Aprendizaje supervisado para la detección de amenazas Web

Juan Carlos Moscardó Pérez
Máster en Seguridad de las TIC
Aplicación de técnicas de Machine Learning a la Seguridad

Enric Hernández Jiménez
Víctor García Font

12/2018

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Aprendizaje supervisado para la detección de amenazas Web*
Nombre del autor:	*Juan Carlos Moscardó Pérez*
Nombre del consultor/a:	*Enric Hernández Jiménez*
Nombre del PRA:	*Víctor García Font*
Fecha de entrega (mm/aaaa):	12/2018
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):

El objetivo de este trabajo es el diseño y desarrollo de un detector predictivo de ataques web. Este detector será accesible vía webservice.

El conjunto de datos elegido fue "HTTP DATASET CSIC 2010". Este conjunto de datos contiene el tráfico generado dirigido a una aplicación web de comercio electrónico desarrollada en el departamento CSIC. El 85% del conjunto de datos se utilizó para aprendizaje y el 15% restante para pruebas y verificación.

El trabajo incluye diferentes etapas: estudio y preprocesado del dataset, selección de características, selección y optimización del algoritmo de aprendizaje, diseño, entrenamiento, verificación y pruebas. En algoritmo finalmente elegido para desarrollar el modelo en Python y Scikit-learn fue el árbol de decisión.

El modelo predictivo generado se construyó en ~0.5s y tardó ~0.05s para clasificar 14,560 instancias (~ 290,000 clasificaciones por segundo) y un 98.86% de exactitud.

Abstract (in English, 250 words or less):

The aim of this work is the design and development of a web attacks predictive detector. This detector will be accessible via webservice.

The dataset chosen was "HTTP DATASET CSIC 2010". This dataset contains the generated traffic targeted to an e-commerce web application developed at the CSIC department. The 85% of the dataset was used for training purposes and the remaining 15% for testing and verification.

The work includes different stages: dataset study, dataset preprocessing, features selection, algorithm selection and optimization, design, training, verification and testing. The project has been made through Decision Tree Classifier algorithm in Python and Scikit-learn.

The predictive model generated was build in ~0.5s and spent ~0.05s to classify 14,560 instances (~290,000 classifications per second) with 98,86% accuracy.

Índice

Ilustraciones

1. Introducción

1.1. Contexto

Cada año aumenta el volumen de información transmitida y de sistemas interconectados. Además, los ciberataques son más frecuentes, sofisticados y dañinos. Por estos motivos, los métodos tradicionales de seguridad[1] resultan insuficientes a la hora de dar una respuesta rápida y eficaz ante las nuevas amenazas.

En los sistemas de seguridad tradicionales, cuando se detecta un ataque es porque ya ha habido una infección y esta ha sido reportada por la victima. La organización[2] encargada del mantenimiento de la base de datos de firmas y patrones tendrá que estudiar el ataque, generar el patrón y/o firma y publicarlo, para que los usuarios actualicen su base de datos y estén protegidos.

Las principales desventajas de los sistemas tradicionales son:

- No detectan ataques de día cero. Estos son nuevos ataques que no se tiene información sobre ellos.
- El ataque, puede tardar días, meses o incluso no ser detectado. Facilitando la posibilidad de que los hackers puedan acceder a los sistemas comprometidos para realizar actividades ilícitas.
- Crecimiento constante de la base de datos de firma y patrones, repercutiendo en el espacio y proceso de los sistemas informáticos.

Para solucionar los inconvenientes anteriores, existen sistemas de seguridad basados en aprendizaje automático. Estos sistemas, generan a partir de datos históricos previamente tratados, modelos predictivos para detectar ataques de día cero. Estos modelos solucionan los problemas de los sistemas tradicionales:

- No necesitan una base de datos de firmas.
- El modelo predictivo detecta rápida y eficazmente ataques de día cero, comportamientos anómalos, ficheros maliciosos, spam, etc... analizando similitudes y comportamientos con ataques.

1 Los sistemas tradicionales son los basados en firmas y en detección de patrones.

2 Un ejemplo de organización: La empresa de un antivirus.

Los sistemas basados en aprendizaje automático se pueden utilizar por separado o conjuntamente con los sistemas tradicionales añadiendo otra capa más de seguridad.

Este trabajo se centra en la detección de amenazas Web creando un modelo predictivo supervisado accesible por webservice que clasifica peticiones HTTP como normales o anómalas (posible ataque). En el futuro, este webservice se podrá utilizar como motor de un sistema WAF (Web application Firewall) predictivo

La decisión de centrase sólo en el protocolo HTTP ha sido motivada por: ser el más utilizado para ofrecer servicios online, sus servicios suelen estar abiertos a todos[3] y existir una gran cantidad plataformas, paquetes y software compartido que podrían contener vulnerabilidades de seguridad.

Este trabajo servirá de base para poder desarrollar un sistema WAF predictivo de fácil implantación para proteger aplicaciones Web.

1.2. Objetivos del Trabajo

- Estudiar los tipos de algoritmos de aprendizaje automático y cuando aplicar cada uno de ellos.
- Trabajar con herramientas que permitan analizar el conjunto de datos que se va a utilizar.
- Aplicar una metodología de resolución de problemas utilizando el aprendizaje automático supervisado.
- Estudiar los método más comunes de selección de características.
- Estudiar los algoritmos más comunes de aprendizaje supervisado.
- Estudiar y desarrollar una aplicación que genere un modelo predictivo supervisado para la detección de amenazas Web en Python con la librería Scikit-learn
- Implementar un webservice que utilice el modelo predictivo generado para clasificar las peticiones HTTP como normales o anómalas.

3 En comparación con otros servicios más restringidos como: ssh, bases de datos, etc...

1.3. Enfoque y método seguido

Este trabajo está dividido en 3 fases:

- Investigación y estudio sobre el aprendizaje automático, sus algoritmos, aplicaciones y metodología para resolver problemas con aprendizaje supervisado.
- Desarrollo del caso práctico.
  - Uso de herramientas necesarias para analizar el conjunto de datos (en inglés, dataset) elegido.
  - Desarrollar un script en Python que construya un modelo para clasificar las conexiones como normales o anómalas.
  - Guardar el modelo.
  - Cargar el modelo generado e integrarlo en un webservice.
- Documentar el proceso seguido durante todo el desarrollo de este trabajo, conclusiones y trabajo futuro.

1.4. Planificación del Trabajo

La planificación para llevar a cabo este trabajo ha sido la siguiente:

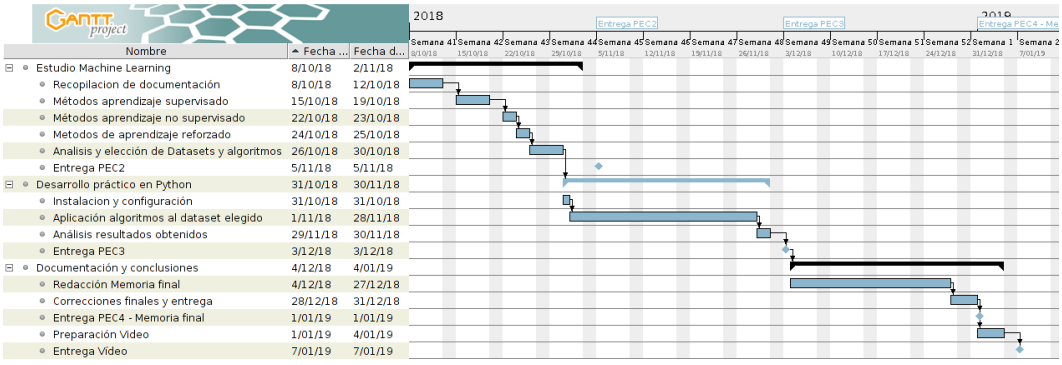


Ilustración 1: Planificación

1.5. Propuesta tecnológica

Para la realización del trabajo práctico se ha utilizado una máquina virtual de 4 cores i7 @2.4GHz, 4GB de memoria RAM y 6GB de disco SSD.

El sistema operativo instalado es Ubuntu 16.04 LTS, y los lenguajes de programación instalados son Python 3 y Java 1.8. También se ha utilizado el reenvío del gestor de ventanas X Window (X11) para ejecutar el entorno gráfico.

Las librerías principales utilizadas para el desarrollo del trabajo son las siguientes:

1.5.1. Scikit-learn

Es una librería de código abierto con licencia BSD de aprendizaje automático en Python. Con las herramientas de esta librería se puede realizar de una forma simple y eficiente el minado y análisis de datos. Es fácil de utilizar y se puede reutilizar e integrar en varios entornos. Está construido sobre NumPy, SciPy y matplotlib.

https://scikit-learn.org

1.5.2. Pandas

Es una librería de código abierto con licencia BSD que proporciona estructuras de datos de alto rendimiento y fáciles de usar y herramientas de análisis de datos para el lenguaje de programación Python.

https://pandas.pydata.org/

1.5.3. NumPy

Es la librería de código abierto con licencia BSD fundamental para la computación científica en Python y añade sofisticadas funciones y objetos para mejorar el tratamiento de vectores y matrices.

http://www.numpy.org/

1.5.4. Weka

Weka (Waikato Environment for Knowledge Analysis, en español «entorno para análisis del conocimiento de la Universidad de Waikato») es una plataforma de software para el aprendizaje automático y la minería de datos

escrito en Java y desarrollado en la Universidad de Waikato. Weka es software libre distribuido bajo la licencia GNU-GPL.

https://www.cs.waikato.ac.nz/ml/weka/

1.5.5. Flask

Flask es un framework minimalista escrito en Python que permite crear aplicaciones web rápidamente y con un mínimo número de líneas de código. Está basado en la especificación WSGI de Werkzeug y el motor de templates Jinja2 y tiene una licencia BSD.

http://flask.pocoo.org/

1.6. Breve sumario de productos obtenidos

Los productos obtenidos en este trabajo son los siguientes:

- Dataset completo en CSV con nuevas características añadidas.
- Una muestra en CSV con el 15% de instancias para realizar pruebas en la herramienta Weka.
- Dos muestras en CSV para Scikit-learn. Una con el 85% de las instancias para entrenamiento y otra con el 15% restante para verificar la precisión de la clasificación.
- Código fuente:
  - Script de importación y transformación del dataset original a CSV.
  - Script de la generación del modelo predictivo
  - Script del webservice detector de amenazas web.
  - Script de test del modelo y del webservice
- Memoria del trabajo.
- Presentación del trabajo.

1.7. Breve descripción de los otros capítulos de la memoria

En el 2º capítulo se introduce el concepto de Machine Learning o aprendizaje automático. Se describen los tipos de clasificaciones, algoritmos y su estado del arte.

En el 3º capítulo se describe y desarrolla la generación del modelo predictivo de clasificación y el webservice.

En el 4º capítulo se enumeran las conclusiones y el trabajo futuro.

Y por último, el glosario, la bibliografía y los anexos.

2. Machine Learning

El aprendizaje automático (en inglés, Machine learning) es una rama de la Inteligencia artificial cuyo objetivo es la creación de modelos capaces de generalizar comportamientos, a partir de una muestra de información suministrada como ejemplo.

Formalmente se puede definir como el proceso computacional de inferir automáticamente y generalizar un modelo de aprendizaje a partir de una muestra de datos. Los modelos de aprendizaje utilizan funciones estadísticas o reglas, para describir las dependencias sobre datos, casualidades y correlaciones entre la entrada y la salida.

Los modelos creados tienen que ser evaluados empíricamente debido a que su rendimiento y precisión están estrechamente relacionados con la cantidad y calidad de las instancias de entrenamiento. Por este motivo se utiliza el mismo conjunto de instancias para comparar los algoritmos.

2.1. Tipos de algoritmos

Los algoritmos de aprendizaje automático se pueden dividir en categorías según sea su propósito y la tipología de problema que se pretenda resolver:

2.1.1. Aprendizaje supervisado (Supervised learning)

En este tipo de aprendizaje se utiliza un conjunto de datos de entrada que han sido previamente clasificados. Con estos datos se entrena a un algoritmo, que construye un modelo para predecir datos desconocidos con

cierta eficacia[4]. Podríamos definir este modelo de aprendizaje como una función que según unos parámetros de entrada nos devolverá una salida.

Dependiendo del dominio de esa salida podemos dividir estos algoritmos en 2 tipos:

- **Clasificación**: Si el resultado de datos es de tipo discreto. Por ejemplo:
  - Si es un ataque o no.
  - Catalogar un artículo según un texto que lo describa,
  - Predecir si estará despejado, nuboso o lloverá.

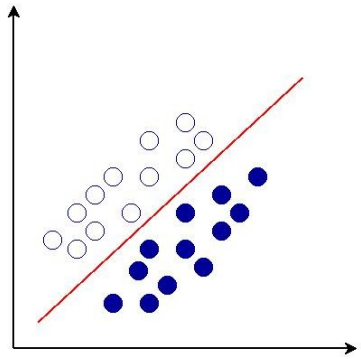


Ilustración 2: Clasificación lineal

- **Regresión**: Si es un valor continuo. Por ejemplo:
  - El salario de un trabajador,
  - El valor de venta de un inmueble.

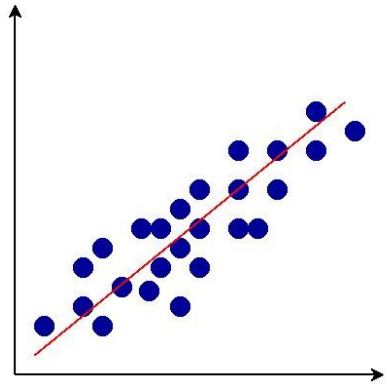


Ilustración 3: Regresión lineal

4 La eficacia dependerá de la cantidad y calidad del conjunto de datos de entrenamiento, y del conocimiento y especificaciones del problema.

Los algoritmos de aprendizaje supervisado a través de la fase de entrenamiento, modelan reglas y/o relaciones de dependencia, que pueden predecir una salida dependiendo de las características de entrada.

Los algoritmos más utilizados para resolver problemas con técnicas de aprendizaje supervisado son:

2.1.1.1. Aprendizaje basado en Regresión logística

Estos algoritmos utilizan un tipo de análisis estadístico de regresión para poder predecir el resultado de una categoría, en función de unos atributos independientes de entrada.

2.1.1.2. Algoritmos basados en Instancia

Estos algoritmos clasifican una nueva instancia, buscando las instancias más parecidas en el conjunto de entrenamiento.

El algoritmo más usado:

- k-Nearest Neighbor

2.1.1.3. Aprendizaje basado en Árboles de decisión

Este tipo de aprendizaje utiliza la estructura de los árboles de decisión para modelar predicciones y/o clasificaciones. Mediante el conjunto de instancias de entrenamiento se genera el modelo predictivo, donde las ramas, representan la conjunción de los atributos que conducen a las hojas, que representan la etiqueta de clasificación.

Los algoritmos más importantes son:

- ID3 (Iterative Dichotomiser 3)
- C.45 (sucesor del ID3)
- CART (Classification And Regression Tree)
- CHAID (CHi-squared Automatic Interaction Detector)

2.1.1.4. Algoritmos Bayesianos

Estos algoritmos utilizan el Teorema de Bayes de probabilidad para crear un modelo predictivo. Este modelo encuentra la hipótesis más probable según el conjunto de datos de entrenamiento y la probabilidad de cada hipótesis.

Los algoritmos más utilizados son:

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

2.1.1.5. Algoritmos SVM (Support Vector Machine)

Estos algoritmos se construyen utilizando el conjunto de datos de entrenamiento como vectores de (k-atributos), un hiperplano o conjunto de hiperplanos que maximizan la distancia (margen) entre las diferentes clases. Es decir, los puntos del vector etiquetados con una categoría estarán a un lado del hiperplano y los otros vectores clasificados con otra categoría estarán en el lado opuesto del hiperplano.

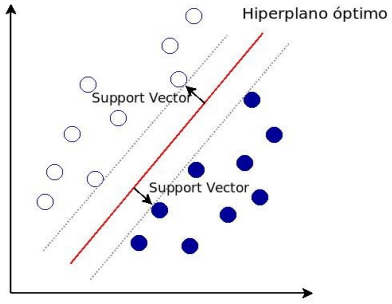


Ilustración 4: Ejemplo SVM

2.1.1.6. Redes neuronales

Una red neuronal artificial es una colección de nodos llamados neuronas artificiales (intentan parecerse a un cerebro biológico). Una neurona artificial recibe una señal de entrada que puede procesar y luego enviar el resultado a otras neuronas a las que está "conectada".

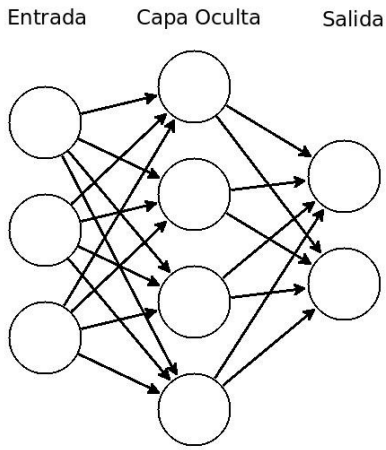


Ilustración 5: Estructura red neuronal artificial

La señal mas común entre neuronas es un número real. La salida de cada neurona artificial se calcula mediante una función no lineal de sus datos de entrada. Las neuronas y las conexiones suelen tener un peso que se va ajustando durante el aprendizaje.

A las redes neuronales artificiales se le añaden diferentes capas que pueden realizar diferentes transformaciones de sus entradas. Las "señales" van desde la primera capa (la de los datos de entrada) hasta la última capa (datos de salida).

2.1.1.7. Algoritmos de Deep learning

Estos algoritmos son una evolución avanzada de los algoritmos de redes neuronales. Estos algoritmos tienen un mayor numero de capas ocultas interconectadas y su estructura de neuronas artificiales tienen la ventaja, que pueden trabajar en paralelo[5], mejorando considerablemente el proceso de grandes volúmenes de información. Estos algoritmos son óptimos para resolver problemas donde se generan grandes volúmenes de información, como por ejemplo: encontrar patrones, reconocimiento de imágenes, conducción autónoma, reconocimiento de voz y traducción automática

5 Utilizando múltiples cores tanto de la CPU como de la GPU.

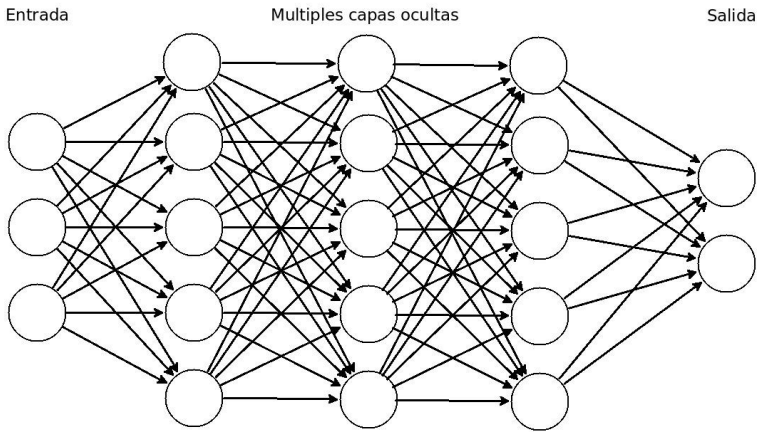


Ilustración 6: Estructura de una red de aprendizaje profundo

2.1.2. Aprendizaje no supervisado (Unsupervised learning)

En este tipo de aprendizaje no hay clases etiquetadas en el dataset. Se explora la información no etiquetada, sin ninguna guía sobre que buscar o agrupar. Las agrupaciones (clusters) o asociaciones encontradas podrán ser útiles y/o ayudar a comprender mejor las relaciones entre los datos del dataset analizado.

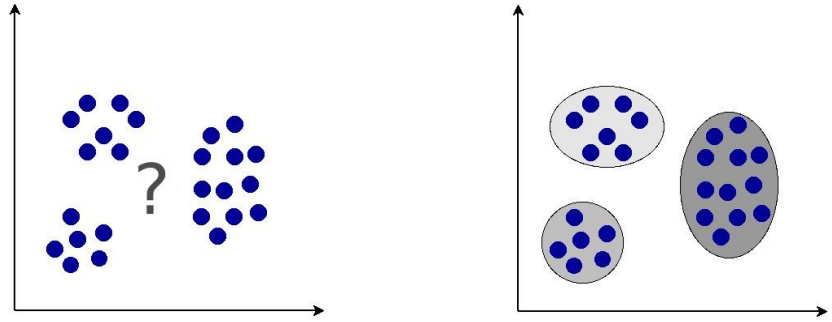


Ilustración 7: Agrupación de instancias con aprendizaje no supervisado

El aprendizaje no supervisado también se utiliza para simplificar datasets reduciendo el número de dimensiones (características). Los tipos de algoritmos más comunes son:

2.1.2.1. Algoritmos de Clustering (agrupación)

Estos algoritmos se utilizan para agrupar una serie de vectores de acuerdo con un criterio. Los criterios más utilizados son la distancia[6] y similitud[7].

Los vectores agrupados comparten propiedades comunes y el conocimiento de estos grupos, pueden aportar una descripción sintética de un conjunto dimensional complejo. Esta descripción se consigue sustituyendo los elementos de esa agrupación, por la de un representante característico del mismo.

Los algoritmos más usados son:

- K-Means
- K-Medians
- Hierarchical Clustering

2.1.2.2. Análisis de componentes principales

En inglés, PCA (Principal Component analysis), este algoritmo realiza una reducción de atributos iniciales, de tal forma, que consigue representar la máxima información con el mínimo número de atributos posibles. Es decir, convierte un conjunto de valores de características posiblemente correlacionados, en un conjunto de valores de características sin correlación, llamados componentes principales.

2.1.3. Aprendizaje semi supervisado (Semi-supervised learning)

En muchas ocasiones, el coste de etiquetar todas las instancias del dataset es bastante alto. La principal causa es que se requieren personas expertas que etiqueten o supervisen el etiquetado de estos datos.

En estos casos, se utiliza este el aprendizaje semi supervisado para construir el modelo predictivo eficiente, con un número bajo de instancias etiquetadas y un número elevado de instancias sin etiquetar.

6 Como la función de distancia euclidiana.

7 Como la matriz de correlación o la maximización de la verosimilitud, propiedad estadística.

Estos algoritmos se basan en la idea de que aunque las instancias no estén etiquetadas, pueden aportar información importante a las instancias etiquetadas que están en el mismo grupo o cluster.

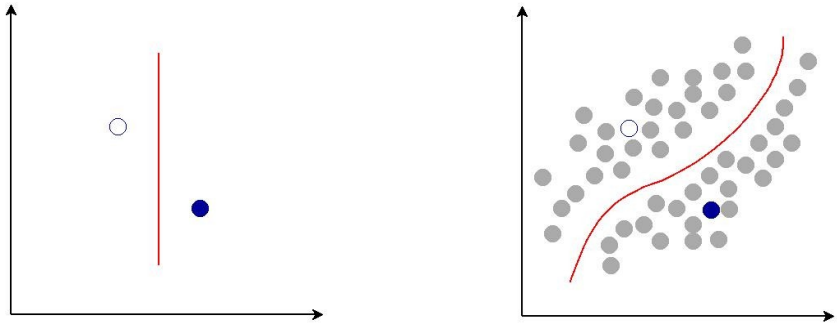


Ilustración 8: Mejora de la predicción con instancias sin etiquetar (derecha).

2.1.4. Aprendizaje reforzado (Reinforcement learning)

Los algoritmos de aprendizaje reforzado, también llamados agentes, crean un modelo que mejora su eficiencia aprendiendo del entorno de una forma iterativa.

Este agente determina automáticamente el comportamiento ideal según un entorno especifico y mediante una retroalimentación simple de recompensa[8], para ir aprendiendo su comportamiento.

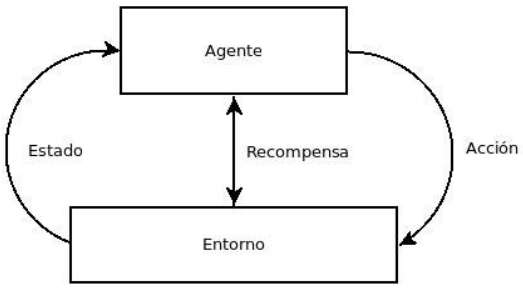


Ilustración 9: Flujo básico de aprendizaje reforzado

Se puede simplificar el funcionamiento del agente de la siguiente forma:

1. Recibe el entorno como entrada.

8 También se conoce como señal de refuerzo.

2. Genera la decisión dependiendo de esa entrada y realiza una acción.
3. Recibe una recompensa (puede ser negativa) para interpretar si ha sido una buena o mala decisión.
4. Almacena el estado, la acción junto con la recompensa recibida del entorno y vuelve al punto 1.

2.2. Estado del arte

Actualmente el aprendizaje automático se está desarrollando y aplicando en estos campos:

Seguridad informática

Detección de ataques, de intrusos, bots, ayuda a la detección de fraudes online y con tarjeta de crédito, detecciones de ataques de día cero u otras amenazas dependiendo de los vectores de infección, Antivirus, detección de spam, detección de anomalías y comportamientos anómalos de conexiones o aplicaciones

Reconocimiento de imágenes

El aprendizaje automático se utiliza para el reconocimiento de imágenes u otros tipos de patrones. Por ejemplo: reconocimiento facial, dactilar, de objetos en un espacio, de voz y/o de escritura...

Los algoritmos Deep learning consiguen una alta precisión en la catalogación de imágenes.

Conducción autónoma

Es un campo en expansión gracias a los algoritmos deep learning. Estos algoritmos entre otras funciones pueden: reconocer imágenes en tiempo real, detectar obstáculos y señales de tráfico, predecir velocidades o adelantarse a posibles accidentes según la forma en que se muevan los vehículo.

Salud

Estos algoritmos pueden ayudar a realizar un diagnostico médico, analizar grandes volúmenes de imágenes en busca de algún tipo de enfermedad o predecir la probabilidad de tener una

enfermedad dependiendo de una serie de características del paciente.

Análisis del mercado de valores

Los modelos generados pueden ayudar a los analistas financieros a predecir el precio de determinadas acciones, aconsejar sobre la mejor acción a realizar sobre un determinado valor,etc.

Motores de Recomendación

Los modelos de aprendizaje automático catalogan a un usuario según unos determinados parámetros y se les hace recomendaciones vinculadas con su catalogación.

Motores de búsqueda

Google utiliza esta tecnología para poder buscar en su banco de imágenes automáticamente catalogadas. También puede buscar imágenes o patrones parecidos.

Reconocimiento del habla

Los algoritmos de aprendizaje automático, en especial los algoritmos Deep Learning, pueden trabajar paralelamente. Sus aspectos destacados son:

- Mejorar la calidad del audio eliminado ruidos.
- Ayudar a la transcripción de textos.
- Permitir detectar sonidos o canciones.

Lingüística computacional

Estas técnicas ayudan a las máquinas a mejorar la interactividad y la comunicación natural con los humanos. Por ejemplo:

- Chatbots.
- Traducción automática.
- Redacción automática de textos y noticias.
- Clasificación de textos para vincularlos a un contexto determinado.

La mayoría de las aplicaciones que utilizan un gran volumen de datos de aprendizaje independiente (con mínima interactividad humana), aprendizaje continuo, proceso en tiempo real y toma de decisiones complejas donde intervienen muchos factores, implementan algoritmos de Deep Learning.

3. Caso práctico

3.1. Desarrollo del modelo de clasificación.

3.1.1. Conjunto de datos

El dataset utilizado para desarrollar el caso práctico es el "HTTP DATASET CSIC 2010". Este conjunto de datos contiene el tráfico generado dirigido a una aplicación web de comercio electrónico desarrollada en el departamento CSIC

Este dataset contiene un total de 97.065 instancias, de ellas, 25.000 son ataques HTTP, que han sido generados automáticamente con las herramientas Paros y W3AF.

Estos son los 3 tipos de peticiones anómalas[9] del dataset:

1. **Ataques estáticos:** Tratar de acceder a recursos ocultos o inexistentes. Solicitudes de archivos obsoletos, ID de sesión en la reescritura de la URL, archivos de configuración, archivos predeterminados, etc.
2. **Ataques dinámicos:** Modificación de los argumentos de una petición válida. Inyección SQL, inyección CRLF, XSS, desbordamiento del buffer, etc.
3. **Solicitudes ilegales involuntarias:** Estas solicitudes no tienen una intención maliciosa, sin embargo, no siguen el

9 Más información en OWASP Top Ten Project 2010 (el dataset es del año 2010)

comportamiento normal de la aplicación web y por lo tanto no tienen la misma estructura que los valores de parámetros normales. Por ejemplo:

Un número de teléfono compuesto por letras.

Este conjunto de datos está compuesto por 3 ficheros (normalTrafficTraining, anomalousTrafficTest y normalTrafficTest). A continuación se muestra un ejemplo del formato de estos ficheros:

```
GET http://localhost:8080/tienda1/publico/anadir.jsp?id=1&nombre=Jam%F3n+Ib%E9rico&precio=39&cantidad=41&B1=A%F1adir+al+carrit
User-Agent: Mozilla/5.0 (compatible; Konqueror/3.5; Linux) KHTML/3.5.8 (like Gecko)
Pragma: no-cache
Cache-control: no-cache
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Encoding: x-gzip, x-deflate, gzip, deflate
Accept-Charset: utf-8, utf-8;q=0.5, *;q=0.5
Accept-Language: en
Host: localhost:8080
Cookie: JSESSIONID=54E25FF4B7F0E4E855B112F882E9EEA5
Connection: close


POST http://localhost:8080/tienda1/publico/anadir.jsp HTTP/1.1
User-Agent: Mozilla/5.0 (compatible; Konqueror/3.5; Linux) KHTML/3.5.8 (like Gecko)
Pragma: no-cache
Cache-control: no-cache
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Encoding: x-gzip, x-deflate, gzip, deflate
Accept-Charset: utf-8, utf-8;q=0.5, *;q=0.5
Accept-Language: en
Host: localhost:8080
Cookie: JSESSIONID=788887A0F479749C4CEEA1E268B4A501
Content-Type: application/x-www-form-urlencoded
Connection: close
Content-Length: 74

id=1&nombre=Jam%F3n+Ib%E9rico&precio=39&cantidad=41&B1=A%F1adir+al+carrito
```

Ilustración 10: Estructura de los ficheros del dataset HTTP CSIC 2010

Los datos de cada instancia del dataset son los siguientes:

- **method**: Método HTTP (Ej.: GET, POST, PUT).
- **url**: Url de llamada HTTP.
- **Accept-Encoding**: Codificación aceptada de la llamada HTTP.
- **Cookie**: Valor de las cookies (en este dataset sólo aparece la cookie de sesión).
- **Accept-Language**: Idioma que acepta el navegador.
- **User-Agent**: Información del navegador.
- **Cache-control**: Configuración de la cache.
- **Connection**: Estado de la conexión HTTP.
- **Accept**: Tipo de petición HTTP que aceptará el navegador como respuesta.

- **Accept-Charset**: Tipo de juego de caracteres que aceptará el navegador.
- **Pragma**: Configuración de la cache.
- **Host**: Nombre del host que recibe la petición HTTP.
- **Content-Type**: Tipo de contenido en el cuerpo de la petición HTTP.
- **Content-Length**: Longitud del contenido del cuerpo de la petición HTTP.
- **args**: Argumentos pasados como POST (en el cuerpo de la llamada).

Estos 3 ficheros se han unido en único en CSV y se han generado otros 3 ficheros CSV que se utilizarán a lo largo del caso práctico:

1. Fichero con un conjunto de muestras del 15% de instancias para análisis. Selección aleatoria para utilizarlo con la herramienta Weka en los procesos de selección de características y del algoritmo.
2. Fichero con un conjunto de entrenamiento con el 85% de instancias. Estas instancias se utilizarán para entrenar el algoritmo y generar el modelo predictivo.
3. Fichero con un conjunto de test con el 15% de instancias restantes. Estas instancias servirán para verificar la exactitud en las predicciones del modelo generado. Ninguna de estas instancias aparece en el conjunto de instancias de entrenamiento.

method	url	Accept-E	Cookie	Acce	User-Age	Cache-co	Connecti	Accept	Accept-C	Pragma	Host	Conte	Content-l	args
GET	http://localhost:8080/tienda1/miembros/fotos.jsp.bak	x-gzip, x-	JSESSION	en	Mozilla/5	no-cache	close	text/xml,	utf-8, utf-	no-cache	localhost:8080			
GET	http://localhost:8080/tienda1/publico/entrar.jsp?errorMsg=Crede	x-gzip, x-	JSESSION	en	Mozilla/5	no-cache	close	text/xml,	utf-8, utf-	no-cache	localhost:8080			
GET	http://localhost:8080/tienda1/publico/productos.jsp	x-gzip, x-	JSESSION	en	Mozilla/5	no-cache	close	text/xml,	utf-8, utf-	no-cache	localhost:8080			
GET	http://localhost:8080/tienda1/publico/anadir.jsp?idA=3&nombre=	x-gzip, x-	JSESSION	en	Mozilla/5	no-cache	close	text/xml,	utf-8, utf-	no-cache	localhost:8080			
GET	http://localhost:8080/tienda1/.inc	x-gzip, x-	JSESSION	en	Mozilla/5	no-cache	close	text/xml,	utf-8, utf-	no-cache	localhost:8080			
POST	http://localhost:8080/tienda1/publico/anadir.jsp	x-gzip, x-	JSESSION	en	Mozilla/5	no-cache	close	text/xml,	utf-8, utf-	no-cache	localhost	appli	73.0	id=3&nc
GET	http://localhost:8080/tienda1/miembros/fotos.jsp	x-gzip, x-	JSESSION	en	Mozilla/5	no-cache	close	text/xml,	utf-8, utf-	no-cache	localhost:8080			
GET	http://localhost:8080/tienda1/publico/caracteristicas.jsp?id=1	x-gzip, x-	JSESSION	en	Mozilla/5	no-cache	close	text/xml,	utf-8, utf-	no-cache	localhost:8080			
GET	http://localhost:8080/tienda1/publico/vaciar.jsp?B2=Vaciar+carr	x-gzip, x-	JSESSION	en	Mozilla/5	no-cache	close	text/xml,	utf-8, utf-	no-cache	localhost:8080			
POST	http://localhost:8080/tienda1/publico/pagar.jsp	x-gzip, x-	JSESSION	en	Mozilla/5	no-cache	close	text/xml,	utf-8, utf-	no-cache	localhost	appli	38.0	modo=i
POST	http://localhost:8080/tienda1/publico/pagar.jsp	x-gzip, x-	JSESSION	en	Mozilla/5	no-cache	close	text/xml,	utf-8, utf-	no-cache	localhost	appli	38.0	modo=i

Ilustración 11: Dataset transformado a CVS

Por las características del problema y por el dataset, donde todas las instancias están etiquetadas, se desarrollará el modelo predictivo con aprendizaje automático supervisado.

3.1.2. Metodología

Los pasos para resolver este caso práctico son los siguientes:

1. Obtener la información del dataset elegido y preprocesarlo para obtener características que puedan ayudar a mejorar la predicción.
2. Determinar que características se van a utilizar para entrenar el modelo. Estas características serán las necesarias para generar un modelo predictivo preciso. Es importante evitar que haya un numero elevado de características para que el modelo no sea muy complejo, pero tiene que haber suficientes características para poder generar una predicción precisa.
3. Seleccionar el algoritmo de aprendizaje a utilizar. Estudiar varios algoritmos de aprendizaje supervisado utilizando el fichero con el 15% de instancias del dataset. Elección del algoritmo con mejor rendimiento.
4. Mejorar el algoritmo de aprendizaje seleccionado ajustando sus parámetros
5. Una vez ajustados los parámetros del algoritmo, se implementa y se evalúa su exactitud.

3.1.3. Paso 1. Preprocesado del dataset y sus características

Debido al formato del dataset, ha sido necesario convertirlo a CSV para poder ser utilizado en Weka y Scikit-learn.

Analizando el dataset, se descarta realizar una extracción de las palabras de las cadenas de texto, al ampliarse considerablemente el numero de dimensiones[10].

Se decide generar nuevas características parametrizando y simplificando los campos de texto, para mantener el numero de características que utilizará el algoritmo lo más bajo posible.

Para valorar con que nuevas características se enriquece el dataset es importante tener conocimiento sobre el conjunto de datos elegido y sobre la problemática que se quiere resolver.

Las nuevas características creadas a partir de los datos originales son:

- **feature-url-path**: url path sin los parámetros Get.
- **feature-url-path-length**: longitud de url path.
- **feature-extension-name**: extensión del tipo de recurso en la llamada HTTP (contenido del path que hay después del primer punto).
- **feature-extension-name-length**: longitud de la extensión, normalmente 3 (jsp, gif,...).
- **feature-request-length**: longitud total de la URL y parámetros Get y/o Post.
- **feature-arguments-values-length**: longitud total del contenido de todos los parámetros.
- **feature-arguments-values-type**: Simplificación del tipo de dato de cada parámetro. Por ejemplo, si los parámetros son los siguientes:

  id=2&nombre=Vino+Rioja&precio=39&cantidad=69

  El valor de de este campo será:

  [NUMERIC][ALPHA][NUMERIC][NUMERIC]

10 Cada token o palabra es una nueva característica que se tiene que utilizar para generar el modelo.

- **feature-number-arguments**: número de parámetros en la petición HTTP.
- **feature-arguments-names**: lista ordenada de los nombres de los parámetros. Por ejemplo, si la cadena de parámetros es:

  id=2&nombre=Vino+Rioja&precio=39&cantidad=69

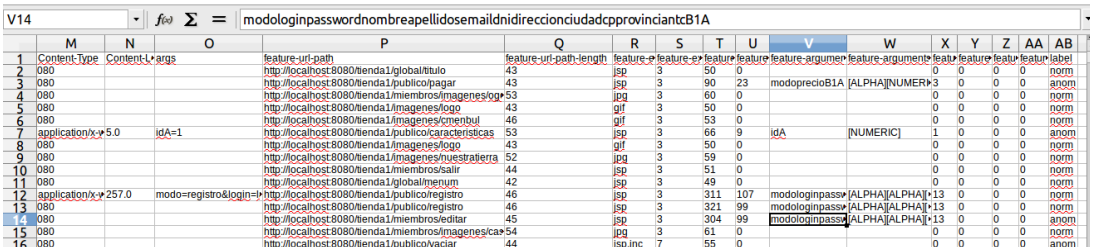
  El valor de de este campo será:

  idnombrepreciocantidad

  Junto con feature-arguments-values-type se consigue agrupar todos los nombres de variables y los dominios de sus valores en dos características.[11]
- **feature-number-special-chars-path**: número de caracteres *especiales*[12] en el path.
- **feature-number-special-chars-extension**: número de caracteres especiales en el campo 'feature-extension-name.'
- **feature-number-special-chars-arguments**: número de caracteres especiales en los valores de los parámetros Get o Post.
- **label**: etiqueta de clasificación de la petición HTTP. Se utiliza para la fase de entrenamiento y verificación. Los valores son:

  norm: Conexión normal,

  anom: Conexión anómala o ataque



	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB
1	Content-Type	Content-L	args	feature-url-path	feature-url-path-length	feature-e	feature-ex	feature	feature	feature-argumen	feature-arguments	featu	feature	featu	featur	label
2	080			http://localhost:8080/tienda1/global/titulo	43	jsp	3	50	0			0	0	0	0	norm
3	080			http://localhost:8080/tienda1/publico/pagar	43	jsp	3	90	23	modoprecioB1A	[ALPHA][NUMERI	3	0	0	0	anom
4	080			http://localhost:8080/tienda1/miembros/imagenes/og	53	jpg	3	60	0			0	0	0	0	norm
5	080			http://localhost:8080/tienda1/imagenes/logo	43	gif	3	50	0			0	0	0	0	norm
6	080			http://localhost:8080/tienda1/imagenes/cmenbul	46	gif	3	53	0			0	0	0	0	norm
7	application/x-	5.0	idA=1	http://localhost:8080/tienda1/publico/caracteristicas	53	jsp	3	66	9	idA	[NUMERIC]	1	0	0	0	anom
8	080			http://localhost:8080/tienda1/imagenes/logo	43	gif	3	50	0			0	0	0	0	norm
9	080			http://localhost:8080/tienda1/imagenes/nuestratierra	52	jpg	3	59	0			0	0	0	0	norm
10	080			http://localhost:8080/tienda1/miembros/salir	44	jsp	3	51	0			0	0	0	0	norm
11	080			http://localhost:8080/tienda1/global/menum	42	jsp	3	49	0			0	0	0	0	norm
12	application/x-	257.0	modo=registro&login=	http://localhost:8080/tienda1/publico/registro	46	jsp	3	311	107	modologinpassv	[ALPHA][ALPHA][	13	0	0	0	norm
13	080			http://localhost:8080/tienda1/publico/registro	46	jsp	3	321	99	modologinpassv	[ALPHA][ALPHA][	13	0	0	0	norm
14	080			http://localhost:8080/tienda1/miembros/editar	45	jsp	3	304	99	modologinpassv	[ALPHA][ALPHA][	13	0	0	0	anom
15	080			http://localhost:8080/tienda1/miembros/imagenes/ca	54	jpg	3	61	0			0	0	0	0	norm
16	080			http://localhost:8080/tienda1/publico/vaciar	44	jsp.inc	7	55	0			0	0	0	0	anom

Ilustración 12: Extracto del CSV generado con las nuevas características

11 Se evita crear un modelo a partir de un conjunto de datos con un gran numero de características. Si los argumentos se dividen en palabras, cada característica sería una palabra.

12 Previo análisis visual del dataset, se consideran caracteres especiales: +*=^<>%(\"'~'

3.1.4. Paso 2. Selección de las características o atributos

En este paso se hace un estudio sobre la importancia de las características del modelo para una correcta clasificación. Para este estudio se utilizará la herramienta Weka con una muestra del 15% del dataset.

Al haber varios métodos de selectores de características se han hecho los estudios con los mas utilizados. De los resultados obtenidos se han seleccionado las características comunes más representativas de cada uno. Hay que tener en cuenta que las características cookie, url y args tienen una gran cantidad de valores únicos que podrían influir en el sesgo y la varianza del algoritmo predictivo. (ver Anexo 1)

Por otra parte, se pueden ignorar las características donde solo exista un único valor, que al final no añade información relevante al modelo predictivo. Para ayudar a la selección de características se muestra a continuación una pantalla de la herramienta Weka, donde se visualizan la distribución de las etiquetas norm y anom en cada característica.

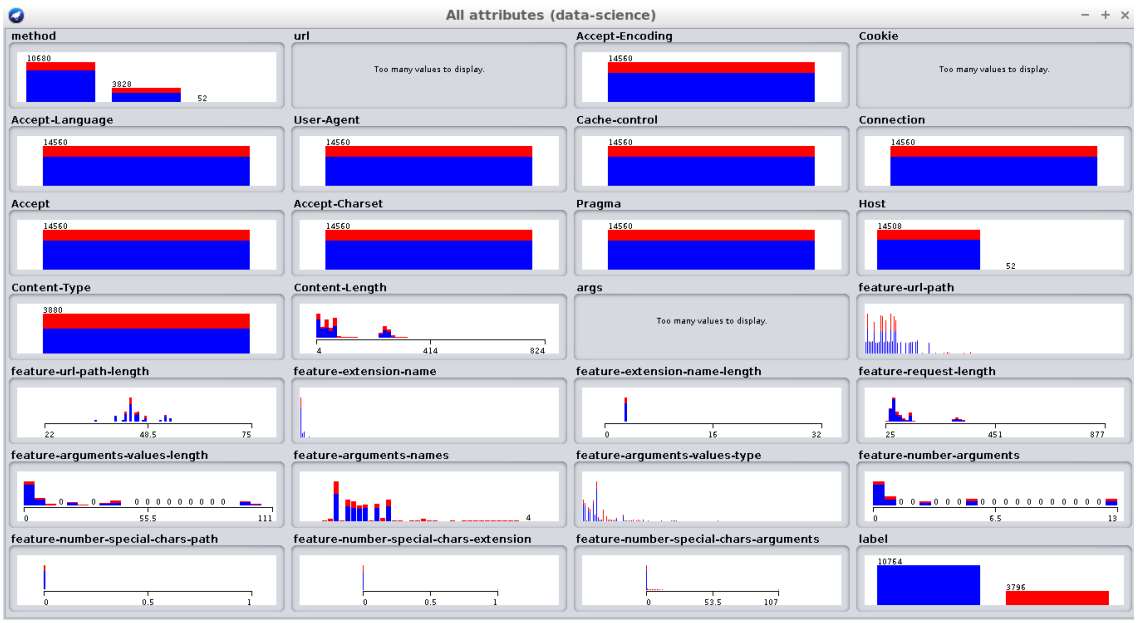


Ilustración 13: Distribución de las características del dataset

Se marca como objetivo utilizar entre 6 a 14 características, que significa una reducción de entre un 50% a 75%[13] de características, sin que afecte a la eficacia del modelo.

Relation: dataset-15%-sample
Instances: 14560
Attributes: 28 (27 + etiqueta label)
Evaluation mode: evaluate on all training data

3.1.4.1. Estudio 1. Correlation Ranking Filter

```
Evaluator:    weka.attributeSelection.CorrelationAttributeEval
Search:       weka.attributeSelection.Ranker -T -1.7976931348623157E308 -N -1
Ranked attributes:
 0.25556   20 feature-request-length
 0.22396   21 feature-arguments-values-length
 0.21149   19 feature-extension-name-length
 0.20649   24 feature-number-arguments
 0.20559   23 feature-arguments-values-type
 0.1937    27 feature-number-special-chars-arguments
 0.17005   22 feature-arguments-names
 0.16353    1 method
 0.15699   15 args
 0.1067    18 feature-extension-name
 0.10081   12 Host
 0.0948    25 feature-number-special-chars-path
...
...
Selected attributes:
20,21,19,24,23,27,22,1,15,18,12,25,14,16,26,2,17,4,3,6,5,7,13,11,9,8,10 : 27
```

3.1.4.2. Estudio 2. Information Gain Ranking Filter

```
Evaluator:    weka.attributeSelection.InfoGainAttributeEval
Search:       weka.attributeSelection.Ranker -T -1.7976931348623157E308 -N -1
Ranked attributes:
 0.82782    4 Cookie
 0.57343    2 url
```

13 Número total de características: 27. No se tiene en cuenta la etiqueta 'label' que almacena la clasificación de la instancias)

```
 0.45112   20 feature-request-length
 0.2199    21 feature-arguments-values-length
 0.21183   16 feature-url-path
 0.16836   18 feature-extension-name
 0.16581   27 feature-number-special-chars-arguments
 0.09537   23 feature-arguments-values-type
 0.09116   17 feature-url-path-length
 0.08248   19 feature-extension-name-length
 0.06264   24 feature-number-arguments
 0.06061   15 args
...
...
Selected attributes:
4,2,20,21,16,18,27,23,17,19,24,15,22,14,1,12,25,26,6,5,3,13,8,11,10,9,7 : 27
```

3.1.4.3. Estudio 3. OneR feature evaluator

```
Evaluator:    weka.attributeSelection.OneRAttributeEval -S 1 -F 10 -B 6
Search:       weka.attributeSelection.Ranker -T -1.7976931348623157E308 -N -1
Ranked attributes:
88.9011    20 feature-request-length
84.25137   23 feature-arguments-values-type
83.87363   21 feature-arguments-values-length
82.23901   27 feature-number-special-chars-arguments
81.76511   22 feature-arguments-names
79.63599   14 Content-Length
79.57418    2 url
79.40247   18 feature-extension-name
78.4272    19 feature-extension-name-length
77.59615   16 feature-url-path
76.90247   17 feature-url-path-length
76.65522   15 args
...
...
Selected attributes:
20,23,21,27,22,14,2,18,19,16,17,15,12,1,25,26,6,3,4,5,24,7,8,13,10,9,11 : 27
```

3.1.4.4. Estudio 4. Classifier feature evaluator

Se utiliza el esquema de aprendizaje del árbol de decisión J48

```
Evaluator:    weka.attributeSelection.ClassifierAttributeEval -execution-slots 1 -B
weka.classifiers.trees.J48 -F 5 -T 0.01 -R 1 -E DEFAULT -- -C 0.25 -M 2

Search:       weka.attributeSelection.Ranker -T -1.7976931348623157E308 -N -1

Learning scheme: weka.classifiers.trees.J48

Scheme options: -C 0.25 -M 2

Ranked attributes:

 0.14317   20 feature-request-length

 0.09341   21 feature-arguments-values-length

 0.08685   23 feature-arguments-values-type

 0.07706   27 feature-number-special-chars-arguments

 0.07232   22 feature-arguments-names

 0.03894   19 feature-extension-name-length

 0.03843   18 feature-extension-name

 0.02984   16 feature-url-path

 0.0237    17 feature-url-path-length

 0.00474   12 Host

 0.00474    1 method

 0.00282   25 feature-number-special-chars-path

...

...

Selected attributes:
20,21,23,27,22,19,18,16,17,12,1,25,7,5,4,24,3,2,6,8,15,13,26,11,9,10,14 : 27
```

3.1.4.5. Elección de características

Con la información resultante se muestran las características mejor posicionadas y se seleccionan las que aparecen repetidas en los 4 estudios.

Estudio	Ranking 1 al 14													
1	**20**	**21**	**19**	24	**23**	**27**	**22**	1	15	**18**	12	25	14	**16**
2	4	2	**20**	**21**	**16**	**18**	**27**	**23**	17	**19**	24	15	**22**	14
3	**20**	**23**	**21**	**27**	**22**	14	2	**18**	**19**	**16**	17	15	12	1
4	**20**	**21**	**23**	**27**	**22**	**19**	**18**	**16**	17	12	1	25	7	5

Tabla 1: Ranking de las mejores características de los 4 estudios

Las características seleccionadas son:

```
16 feature-url-path
18 feature-extension-name
```

```
19 feature-extension-name-length
20 feature-request-length
21 feature-arguments-values-length
22 feature-arguments-names
23 feature-arguments-values-type
27 feature-number-special-chars-arguments
```

3.1.5. Paso 3. Definir el algoritmo de aprendizaje a utilizar

En este apartado se realiza un estudio de la eficacia de los algoritmos más utilizados en una muestra del 15% del dataset. La forma de verificar el algoritmo ha sido mediante la división de la muestra, en un conjunto de datos de entrenamiento (85%) y de test (15%)

Para evaluar el grado de desempeño del algoritmo se utiliza la Matriz de confusión (en inglés, confusion matrix). Esta herramienta permite saber con que precisión ha clasificado un algoritmo de aprendizaje supervisado.

Cada columna representa en número de predicciones de cada categoría, y las filas representan las instancias en la clase real.

		Predicción	
		Normal	Anómala
Real	Normal	TN Verdaderos negativos	FP Falsos Positivos
	Anómala	FN Falsos Negativos	TP Verdaderos positivos

Ilustración 14: Formato de la matriz de confusión.

La Información que se puede obtener de una matriz de confusión es:

Exactitud (AC):

Es la proporción de casos que fueron clasificados correctamente.

$$AC = \frac{TN+TP}{TN+FN+FP+TP}$$

Tasa de errores:

Es la proporción de casos que no fuero clasificados correctamente.

$$ER = \frac{FN+FP}{TN+FN+FP+TP}$$

Tasa de verdaderos positivos (TPR) o sensibilidad:

Es la proporción de casos positivos que se predijeron correctamente.

$$TPR = \frac{TP}{FN+TP}$$

Tasa de falsos positivos (FPR):

Es la proporción de instancias que se han clasificado incorrectamente como positivos.

$$FPR = \frac{FP}{FP+TN}$$

Tasa de negativos verdaderos (TNR) o especificidad:

Es la proporción de los casos negativos que se clasificaron correctamente.

$$TNR = \frac{TN}{FP+TN}$$

Tasa de falsos negativos (FNR):

Es la proporción de los casos positivos que se clasificaron incorrectamente como negativos.

$$FNR = \frac{FN}{FN+TP}$$

Precisión o valor predictivo positivo (P):

Es la proporción de los casos positivos pronosticados correctamente.

$$P = \frac{TP}{FP+TP}$$

Para simplificar la salida de la información se valorarán los algoritmos por: tiempo de construcción del modelo, tiempo de validación del conjunto de test y exactitud (tasa de aciertos).

Para todos los estudios:

Instances: 14560

Attributes: 9 (8 + etiqueta label)

Test mode: split 85.0% train, remainder test

3.1.5.1. Logistic Regression

```
Time taken to build model: 41.17 seconds

Time taken to test model on test split: 0.04 seconds

=== Summary ===

Correctly Classified Instances        2113               96.7491 %

Incorrectly Classified Instances        71                3.2509 %

Kappa statistic                          0.9128

Mean absolute error                      0.0432

Root mean squared error                  0.1714

Relative absolute error                 11.3318 %

Root relative squared error             39.1415 %

Total Number of Instances             2184

=== Confusion Matrix ===

    a    b   <-- classified as

 1608   11 |    a = norm

   60  505 |    b = anom
```

K-Nearest Neighbours

```
IB1 instance-based classifier

using 1 nearest neighbour(s) for classification

Time taken to build model: 0.01 seconds

Time taken to test model on test split: 1.51 seconds

=== Summary ===

Correctly Classified Instances        2119               97.0238 %

Incorrectly Classified Instances        65                2.9762 %

Kappa statistic                          0.921

Mean absolute error                      0.0324

Root mean squared error                  0.1671

Relative absolute error                  8.4916 %

Root relative squared error             38.157  %

Total Number of Instances             2184

=== Confusion Matrix ===

    a    b   <-- classified as

 1602   17 |    a = norm
```

```
   48  517 |    b = anom
```

3.1.5.2. Árbol de decisión. J48 unpruned tree

```
Number of Leaves  : 257
Size of the tree :  513
Time taken to build model: 0.23 seconds
Time taken to test model on test split: 0 seconds
=== Summary ===
Correctly Classified Instances        2137               97.848  %
Incorrectly Classified Instances        47                2.152  %
Kappa statistic                          0.9428
Mean absolute error                      0.027
Root mean squared error                  0.1303
Relative absolute error                  7.0856 %
Root relative squared error             29.7639 %
Total Number of Instances             2184
=== Confusion Matrix ===
    a    b   <-- classified as
 1612    7 |    a = norm
   40  525 |    b = anom
```

3.1.5.3. Árbol de decisión. RandomTree

```
Size of the tree : 16645
Time taken to build model: 0.45 seconds
Time taken to test model on test split: 0.02 seconds
=== Summary ===
Correctly Classified Instances        2077               95.1007 %
Incorrectly Classified Instances       107                4.8993 %
Kappa statistic                          0.8665
Mean absolute error                      0.0481
Root mean squared error                  0.1966
Relative absolute error                 12.6098 %
Root relative squared error             44.8846 %
Total Number of Instances             2184
=== Confusion Matrix ===
    a    b   <-- classified as
 1603   16 |    a = norm
   91  474 |    b = anom
```

3.1.5.4. Árboles de decisión. RandomForest

```
Bagging with 100 iterations and base learner

weka.classifiers.trees.RandomTree -K 0 -M 1.0 -V 0.001 -S 1 -do-not-check-capabilities

Time taken to build model: 7.77 seconds

Time taken to test model on test split: 0.58 seconds

=== Summary ===

Correctly Classified Instances        2111               96.6575 %
Incorrectly Classified Instances        73                3.3425 %
Kappa statistic                          0.9109
Mean absolute error                      0.0517
Root mean squared error                  0.1634
Relative absolute error                 13.5664 %
Root relative squared error             37.3098 %
Total Number of Instances             2184

=== Confusion Matrix ===

    a    b   <-- classified as
 1602   17 |    a = norm
   56  509 |    b = anom
```

3.1.5.5. Naive Bayes Classifier

```
Time taken to build model: 0.02 seconds

Time taken to test model on test split: 0.04 seconds

=== Summary ===

Correctly Classified Instances        1968               90.1099 %
Incorrectly Classified Instances       216                9.8901 %
Kappa statistic                          0.7209
Mean absolute error                      0.112
Root mean squared error                  0.2559
Relative absolute error                 29.385  %
Root relative squared error             58.4308 %
Total Number of Instances             2184

=== Confusion Matrix ===

    a    b   <-- classified as
 1577   42 |    a = norm
  174  391 |    b = anom
```

3.1.5.6. SVM. Stochastic Gradient Descent Classifier.

```
Time taken to build model: 9.95 seconds

Time taken to test model on test split: 0.02 seconds

=== Summary ===

Correctly Classified Instances        2117               96.9322 %
Incorrectly Classified Instances        67                3.0678 %
Kappa statistic                          0.9191
Mean absolute error                      0.0307
Root mean squared error                  0.1752
Relative absolute error                  8.046  %
Root relative squared error             39.9937 %
Total Number of Instances             2184

=== Confusion Matrix ===

    a    b   <-- classified as
 1595   24 |    a = norm
   43  522 |    b = anom
```

3.1.5.7. Artificial Neural Networks (ANN). MultilayerPerceptron.

```
Time taken to build model: 90.26 seconds

Time taken to test model on test split: 3.61 seconds

=== Summary ===

Correctly Classified Instances        2052               93.956  %
Incorrectly Classified Instances       132                6.044  %
Kappa statistic                          0.8322
Mean absolute error                      0.1826
Root mean squared error                  0.2482
Relative absolute error                 47.903  %
Root relative squared error             56.6705 %
Total Number of Instances             2184

=== Confusion Matrix ===

    a    b   <-- classified as
 1606   13 |    a = norm
  119  446 |    b = anom
```

3.1.5.8. Deep Neural Network (DNN). Dl4jMlpClassifier

```
Time taken to build model: 73.84 seconds

Time taken to test model on test split: 0.28 seconds

=== Summary ===
```

```
Correctly Classified Instances          69               3.1593 %
Incorrectly Classified Instances      2115              96.8407 %
Kappa statistic                         -0.5614
Mean absolute error                      0.9567
Root mean squared error                  0.9676
Relative absolute error                250.9053 %
Root relative squared error            220.9417 %
Total Number of Instances             2184
=== Confusion Matrix ===
    a    b   <-- classified as
   26 1593 |    a = norm
  522   43 |    b = anom
```

3.1.5.9. Resultados y selección del algoritmo

A continuación se resumen los resultados de las pruebas con el conjunto de muestra del 15%:

Algoritmo	Tiempo (segundos)		Confusion matrix				Clasificación			
	Build Model	Test Model	norm (negative)		anom (positive)		Incorrecta		Correcta	
			True N.	False N.	False P.	True P.	Número	%	Número	%
Logistic Regression	41,17	0,04	1608	11	60	505	71	3,2509%	2113	96,7491%
K-Nearest Neighbours	0,01	1,51	1602	17	48	517	65	2,9762%	2119	97,0238%
J48 unpruned tree	0,23	*0,00*	1612	7	40	525	47	2,1520%	2137	97,8480%
RandomTree	0,45	*0,02*	1603	16	91	474	107	4,8993%	2077	95,1007%
RandomForest	7,77	*0,58*	1602	17	56	509	73	3,3425%	2111	96,6575%
Naive Bayes Classifier	0,02	*0.04*	1577	42	174	391	216	9,8901%	1968	90,1099%
Stochastic Gradient Descent Classifier	9,95	0,02	1595	24	43	522	67	3,0678%	2117	96,9322%
Artificial Neural Networks	90,26	3,61	1606	13	119	446	132	6,0440%	2052	93,9560%
Deep Neural Network	73,84	0,28	26	1593	522	43	2115	96,8407%	69	3,1593%

Tabla 2: Resumen de los resultados de los algoritmos de aprendizaje

En el siguiente gráfico se puede comparar los porcentajes de las clasificaciones correctas:

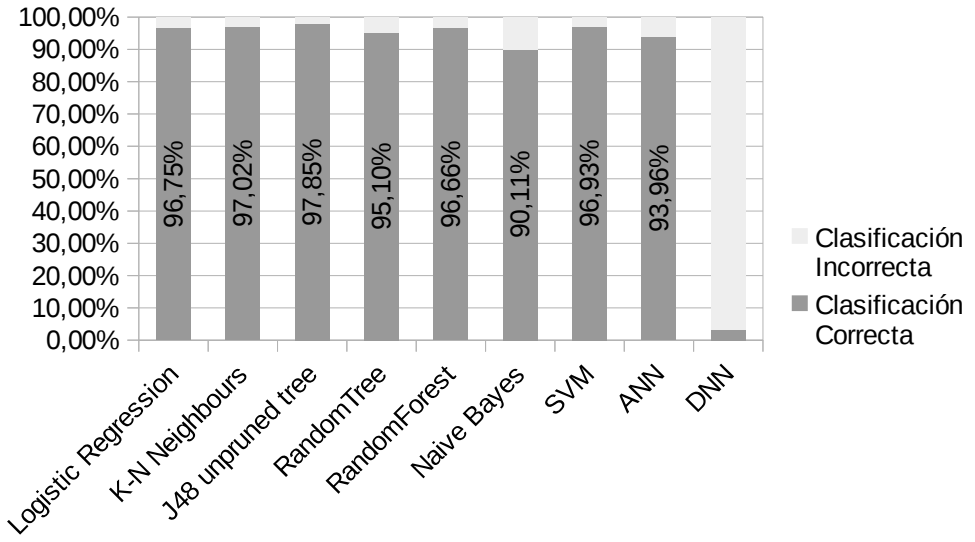


Ilustración 15: Porcentaje de aciertos de los algoritmos estudiados

Según estos resultados, el algoritmo con el mejor porcentaje de clasificaciones correctas, es el Árbol de decisión J48[14] con un 97,8480%, seguido del K-Nearest Neighbours con un 97,0238%.

K-Nearest Neighbours es el algoritmo que genera el modelo más rápidamente (0,01s), seguido de Naive Bayes (0,02s) y el tercero sería el J48 (0,23).

Sobre el tiempo en el que se verifica el modelo con el conjunto de datos de test, el más rápido es J48 (0s), le sigue RandomTree y SVM (los dos con 0,02s)

El valor atípico del algoritmo Deep Neural Network puede ser debido a que los datos nominales no están normalizados o los parámetros del algoritmo no son los adecuados. No se ha dedicado más tiempo a la optimización de este algoritmo porque junto con Logistic Regression y Artificial Neural Networks son los algoritmos que más tardan en generar el modelo.

14 El algoritmo J48 es una implementación open-source del algoritmo C4.5

En este estudio se está buscando el algoritmo con mejor porcentaje de aciertos, menor tiempo de creación del modelo y de verificación del conjunto de test.

Por consiguiente y con los datos enumerados, el algoritmo seleccionado y que mejor **se adapta** a este tipo de clasificación es el **Árbol de decisión J48.** Debido a que tiene el mayor porcentaje de aciertos, la creación del modelo es muy rápida (0,23s) y la verificación del conjunto de datos de test es casi instantánea (0s).

3.1.6. Paso 4. Mejora del algoritmo de aprendizaje

Una vez elegido el algoritmo se procede a ajustar sus parámetros para ver si se puede mejorar su precisión. Para esto, se vuelve a utilizar el método anterior de dividir la muestra en un conjunto de entrenamiento con 85% de instancias y el resto en el conjunto de test para verificar la correcta clasificación.

Se han realizado varias pruebas pero no se ha podido mejorar los resultados del Árbol de decisión J48.

Por lo que la configuración del algoritmo J48 (Árbol de decisión) es la siguiente:

- Unpruned
- Binary
- BatchSize 100
- (weka.classifiers.trees.J48 -U -B -M 2)

3.1.7. Implementación y evaluación de la exactitud

Una vez seleccionadas las características y la mejor configuración del árbol de decisión[15]. Se implementa el algoritmo utilizando la librería Scikit-learn,

15 Scikit-learn implementa una versión mejorada del algoritmo de árbol de decisión CART.

se entrena con el 85% de instancias (dataset-training-85%.csv) del dataset y se evalúa su exactitud con el 15% de instancias restantes (dataset-test-15%.csv).

```
X_train_all_features= pd.read_csv("dataset-training-85%.csv")
y_train=X_train_all_features['label']
X_test_all_features= pd.read_csv("dataset-test-15%.csv")
y_test=X_test_all_features['label']
```

Se eliminan todos los atributos menos los seleccionados en el paso 2:

```
tfm_common.features=['feature-number-special-chars-arguments',
      'feature-request-length', 'feature-arguments-names',
      'feature-arguments-values-length', 'feature-arguments-values-type',
      'feature-extension-name-length', 'feature-extension-name',
      'feature-url-path']
X_train = X_train_all_features[tfm_common.features]
X_test = X_test_all_features[tfm_common.features]
Sefactoriaza
```

Se factorizan los atributos de texto utilizando pandas.factorize[16] en vez de pandas.get_dummies[17]. Este proceso convierte las cadenas de texto en datos numéricos. Es necesario porque el algoritmo sólo acepta entradas numéricas.

Se entrena el modelo[18]:

```
dtc = tree.DecisionTreeClassifier(criterion='entropy',
      presort=True,min_samples_split=2)
dtc.fit(X_train, y_train)
```

Se realiza una predicción con el conjunto de datos de test. El 15% de las instancias del dataset.

```
y_pred = dtc.predict(X_test)
```

16 Pandas factorize funciona igual que Scikit-learn LabelEncoder. Estas funciones codifican una característica de etiquetas a valores numéricos, el resultado tendrá 1 dimensión.

17 Pandas get_dummies funciona igual que Scikit-learn OneHotEncoder. El resultado tendrá n-dimensiones, una por cada etiqueta distinta de la característica codificada.

18 Ver el árbol de decisión generado en el Anexo 7.3

Se obtiene la matriz de confusión entre y_test (etiquetas de test) e y_pred (etiquetas clasificadas) para saber la precisión de modelo.

```
con_matrix=confusion_matrix(y_test, y_pred,labels=confusion_matrix_labels)
```

Se muestran los resultados de esta verificación

```
Time to prepare the factorized datasets : 4.226s


DecisionTreeClassifier
======
Time taken to build model: 0.492s
Time taken to test model on test split: 0.047s


Correctly Classified Instances:   14394         98.860%
Incorrectly Classified Instances: 166           1.140%


Total Instances: 14560


Confusion Matrix
TN:  10827      FP:  43
FN:  123        TP:  3567
N (Negative): Normal request
P (Positive): Anomaly request
```

Se guarda el modelo de clasificación y los índices para que el webservice pueda reutilizar el modelo y factorizar las características.

```
to_serialize=[
    dtc,
    factorize_urls_index,
    factorize_extensions_index,
    factorize_arguments_index,
    factorize_aguments_type_index,
    ]
joblib.dump(to_serialize, 'model_and_resources.pkl')
```

La exactitud en la clasificación del modelo generado con el árbol de decisión CART en Python con Scikit-learn es del 98,86%. El tiempo de

clasificación de las 14.560 instancias fue de ~0,05s (~290.000 clasificaciones por segundo).

3.2. Webservice detector supervisado de amenazas

Este script crea un webservice cuya funcionalidad es la clasificación (normal o anómala) de peticiones HTTP de la aplicación de donde se ha extraído el dataset[19]. Se utilizará el modelo predictivo generado en el punto anterior.

Este webservice recibe por Get o Post los parámetros Method, Url y Args y retorna su catalogación: norm (normal) o anom (anómala). La aplicación que ha consumido el webservice actuará dependiendo del resultado permitiendo o bloqueando la conexión.

3.2.1. Funcionalidades

Las funcionalidades de este script son las siguientes:

- Cargar en memoria el modelo entrenado que está almacenado en un fichero
- Procesar las peticiones recibidas a la url "/predict" del puerto 8888, predecir si es una llamada normal o anómala y retornar el valor al cliente. Por ejemplo:

  ```
  http://127.0.0.1:8888/predict?
  method=GET&url=http://localhost:8080/tienda1/index.jsp
  ```

  Retorna:

  ```
  result=norm
  ```

3.2.2. Arquitectura

Este webservice está pensado para funcionar independientemente, por lo que es muy flexible a la hora de consumirlo.

19 El Dataset CSIC HTTP 2001 se generó con una aplicación de prueba desarrollada por CSIC. Para proteger otra aplicación web será necesario extraer un conjunto de datos con llamadas especificas de esa aplicación y realizar el proceso descrito en el caso práctico.

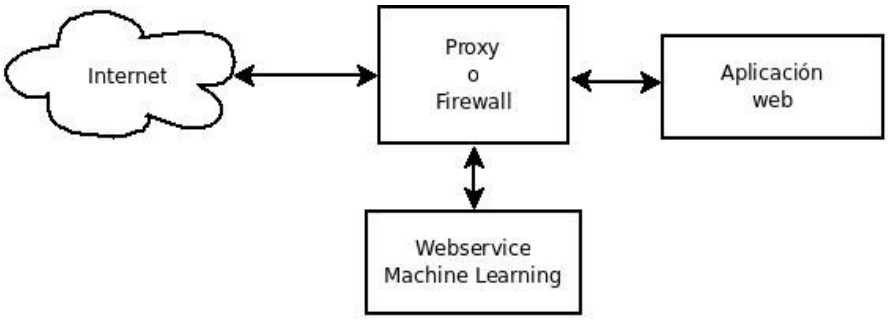


Ilustración 16: Servicio consumido por un firewall/proxy o pasarela

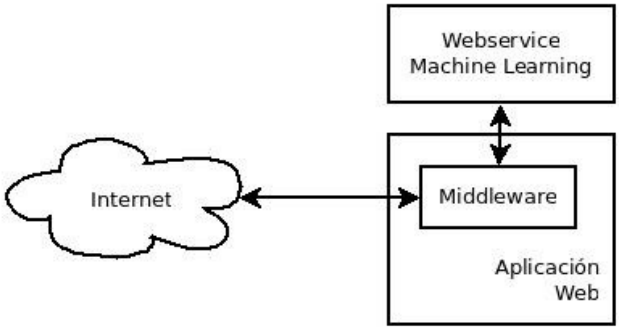


Ilustración 17: Servicio consumido por un middleware de la aplicación

3.2.3. Implementación

Este script está dividido en dos partes:

- La carga del modelo predictivo y los índices para poder factorizar las cadenas.

```
models = joblib.load('model_and_resources.pkl')
dtc = models[0]
factorize_urls_index = models[1]
factorize_extensions_index = models[2]
factorize_arguments_index = models[3]
factorize_aguments_type_index = models[4]
```

- Ejecución del servidor web en el puerto 8888 y creación de la ruta "/predict" que aceptará los parámetros (method,url y args) de la petición HTTP que se quiere catalogar. Generación de la instancia con los datos factorizados, llamada al modelo y retorno si es anómala o no.

```
app = Flask(__name__)
@app.route('/predict', methods=['GET','POST'])
def waf():
    #...
    #leer los parámetros de entrada
    #...
    to_predict = pd.DataFrame(columns=tfm_common.features)
    values=tfm_common.calculate_feature_values(method,url,args)
    res="anom"
    #...
    #factorizar los parámetros y llamar al modelo
    #...
    res = dtc.predict(to_predict)[0]
    #devolver resultado
    return "result="+res
#ejecutar servidor
app.run(host='0.0.0.0', port=8888)
```

4. Conclusiones

Una vez realizado el estudio y el caso práctico se ha llegado a las siguientes conclusiones:

- Es muy importante tener conocimientos sobre el problema que se quiere resolver.
- El preprocesado de las características es muy importante para obtener un modelo predictivo ágil y efectivo.
- Los resultados están relacionados directamente con la cantidad y calidad de la muestra de entrenamiento.
- La selección de las características y del algoritmo depende del conjunto de datos y del problema que se pretende solucionar.
- Hay que realizar pruebas con diferentes algoritmos y con los mismos datos de entrenamiento. Una vez valorados los resultados se elegirá el algoritmo que mejor se haya comportado.
- Varios algoritmos pueden resolver el mismo problema.
- Es importante conocer como funcionan los algoritmos para poder ajustar sus parámetros y mejorar la creación del modelo, su complejidad y tasa se aciertos.
- El almacenamiento de un modelo favorece su reutilización y actualización. Por ejemplo:

  Se puede generar un modelo con 100GB en datos de entrenamiento en local con una máquina muy potente y utilizar el modelo generado de, por ejemplo, 1MB en un hosting compartido mucho más limitado.

Los objetivos marcados en el punto 1.2 de este trabajo se han alcanzado satisfactoriamente a lo largo del desarrollo de este trabajo.

Se ha generado un modelo predictivo de clasificación[20] con un porcentaje de aciertos del 98,86%. El árbol de decisión fue entrenado con el 85% de las instancias del dataset y verificado con el 15% restante. El modelo predictivo se generó en ~0,5s y realizó la predicción de clasificación de 14.560 instancias de test en ~0,05s, que hace una media de ~290.000 predicciones por segundo.

Como trabajo futuro se podría realizar las siguientes tareas:

20 Desarrollado en el punto 3.1

- Probar el modelo con un caso real.
- Generar un modelo con un dataset que tenga marcas de tiempo y seguimiento de usuarios, para que el modelo tenga en cuenta el comportamiento de navegación.
- Ampliar los conocimientos de los algoritmos utilizados y sobre todo, los relacionados con Deep learning por sus aplicaciones y sus bueno resultados. En este trabajo ha sido el algoritmo con la peor tasa de aciertos, pero puede ser porque no están los datos correctamente normalizados y/o sea necesario dedicarle más tiempo para ajustar correctamente los parámetros del algoritmo.
- Ampliar conocimientos sobre la detección de anomalías con una sola clase en el dataset. Esa clase sería "normal", las peticiones anómalas se clasificarían si no se detectasen como normales.
- Mejorar el webservice desarrollado para que automáticamente detecte si el modelo ha cambiado y vuelva a recargar, y para optimizar y añadir nuevos webservices. Por ejemplo: que se pueda almacenar una instancia en un CSV para añadirla en el conjunto de instancias de entrenamiento.

5. Glosario

Ataque de día cero (en inglés zero-day attack o 0-day attack): Es un ataque contra una aplicación o sistema aprovechando vulnerabilidades que, por lo general, son desconocidas para los usuarios y fabricante del producto. Estos ataques son desconocidos hasta que finalmente se publican en algún foro público.

Conjunto de datos o **dataset**: Es un conjunto de datos observados (histórico) que se usa como base para entrenar al algoritmo de aprendizaje. El conjunto de datos se compone de instancias y esas instancias de características. Dependiendo del sistema de aprendizaje, las instancias pueden estar etiquetadas o no.

Modelo: Un modelo de aprendizaje automático es una representación matemática de un proceso del mundo real. Para generar este modelo es necesario proporcionar datos de entrenamiento a un algoritmo de aprendizaje.

Entrenamiento o **aprendizaje**: Es un proceso en el aprendizaje automático, donde se le pasa a un algoritmo, unos datos observados. El algoritmo de aprendizaje procesa estos datos en busca de patrones o reglas, para obtener un objetivo. Una vez realizado este proceso, se podrán hacer predicciones, aplicando los patrones observados en esta fase.

Características o **atributos**: Son variables individuales independientes que actúan como entrada en un sistema de aprendizaje automático.

Instancia: Conjunto de características que definen una observación. Vector de características.

Etiqueta: es la parte de la respuesta de una instancia. La etiqueta puede basarse en un hecho observado o en una predicción.

6. Bibliografía

[1] Murat Aydos. A Review on Cyber Security Datasets for Machine Learning Algorithms.2017

[2] Sara Althubiti, Xiaohong Yuan, Albert Esterline. Analyzing HTTP requests for web intrusion detection. Kennesaw State University. 2017

[3] Rafal Kozik, Michal Choraś, Rafal Renk, Witold Holubowicz. A Proposal of Algorithm for Web Applications Cyber Attack Detection. 2017

[4] A Self-learning Anomaly-Based Web Application Firewall

[5] Jason Brownlee, A Tour of Machine Learning Algorithms, 2013, https://machinelearningmastery.com/a-tour-of-machine-learning-algorithms/

[6] Carmen Torrano Giménez, Alejandro Pérez Villegas, Gonzalo Álvarez Marañón, HTTP DATASET CSIC 2010, 2010, http://www.isi.csic.es/dataset/

[7] Jain, A.K., R.P.W. Duin, and J. Mao, Statistical pattern recognition: A review., 2000

[8] Sumeet Dua and Xian Du, Data Mining andMachine Learningin Cybersecurity, 2011

[9] Devin Soni. Supervised vs. Unsupervised Learning https://towardsdatascience.com/supervised-vs-unsupervised-learning-14f68e32ea8d

[10] Wikipedia. Aprendizaje por refuerzo. https://es.wikipedia.org/wiki/Aprendizaje_por_refuerzo

[11] Daniil Korbut. Machine Learning Algorithms: Which One to Choose for Your Problem https://blog.statsbot.co/machine-learning-algorithms-183cc73197c

[12] Wikipedia. Semi-supervised learning https://en.wikipedia.org/wiki/Semi-supervised_learning

[13] Rushikesh Pupale. Support Vector Machines(SVM)—An Overview
https://towardsdatascience.com/https-medium-com-pupalerushikesh-svm-f4b42800e989

[14] Wikipedia. Support vector machine
https://en.wikipedia.org/wiki/Support_vector_machine

[15] Juan Ignacio Bagnato. Comprende Principal Component Analysis
http://www.aprendemachinelearning.com/comprende-principal-component-analysis/

[16] Wikipedia. Análisis de componentes principales
https://es.wikipedia.org/wiki/An%C3%A1lisis_de_componentes_principales

[17] Ciberamenazas y tendencias 2018. CCN CERT
https://www.ccn-cert.cni.es/informes/informes-ccn-cert-publicos/2856-ccn-cert-ia-09-18-ciberamenazas-y-tendencias-2018-resumen-ejecutivo-2018/file.html

[18] The OWASP Foundation. Top 10 2010-Main
https://www.owasp.org/index.php/Top_10_2010-Main

[19] Wikipedia. Regresión logística
https://es.wikipedia.org/wiki/Regresi%C3%B3n_log%C3%ADstica

[20] Wikipedia. Deep learning
https://en.wikipedia.org/wiki/Deep_learning

[21] Juan Ignacio Bagnato. Aplicaciones del Machine Learning
http://www.aprendemachinelearning.com/aplicaciones-del-machine-learning/

[22] Marvin G. Soto. Inteligencia artificial (AI): Estado del arte…
https://planetachatbot.com/inteligencia-artificial-ai-estado-del-arte-67b325c7bbfc

[23] Wikipedia. Aprendizaje automático. Wikipedia.
https://es.wikipedia.org/wiki/Aprendizaje_autom%C3%A1tico

[24] Wikipedia. Machine learning. Wikipedia.
https://en.wikipedia.org/wiki/Machine_learning

[25] David Fumo. Types of Machine Learning Algorithms You Should Know.
https://towardsdatascience.com/types-of-machine-learning-algorithms-you-should-know-953a08248861

[26] Jason Brownlee. How to Perform Feature Selection With Machine Learning Data in Weka
https://machinelearningmastery.com/perform-feature-selection-machine-learning-data-weka/

[26] Wikipedia. Feature selection
https://en.wikipedia.org/wiki/Feature_selection

[27] Wikipedia. Matriz de confusión
https://es.wikipedia.org/wiki/Matriz_de_confusi%C3%B3n

7. Anexos

7.1. Sesgo y varianza en el aprendizaje supervisado

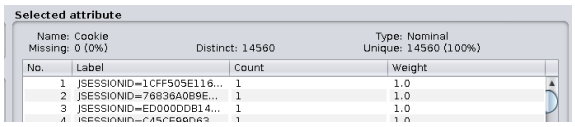
En un modelo de aprendizaje supervisado tiene que haber un equilibrio entre el sesgo y la varianza.

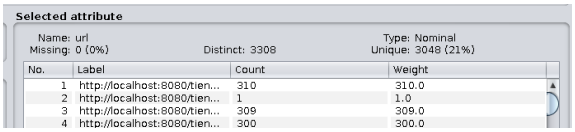
El **sesgo** de un modelo es la diferencia del error entre la clasificación que debería ser (la real) y la que predice el modelo.

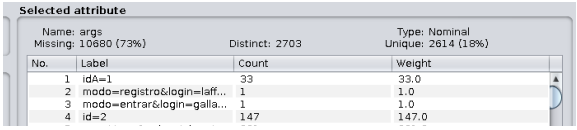
La **varianza** de un modelo es la variabilidad de la predicción si se estimase con un conjunto de datos diferentes.

Para un modelo, cuanto más simple mayor sesgo y cuando más complejo mayor varianza.

Por lo tanto, hay que tener en cuenta las características cookie, args y url porque son cadenas de texto muy variables (alto numero de valores únicos) puede aumentar la varianza del modelo y que no clasifique correctamente con un nuevo conjunto de datos aunque las instancias tengan valores muy parecidos.







A modo de ejemplo y utilizando Weka. Se han elegido dos instancias "anómalas" y se han creado 2 nuevas añadiendo añadiendo sólo un "0" al final:

```
GET   http://localhost:8080/tienda1/miembros/fotos.jsp.bak0
GET   http://localhost:8080/tienda1/publico/anadir.jsp?
idA=3&nombre=Vino+Rioja&precio=85&cantidad=76&B1=A%F1adir+al+carrito0
```

Cuando clasificamos estas instancias utilizando Weka, no las clasifica correctamente, es decir como anómalas, demostrando así que el modelo es muy sensible a pequeños cambios del conjunto de datos de entrenamiento.

7.2. Código fuente Python

7.2.1. Funciones comunes (tfm_common.py)

```
from urllib.parse import urlparse, parse_qsl
from validate_email import validate_email
import re
import sys

features=['feature-number-special-chars-arguments', 'feature-request-length',
          'feature-arguments-names', 'feature-arguments-values-length',
          'feature-arguments-values-type', 'feature-extension-name-length',
          'feature-extension-name', 'feature-url-path']

def args_extractor(args):
    params = parse_qsl(args,encoding='ISO-8859-1',keep_blank_values=True)
    return params

def special_chars_counter(s,is_path=False,is_args=False):
    p='\+|\*|=|\^|<|>|%'
    p= p + '|\(|\|\"|\'|\~|:'
    r= re.findall(p,s)
    return r

def get_content_type(content):
    content = content.replace(' ','')
    if content.isnumeric():
```

```
        return '[NUMERIC]'
    elif content.isalpha():
        return '[ALPHA]'
    elif content.isalnum():
        return '[ALNUM]'
    elif validate_email(content):
        return '[EMAIL]'
    elif content.isprintable():

        return '[PRINTABLE]'
    else:
        return '[OTHER]'



def calculate_feature_values(method,url,args):
    row=dict()
    row['feature-request-length']=len(method+url+args)
    url_obj=urlparse(url)
    get_args= args_extractor(url_obj.query)
    n_args=len(get_args)
    post_args = args_extractor(args)
    n_args=n_args + len(post_args)
    row['feature-number-arguments']=n_args
    try:
        dot_pos=url_obj.path.index('.')
        if(dot_pos):
            row['feature-extension-name']= url_obj.path[dot_pos + 1:]
            row['feature-url-path']= url_obj.scheme+ '://'+url_obj.netloc
+url_obj.path[:dot_pos]
    except ValueError:
        row['feature-extension-name']=''
        row['feature-url-path']=  url_obj.scheme+ '://'+url_obj.netloc +url_obj.path


    row['feature-url-path-length']=len(row['feature-url-path'])
    row['feature-extension-name-length']=len(row['feature-extension-name'])
    feature_list_special_chars_path= special_chars_counter(url_obj.path,is_path=True)
    feature_list_special_chars_extension = special_chars_counter(row['feature-extension-
name'],is_path=False)
    args_sorted_names='';
    args_sorted_content='';
    args_sorted_content_type='';
```

```
    for x in (post_args+ get_args):
        args_sorted_names=args_sorted_names + str(x[0])
        args_sorted_content=args_sorted_content + str(x[1])
        args_sorted_content_type=args_sorted_content_type+get_content_type(str(x[1]))

    feature_list_special_chars_arguments =
special_chars_counter(args_sorted_content,is_args=True)
    row['feature-arguments-names']=args_sorted_names
    row['feature-arguments-values-type']=args_sorted_content_type
    row['feature-arguments-values-length']=len(args_sorted_content_type)
    row['feature-number-special-chars-path']= len(feature_list_special_chars_path)
    row['feature-number-special-chars-extension']=
len(feature_list_special_chars_extension)
    row['feature-number-special-chars-arguments']=
len(feature_list_special_chars_arguments)

    return row

def get_factorized_key(index,key):
    try:
        return index.get_loc(key)
    except KeyError:
        return sys.maxsize
```

7.2.2. Conversión a csv y añadir características (create_csv.py)

```
import csv
import pandas as pd
import os
from sklearn.model_selection import train_test_split
import tfm_common

def convert_to_csv(dataset_file, csv_file,label):
    print("Convert file ", dataset_file, " to CSV ", csv_file)
    n_lines = 0
    fieldnames = ['method', 'url', 'Accept-Encoding', 'Cookie',
                  'Accept-Language', 'User-Agent', 'Cache-control',
                  'Connection', 'Accept', 'Accept-Charset', 'Pragma',
                  'Host', 'Content-Type', 'Content-Length', 'args',
```

```
                'feature-url-path', 'feature-url-path-length',
                'feature-extension-name','feature-extension-name-length',
                'feature-request-length','feature-arguments-values-length',
                'feature-arguments-names',
                'feature-arguments-values-type',
                'feature-number-arguments',
                'feature-number-special-chars-path',
                'feature-number-special-chars-extension',
                'feature-number-special-chars-arguments',
                'label']
    with open(dataset_file) as f:
        with open(csv_file, 'w') as csvf:
            csv_writer = csv.DictWriter(csvf, fieldnames=fieldnames,
                                        quoting=csv.QUOTE_ALL)
            csv_writer.writeheader()
            for line in f:
                if line.startswith(("POST", "GET","PUT")):
                    # Inicio Request
                    request_length=len(line)
                    splt = line.rstrip('\r\n').split(" ")
                    n_lines = n_lines + 1
                    row = {
                       'method': splt[0],
                       'url': splt[1]
                    }
                    row["args"]=''
                    for inner_line in f:
                        request_length += len(inner_line)
                        splt_inner = inner_line.rstrip('\r\n').split(":",1)
                        if len(splt_inner) == 2:
                            row[splt_inner[0].strip()] = splt_inner[1].strip()
                        elif len(splt_inner[0]) == 0 :
                            next_line=next(f).rstrip('\r\n')
                            if len(next_line) != 0 :
                                row["args"] = next_line
                            break
                    features =
tfm_common.calculate_feature_values(row["method"],row["url"],row["args"])
                    row.update(features)

                    row['label']=label
```

```
                    csv_writer.writerow(row)


convert_to_csv('anomalousTrafficTest.txt','anomalousTrafficTest.csv','anom')
convert_to_csv('normalTrafficTest.txt','normalTrafficTest.csv','norm')
convert_to_csv('normalTrafficTraining.txt','normalTrafficTraining.csv','norm')

#merge CSV
df1 = pd.read_csv('anomalousTrafficTest.csv')
df2 = pd.read_csv('normalTrafficTest.csv')
df3 = pd.read_csv('normalTrafficTraining.csv')
fulldf = pd.concat([df1, df2, df3], axis=0, join='inner', ignore_index=True)
fulldf.reset_index()
fulldf.to_csv('dataset-full.csv',index=False,quoting=csv.QUOTE_ALL)

max=len(fulldf)
print ('Full Dataframe rows: %i'%(max))

sampledf = fulldf.sample(frac=0.15)
sampledf.to_csv('dataset-15%-sample.csv',index=False,quoting=csv.QUOTE_ALL)

X_train_all, X_test_all, y_train, y_test = train_test_split(fulldf, fulldf['label'],
test_size=0.15,train_size=0.85)

X_train_all.to_csv('dataset-training-85%.csv',index=False,quoting=csv.QUOTE_ALL)
X_test_all.to_csv('dataset-test-15%.csv',index=False,quoting=csv.QUOTE_ALL)

os.remove('anomalousTrafficTest.csv')
os.remove('normalTrafficTest.csv')
os.remove('normalTrafficTraining.csv')
```

7.2.3. Creación del modelo de aprendizaje supervisado (model.py)

```
import pandas as pd
import numpy as np
from sklearn import tree
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.externals import joblib
import time
import sys
```

```
import tfm_common

from sklearn.ensemble import RandomForestClassifier

from sklearn.linear_model import LogisticRegression

from sklearn.linear_model import SGDClassifier

from sklearn.naive_bayes import GaussianNB


def printConfusionMatrix(classifierName,duration1, duration2, tn, fp, fn, tp):

    print('\n%s\n======\nTime taken to build model: %.3fs\nTime taken to test model on
test split: %.3fs'%(classifierName,duration1,duration2))

    total=tn + fn +fp + tp

    ac=(tn+tp)/total*100

    print('\nCorrectly Classified Instances:   %i\t\t%.3f%%'%(tn+tp,ac))

    er=(fn+fp)/total*100

    print('Incorrectly Classified Instances: %i\t\t%.3f%%'%(fn+fp,er))

    print('\nTotal Instances: %i'%(total))

    print('\nConfusion Matrix')

    print('TN:  %i\tFP:  %i'%(tn,fp))

    print('FN:  %i\tTP:  %i'%(fn,tp))

    print('N (Negative): Normal request\nP (Positive): Anomaly request\n')

    sys.stdout.flush()


start = time.time()

X_train_all_features= pd.read_csv("dataset-training-85%.csv")

y_train=X_train_all_features['label']

X_test_all_features= pd.read_csv("dataset-test-15%.csv")

y_test=X_test_all_features['label']



X_train_all_features.fillna('',inplace=True)

X_test_all_features.fillna('',inplace=True)


labels1, factorize_arguments_index = pd.factorize(X_train_all_features['feature-
arguments-names'],na_sentinel=sys.maxsize)

X_train_all_features['feature-arguments-names']=labels1

labels2, factorize_urls_index = pd.factorize(X_train_all_features['feature-url-
path'],na_sentinel=sys.maxsize)

X_train_all_features['feature-url-path']=labels2

labels3, factorize_extensions_index = pd.factorize(X_train_all_features['feature-
extension-name'],na_sentinel=sys.maxsize)

X_train_all_features['feature-extension-name']=labels3

labels4, factorize_aguments_type_index = pd.factorize(X_train_all_features['feature-
arguments-values-type'],na_sentinel=sys.maxsize)
```

```
X_train_all_features['feature-arguments-values-type']=labels4



X_train = X_train_all_features[tfm_common.features]

X_test = X_test_all_features[tfm_common.features]



for index, row in X_test_all_features.iterrows():

    old_value_1=X_test.at[index,'feature-arguments-names']

    old_value_2=X_test.at[index,'feature-url-path']

    old_value_3=X_test.at[index,'feature-extension-name']

    old_value_4=X_test.at[index,'feature-arguments-values-type']

    X_test.at[index,'feature-arguments-
names']=tfm_common.get_factorized_key(factorize_arguments_index,old_value_1)

    X_test.at[index,'feature-url-
path']=tfm_common.get_factorized_key(factorize_urls_index,old_value_2)

    X_test.at[index,'feature-extension-
name']=tfm_common.get_factorized_key(factorize_extensions_index,old_value_3)

    X_test.at[index,'feature-arguments-values-
type']=tfm_common.get_factorized_key(factorize_aguments_type_index,old_value_4)


confusion_matrix_labels=['norm','anom']

duration= time.time() - start

print('\nTime to prepare the factorized datasets : %.3fs'%(duration))


start = time.time()

dtc =
tree.DecisionTreeClassifier(criterion='entropy',presort=True,min_samples_split=2,random_
state=0)

dtc.fit(X_train, y_train)

duration1= time.time() - start


start = time.time()

y_pred = dtc.predict(X_test)

y_pred = y_pred.astype(str)

con_matrix=confusion_matrix(y_test, y_pred,labels=confusion_matrix_labels)

tn, fp, fn, tp = con_matrix.ravel()

duration2= time.time() - start

printConfusionMatrix('DecisionTreeClassifier',duration1,duration2,tn, fp, fn, tp)


tree.export_graphviz(dtc,out_file='tree-ml4-
v10_1.dot',feature_names=tfm_common.features,max_depth=5)
```

```
to_serialize=[
    dtc,
    factorize_urls_index,
    factorize_extensions_index,
    factorize_arguments_index,
    factorize_aguments_type_index,
    ]
joblib.dump(to_serialize, 'model_and_resources.pkl')
```

7.2.4. Webservice del modelo creado (webservice.py)

```
from flask import Flask,request
from sklearn.externals import joblib
import tfm_common
import pandas as pd

models = joblib.load('model_and_resources.pkl')

dtc = models[0]
factorize_urls_index = models[1]
factorize_extensions_index = models[2]
factorize_arguments_index = models[3]
factorize_aguments_type_index = models[4]

print('Model and resources loaded')

app = Flask(__name__)

@app.route('/predict', methods=['GET','POST'])
def waf():
    method = request.args.get('method','GET').upper()
    url = request.args.get('url','')
    args = request.args.get('args','')
    if request.method == 'POST':
        method = request.form.get('method','GET').upper()
        url = request.form.get('url','')
        args = request.form.get('args','')

    to_predict = pd.DataFrame(columns=tfm_common.features)
    values=tfm_common.calculate_feature_values(method,url,args)
```

```
    res="anom"


    values['feature-url-path'] =
tfm_common.get_factorized_key(factorize_urls_index,values['feature-url-path'])
    values['feature-extension-name'] =
tfm_common.get_factorized_key(factorize_extensions_index,values['feature-extension-
name'])
    values['feature-arguments-names'] =
tfm_common.get_factorized_key(factorize_arguments_index,values['feature-arguments-
names'])
    values['feature-arguments-values-type'] =
tfm_common.get_factorized_key(factorize_aguments_type_index,values['feature-arguments-
values-type'])


    to_predict=to_predict.append(values,ignore_index=True)[tfm_common.features]
    res = dtc.predict(to_predict)[0]


    return 'result='+res


app.run(host='0.0.0.0', port=8888,debug=True)
```

7.2.5. Tests (test_model_and_webservice.py)

```
# -*- coding: utf-8 -*-
from sklearn.externals import joblib
import tfm_common
import pandas as pd
import numpy as np
from sklearn.metrics import confusion_matrix
import urllib.request
import urllib.parse


models = joblib.load('model_and_resources.pkl')


dtc = models[0]
factorize_urls_index = models[1]
factorize_extensions_index = models[2]
factorize_arguments_index = models[3]
factorize_aguments_type_index = models[4]


test_requests=A = np.array([
    ['GET','http://localhost:8080/tienda1/index.jsp','','norm'],
```

```
    ['GET','http://localhost:8080/tienda1/index.jsp?test','','anom'],
    ['GET','http://localhost:8080/','test05','anom'],

['POST','http://localhost:8080/tienda1/publico/anadir.jsp','id=2&nombre=Vino+Rioja&preci
o=39&cantidad=69&B1=A%F1adir+al+carritoany%250D%250ASet-cookie%253A%2BTamper
%253D57652055672348762 35','anom'],
    ['POST','http://localhost:8080/tienda1/publico/caracteristicas.jsp','id=2','norm'],
    ['POST','http://localhost:8080/tienda1/publico/caracteristicas.jsp','id=aa','anom']
])

y_test=test_requests[:,3]
to_predict = pd.DataFrame(columns=tfm_common.features)

print('\nTesting Model')
for req in test_requests:
    values=tfm_common.calculate_feature_values(req[0],req[1],req[2])
    values['feature-url-path'] =
tfm_common.get_factorized_key(factorize_urls_index,values['feature-url-path'])
    values['feature-extension-name'] =
tfm_common.get_factorized_key(factorize_extensions_index,values['feature-extension-
name'])
    values['feature-arguments-names'] =
tfm_common.get_factorized_key(factorize_arguments_index,values['feature-arguments-
names'])
    values['feature-arguments-values-type'] =
tfm_common.get_factorized_key(factorize_aguments_type_index,values['feature-arguments-
values-type'])
    to_predict=to_predict.append(values,ignore_index=True)[tfm_common.features]

y_pred = dtc.predict(to_predict)
confusion_matrix_labels=['norm','anom']
con_matrix=confusion_matrix(y_test, y_pred,labels=confusion_matrix_labels)

tn, fp, fn, tp = con_matrix.ravel()

total=tn + fn +fp + tp
ac=(tn+tp)/total*100
print('Correctly Classified Instances:   %i\t\t%.3f%%'%(tn+tp,ac))
er=(fn+fp)/total*100
print('Incorrectly Classified Instances: %i\t\t%.3f%%'%(fn+fp,er))
print('Total Instances: %i'%(total))

print('\nTesting webservice. Webservice must be started')
base_url='http://127.0.0.1:8888/predict'
```

```
for item in test_requests:
        url=base_url + '?method='+ urllib.parse.quote(item[0], safe='') +
"&url="+urllib.parse.quote(item[1], safe='') + "&args="+ urllib.parse.quote(item[2],
safe='')
        #print(url)
        req = urllib.request.Request(url)
        try:
            response = urllib.request.urlopen(req)
            result = response.read().decode()
            if(result == 'result='+item[3]):
                print('Response OK %s. Recived: %s'%(item[3],result))
            else:
                print('Resonse ERROR must be %s. Recived: %s '%(item[3],result))
        except urllib.error.URLError as e:
            print(e.reason)
```

7.3. Árbol de decisión. max_depth=5