *Frameworks***. Caffe y Tensorflow son dos *frameworks* de los más populares hoy en día. Caffe es compatible con Matlab y Python. Tensorflow es compatible con R y Python. Además, entre otros *frameworks* de DNN menos populares, como Theano, Python es el único lenguaje que tiene soporte universal. Esto da a Python una ventaja clara sobre otros lenguajes.

- **Preparado para producción**. R es más adecuado para el análisis estadístico. Matlab y Octave son más adecuados para tareas relacionadas con la visión por computador, pero Python es más adecuado para cualquier tarea genérica, como el procesamiento previo de datos y el postprocesado de resultados. Además, Python es lo suficientemente genérico que lo hace más adecuado si existe la necesidad de integrar ML con otro software.

Teniendo en cuenta las métricas anteriores si se valora con 1 punto al lenguaje en el que tiene una ventaja sobre los demás, el resumen queda de la siguiente forma:

Lenguaje	Velocidad	Curva de aprendizaje	Preparado para producción	Soporte de la comunidad	Coste económico	Soporte *frameworks* DNN	Total
Python	0	1	1	1	1	1	5
R	1	1	0	1	1	0	4
Octave	1	1	0	0	1	0	3
Matlab	1	1	0	1	0	0	3

Tabla 11. Valoraciones de las métricas

En base a las métricas mencionada, es evidente que Python es la mejor opción de los 4 lenguajes, principalmente porque es lo suficientemente genérico como para aplicarse no solo para tareas relacionadas con el aprendizaje estadístico / automático, sino también para otras tareas genéricas y dispone mejor soporte para todos los *frameworks* DNN, como Tensorflow o Caffe. Pero, R puede ser muy útil para un prototipo rápido que no necesita usar *frameworks* DNN.

En base a estas conclusiones, se toma la decisión de utilizar Python en este proyecto. El producto que se entrega junto con esta memoria contiene el

documento “Descarga e instalación de Python 3.pdf” con las instrucciones para descargar e instalar Python 3.6 en un sistema operativo (SO) CentOS 7.

4.3. Librerías

A continuación se describen brevemente cuatro de las librerías de *Deep Learning* más populares en la actualidad.

4.3.1. TensorFlow

Es una biblioteca de software de código abierto para *Machine Intelligence*. TensorFlow fue desarrollado por el equipo de Google Brain y lo convirtieron en código abierto el 9 de noviembre de 2015. Es un sucesor de DistBelief Net que tiene un sistema de aprendizaje automático basado en redes neuronales. Es muy eficiente para el cálculo numérico usando gráficos de flujo de datos. Los nodos en el gráfico representan operaciones matemáticas, mientras que los bordes del gráfico representan los conjuntos de datos multidimensionales (tensores) comunicados entre ellos. La arquitectura flexible le permite implementar cálculos en una o más CPU o GPU en una computadora de escritorio, servidor o dispositivo móvil con una sola API.

Fig. 11. Logo de TensorFlow

4.3.2. Theano

Es una biblioteca de Python que permite definir, optimizar y evaluar expresiones matemáticas que involucran matrices multidimensionales de manera eficiente. Es similar a NumPy, junto con operaciones y expresiones matemáticas. Theano fue desarrollado por el grupo de *Machine Learning* de la Université de Montréal. La biblioteca también optimiza el uso de la GPU y la CPU y hace que el rendimiento de los cálculos intensivos de datos sea aún más rápido. En realidad, sirve como los bloques de construcción para las redes neuronales, mientras que NumPy sirve los bloques de construcción para la informática científica.

Fig. 12. Logo de Theano

4.3.3. Caffe

Es un marco de aprendizaje profundo hecho con expresión, velocidad y modularidad en mente. Es desarrollado por el Centro de Visión y Aprendizaje de Berkeley (BVLC) y por colaboradores de la comunidad. DeepDream de Google está basado en Caffe Framework. Caffe no es una biblioteca de Python, pero proporciona enlaces en el lenguaje de programación de Python. En realidad, es una biblioteca C ++ con licencia de BSD que proporciona una interfaz de python.

Fig. 13. Logo de Caffe

4.3.4. MXNet

Es una biblioteca flexible y eficiente para *Deep Learning*. Destaca que admite más de 7 enlaces de idiomas diferentes, incluidos C ++, Python, R, Javascript e incluso Matlab. Si se desea velocidad de entrenamiento en múltiples CPU o GPU, MXNet es ideal, ya que admite la computación distribuida.

Fig. 14. Logo de MXNet

4.4. Elección de librería

Para este proyecto se decide que se va a utilizar la librería TensorFLow por los motivos siguientes:

- **Implementación fácil**. Una de las desventajas de alguna librerías como Caffe, por ejemplo, es que no existe un mecanismo fácil para la instalación, como el administrador de paquetes pip de Python implementado por Tensorflow. Esto evita tener que compilar siempre desde la fuente.

- **API de alto nivel para usar y compartir API**. Dispone de una API de alto nivel para construir modelos. Se pueden obtener modelos pre-entrenados con una sola línea de Python. También

ofrece una instrucción para evaluar una gama de modelos en un paquete tipo *scikit*.

- **Gestión del ciclo de vida para desarrolladores**. Tiene una interfaz buena para Python, que se está convirtiendo cada vez más en el lenguaje de elección para los científicos de datos. La interfaz de otras librerías como Caffe, por ejemplo, es mucho más centrada en C ++, lo que requiere que los usuarios realicen tareas como crear archivos de configuración y plantarlos en el disco para cada nuevo trabajo de aprendizaje automático.

- **Soporte para GPU**. Todos los ajustes necesarios se realizan a través de *tf.device* (), donde se designa el uso de GPU. No se necesita documentación adicional, ni se requieren cambios en la API. Además, TensorFlow es más flexible en términos de arquitectura: puede ejecutar dos copias de un modelo en dos GPU o un único modelo grande en dos GPU.

- **Mejor soporte para configuraciones de máquinas múltiples**. El soporte para múltiples máquinas es igualmente fácil con TensorFlow. Ofrece una manera fácil de configurar trabajos para trabajos de múltiples nodos, simplemente ajustando *tf.device* () con la cantidad de máquinas en las que se puede ejecutar el trabajo.

El producto que se entrega junto con esta memoria contiene el documento “Descarga e instalación de TensorFlow.pdf” con las instrucciones para instalar la librería TensorFlow en Python 3.

5. Diseño e implementación del algoritmo

En este apartado, se detallan los pasos y justifican las decisiones que se han llevado a cabo para realizar el diseño e implementación del algoritmo predictivo de tipo clasificador binomial, mediante el uso de redes neuronales profundas de TensorFlow.

5.1. Importación de datos

El planteamiento inicial era hacer el entrenamiento con los datos de la tabla “train_data”, una primera validación con los datos de la tabla “validation_data” y una segunda validación con los datos de la tabla “test_data”. Después de realizar varias pruebas con el algoritmo, se ha cambiado el planteamiento a realizar la fase de entrenamiento con la tabla “validation_data” que, tal y como se ha demostrado en el apartado 2.10 de esta memoria, es una representación fiel del ~10% de la tabla “train_data”, una primera validación con un subconjunto (20%) aleatorio de la misma tabla y una segunda validación con la tabla “test_data”, ésta última sin cambios respecto al planteamiento inicial.

Este cambio se ha realizado porque el objetivo del proyecto es conseguir un algoritmo con un porcentaje de acierto elevado pero a su vez, lo más eficiente posible. Las pruebas de entrenamiento realizadas con la tabla “validation_data” han devuelto porcentajes de acierto casi idénticos a los conseguidos con la tabla “train_data”, además de que el tiempo de entrenamiento se ha reducido en unas 10 veces usando la primera tabla en comparación con la segunda. Solo en la fase de importación de datos, la tabla “train_data” necesita casi 30 minutos, mientras la tabla “validation_data” lo hace en menos de 2’. El tiempo total empleado por el algoritmo para cubrir todas las fases de creación del modelo con la tabla “validation_data” es de ~5 minutos, mientras que con la tabla “train_data” superaba los 50’.

La importación de los datos se realiza mediante Pandas, que es una librería de Python destinada al análisis de datos proporcionando unas estructuras de datos flexibles que permiten trabajar con los mismos de forma muy eficiente. Los datos de las tablas se importan a *DataFrame* (estructuras de datos similares a las tablas de bases de datos relacionales como SQL) desde los ficheros CSV que contienen las exportaciones de las tablas “validation_data” y “test_dada” de la BD Cassandra.

A continuación se listan los pasos realizados para la importación de los datos:

- Exportación de las tablas “validation_data” y “test_data” a ficheros CSV mediante el *script* **tables_data_export.sh** (Anexo 2).

- Importación de los ficheros CSV generados en el paso anterior a *DataFrame* mediante los comandos siguientes:
  df_train = pd.read_csv('./validation_data_out.csv')
  df_test = pd.read_csv('./test_data_out.csv')

- Separación de las columnas de características de la columna de las etiquetas.

  DataFrame para el entrenamiento:

  ```
  x_train = df_train.iloc[ : , 3:] #Todas las columnas menos label_class, label y id
  y_train = df_train.iloc[ : , 0:1] #Solo la columna label_class
  ```

  DataFrame para la primera validación:

  ```
  df_validation = df_train.sample(frac=0.2) #muestra aleatoria del 20% del DataFrame de entrenamiento
  x_validation = df_validation.iloc[ : , 3:] #Todas las columnas menos label_class, label y id
  y_validation = df_validation.iloc[ : , 0:1] #Solo la columna label_class
  ```

  DataFrame para la segunda validación y predicción:

  ```
  x_test = df_test.iloc[ : , 3:] #Todas las columnas menos label_class, label y id
  y_test = df_test.iloc[ : , 0:1] #Solo la columna label_class
  ```

5.2. Asignación de los valores iniciales a los hiperparámetros

Los hiperparámetros se utilizan para parametrizar el proceso de instanciación del modelo y no para modelar los datos directamente, pero influyen en la capacidad y características del aprendizaje. Los hiperparámetros de un clasificador o estimador no son directamente aprendidos en el paso de aprendizaje estadístico de los datos de entrenamiento, sino que son optimizados de forma separada.

El objetivo de la optimización de los hiperparámetros es el de mejorar el desempeño del clasificador y lograr una buena generalización del algoritmo de aprendizaje. En la fase de ajuste del algoritmo se realizan pruebas de ejecución con combinaciones de valores diferentes asignadas a los hiperparámetros. Al final de todas las ejecuciones se analiza cual es la combinación de valores que obtiene los mejores resultados y se establece como el modelo más óptimo.

A continuación se describen brevemente cada uno de los hiperparámetros que se han usado en el diseño del clasificador de este proyecto, y se indican los valores en la asignación inicial.

- ***Epochs* (iteraciones)**. Es la cantidad de veces que recorre el conjunto de datos completo extrayendo lotes para alimentar el algoritmo de aprendizaje. Este parámetro de ajusta en orden ascendente, empezando con valores bajos/medios. En el algoritmo se han definido los dos siguientes:

*train_num_epochs***.** Valor inicial: 10. Definido para la fase de entrenamiento.
validation_num_epochs. Valor inicial: 1. Definido para la fase de la primera validación. En la segunda validación se deja el valor por defecto: 1.

- ***Batch size* (tamaño del lote)**. Valor inicial: 128 (por defecto). Es el tamaño del trozo de datos que alimenta en cada ciclo el algoritmo de aprendizaje. Puede alimentar todo el conjunto de datos, en cuyo caso el tamaño del lote es igual al tamaño del conjunto de datos. También puede alimentar un ejemplo a la vez o un número N de ejemplos.

- ***Steps* (pasos)**. Valor inicial: 1.000. Es el número de veces que se ejecutará el ciclo de entrenamiento en su algoritmo de aprendizaje para actualizar los parámetros en el modelo. En cada iteración de ciclo, procesará un fragmento de datos, que es básicamente un lote. Generalmente, este ciclo se basa en el algoritmo de gradiente descendente. Se ajusta en orden ascendente. El número de pasos máximos viene dado por la fórmula siguiente: máx.(pasos) = (tamaño conjunto entreno / tamaño lote) * número de iteraciones.

- ***Learning rate* (ratio de aprendizaje)**. Valor inicial: 0.001. La tasa de aprendizaje es uno de los hiperparámetros más importantes de ajuste en el entrenamiento de redes neuronales profundas. Los modelos de aprendizaje profundo generalmente son entrenados por un optimizador de descenso de gradiente estocástico. Hay muchas variaciones de descendencia de gradiente estocástica: Adam, RMSProp, Adagrad, etc. Todas permiten establecer la velocidad de aprendizaje. Este parámetro le dice al optimizador qué tan lejos debe mover los pesos en la dirección del gradiente para un mini lote. Si la tasa de aprendizaje es baja, el entrenamiento es más confiable, pero la optimización llevará mucho tiempo porque los pasos hacia el mínimo de la función de pérdida son pequeños. Si la tasa de aprendizaje es alta, el entrenamiento puede no converger o incluso divergir. Los cambios de peso pueden ser tan grandes que el optimizador sobrepasa el mínimo y empeora la pérdida. En primer lugar, con tasas bajas de aprendizaje, la pérdida mejora lentamente, luego el entrenamiento se acelera hasta que la tasa de aprendizaje se vuelve demasiado grande y la pérdida aumenta: el proceso de entrenamiento diverge. Es necesario seleccionar un punto con la disminución más rápida de pérdida. Se ajusta en orden ascendente.

- **Número de capas ocultas**. Valor inicial: 2. Lo que diferencia una red neuronal de una red neuronal profunda son las capas de neuronas ocultas. La capa oculta es donde la red guarda la

representación abstracta interna de los datos de entrenamiento. Una red con más capas se vuelve más profunda, lo que se traduce en mayor aprendizaje, pero también más complejidad. Hay que ajustar el número de capas de manera que se consiga un porcentaje de acierto elevado pero de la forma más eficiente posible. Se ajusta en orden ascendente.

- **Número de neuronas en las capas ocultas**. Valor inicial: 32. El número de neuronas es más o menos ilimitado, es decir, puede ser más pequeño o más grande que el tamaño de la capa de entrada. No hay una regla universal que se pueda aplicar para calcular el número adecuado de neuronas en las capas ocultas, pero una de las reglas empíricas, dice que el tamaño de la capa oculta debe estar comprendido entre las capas de entrada y salida. Entre no significa en el medio, de hecho, generalmente es mejor establecer el tamaño de la capa oculta más cerca del tamaño del vector de entrada. La razón es que si la capa oculta es demasiado pequeña, la red podría tener una convergencia difícil. Una capa oculta más grande le da a la red más capacidad que la ayuda a converger, en comparación con una capa oculta más pequeña. Para calcular el número de neuronas de las capas ocultas se ha seguido la fórmula: (nº de características de entrada + nº de clases de salida) * 2/3 => (41+2)*(2/3) = ~28. Redondeado a potencia de 2 => 32.

- **Número de neuronas de la capa de entrada**. Coinciden con el número de variables de entrada del conjunto de entrenamiento (41).

- **Número de neuronas de la capa de salida**. Coinciden con el número de clases en la que se quieren clasificar los datos (2).

5.3. Transformación de las variables categóricas

Las redes neuronales operan únicamente con números, ya que cada neurona realiza operaciones de multiplicación y suma en los pesos y datos de entrada. Sin embargo, los datos de entrada en la vida real a menudo contienen datos no numéricos (categóricos), tales como género (masculino / femenino), nacionalidad (ESP, FRA, GER, …).

Entre las 41 variables de entrada de los conjuntos de entrenamiento, validación y test, que se usan en este proyecto, las siguientes variables son de tipo categórico:

- flag
- protocol_type
- service

Existen numerosas alternativas para transformar variables categóricas en numéricas, tales como crear un diccionario que asigne un valor numérico a cada una de las variables categóricas (ejemplo: 1: femenino, 2: masculino) o crear tantas variables como valores diferentes haya de cada una de las categorías, por ejemplo, la variable “género” se separaría en dos variables, cada una con el nombre de una categoría, es decir, “masculino” y “femenino”, y se asignaría el valor “1” a la columna que correspondiera en cada caso, y “0” al resto. El género masculino se representaría con un “1” en la variable “masculino” y un “0” en la variable “femenino”, y viceversa.

Las alternativas anteriores son viables si la cantidad de valores posibles en cada variable categórica es muy reducida, pero el proceso se hace muy tedioso si existen muchos valores diferentes en una misma variable, por ejemplo, “nacionalidad”.

Tensorflow dispone de “*feature_column*” (columnas de características), que actúan como intermediarias entre los datos en bruto y los estimadores. Permiten transformar una amplia gama de datos sin procesar en formatos que los estimadores pueden usar, lo que permite una experimentación fácil. A continuación se listan las funciones disponibles de TensorFlow para transformar variables categóricas a numéricas:

- ***Categorical identity column*** **(columna de identidad categórica)**. Cada vector representa un entero único. Por ejemplo, para representar el rango del 0 al 3 se haría de la siguiente forma:

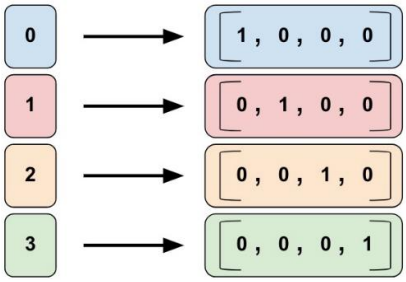


Fig. 15. Representación de columna de identidad categórica

Función: *tf.feature_column.categorical_column_with_identity*

- ***Categorical vocabulary column*** **(columna de vocabulario categórico)**. Las columnas de vocabulario categórico proporcionan una buena manera de representar cadenas como un vector único. Por ejemplo:

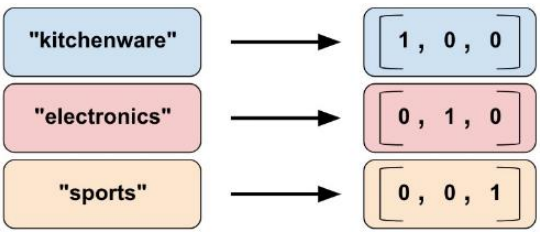


Fig. 16. Representación de columna de vocabulario categórico

Funciones: *tf.feature_column.categorical_column_with_vocabulary_list*
tf.feature_column.categorical_column_with_vocabulary_file

- ***Hashed column*** **(columna hash)**. Permite especificar el número de categorías. El modelo calcula un valor hash de la entrada y luego lo coloca en una de las categorías *hash_bucket_size* utilizando el operador de módulo.

  Función: *tf.feature_column.categorical_column_with_hash_bucket*

En este proyecto se ha usado la función de columna hash ya que, a pesar de que solo hay tres columnas de tipo categóricas, la variable "service" tiene 66 categorías diferentes.

```
>>> x_train['flag'].unique()
array(['SF', 'REJ', 'RSTR', 'RSTO', 'S1', 'S3', 'S0', 'S2', 'SH', 'OTH',
       'RSTOS0'], dtype=object)
>>> len(x_train['flag'].unique())
11
>>> x_train['protocol_type'].unique()
array(['tcp', 'udp', 'icmp'], dtype=object)
>>> len(x_train['protocol_type'].unique())
3
>>> x_train['service'].unique()
array(['http', 'smtp', 'other', 'ftp_data', 'ecr_i', 'ftp', 'domain_u',
       'private', 'ntp_u', 'auth', 'urp_i', 'IRC', 'telnet', 'eco_i',
       'finger', 'urh_i', 'time', 'X11', 'pop_3', 'login', 'imap4',
       'link', 'rje', 'name', 'mtp', 'gopher', 'remote_job', 'whois',
       'domain', 'ssh', 'discard', 'uucp_path', 'courier', 'exec',
       'sql_net', 'vmnet', 'netbios_ssn', 'nnsp', 'bgp', 'supdup',
       'http_443', 'netstat', 'uucp', 'sunrpc', 'printer', 'echo', 'ldap',
       'csnet_ns', 'klogin', 'pop_2', 'nntp', 'hostnames', 'kshell',
       'ctf', 'netbios_dgm', 'efs', 'Z39_50', 'systat', 'daytime',
       'shell', 'netbios_ns', 'iso_tsap', 'red_i', 'tim_i', 'tftp_u',
       'pm_dump'], dtype=object)
>>> len(x_train['service'].unique())
66
>>>
```

Fig. 17. Detalle y cantidad de variables categóricas

La transformación de las columnas se ha realizado mediante un bucle (Figura 17) en el que se recorren todas las variables de entrada y se comprueba el tipo al que corresponde cada una de ellas. Si es de tipo texto (categórica), la convierte mediante la función de columna hash, definiendo el número de categorías como la cantidad de cadenas de texto diferentes en la variable, y la dimensión como la raíz cuarta del valor anterior.

5.4. Selección del modelo más óptimo

La metodología seguida para seleccionar el algoritmo más óptimo, ha consistido en realizar 3 ejecuciones por cada una de las combinaciones de valores diferentes asignados a los hiperparámetros.

Los resultados de las diferentes ejecuciones se han registrado en una tabla (Tabla 12). Cada una de las filas contiene los valores asignados a los hiperparámetros más los campos “Precisión val.”, “Precisión test” y “Tiempo”, calculados como la media de 3 ejecuciones por cada combinación de valores. Para determinar cuál ha sido el modelo más óptimo, se ha fijado como objetivo que la precisión en la validación y en el test estuvieran cercanos al 100% y el tiempo total empleado por el algoritmo sea mínimo.

A continuación se muestra la tabla con los resultados de las diferentes ejecuciones:

Modelo	*Train num epochs*	*Validation num epochs*	*Batch size*	*Steps*	*Learning rate*	Precisión val.	Precisión test	Tiempo (min:sec)
Mod1	10	1	128	1.000	0,001	98,85	81,56	6:04
Mod2	10	1	128	1.000	0,01	98,83	81,47	5:55
Mod3	10	1	128	1.000	0,1	88,66	74,42	5:24
Mod4	10	1	128	10.000	0,001	98,62	82,42	5:10
Mod5	10	1	128	10.000	0,01	98,97	90,13	5:25
Mod6	10	1	128	10.000	0,1	80,35	80,52	5:40
Mod7	10	1	128	20.000	0,01	98,84	90,95	5:39
Mod8	10	1	128	20.000	0,001	98,55	85,37	5:34
Mod9	10	1	128	25.000	0,01	99,26	92,50	5:30
Mod10	10	10	128	25.000	0,01	99,06	92,15	6:29

Tabla 12. Registro de las ejecuciones

Tal y como se observa en la tabla anterior, la precisión en la fase de validación es muy buena con todas las combinaciones de valores probadas, obteniendo los peores resultados cuando se ha asignado el valor 0,1 a la tasa de aprendizaje (*learning_rate*). Esto es debido a que si la tasa de aprendizaje es alta, el entrenamiento puede no converger, incluso divergir. Los cambios de peso pueden ser tan grandes que el optimizador sobrepasa el mínimo y empeora la pérdida.

También se observa que cuando se asignan valores bajos (0,001) a *learning_rate*, la precisión en el conjunto de test no es tan buena como cuando se ha asignado 0,01. El motivo de que esto ocurra es que con tasas de aprendizaje muy bajas, el entrenamiento es más confiable pero la optimización llevará mucho más tiempo, con lo que el número de pasos introducidos debe ser muy superior al introducido en las pruebas, cosa que incrementa considerablemente el tiempo de entrenamiento y la mejora en la precisión es mínima.

Por lo tanto, lo mejores resultados de precisión obtenidos en el menor tiempo se han conseguido con el valor 0,01 asignado a la tasa de aprendizaje.

La precisión en la validación también ha aumentado cuando el número de pasos (*steps*) se ha incrementado, sin sufrir apenas un incremento de tiempo en la ejecución total del algoritmo. Al pasar de 1.000 pasos a 10.000 ha mejorado la precisión de ~80% a ~90%. El incremento de precisión cuando se ha pasado de 10.000 a 25.000 solo ha sido de ~2%. Por lo tanto, aumentar los

pasos a partir de 25.000 subirá el tiempo de ejecución y casi nada la precisión. De ahí que se haya fijado el número máximo de pasos en 25.000.

En el modelo 10 se ha aumentado a 10 el número de iteraciones a realizar sobre el conjunto de validación, pero no ha habido mejora en la precisión y ha subido un poco el tiempo de proceso. Por lo tanto se fija el número de iteraciones para el conjunto de validación a 1, tal y como estaba al inicio.

Después de las pruebas realizadas se decide que el mejor modelo es el 9, que se ha salvaguardado en el directorio "mod9**"** dentro de "scripts_clasificador". El modelo está incluido en el producto con el que se entrega esta memoria, para que se pueda reutilizar durante las posibles pruebas que se realicen en la laboratorio del producto final.

5.5. Evaluación del modelo. Matriz de confusión

El instrumento más usual para evaluar la exactitud de una clasificación es la matriz de confusión, también llamada matriz de error o de contingencia. Es una matriz cuadrada de n x n, donde n es el número de clases. Dicha matriz muestra la relación entre dos series de medidas correspondientes al área en estudio. Las filas se corresponden con las clases reales (etiqueta de la tabla de test) y las columnas, según las clases provistas por el modelo (predicciones).

La matriz de confusión sirve para mostrar de forma explícita cuando una clase es confundida con otra. Por eso, permite trabajar de forma separada con distintos tipos de error. La diagonal principal de la matriz contiene la suma de todas las predicciones correctas, es decir la exactitud (*acurracy*). La otra diagonal refleja los errores del clasificador: los falsos positivos y los falsos negativos.

La matriz de confusión obtenida con el último entrenamiento del modelo seleccionado en el apartado anterior es la siguiente:

		Predicción		
		Positivos Pred.	Negativos Pred.	
Observación	Positivos Obs.	**Verdaderos Positivos (VP) = 227.632**	**Falsos Negativos (FN) (Error Tipo II) = 22.804**	Valor Predictivo Positivo = VP / Positivos Obs. = **90,89%**
	Negativos Obs.	**Falsos Positivos (FP) (Error Tipo I) = 536**	**Verdaderos Negativos (VN) = 60.057**	Valor Predictivo Negativo = VN / Negativos Obs. = **99,12%**
		Sensibilidad = VP / Positivos Pred. = **99,77%**	**Especificidad** = VN / Negativos Pred. = **72,48%**	

Fig. 18. Matriz de confusión del modelo 9

```
***** TIEMPOS *****
Inicio del proceso: 02/06/2018 16:58:48
Tiempo empleado en la importación de datos de entrenamiento: 0:00:01.528826
Tiempo empleado en la importación de datos de validación: 0:00:00.078975
Tiempo empleado en la importación de datos de test: 0:00:01.245614
Tiempo empleado en la fase de entrenamiento: 0:04:33.822855
Tiempo empleado en la fase de validación: 0:00:07.887856
Tiempo empleado en la fase de predicción: 0:00:21.262381
***** PRECISIÓN *****
VN= 60057, FP = 536, FN = 22804, VP = 227632
Valor de predicción positivo: 90.89428037502596
Valor de predicción negativo: 99.11540937071939
Sensibilidad: 99.76508537568809
Especificidad: 72.47921217460566
Error Tipo I: 536
Error Tipo II: 22804
Precisión en la validación: 0.9926318526268005
Precisión en el test: 0.9249587655067444
Fin del proceso: 02/06/2018 17:04:19
Tiempo total empleado en la ejecución: 0:05:30.885228
***** FIN *****
[dmt88@localhost scripts_clasificador]$
```

Fig. 19. Resultados de la última ejecución de entrenamiento del modelo 9 (seleccionado)

A continuación se describen las métricas de la matriz de confusión:

- **Verdaderos Positivos (VP)**. Es la cantidad de positivos que han sido clasificados correctamente como positivos por el modelo.

- **Verdaderos Negativos (VN)**. Es la cantidad de negativos que han sido clasificados correctamente como negativos por el modelo.

- **Falso Negativos (FN)**. Es la cantidad de positivos que han sido clasificados incorrectamente como negativos.

- **Falsos Positivos (FP)**. Es la cantidad de negativos que han sido clasificados incorrectamente como positivos.

- **Exactitud (*Acurracy*)**. Es el porcentaje de datos que ha clasificado correctamente el modelo. Se calcula según:

$$Exactitud = \frac{VP + VN}{Total}$$

- **Tasa de error (*Misclassification Rate*)**. Es el porcentaje de datos que ha clasificado incorrectamente el modelo. Se calcula según:

$$\text{Tasa de error} = \frac{FP + FN}{Total}$$

- **Tasa de verdaderos positivos (*True Positive Rate*)**. Es el porcentaje de positivos que el modelo ha clasificado correctamente. Se calcula según:

$$\text{Sensibilidad} = \frac{VP}{\text{Total Positivos}}$$

- **Tasa de verdaderos negativos (*True Negative Rate*)**. Es el porcentaje de negativos que el modelo ha clasificado correctamente. Se calcula según:

$$\text{Especificidad} = \frac{VN}{\text{Total Negativos}}$$

- **Valor de predicción negativo (*Precision*)**. Es el porcentaje que ha clasificado correctamente durante la clasificación de positivos. Se calcula según:

$$\text{Precisión} = \frac{VP}{\text{Total clasificados positivos}}$$

- **Valor de predicción negativo**. Es el porcentaje que ha clasificado correctamente durante la clasificación de negativos. Se calcula según:

$$\text{VPN} = \frac{VN}{\text{Total clasificados negativos}}$$

5.6. Reutilización del modelo

Una vez obtenido el modelo más óptimo, ha llegado el momento de utilizarlo para realizar predicciones de tramas de conexiones nuevas (sin etiquetar). Para esta fase se ha utilizado el conjunto de datos del fichero **kddcup.newtestdata_10_percent_unlabeled** que, tal y como se indicaba en el apartado 2.2, viene sin el campo de etiqueta.

En primer lugar, se ha creado la tabla “predicted_data” en la BD Cassandra, con la estructura del fichero anterior más los campos “uuid” y “label_class”, el primero para añadir el identificador único de cada trama y el segundo para guardar el resultado de la predicción: “ATTACK” o “NORMAL”. El script que crea la tabla es “predicted_data_create.cql” (Anexo 5).

En Tensorflow, para reutilizar un modelo clasificador que ha sido salvado previamente, hay que pasar la ruta del punto de control (*checkpoint_path*) donde se encuentra el modelo al comando *predict*. En el caso de que no se introduzca ningún punto de control, coge el valor del parámetro “model_dir” de la definición del modelo. Dicha ruta es la que está definida en el algoritmo “intrusion_pred_model.py” (Figura 20).

```
# Definición del modelo clasificador de red neural profunda
modelo = tf.estimator.DNNClassifier(
        hidden_units=[n_hidden_1, n_hidden_2],
    feature_columns=feature_columns,
    model_dir='./mod9',
    n_classes=2,
    label_vocabulary=["NORMAL", "ATTACK"],
    optimizer=tf.train.ProximalAdagradOptimizer(
        learning_rate=learning_rate,
        l1_regularization_strength=0.001))
```

Fig. 20. Definición del directorio donde se guarda el modelo

Los datos del fichero **data_to_predict.csv** no se tratan directamente desde el modelo clasificador, sino que sufren un tratamiento previo. En primer lugar se procesa mediante el script "add_uuid_pred.py" (Anexo 5), para añadir un identificador único al final de cada registro (uuid) y un campo con el texto "NULL" a continuación. Después se crea la tabla "data_to_predict" en la BD, se importa el fichero a la misma y se exporta a CSV de nuevo (**data_to_predict_out.csv**). Este procedimiento se hace para que la importación al modelo clasificador con Pandas sea más sencilla.

A continuación se ejecuta el script de Python **classify_connections.py**, que utilizando el modelo 9, clasifica las conexiones nuevas. El clasificador exporta los resultados al fichero **predicted_data_csv**.

El último paso es la importación del fichero con los resultados a la tabla predicted_data. Esto lo hace el script cql **predicted_data_table_import.cql**.

La Figura 21 muestra el contenido de la tabla predicted_data con las nuevas conexiones clasificadas.

```
cqlsh:intrusion_detector> select label_class, count(*) from predicted_data group by label_class;

 label_class | count
-------------+--------
      ATTACK | 228203
      NORMAL |  82876

(2 rows)

Warnings :
Aggregation query used without partition key
```

Fig. 21. Contenido de la tabla predicted_data con las tramas agrupadas por label_class

El tiempo medio en realizar las tres fases mencionadas no supera los 3 minutos con el entorno de trabajo en el que se ha realizado todo el proyecto (ver Anexo 7).

6. Conclusiones

A lo largo de este proyecto se han ido extrayendo una serie de conclusiones que vienen a representar el fruto del trabajo realizado, y que han tenido que partir del análisis de las necesidades a cubrir para la empresa Ancert. Se puede concluir que:

- ✓ No hay una solución única para un mismo proyecto.
- ✓ De todas las alternativas posibles para resolver un problema hay que escoger aquella en la que la persona que vaya a llevarla a cabo se sienta más cómoda.
- ✓ Es importante invertir más tiempo en investigar el cómo hacerlo que el hacerlo directamente.
- ✓ Lo sistemas de BD NoSQL son ideales para digerir enormes cantidades de datos, pero no son buenos para realizar consultas de agrupaciones de datos.
- ✓ El resultado del entrenamiento de un algoritmo predictivo será mejor si la calidad de los datos también lo es. Esto quiere decir que los datos sean una muestra fiel de la realidad que representan.
- ✓ La cantidad de datos que se use para entrenar el modelo también es importante, pero se pueden conseguir porcentajes de acierto excelentes con volúmenes intermedios, que reducen considerablemente el tiempo de entrenamiento.

Se ha alcanzado el objetivo principal del proyecto, que era crear un clasificador predictivo eficiente, capaz de distinguir conexiones de red fraudulentas de conexiones lícitas. Todo ello, con un impacto ínfimo en cuanto al repudio de usuarios lícitos (<0,30% de falsos positivos). En tan solo 5’ 30” es capaz de realizar un entrenamiento completo, con el que se obtiene un modelo muy preciso.

El modelo creado clasifica ~300.000 tramas de red, en menos de 3 minutos (~1.666/s), incluidos los procesos previos de preparado de datos de entrada y la inserción en la BD de los resultados de las predicciones.

Se ha seguido el plan de trabajo establecido al inicio del proyecto, cumpliendo con todas las fechas de los hitos.

Solo ha habido un cambio en el proyecto respecto a la planificación inicial. Los datos con los que se ha entrenado el algoritmo no han sido los del fichero que se había establecido, sino que se han realizado con el fichero que es ~10% del mismo. Este cambio se ha realizado durante la creación del modelo porque se ha observado que con el fichero pequeño se conseguían los mismos resultados que con el grande, reduciendo el tiempo de proceso unas 10 veces.

Trabajos futuros deberían comprender el diseño e implementación de una API, con métodos para entrenar el algoritmo, modificar hiperparámetros, realizar predicciones de diferentes ficheros, etc.

7. Glosario

DDos (*Distributed Denial of Service*). Es un ataque a un sistema de computadoras o red que causa que un servicio o recurso sea inaccesible a los usuarios legítimos.

Deep Learning. Aprendizaje profundo. Es un conjunto de algoritmos de clase aprendizaje automático, que intenta modelar las abstracciones de alto nivel en datos, usando arquitecturas compuestas de transformaciones no lineales múltiples.

Desarrollador *Full Stack*. Persona técnica que conoce bien todos los aspectos del *front-end*, *back-end*, los diferentes sistemas operativos y componentes que quedan en el medio.

Hinted Handof. Técnica de Cassandra que sirve para compensar el clúster en el momento que un nodo falla. Los vecinos del nodo fallido tomarán relevo y realizarán el trabajo de ese nodo permitiendo al clúster trabajar de forma normal. Esto puede considerarse una forma de auto-curación.

IDE (*Integrated Development Environment*). Entorno de desarrollo integrado. Es una aplicación informática que proporciona servicios integrales para facilitarle al desarrollador o programador el desarrollo de software.

***Machine Learning*.** Aprendizaje automático. Es el subcampo de las ciencias de la computación y una rama de la inteligencia artificial cuyo objetivo es desarrollar técnicas que permitan a las computadoras aprender.

NoSQL. Es una amplia clase de sistemas de gestión de bases de datos que difieren del modelo clásico de SGBDR (Sistema de Gestión de Bases de Datos Relacionales) en aspectos importantes, siendo el más destacado que no usan SQL como lenguaje principal de consultas.

RDBMS (*Relational Database Management System*). Sistema de base de datos relacional. Es una colección de elementos de datos organizados en un conjunto de tablas formalmente descritas, desde la que se puede acceder a los datos o volver a montarlos de muchas maneras diferentes sin tener que reorganizar las tablas base.

Read Repair. Operación de Cassandra que puede utilizarse por parte del usuario para asegurar que todos los nodos son consistentes, es decir, que no hay réplicas con diferentes valores.

SQL (*Structured Query Language*). Lenguaje de consultas estructuradas. Es un lenguaje específico del domino que da acceso a un

sistema de gestión de bases de datos relacionales, que permite especificar diversos tipos de operaciones en ellos.

VPN (*Vritual Private Network*). Red privada virtual. Es una tecnología de red de computadoras que permite una extensión segura de red de área local (LAN) sobre una red pública o no controlada como Internet.

WAN (*Wide Area Network*). Red de área amplia, es una red de computadoras que une varias redes locales, aunque sus miembros no estén en una misma ubicación física.

8. Bibliografía

http://www.ancert.com/liferay/web/ancert/sobre-nosotros [26 de Febrero de 2018].

http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html [26 de Febrero de 2018].

https://es.wikipedia.org [27 de Febrero de 2018].

https://blogthinkbig.com/artificial-intelligence-machine-learning-y-deep-learning-conoces-las-diferencias [1 de Marzo de 2018].

http://www.aprendemachinelearning.com/aprendizaje-profundo-una-guia-rapida/ [4 de Marzo de 2018].

http://www.aprendemachinelearning.com/7-pasos-machine-learning-construir-maquina/ [5 de Marzo de 2018]

http://slashmobility.com/blog/2016/12/nosql-vs-sql-cual-elegir/ [19 de Marzo de 2018]

http://latamdigital.softtek.co/9-aspectos-que-debe-conocer-un-full-stack-developer [19 de Marzo de 2018]

https://blog.pandorafms.org/es/bases-de-datos-nosql/ [22 de Marzo de 2018]

https://www.acens.com/wp-content/images/2014/02/bbdd-nosql-wp-acens.pdf [23 de Marzo de 2018]

https://www.solvetic.com/tutoriales/article/4787-como-instalar-base-de-datos-cassandra-centos-7/ [24 de Marzo de 2018]

http://exabig.com/blog/2018/02/28/setup-cassandra-cluster-centos-7/ [24 de Marzo de 2018]

https://docs.datastax.com [25 de Marzo de 2018]

https://blogthinkbig.com/artificial-intelligence-machine-learning-y-deep-learning-conoces-las-diferencias [11 de Abril de 2018]

https://searchdatacenter.techtarget.com/es/definicion/Inteligencia-artificial-o-AI [13 de Abril de 2018]

http://www.fgcsic.es/lychnos/es_es/articulos/inteligencia_artificial [14 de Abril de 2018]

http://cleverdata.io/que-es-machine-learning-big-data/ [18 de Abril de 2018]

http://www.raona.com/machine-learning-tipos-machine-learning/ [21 de Abril de 2018]

https://blog.adext.com/es/machine-learning-guia-completa [21 de Abril de 2018]

http://www.aprendemachinelearning.com/principales-algoritmos-usados-en-machine-learning/ [23 de Abril de 2018]

https://www.indracompany.com/es/blogneo/deep-learning-sirve [25 de Abril de 2018]

https://planetachatbot.com/deep-learning-f%C3%A1cil-con-deepcognition-9af43b2319ba [26 de Abril de 2018]

http://noticias.universia.es/ciencia-tecnologia/noticia/2017/07/19/1154393/sirve-phyton.html [27 de Abril de 2018]

http://synergy.vision/corpus/R/Intro-R/ [27 de Abril de 2018]

http://softlibre.unizar.es/manuales/aplicaciones/octave/octave.pdf [27 de Abril de 2018]

ftp://ftp.unicauca.edu.co/Facultades/FIET/DEIC/Materias/Identificacion/matlab_seminar/docs/Matlab6xConatec.pdf [27 de Abril de 2018]

https://medium.com/@UdacityINDIA/machine-learning-programming-languages-why-is-the-best-and-why-56f9f370cb99 [27 de Abril de 2018]

http://makemeanalyst.com/5-popular-python-libraries-machine-learning-deep-learning-2017/ [28 de Abril de 2018]

https://thenewstack.io/three-ways-google-tensorflow-beats-caffe/ [28 de Abril de 2018]

https://www.tensorflow.org [29 de Abril de 2018]

https://towardsdatascience.com/estimating-optimal-learning-rate-for-a-deep-neural-network-ce32f2556ce0 [7 de Mayo de 2018]

https://www.toptal.com/machine-learning/un-tutorial-de-aprendizaje-profundo-de-perceptrones-a-redes-profundas/es [9 de Mayo de 2018]

http://www.teledet.com.uy/tutorial-imagenes-satelitales/clasificacion-matriz-confusion.htm [16 de Mayo de 2018]

https://rpubs.com/chzelada/275494 [18 de Mayo de 2018]

9. Anexos

Anexo 1. Scripts previos y de importación de datos a la BD Cassandra

Los scripts siguientes se encuentran dentro de los directorios "**scripts_cassandra**" y "**script_cassandra/scripts_cqlsh**", contenidos éstos a su vez en el interior del directorio "**producto**", que se entrega junto con esta memoria.

load_data.sh

- Descomprime los ficheros de datos descargados de la página de KDD CUP.
- Llama al script **add_uuid.py** con los nombres de los ficheros de entrada y salida como argumentos del script.
- Ejecuta el script **cassandra.sh**.

```
#!/bin/bash

DIR='/usr/share/cassandra/scripts/'

# Descomprime los ficheros de datos
gunzip ${DIR}*.gz

# Añade un UUID al final de cada registro de los ficheros
python ${DIR}add_uuid.py ${DIR}kddcup.data ${DIR}train_data.csv
python ${DIR}add_uuid.py ${DIR}kddcup.data_10_percent ${DIR}validation_data.csv
python ${DIR}add_uuid.py ${DIR}corrected ${DIR}test_data.csv

# Crea el keyspace, las tablas e importa los datos de los ficheros a las tablas
${DIR}scripts_cqlsh/cassandra.sh
```

add_uuid.py

- Abre en modo lectura el fichero introducido como primer argumento.
- Abre en modo escritura el fichero introducido como segundo argumento.
- Añade un campo con el texto "NORMAL", si el último campo del fichero original tiene el texto "normal.", o con el texto "ATTACK" si contiene cualquier otro texto.
- Añade un campo al final de cada registro con un UUID (*Universally Unique IDentifier*).

```
import csv
import uuid
import sys

filein=str(sys.argv[1])
fileout=str(sys.argv[2])

fin = open(filein, 'r')
fout = open(fileout, 'w')

reader = csv.reader(fin, delimiter=',')
writer = csv.writer(fout, delimiter=',')

for row in reader:
    if (row[41] == "normal."):
        row.append("NORMAL")
    else:
        row.append("ATTACK")
    row.append(uuid.uuid4())
    writer.writerow(row)
fout.close()
fin.close()
```

cassandra.sh

- Ejecuta los scripts de borrado y creación del keyspace y las tablas e importación de los datos de los ficheros a las tablas.

```
#!/bin/bash

DIR='/usr/share/cassandra/scripts/scripts_cqlsh/'

# Borra el keyspace intrusion_detector si existe
cqlsh -f ${DIR}keyspace_drop.cql

# Crea el keyspace intrusion_detector
cqlsh -f ${DIR}keyspace_create.cql

# Crea la tabla train_data
cqlsh -f ${DIR}train_data_table_create.cql

# Crea la tabla validation_data
cqlsh -f ${DIR}validation_data_table_create.cql

# Crea la tabla test_data
cqlsh -f ${DIR}test_data_table_create.cql

# Importa los datos del fichero kddcup.data a la tabla train_data
cqlsh -f ${DIR}train_data_table_import.cql

# Importa los datos del fichero kddcup.data_10_percent a la tabla validation_data
cqlsh -f ${DIR}validation_data_table_import.cql

# Importa los datos del fichero corrected a la tabla test_data
cqlsh -f ${DIR}test_data_table_import.cql
```

keyspace_drop.cql

- Elimina el keyspace **intrusion_detector** si existe.

```
DROP KEYSPACE IF EXISTS intrusion_detector;
```

keyspace_create.cql

- Crea el keyspace **intrusion_detector** si no existe. Lo crea en un solo nodo y con factor de replicación 3.

```
CREATE KEYSPACE IF NOT EXISTS intrusion_detector
  WITH REPLICATION = {
      'class' : 'SimpleStrategy', 'replication_factor' : 3 };
```

train_data_create.cql

- Crea la tabla **train_data** en el keyspace de **intrusión_detector** con clave primaria compuesta por **label_class**, **label** e **id**, la primera como clave de partición.

```
CREATE TABLE IF NOT EXISTS intrusion_detector.train_data (
    id UUID,
    duration int,
    protocol_type text,
    service text,
    flag text,
    src_bytes int,
    dst_bytes int,
    land text,
    wrong_fragment int,
    urgent int,
    hot int,
    num_failed_logins int,
    logged_in text,
    num_compromised int,
    root_shell int,
    su_attempted int,
    num_root int,
    num_file_creations int,
    num_shells int,
    num_access_files int,
    num_outbound_cmds int,
    is host login text,
```

```
    is_guest_login text,
    count int,
    srv_count int,
    serror_rate float,
    srv_serror_rate float,
    rerror_rate float,
    srv_rerror_rate float,
    same_srv_rate float,
    diff_srv_rate float,
    srv_diff_host_rate float,
    dst_host_count int,
    dst_host_srv_count int,
    dst_host_same_srv_rate float,
    dst_host_diff_srv_rate float,
    dst_host_same_src_port_rate float,
    dst_host_srv_diff_host_rate float,
    dst_host_serror_rate float,
    dst_host_srv_serror_rate float,
    dst_host_rerror_rate float,
    dst_host_srv_rerror_rate float,
    label text,
    label_class text,
    PRIMARY KEY (label_class,label,id));
```

validation_data_create.cql

- Crea la tabla **validation_data** en el keyspace de **intrusión_detector** con clave primaria compuesta de **label_class**, **label** e **id**, la primera como clave de partición.

```
CREATE TABLE IF NOT EXISTS intrusion_detector.validation_data (
    id UUID,
    duration int,
    protocol_type text,
    service text,
    flag text,
    src_bytes int,
    dst_bytes int,
    land text,
    wrong_fragment int,
    urgent int,
    hot int,
    num_failed_logins int,
    logged_in text,
    num_compromised int,
    root_shell int,
    su_attempted int,
    num_root int,
    num_file_creations int,
    num_shells int,
    num_access_files int,
    num_outbound_cmds int,
    is_host_login text,
```

```
    is_guest_login text,
    count int,
    srv_count int,
    serror_rate float,
    srv_serror_rate float,
    rerror_rate float,
    srv_rerror_rate float,
    same_srv_rate float,
    diff_srv_rate float,
    srv_diff_host_rate float,
    dst_host_count int,
    dst_host_srv_count int,
    dst_host_same_srv_rate float,
    dst_host_diff_srv_rate float,
    dst_host_same_src_port_rate float,
    dst_host_srv_diff_host_rate float,
    dst_host_serror_rate float,
    dst_host_srv_serror_rate float,
    dst_host_rerror_rate float,
    dst_host_srv_rerror_rate float,
    label text,
    label_class text,
    PRIMARY KEY (label_class,label,id));
```

test_data_create.cql

- Crea la tabla **test_data** en el keyspace de **intrusión_detector** con clave primaria compuesta de **label_class**, **label** e **id**, la primera como clave de partición.

```
CREATE TABLE IF NOT EXISTS intrusion_detector.test_data (
    id UUID,
    duration int,
    protocol_type text,
    service text,
    flag text,
    src_bytes int,
    dst_bytes int,
    land text,
    wrong_fragment int,
    urgent int,
    hot int,
    num_failed_logins int,
    logged_in text,
    num_compromised int,
    root_shell int,
    su_attempted int,
    num_root int,
    num_file_creations int,
    num_shells int,
    num_access_files int,
    num_outbound_cmds int,
    is_host_login text,
```

```
    is_guest_login text,
    count int,
    srv_count int,
    serror_rate float,
    srv_serror_rate float,
    rerror_rate float,
    srv_rerror_rate float,
    same_srv_rate float,
    diff_srv_rate float,
    srv_diff_host_rate float,
    dst_host_count int,
    dst_host_srv_count int,
    dst_host_same_srv_rate float,
    dst_host_diff_srv_rate float,
    dst_host_same_src_port_rate float,
    dst_host_srv_diff_host_rate float,
    dst_host_serror_rate float,
    dst_host_srv_serror_rate float,
    dst_host_rerror_rate float,
    dst_host_srv_rerror_rate float,
    label text,
    label_class text,
    PRIMARY KEY (label_class,label,id));
```

train_data_table_import.cql

- Importa los datos del fichero **train_data.csv** a la tabla **train_data**.

```
COPY intrusion_detector.train_data (
     duration, protocol_type, service, flag, src_bytes, dst_bytes, land, wrong_fragment, urgent,
     hot, num_failed_logins, logged_in, num_compromised, root_shell, su_attempted, num_root,
     num_file_creations, num_shells, num_access_files, num_outbound_cmds, is_host_login, is_guest_login,
     count, srv_count, serror_rate, srv_serror_rate, rerror_rate, srv_rerror_rate, same_srv_rate,
     diff_srv_rate, srv_diff_host_rate, dst_host_count, dst_host_srv_count, dst_host_same_srv_rate,
     dst_host_diff_srv_rate, dst_host_same_src_port_rate, dst_host_srv_diff_host_rate, dst_host_serror_rate,
     dst_host_srv_serror_rate, dst_host_rerror_rate, dst_host_srv_rerror_rate, label, label_class, id)
FROM '/usr/share/cassandra/scripts/train_data.csv' WITH DELIMITER=',';
```

validation_data_table_import.cql

- Importa los datos del fichero **validation_data.csv** a la tabla **validation_data**.

```
COPY intrusion_detector.validation_data (
     duration, protocol_type, service, flag, src_bytes, dst_bytes, land, wrong_fragment, urgent,
     hot, num_failed_logins, logged_in, num_compromised, root_shell, su_attempted, num_root,
     num_file_creations, num_shells, num_access_files, num_outbound_cmds, is_host_login, is_guest_login,
     count, srv_count, serror_rate, srv_serror_rate, rerror_rate, srv_rerror_rate, same_srv_rate,
     diff_srv_rate, srv_diff_host_rate, dst_host_count, dst_host_srv_count, dst_host_same_srv_rate,
     dst_host_diff_srv_rate, dst_host_same_src_port_rate, dst_host_srv_diff_host_rate, dst_host_serror_rate,
     dst_host_srv_serror_rate, dst_host_rerror_rate, dst_host_srv_rerror_rate, label, label_class, id)
FROM '/usr/share/cassandra/scripts/validation_data.csv' WITH DELIMITER=',';
```

test_data_table_import.cql

- Importa los datos del fichero **test_data.csv** a la tabla **test_data**.

```
COPY intrusion_detector.test_data (
     duration, protocol_type, service, flag, src_bytes, dst_bytes, land, wrong_fragment, urgent,
     hot, num_failed_logins, logged_in, num_compromised, root_shell, su_attempted, num_root,
     num_file_creations, num_shells, num_access_files, num_outbound_cmds, is_host_login, is_guest_login,
     count, srv_count, serror_rate, srv_serror_rate, rerror_rate, srv_rerror_rate, same_srv_rate,
     diff_srv_rate, srv_diff_host_rate, dst_host_count, dst_host_srv_count, dst_host_same_srv_rate,
     dst_host_diff_srv_rate, dst_host_same_src_port_rate, dst_host_srv_diff_host_rate, dst_host_serror_rate,
     dst_host_srv_serror_rate, dst_host_rerror_rate, dst_host_srv_rerror_rate, label, label_class, id)
FROM '/usr/share/cassandra/scripts/test_data.csv' WITH DELIMITER=',';
```

Anexo 2. Scripts de exportación de datos a ficheros CSV desde la BD Cassandra

Los scripts siguientes se encuentran dentro de los directorios **scripts_cassandra** y **script_cassandra/scripts_cqlsh**, contenidos éstos a su vez en el interior del directorio **producto**, que se entrega junto con esta memoria.

tables_data_export.sh

- Ejecuta los scripts de exportación de datos a ficheros CSV de las tablas **validation_data** y **test_data**.

```
#!/bin/bash

DIR='/usr/share/cassandra/scripts/scripts_cqlsh/'

# Exporta los datos de la tabla validation_data al fichero validation_data_out.csv
cqlsh -f ${DIR}validation_data_table_export.cql

# Exporta los datos de la tabla test_data al fichero test_data_out.csv
cqlsh -f ${DIR}test_data_table_export.cql
```

validation_data_table_export.cql

- Exporta los datos de la tabla **validation_data** del keyspace **intrusión_detector** al **fichero validation_data_out.csv**.

```
COPY intrusion_detector.validation_data
TO '/usr/share/cassandra/scripts/validation_data_out.csv'
WITH HEADER = true;
```

test_data_table_export.cql

- Exporta los datos de la tabla **test_data** del keyspace **intrusión_detector** al **fichero test_data_out.csv**.

```
COPY intrusion_detector.test_data
TO '/usr/share/cassandra/scripts/test_data_out.csv'
WITH HEADER = true;
```

Anexo 3. Script de creación del modelo predictivo clasificador

El script siguiente se encuentra dentro del directorio "**scripts_cqlsh**", contenido éste a su vez en el interior del directorio "**producto/scripts_cassandra**", que se entrega junto con esta memoria.

tables_data_export.sh

- Exporta el contenido de las tablas **validation_data** y **test_data** desde la BD a los ficheros **validation_data_out.csv** y **test_data_out.csv**.

```
#!/bin/bash

DIR='/usr/share/cassandra/scripts/scripts_cqlsh/'

# Exporta los datos de la tabla validation_data al fichero validation_data_out.csv
cqlsh -f ${DIR}validation_data_table_export.cql

# Exporta los datos de la tabla test_data al fichero test_data_out.csv
cqlsh -f ${DIR}test_data_table_export.cql
```

Anexo 4. Script de creación del modelo predictivo clasificador

El script siguiente se encuentra dentro del directorio **scripts_clasificador**, contenido éste a su vez en el interior del directorio **producto/scripts_cassandra**, que se entrega junto con esta memoria.

intrusión_pred_model.sh

- Importación de las librerías necesarias.

```
# Importación de librerias
import pandas as pd
import numpy as np
import datetime
import logging
logging.getLogger('tensorflow').disabled = True
import tensorflow as tf
import pandas.api.types as ptypes
from sklearn.metrics import confusion_matrix
```

- Importación de los datos de las tablas **validation_data** y **test_data** desde los ficheros **validation_data_out.csv** y **test_data_out.csv** generados por el script **tables_data_export.sh**.

```
# Control de tiempo
start_time = datetime.datetime.now()
print("Inicio del proceso: " + start_time.strftime("%d/%m/%Y %H:%M:%S"))

print("***** INICIO DE LA FASE DE IMPORTACIÓN DE DATOS *****")
# Importación de los datos para el entrenamiento
print("Importando los datos para la fase de entrenamiento...")
df_train = pd.read_csv('../validation_data_out.csv')
print("Preparando dataframe con las columnas de características...")
x_train = df_train.iloc[ : , 3:]
print("Preparando dataframe con la columna de etiquetas...")
y_train = df_train.iloc[ : , 0:1]
print("Filas:" + str(len(x_train)))
print("Columnas:" + str(len(x_train.columns)))
# Control de tiempo. Importacion del fichero de datos de entrenamiento
train_data_import_time = datetime.datetime.now()
```

```
# Importación de los datos para el test
print("Importando los datos para la fase de test...")
print("Preparando dataframe con las columnas de características...")
df_test = pd.read_csv('../test_data_out.csv')
x_test = df_test.iloc[ : , 3:]
print("Preparando dataframe con la columna de etiquetas...")
y_test = df_test.iloc[ : , 0:1]
print("Filas:" + str(len(x_test)))
print("Columnas:" + str(len(x_test.columns)))
# Control de tiempo. Importacion del fichero de datos de test
test_data_import_time = datetime.datetime.now()
print("***** FIN DE LA FASE DE IMPORTACIÓN DE DATOS *****")
```

- Creación del subconjunto del 20% de **validation_data** para utilizar en la primera validación.

```
# Importación de los datos para la fase de validación
print("Importando los datos para la fase de validación...")
df_validation = df_train.sample(frac=0.2)
print("Preparando dataframe con las columnas de características...")
x_validation = df_validation.iloc[ : , 3:]
print("Preparando dataframe con la columna de etiquetas...")
y_validation = df_validation.iloc[ : , 0:1]
print("Filas:" + str(len(x_validation)))
print("Columnas:" + str(len(x_validation.columns)))
# Control de tiempo. Importacion del fichero de datos de validacion
validation_data_import_time = datetime.datetime.now()
```

- Definición de hiperparámetros.

```
print("***** INICIO DE LA FASE DE DEFINICIÓN DE PARÁMETROS DE AJUSTE Y MODELO *****")
# Hiperparámetros
train_num_epochs=10 # Ajustar en orden ascendente
validation_num_epochs=1 # Ajustar en orden ascendente
steps=25000 # Ajustar en orden ascendente.
# max(steps) = (len(x_train) / batch_size(default:128)) * train_num_epochs
learning_rate = 0.01 # Entre 0 y 1. Ajustar en orden descendente
# Cantidad de neuronas en las capas ocultas calculadas según:
# (num. características de entrada + num. clases de salida) * (2/3)
# (41+2)*(2/3) = ~28. Redondeado a potencia de 2 => 32
n_hidden_1 = 32 # Número de neuronas de la primera capa oculta
n_hidden_2 = 32 # Número de neuronas de la segunda capa oculta
```

- Definición de las funciones para alimentar al modelo en el entrenamiento, validación y test.

```
# Función de entrada para alimentar el modelo en la fase de entrenamiento
train_input_fn = tf.estimator.inputs.pandas_input_fn(
    x=x_train,
    y=y_train,
    num_epochs=train_num_epochs,
    shuffle=True)

# Función de entrada para alimentar el modelo en la fase de validación
validation_input_fn = tf.estimator.inputs.pandas_input_fn(
    x=x_validation,
    y=y_validation,
    num_epochs=validation_num_epochs,
    shuffle=False)

# Función de entrada para alimentar el modelo en la fase de predicción
test_input_fn = tf.estimator.inputs.pandas_input_fn(
    x=x_test,
    y=y_test,
    shuffle=False)
```

- Transformación de las variables de características categorícas y numéricas a tensor.

```
# Transformación de las columnas de características
feature_columns = []
for col in x_train.columns:
    if ptypes.is_string_dtype(x_train[col]): # Para las columnas con valores categóricos
        # hash_bucket_size = número de categorías diferentes en la columna
        # dimension = raíz cuarta del número de categorías diferentes en la columna
        feature_columns.append(tf.feature_column.embedding_column(
                tf.feature_column.categorical_column_with_hash_bucket(
                    col, hash_bucket_size=len(x_train[col].unique())),
                        dimension=round(len(x_train[col].unique())**0.25)))

    elif ptypes.is_numeric_dtype(x_train[col]): # Para las columnas con valores numéricos
        feature_columns.append(tf.feature_column.numeric_column(col))
```

- Definición del modelo.

```
# Definición del modelo clasificador de red neural profunda
modelo = tf.estimator.DNNClassifier(
        hidden_units=[n_hidden_1, n_hidden_2],
    feature_columns=feature_columns,
    model_dir='./mod9',
    n_classes=2,
    label_vocabulary=["NORMAL", "ATTACK"],
    optimizer=tf.train.ProximalAdagradOptimizer(
        learning_rate=learning_rate,
        l1_regularization_strength=0.001))
print("***** FIN DE LA FASE DE DEFINICIÓN DE PARÁMETROS DE AJUSTE Y MODELO *****")
```

- Ejecución de las fases de entrenamiento, primera validación con los datos de validación, segunda validación con los datos de test y predicción con los datos de test.

```
# Fase de entrenamiento
print("***** INICIO DE LA FASE DE ENTRENAMIENTO *****")
modelo.train(input_fn=train_input_fn);
print("***** FIN DE LA FASE DE ENTRENAMIENTO *****")
# Control de tiempo de la fase de entrenamiento
train_time = datetime.datetime.now()

# Fase de validación con los datos de validación
print("***** INICIO DE LA FASE DE VALIDACIÓN *****")
validation_eval_result = modelo.evaluate(input_fn=validation_input_fn)
print("***** FIN DE LA FASE DE VALIDACIÓN *****")
# Control de tiempo de la fase de validación
validation_time = datetime.datetime.now()

# Fase de validación predicción con los datos de test
print("***** INICIO DE LA FASE DE TEST *****")
test_eval_result = modelo.evaluate(input_fn=test_input_fn)
predict_results = modelo.predict(input_fn=test_input_fn)
print("***** FIN DE LA FASE DE TEST *****")
# Control de tiempo de la fase de predicción
test_time = datetime.datetime.now()
```

- Cálculo e impresión de tiempos parciales.

```
# Tiempos
print("***** TIEMPOS *****")
# Inicio del proceso
print("Inicio del proceso: " + start_time.strftime("%d/%m/%Y %H:%M:%S"))
# Tiempo empleado en la importación de datos de entrenamiento
print("Tiempo empleado en la importación de datos de entrenamiento: " +
        str(train_data_import_time - start_time))
# Tiempo empleado en la importación de datos de validation
print("Tiempo empleado en la importación de datos de validación: " +
        str(validation_data_import_time - train_data_import_time))
# Tiempo empleado en la importación de datos de test
print("Tiempo empleado en la importación de datos de test: " +
        str(test_data_import_time - validation_data_import_time))
# Tiempo empleado en la fase de entrenamiento
print("Tiempo empleado en la fase de entrenamiento: " +
        str(train_time - test_data_import_time))
# Tiempo empleado en la fase de validación
print("Tiempo empleado en la fase de validación: " +
        str(validation_time - train_time))
# Tiempo empleado en la fase de predicción
print("Tiempo empleado en la fase de predicción: " +
        str(test_time-validation_time))
```

- Cálculo e impresión de precisión y métricas de la matriz de confusión.

```
# Resultados de precisión del algoritmo
print("***** PRECISIÓN *****")
test_labels = y_test
test_labels.replace({'NORMAL': 0, 'ATTACK': 1}, inplace=True)
test_labels = np.asarray(test_labels, np.float32)
prediction_ids = [prediction['class_ids'][0] for prediction in predict_results]
predict = pd.DataFrame({"label_class": prediction_ids})
predicted_labels = np.asarray(predict, np.float32)
vn, fp, fn, vp = confusion_matrix(test_labels, predicted_labels).ravel()
print('VN= ' + str(vn) + ', FP = ' + str(fp) + ', FN = ' + str(fn) + ', VP = ' + str(vp))
vpp = (vp / (vp + fn)) * 100
vnp = (vn / (fp + vn)) * 100
sen = (vp / (vp + fp)) * 100
esp = (vn / (fn + vn)) * 100
print("Valor de predicción positivo: " + str(vpp))
print("Valor de predicción negativo: " + str(vnp))
print("Sensibilidad: " + str(sen))
print("Especificidad: " + str(esp))
print("Error Tipo I: " + str(fp))
print("Error Tipo II: " + str(fn))
print ("Precisión en la validación: {accuracy}".format(**validation_eval_result))
print ("Precisión en el test: {accuracy}".format(**test_eval_result))
```

- Cálculo e impresión de tiempos totales.

```
# Control de tiempo.
finish_time = datetime.datetime.now()
print("Fin del proceso: " + finish_time.strftime("%d/%m/%Y %H:%M:%S")) # Fin del proceso
total_time = finish_time - start_time
print("Tiempo total empleado en la ejecución: " + str(total_time)) # Tiempo total empleado
print("***** FIN *****")
```

Anexo 5. Scripts para clasificación de tramas de red nuevas utilizando el modelo predictivo clasificador creado

Los scripts siguientes se encuentran dentro de los directorios **scripts_clasificador** y **scripts_cqlsh**, contenidos éstos a su vez en el interior del directorio **producto/scripts_cassandra**, que se entrega junto con esta memoria.

classify_connections.sh

- Ejecuta el script **load_data_pred.sh**.
- Ejecuta el script de Python **classify_connections.py**.
- Ejecuta el script cql **predicted_data_table_import_cql**.

```
#!/bin/bash

# Control de tiempo
inicio=`date +%s`

# Directorios
DIR='/usr/share/cassandra/scripts/'
DIR_DB='/usr/share/cassandra/scripts/scripts_cqlsh/'
DIR_PRED='/usr/share/cassandra/scripts/scripts_clasificador/'

# Crea las tablas data_to_predict y predicted_data en DB Cassandra
# Añade los campos UUID y label_class al fichero data_to_predict.csv
# Importa los datos del fichero data_to_predict.csv a la tabla data_to_predict
# Exporta los datos de la tabla data_to_predict al fichero data_to_predict_out.csv
echo "Preparando los datos a clasificar..."
${DIR}load_data_pred.sh

# Ejecuta el clasificador con el fichero data_to_predict_out
sudo python3.6 ${DIR_PRED}classify_connections.py

echo "Insertando las tramas clasificadas en la BD..."
# Inserta los datos del fichero predicted_data.csv en la tabla predicted_data
cqlsh -f ${DIR_DB}predicted_data_table_import.cql

# Control de tiempo
fin=`date +%s`
let total=$fin-$inicio
echo "Tiempo total: -$total- segundos"
```

load_data_pred.sh

- Ejecuta el script de Pyhton **add_uuid_pred.py**.
- Ejecuta el script **cassandra_pred.sh**.

```
#!/bin/bash

DIR='/usr/share/cassandra/scripts/'

# Añade un UUID al final de cada registro de los ficheros y el campo NULO
python ${DIR}add_uuid_pred.py ${DIR}kddcup.newtestdata_10_percent_unlabeled ${DIR}data_to_predict.csv

# Crea las tablas data_to_predict y predicted_data
${DIR}scripts_cqlsh/cassandra_pred.sh
```

add_uuid_pred.py

- Añade una columna con el texto “NULL” y otra con un UUID a todas las filas del fichero introducido como primer argumento en la llamada del script y lo guarda en el fichero de salida introducido en el segundo argumento.

```
import csv
import uuid
import sys

filein=str(sys.argv[1])
fileout=str(sys.argv[2])

fin = open(filein, 'r')
fout = open(fileout, 'w')

reader = csv.reader(fin, delimiter=',')
writer = csv.writer(fout, delimiter=',')

for row in reader:
    row.append("NULL")
    row.append(uuid.uuid4())
    writer.writerow(row)
fout.close()
fin.close()
```

cassandra_pred.sh

- Ejecuta el script cql **data_to_predict_ table_create.cql**
- Ejecuta el script cql **predicted_data_table_create_cql**
- Ejecuta el script cql **data_to_predict_table_import_cql**
- Ejecuta el script cql **data_to_predict_table_export_cql**

```
#!/bin/bash

DIR='/usr/share/cassandra/scripts/scripts_cqlsh/'

# Crea la tabla data_to_predict
cqlsh -f ${DIR}data_to_predict_table_create.cql

# Crea la tabla predicted_data
cqlsh -f ${DIR}predicted_data_table_create.cql

# Importa los datos del fichero data_to_predict a la tabla data_to_predict
cqlsh -f ${DIR}data_to_predict_table_import.cql

# Exporta los datos del fichero de la tabla data_to_predict al fichero data_to_predict_out.csv
cqlsh -f ${DIR}data_to_predict_table_export.cql
```

data_to_predict_ table_create.cql

- Borra si existe y crea la tabla **data_to_predict** en el *keyspace* **intrusión_detector** con clave primaria label_class e id.

```
DROP TABLE IF EXISTS intrusion_detector.data_to_predict;

CREATE TABLE IF NOT EXISTS intrusion_detector.data_to_predict (
    id UUID,
    duration int,
    protocol_type text,
    service text,
    flag text,
    src_bytes int,
    dst_bytes int,
    land text,
    wrong_fragment int,
    urgent int,
    hot int,
    num_failed_logins int,
    logged_in text,
    num_compromised int,
    root_shell int,
    su_attempted int,
    num_root int,
    num_file_creations int,
    num_shells int,
    num_access_files int,
    num_outbound_cmds int,
    is_host_login text,
    is_guest_login text,
    count int,
    srv_count int,
    serror_rate float,
    srv_serror_rate float,
    rerror_rate float,
```

```
srv_rerror_rate float,
same_srv_rate float,
diff_srv_rate float,
srv_diff_host_rate float,
dst_host_count int,
dst_host_srv_count int,
dst_host_same_srv_rate float,
dst_host_diff_srv_rate float,
dst_host_same_src_port_rate float,
dst_host_srv_diff_host_rate float,
dst_host_serror_rate float,
dst_host_srv_serror_rate float,
dst_host_rerror_rate float,
dst_host_srv_rerror_rate float,
label_class text,
PRIMARY KEY (label_class,id));
```

predicted_data_table_create.cql

- Crea la tabla **predicted_data** si no existe en el *keyspace* **intrusion_detector** con clave primaria label_class e id.

```
CREATE TABLE IF NOT EXISTS intrusion_detector.predicted_data (
    id UUID,
    duration int,
    protocol_type text,
    service text,
    flag text,
    src_bytes int,
    dst_bytes int,
    land text,
    wrong_fragment int,
    urgent int,
    hot int,
    num_failed_logins int,
    logged_in text,
    num_compromised int,
    root_shell int,
    su_attempted int,
    num_root int,
    num_file_creations int,
    num_shells int,
    num_access_files int,
    num_outbound_cmds int,
    is_host_login text,
    is_guest_login text,
    count int,
    srv_count int,
```

```
  serror_rate float,
  srv_serror_rate float,
  rerror_rate float,
  srv_rerror_rate float,
  same_srv_rate float,
  diff_srv_rate float,
  srv_diff_host_rate float,
  dst_host_count int,
  dst_host_srv_count int,
  dst_host_same_srv_rate float,
  dst_host_diff_srv_rate float,
  dst_host_same_src_port_rate float,
  dst_host_srv_diff_host_rate float,
  dst_host_serror_rate float,
  dst_host_srv_serror_rate float,
  dst_host_rerror_rate float,
  dst_host_srv_rerror_rate float,
  label_class text,
  PRIMARY KEY (label_class,id));
```

data_to_predict_table_import_cql

- Importa los datos del fichero **data_to_predict.csv** a la tabla **data_to_predict**.

```
COPY intrusion_detector.data_to_predict (
    duration, protocol_type, service, flag, src_bytes, dst_bytes, land, wrong_fragment, urgent,
    hot, num_failed_logins, logged_in, num_compromised, root_shell, su_attempted, num_root,
    num_file_creations, num_shells, num_access_files, num_outbound_cmds, is_host_login, is_guest_login,
    count, srv_count, serror_rate, srv_serror_rate, rerror_rate, srv_rerror_rate, same_srv_rate,
    diff_srv_rate, srv_diff_host_rate, dst_host_count,  dst_host_srv_count, dst_host_same_srv_rate,
    dst_host_diff_srv_rate, dst_host_same_src_port_rate, dst_host_srv_diff_host_rate, dst_host_serror_rate,
    dst_host_srv_serror_rate, dst_host_rerror_rate, dst_host_srv_rerror_rate, label_class, id)
FROM '/usr/share/cassandra/scripts/data_to_predict.csv' WITH DELIMITER=',';
```

data_to_predict_table_export_cql

- Exporta los datos de la tabla **data_to_predict** al fichero **data_to_predict_out.csv**.

```
COPY intrusion_detector.data_to_predict TO
'/usr/share/cassandra/scripts/data_to_predict_out.csv'
WITH HEADER = true;
```

classify_connections.py

- Importación de las librerías necesarias.

```
# Importación de librerias
import pandas as pd
import numpy as np
import datetime
import logging
logging.getLogger('tensorflow').disabled = True
import tensorflow as tf
import pandas.api.types as ptypes
from sklearn.metrics import confusion_matrix
```

- Importación a DataFrame de los datos a clasificar desde el fichero **data_to_predict_out.csv**.

```
# Importación de los datos a clasificar
print("Importando los datos para la fase de clasificación predictiva...")
df_pred = pd.read_csv('../data_to_predict_out.csv')
print("Preparando dataframe con las columnas de características...")
x_pred = df_pred.iloc[ : , 2:]
print("Preparando dataframe con la columna de etiquetas...")
y_pred = df_pred.iloc[ : , 0:1]
print("Filas:" + str(len(x_pred)))
print("Columnas:" + str(len(x_pred.columns)))
print("***** FIN DE LA FASE DE IMPORTACIÓN DE DATOS *****")
```

- Definición de la función de entrada para la fase de predicción.

```
# Función de entrada para alimentar el modelo en la fase de predicción
pred_input_fn = tf.estimator.inputs.pandas_input_fn(
    x=x_pred,
    y=y_pred,
    shuffle=False)
```

- Ejecución de la fase de predicción.

```
# Fase de clasificación predictiva
print("***** INICIO DE LA FASE DE CLASIFICACIÓN PREDICTIVA *****")
predicted_results = modelo.predict(input_fn=pred_input_fn)
print("***** FIN DE LA FASE DE CLASIFICACIÓN PREDICTIVA *****")
```

- Exportación de los resultados al fichero **predicted_data.csv**.

```
# Resultados de precisión del algoritmo
print("***** EXPORTACIÓN DE RESULTADOS A FICHERO CSV *****")
prediction_ids = [prediction['class_ids'][0] for prediction in predicted_results]
predicted = pd.DataFrame({"label_class": prediction_ids})
predicted.replace({0: 'NORMAL', 1: 'ATTACK'}, inplace=True)
predicted_out = pd.concat([x_pred, predicted, df_pred.iloc[ : , 1:2]], axis=1)
predicted_out.to_csv('../predicted_data.csv', sep=',', header=None, index=False)
```

- Impresión de los tiempos totales de proceso.

```
# Control de tiempo.
finish_time = datetime.datetime.now()
print("Fin del proceso: " + finish_time.strftime("%d/%m/%Y %H:%M:%S")) # Fin del proceso
total_time = finish_time - start_time
print("Tiempo total empleado en la ejecución: " + str(total_time)) # Tiempo total empleado
print("***** FIN *****")
```

predicted_data_table_import.cql

- Importa los datos del fichero **predicted_data.csv** a la tabla **predicted_data** de la BD.

```
COPY intrusion_detector.predicted_data (
    count, diff_srv_rate, dst_bytes, dst_host_count, dst_host_diff_srv_rate, dst_host_rerror_rate,
    dst_host_same_src_port_rate, dst_host_same_srv_rate, dst_host_serror_rate, dst_host_srv_count,
    dst_host_srv_diff_host_rate, dst_host_srv_rerror_rate, dst_host_srv_serror_rate, duration, flag,
    hot, is_guest_login, is_host_login, land, logged_in, num_access_files, num_compromised, num_failed_logins,
    num_file_creations, num_outbound_cmds, num_root, num_shells, protocol_type, rerror_rate, root_shell,
    same_srv_rate, serror_rate, service, src_bytes, srv_count, srv_diff_host_rate, srv_rerror_rate, srv_serror_rate,
    su_attempted, urgent, wrong_fragment, label_class, id) FROM '/usr/share/cassandra/scripts/predicted_data.csv'
WITH DELIMITER=',';
```

Anexo 6. Mapa del directorio de archivos del producto

A continuación se detalla el contenido del directorio y subdirectorios de la carpeta **producto**:

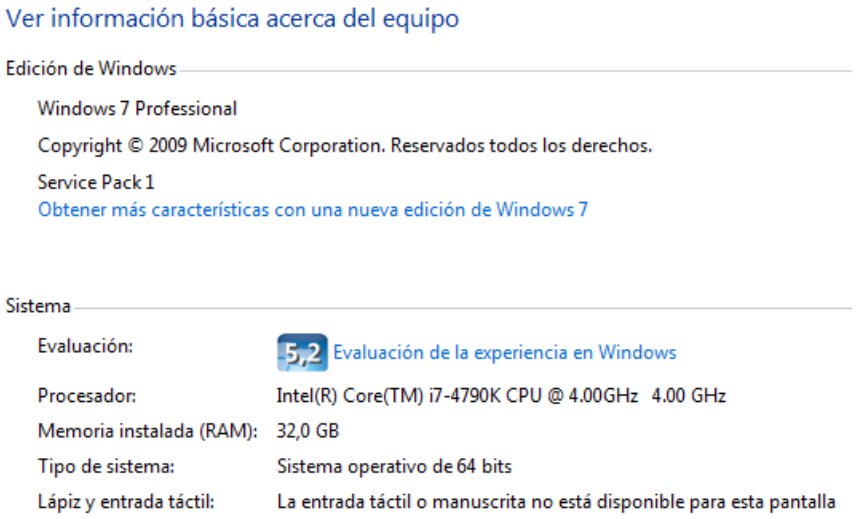
producto
|___Descarga e instalación de Apache Cassandra.pdf
|___Descarga e instalación de Python 3.pdf
|___Descarga e instalación de TensorFlow.pdf
|___LEEME.txt
|___**scripts_cassandra**
    |___add_uuid.py
    |___add_uuid_pred.py
    |___LEEME.txt
    |___load_data.sh
    |___load_data_pred.sh
    |___**kdd_files**
        |___corrected.gz
        |___kddcup.data.gz
        |___kddcup.data_10_percent.gz
        |___kddcup.newtestdata_10_percent_unlabeled.gz
        |___kddcup.testdata.unlabeled.gz
        |___kddcup.testdata.unlabeled_10_percent.gz
    |___**scripts_clasificador**
        |___**mod9**
        |___classify_connections.py
        |___classify_connections.sh
        |___intrusion_pred_model.sh
        |___LEEME.txt
    |___**scripts_cqlsh**
        |___cassandra.sh
        |___cassandra_pred.sh
        |___data_to_predict_table_create.cql
        |___data_to_predict_table_export.cql
        |___data_to_predict_table_import.cql
        |___keyspace_create.cql
        |___keyspace_drop.cql
        |___predicted_data_table_create.cql
        |___predicted_data_table_import.cql
        |___tables_data_export_sh
        |___test_data_table_create.cql
        |___test_data_table_export.cql
        |___test_data_table_import.cql
        |___train_data_table_create.cql
        |___train_data_table_import.cql
        |___validation_data_table_create.cql
        |___validation_data_table_export.cql
        |___validation_data_table_import.cql

Anexo 7. Características del entorno de trabajo

A continuación se detallan las características técnicas del hardware con el que se ha elaborado este proyecto:

Equipo host

- Procesador: Intel i7-7790K 4 GHz (8 procesadores)
- RAM: 32GB
- Sistema operativo: Windows 7 64 bits
- Disco duro: 1 TB



Equipo virtual

- Software de virtualización: VMware Workstation 14 Player
- Nº de procesadores: 4
- RAM: 4 GB
- Disco duro: 50 GB