

**Universidad ORT Uruguay
Facultad de Ingeniería**

**Análisis y desarrollo de modelos
predictivos con redes neuronales para
Web Application Firewall**

**Entregado como requisito para la obtención del
título de Máster en Big Data**

**Deborah Biardo - 234817
Guzmán González - 146966
Sabrina Lanzotti - 201820**

Tutor: Sergio Yovine

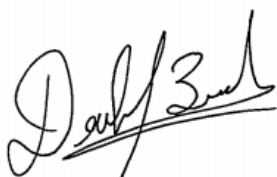
2020

Declaración de Autoría

Nosotros, Deborah Biardo, Guzman Gonzalez y Sabrina Lanzotti, declaramos que el trabajo que se presenta en esa obra es de nuestra propia mano.

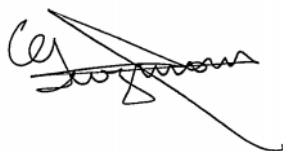
Podemos asegurar que:

- La obra fue producida en su totalidad mientras realizábamos la Tesis;
- Cuando hemos consultado el trabajo publicado por otros, lo hemos atribuido con claridad;
- Cuando hemos citado obras de otros, hemos indicado las fuentes. Con excepción de estas citas, la obra es enteramente nuestra;
- En la obra, hemos acusado recibo de las ayudas recibidas;
- Cuando la obra se basa en trabajo realizado conjuntamente con otros, hemos explicado claramente qué fue contribuido por otros, y qué fue contribuido por nosotros;
- Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto donde se han realizado las aclaraciones correspondientes.



Deborah Biardo

24 de setiembre de 2020



Guzmán González

24 de setiembre de 2020



Sabrina Lanzotti

24 de setiembre de 2020

Agradecimientos

En primer lugar queremos agradecer a todas las personas que nos brindaron su apoyo durante el transcurso de la Maestría.

Agradecemos a Sergio Yovine y Ramiro Visca, quienes nos acompañaron, ofrecieron su experiencia y nos guiaron durante el desarrollo de este trabajo.

Particularmente queremos dar gracias a Atos y Quanam quienes oficiaron de sponsors para la realización de esta Maestría.

Finalmente el agradecimiento más importante va dedicado a nuestras familias y amigos, quienes nos apoyaron incondicionalmente durante la tesis. En especial a Claudio Barreto, Sofía Belén Barreto, Mariella Mirenda, Ana María Filardi, Carmelo Lanzotti, Gonzalo Lanzotti, Federico Machado, Sandra Quintana, Adan Gonzalez, Florencia Barrios, Jennifer Lopez y Alejandro Bordagorria.

Abstract

Los datos de organismos estatales y demás organizaciones son valiosos para la realización de múltiples investigaciones, sin embargo, muchas veces poseen información personal que debe ser protegida, y que por ende dificulta la tarea legal de brindarlos para que estén a disposición de todos.

Por este motivo, anonimizar los datos es una tarea de vital importancia. Esto implica privatizarlos sin restarles poder informativo al momento de ser analizados.

El presente trabajo tiene como objetivo el desarrollo de un modelo predictivo basado en *Bag of Words (BoW)* y redes neuronales, con el fin de realizar una tarea de ciberseguridad predictiva a partir de datos no privatizados, que permita la clasificación de *URIs* como normales o anómalas.

El resultado servirá de punto de partida para poder comparar posteriormente contra modelos obtenidos a partir de datos privatizados de *logs*.

Como antecedente, se tomaron otras investigaciones referentes a esta temática.

Utilizando conjuntos de datos empleados de tales investigaciones y provenientes de otras fuentes, se realizó un tratamiento de estos para la obtención de un conjunto de *URIs*.

A partir del conocimiento de dominio experto, se construyó un *BoW* para cada *URI* que contiene la frecuencia de cada *key* experta asociada a ataques informáticos. Luego, mediante el análisis de las *URIs*, algunas *keys* específicas fueron añadidas.

Se desarrollaron diversas arquitecturas del estilo *Multilayer Perceptron (MLP)* que fueron entrenadas para cada conjunto de datos por separado, y luego para un único *dataset* creado a partir de la unión de ellos.

Se aplicaron distintos criterios de *thresholds* para la clasificación de las *URIs* y a los efectos de evaluar el modelo se emplearon técnicas tales como la matriz de confusión,

el análisis y comparación de las métricas de *True Positive Rate (TPR)* y *True Negative Rate (TNR)*, (priorizando su balanceo y *performance*) y las curvas de aprendizaje.

Con la intención de mejorar la capacidad predictiva del modelo se aplicaron técnicas de *Term Frequency - Inverse Document Frequency (TF-IDF)*, primero utilizando las *keys* expertas identificadas como *tokens*, y luego mediante *feature extraction*, obteniendo los *tokens* en forma dinámica, en modalidad de unigrama, bigrama y mixta.

El mejor resultado obtenido según las métricas mencionadas fue el del modelo de *TF-IDF: Feature Extraction Unigrama*, el cual resultó satisfactorio.

Finalmente se aplicaron técnicas de *ensemble* tales como *stacking* y *majority voting*, pero no generaron mejores resultados que los del modelo *TF-IDF: Feature Extraction Unigrama*.

Palabras Claves: *Machine Learning; Deep Learning; Cybersecurity; ModSecurity; OWASP; WAF; Web Application Firewalls; Anomaly Detection; n-grams; Ensemble; Classification.*

Contenido

1. Introducción.....	9
2. Tratamiento de datos.....	13
2.1 URIs.....	13
2.1.1 Sintaxis de una URI.....	13
2.2 Datasets y tratamiento de datos.....	15
2.2.1 Good Queries/ Bad Queries.....	15
2.2.2 CSIC (2010).....	17
2.2.3 PKDD	20
2.2.4 Bag of words.....	22
3. Evaluación de los modelos.....	26
3.1 Matriz de confusión.....	26
3.2 Análisis de Curvas de Aprendizaje	27
3.3 Definición del threshold y criterios elegidos.....	27
4. Estudio preliminar de modelos	29
4.1 Modelo Good/Bad Queries	29
4.2 Modelo CSIC.....	33
4.3 Modelo PKDD	35
5. Unificación de datasets	38
5.1 Exploración del corpus con el bag of words de Keys Expertas	39
5.1.1 Frecuencia de las Keys Expertas en el Corpus	39
5.1.2 Correlación de las Keys Expertas	44
6. Desarrollo de modelos sobre datasets unificados.....	47
6.1 Modelo Keys Expertas	47
6.1.1 Arquitectura del Modelo.....	47
6.1.2 Análisis de la arquitectura y entrenamiento del modelo.....	48
6.2 N-Grams	53
6.2.1 TF-IDF: Term frequency - inverse document frequency	53
7. Trabajo de relevamiento	70
8. Conclusiones finales	79
9. Trabajo a futuro	81

10. Referencias Bibliográficas	82
Anexo 1	84
Anexo 2	86

1. Introducción

Los datos que poseen los organismos estatales y demás organizaciones se han vuelto cada vez más, un activo valioso como insumo esencial para la extracción de información, mediante la elaboración de modelos predictivos que aplican técnicas de aprendizaje automático, en proyectos de investigación e innovación científica y tecnológica.

Sin embargo, debido al hecho de que tales datos poseen información de carácter personal relativa a las propias organizaciones o a terceros, brindarlos no es tarea sencilla.

De hecho, los datos publicados deben estar sujetos al cumplimiento de la legislación vigente sobre la protección de privacidad de los datos. En el caso de Uruguay, estos aspectos están contemplados en la Ley N° 18.331 Protección de Datos Personales y acción de Habeas Data. En particular, esta ley define expresamente el proceso de disociación como todo tratamiento de datos personales de manera que la información obtenida no pueda vincularse a una persona determinada o determinable.

Surge entonces la necesidad de garantizar la privacidad de los datos sin limitar su utilidad. Esto significa hacerlos disponibles a determinados actores, públicos o privados, en garantía de anonimidad, pero conservando su naturaleza de insumo para el desarrollo de análisis e investigaciones.

Esto significa que las técnicas de anonimización robustas de los datos permitirían que estos pudieran brindarse entre las organizaciones con el fin de reducir costos, mejorar la eficiencia, generar desarrollo en materia de prevención y tratamiento de enfermedades, analizar *logs* con datos personales de terceros, prevenir fraudes, etc.

El objetivo del presente trabajo es construir un modelo predictivo a partir de datos no privatizados que sirva como línea base para comparar la eventual pérdida de poder predictivo de datos privatizados, entendiéndose que la privatización de estos no forma parte de este proyecto.

Por lo tanto, el presente trabajo servirá de punto de partida para poder comparar posteriormente contra modelos obtenidos a partir de datos privatizados de *logs* de sistemas informáticos provenientes de *firewalls* de aplicaciones *web* (*WAF*), provistos por CERTuy de AGESIC, que contienen información personal de terceros. Esto permitirá la evaluación de diferentes mecanismos de privatización de datos.

Dichas técnicas de privatización están siendo estudiadas en el contexto de los proyectos ANII FSDA_1_2018_1_154419 y FMV_1_2019_1_155913 a los cuales este trabajo contribuye, y cuyo objetivo principal es el desarrollo de técnicas de privatización de datos y su evaluación mediante *logs*.

En el caso de AGESIC, a modo de poder impedir ataques contra aplicaciones *web* se despliega y configura un *firewall* para aplicaciones *web*, en este caso *ModSecurity*, que analiza todas las peticiones y respuestas en función de un conjunto de reglas predefinidas. Si el análisis da como válida la petición se procede con su normal funcionamiento, de lo contrario se disparan diferentes acciones.

Por otro lado, *OWASP* (*Open Web Application Security Project*) es un proyecto cuya finalidad es ayudar a combatir ataques informáticos y por consiguiente que las aplicaciones sean seguras.

El *OWASP ModSecurity CRS* o *Core Rule Set* es un conjunto de reglas genéricas de detección de ataques predefinidas para ser usadas en conjunto con *ModSecurity*.

Este proyecto publica cada tres años un documento de los diez riesgos de seguridad más significativos.

Dentro del top 10 *OWASP* que se puede visualizar en el Anexo 1, se consideran los siguientes ataques: *SQL injection*, *buffer overflow*, *information gathering*, *files disclosure*, *CRLF injection*, *Cross-Site Scripting (XSS)*, *server side include*, *parameter tampering*, *LDAP Injection*, *XPATH Injection*, *SSI attacks*, etc.

Por un lado, la motivación del presente trabajo se centra en el objetivo de lograr obtener mejores resultados que los obtenidos por la investigación del Instituto de Computación y la Facultad de Ingeniería, publicado en el año 2018: *Machine learning-assisted virtual patching of web applications*. [7]

El fin buscado de esta investigación, es mejorar la *performance* que tienen herramientas que cumplen con la función de detectar ataques, como es el caso del *firewall WAF* que utiliza el protocolo *HTTP*.

Se eligió el trabajo realizado en [7] como un punto de partida porque abarca tres problemas conocidos que tienen los *firewalls tradicionales* para la detección de ataques. Uno de los problemas más conocido es que el *ModSecurity* utiliza un conjunto de reglas rígidas y estáticas para determinar si un *request* tiene apariencia de ataque o no. En base a esta información, se procesa el mismo de forma diferente, generando como consecuencia no deseada un alto número de falsos positivos (es decir que se identifica un *request* como anómalo cuando en realidad es normal). El alto número de falsos positivos que tiene el *WAF* es un error costoso en ciberseguridad porque en muchas ocasiones puede derivar en una interrupción del servicio.

Otra de las limitaciones de este tipo de *firewall* es su incapacidad de detectar lo que se conoce como *zero days attacks*. Se definen así aquellos ataques considerados nuevos por no ser conocidos por los expertos. Por último, estos sistemas tienen como problema su alta complejidad en el mantenimiento de las reglas definidas para detectar anomalías del tráfico.

Estos tres problemas mencionados en [7] son los mismos que apuntalan esta investigación y se pretende resolver.

Adicionalmente, en [7] se plantean tres escenarios o preguntas disparadoras que utilizan para desarrollar toda su investigación y que han servido a este trabajo como un marco de referencia para realizar una comparación.

Escenario 1: Es posible construir un Sistema de detección de ataques utilizando un conjunto de datos de entrenamiento de otras aplicaciones *webs*.

Escenario 2: Es posible mejorar los resultados del *ModSecurity* de un *WAF* utilizando las técnicas *de machine learning*.

Escenario 3: Tratar de comprender el rendimiento de los métodos de aprendizaje automático en comparación con los obtenidos por el *ModSecurity* del *WAF*.

Los dos primeros escenarios fueron claves al momento de tomar las decisiones de diseño en esta investigación. El primer escenario, fue abordado cuando en la etapa de entrenamiento se definió utilizar datos de diferentes dominios o aplicaciones *web*. Por otro lado, el segundo parte de un intento de mejorar las necesidades que tiene la AGESIC que utiliza un *firewall WAF* para llevar su estrategia de ciberseguridad. Con el objetivo de contestar estas preguntas, [7] emplea dos enfoques para hacer frente a los escenarios: *One-Class Classification* y *Anomaly Detection* usando *n-gram*.

El objetivo principal de este trabajo es poder clasificar las *URIs* como legítimas o anómalas, para lo cual, a partir de los conjuntos de datos empleados se buscará desarrollar un modelo predictivo basado en *BoW* y redes neuronales.

Al evaluar los modelos desarrollados, se buscará que el total de *URIs* predichas correctamente como anómalas en el total de *URIs* anómalas reales, y el total de *URIs* predichas correctamente como normales en el total de tráfico normal, estén balanceados y tengan una buena *performance*.

Se ajustará la arquitectura de tales modelos en función de dicho objetivo.

2. Tratamiento de datos

2.1 URIs

El presente trabajo tiene como objeto de estudio la *URI*, definida en función del *RFC* 3986 relativo a las *URIs*.

Una *URI* proporciona un método simple y extensible para identificar un recurso, refiriéndose este a un documento electrónico, imagen, fuente de información, servicio, etc.

En el contexto de una *URI* el uso del término “*Identifier*” permite distinguir un recurso de todos los demás.

Por último, la uniformidad permite utilizar diferentes identificadores de recursos en el mismo contexto, aun cuando el mecanismo de acceso a los mismos difiere, lo cual posibilita la creación de nuevos tipos de identificadores sin afectar los existentes.

2.1.1 Sintaxis de una URI

Las *URIs* tienen una sintaxis que se compone de una secuencia ordenada y jerárquica de componentes, los cuales son: *schema*, *authority*, *path*, *query* y *fragment*.

El *schema* y el *path* son obligatorios para todas las *URIs*, en cambio los restantes componentes pueden encontrarse o no presentes. Vemos esto en la Ilustración 1.

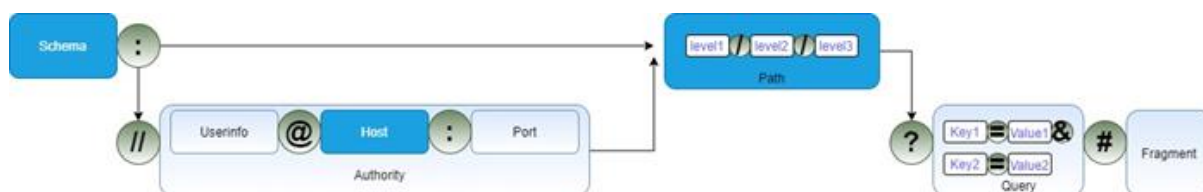


Ilustración 1. Estructura de una *URI*

Schema: Es la especificación para asignar identificadores dentro de ese esquema. Los esquemas se nombran de la siguiente manera: deben comenzar con una letra, seguido de otra combinación de letras, dígitos, signo de más, punto o signo de menos. La forma canónica de los nombres de los esquemas es en minúscula, aunque puede aceptar en forma equivalente mayúsculas. El *Schema* puede identificar el protocolo de acceso al recurso, por ejemplo *HTTP*.

El componente *Authority* está contenido por una barra doble ("//") y puede terminar con los siguientes caracteres "/", "?", o "#". Esta sintaxis genérica proporciona un medio común que permite distinguir los tres subcomponentes. Ellos son el *host* que es una autoridad basada en un nombre registrado o dirección de servidor, el puerto (*port*) y la información del usuario (*userinfo*), siendo estas dos últimas de carácter opcional.

El *userinfo* contiene el nombre del usuario e información específica del esquema para obtener autorización de acceso al recurso. El *port* es un número en decimales que se encuentra definido por el *Schema* utilizado, por ejemplo, el *Schema HTTP* tiene por *default* el puerto 80.

Finalmente, el subcomponente *host* se identifica por medio de un nombre registrado o de una dirección *IP*.

Path: Este componente contiene datos, generalmente organizados en forma secuencial y jerárquica separados por *slash* (/). Junto con los datos en el componente de *query* sirve para identificar un recurso dentro del alcance del *Schema* de *URI* y la *Authority* de denominación (en caso de que exista).

Query: Este componente contiene datos no jerárquicos que ayudan a identificar los recursos. Se inicia con el carácter "?", terminando con un *hashtag* (#) o porque la *URI* finaliza. Se debe tener cuidado porque ambos caracteres pueden encontrarse dentro del componente *query*. A menudo la forma de este componente es una secuencia par de la forma *key-value*.

Fragment: Este componente de la *URI* permite identificar indirectamente un recurso secundario que se encuentre relacionado con el recurso primario, así como también información adicional. Se utiliza para hacer referencia dentro de un contenido. Los caracteres “/” y “?” permiten identificar datos dentro de este componente.

Este trabajo se centra en los siguientes componentes de las *URIs*: **Path**, **Query** y **Fragment**. En la Ilustración 2 se muestra un diagrama del objeto de estudio.

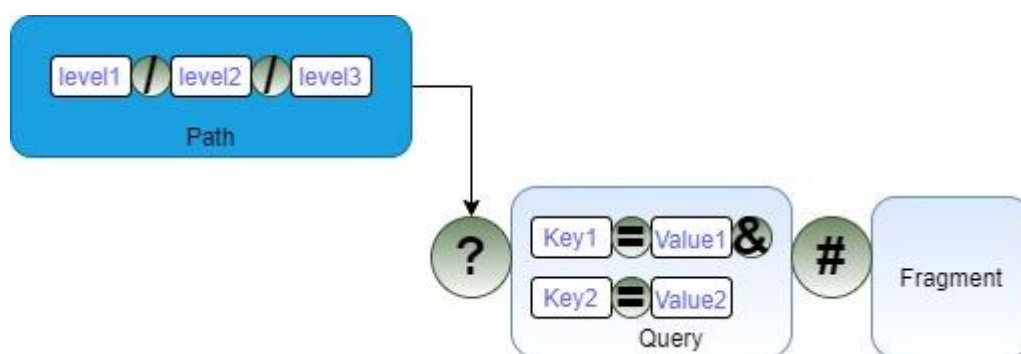


Ilustración 2. *Path*, *Query* y *Fragment* de una *URI*

2.2 Datasets y tratamiento de datos

A continuación, se describen los conjuntos de datos utilizados para el entrenamiento de los modelos:

2.2.1 Good Queries/ Bad Queries

El conjunto de datos se divide en dos *datasets* que contienen un registro de los recursos y parámetros de las *URIs* sin ataques (*Good Queries*) y con ataques (*Bad Queries*).

Dicho conjunto de datos fue disponibilizado en el año 2017 por Faizan Ahmad, *CEO* de FSecurity, en el marco de un trabajo publicado en *kdnuggets*.

Los *datasets* fueron elaborados en base al conjunto de datos *HTTP logs* que se encuentra en el sitio *web SecRepo* y contienen los logs de *request/response* con su respectiva metadata, sin estar etiquetados.

El proceso de identificación de los *request* anómalos se realizó en base al conocimiento experto. La documentación no especifica acerca de este proceso, ni tampoco acerca de la definición de ataques y criterios empleados.

Además, se complementaron estos datos con otros *datasets* obtenidos en repositorios de Github que contienen ataques XSS, *SQL Injection*, etc.

Se observa una muestra de estos datos en la Tabla 1.

goodqueries.txt
<code>/include/phpxd/phpxd.php?appconf[rootpath]=@rfiurl?appconf[rootpath]=@rfiurl?&cmd=id</code>
badqueries.txt
<code>/en-us/account/login.pl?login=ledgersmb_script_code_exec.nasl&script=-e print "content-type: text/plain\x0d\x0a\x0d\x0a";system(id)&action=logout</code>

Tabla 1. Datasets Good/Bad Queries

Transformaciones realizadas:

- Eliminación de *URIs* duplicadas y datos vacíos (o *NAs*)
- Creación de etiqueta, asignando el valor 1 si la *URI* corresponde al archivo `badqueries.txt` y 0 si corresponde al archivo `goodqueries.txt`

2.2.2 CSIC (2010)

El conjunto de datos utilizado fue desarrollado por el *Instituto de Seguridad de la Información* y publicado en el año 2010 por el *Consejo Nacional de Investigación de España*, con el objetivo de probar un sistema que facilite una aplicación *web* destinada a realizar compras online.

El motivo de los investigadores para desarrollar este conjunto de datos se centra en aportar a la comunidad una mejora al *dataset* ampliamente utilizado por DARPA, incluyendo datos más actualizados y ataques más reales.

El conjunto de datos se distribuye en tres *datasets*, uno para entrenamiento que solo contiene *requests* legítimas (no ataque) y otros dos de *test*, de los cuales uno contiene *requests* legítimas y el otro *requests* maliciosas.

El conjunto de datos *HTTP CSIC 2010* contiene un total de 61.000 *HTTP requests* de tráfico generado automáticamente, dirigido a una aplicación *web* de comercio electrónico, en donde los usuarios se registran y realizan compras. Del total de solicitudes, 36.000 corresponden al tráfico de solicitudes válidas y 25.000 corresponden a *requests* anómalos que pueden considerarse ataques.

Los ataques que contiene el *dataset* son generados utilizando las herramientas Paros y W3AF, y pertenecen a las siguientes categorías: *SQL injection*, *buffer overflow*, *information gathering*, *files disclosure*, *CRLF injection*, *XSS*, *server side include*, *parameter tampering*.

Los *requests* son catalogadas como:

Estáticos: Son solicitudes de recursos ocultos o inexistentes, por ejemplo: solicitudes que incluyen archivos obsoletos, *ID* de sesión en reescritura de *URL*, archivos de configuración, archivos predeterminados, etc.

Dinámicos: Son solicitudes que modifican argumentos válidos, por ejemplo: *SQL injection*, *CRLF injection*, *cross-site scripting (XSS)*, *buffer overflows*.

Irregulares no Intencionales: Son solicitudes que no tienen intención maliciosa pero no presentan la estructura esperada.

Se observó como característica en este conjunto de datos que para cada *request* del tipo *GET*, le sigue uno del tipo *PUT*. Sin embargo, analizando el conjunto de *requests* maliciosas se verificó que los ataques del método *PUT* son exactamente iguales a los del *GET*, lo que implica que en la práctica el *dataset* de *requests* maliciosas tiene 9580 registros duplicados.

Se observa una muestra de estos datos en la Tabla 2.

CSIC
<pre>GET http://localhost:8080/tienda1/publico/anadir.jsp?id=2&nombre=Jam%F3n+Ib%E9rico&precio=85&c antidad=%27%3B+DROP+TABLE+usuarios%3B+SELECT+*+FROM+datos+WHERE+nombre+LIKE+%2 7%25&B1=A%F1adir+al+carrito 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=B92A8B48B9008CD29F622A994E0F650D 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</pre>

```
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=AE29AEEBDE479D5E1A18B4108C8E3CE0
Content-Type: application/x-www-form-urlencoded
Connection: close
Content-Length: 146
id=2&nombre=Jam%F3n+Ib%E9rico&precio=85&cantidad=%27%3B+DROP+TABLE+usuarios%3B+SELECT+*+FROM+datos+WHERE+nombre+LIKE+%27%25&B1=A%F1adir+al+carrito
```

Tabla 2. Datasets CSIC

Transformaciones realizadas:

- Filtrado de las *HTTP requests* con método *PUT*
- Extracción del parámetro *URI* para los *HTTP requests* con método *GET*
- Extracción del parámetro *URI* y del body para los *HTTP requests* con método *POST*
- Eliminación del método, [http://localhost: 8080](http://localhost:8080), *HTTP/1.1* para cada *URI*.
- Eliminación de registros duplicados y vacíos (*NAs*)
- Creación de la etiqueta, asignando el valor 1 si la *URI* pertenece al archivo `csic_2010_anomalousTrafficTest.txt` y 0 si la *URI* corresponde a cualquiera de los otros dos archivos.

2.2.3 PKDD

Este *dataset* fue publicado en el año 2007 en el marco de un reto propuesto por *European Conference on Machine Learning (ECML)* y la *European Conference on Principles and Practice of Knowledge Discovery in Database (PKDD)*, para analizar el tráfico *web*.

El reto propuesto consistió en dos objetivos: clasificar las consultas de acuerdo a la clase que pertenecen en base al contexto y por otro lado identificar y aislar el patrón de ataque, para lo cual se proporcionó un *dataset* en formato *XML* de 50.000 *HTTP query logs*.

Un 20% de los *query* que fueron proporcionados se corresponden a distintos tipos de ataques independientes unos de otros.

Cada *HTTP query logs* propuesto está compuesto de tres partes: el contexto, la clase y la descripción de la consulta. El contexto contiene una descripción sobre el ambiente en el que se ejecuta el código e incluye la siguiente información: sistema operativo en el cual se ejecuta el servidor *web*, *HTTP Server targeted by the request*, si el servidor admite *XPATH*, y si hay una base *LDAP* y *SQL* en el servidor *web*.

El conjunto de datos está distribuido en dos *datasets*, uno para *train* y otro para *test*. Sin embargo, en este caso las *requests* legítimas y las maliciosas no se encuentran separadas.

A continuación, se presentan ejemplos de los dos archivos con los datos sin procesar para el *Dataset PKDD* en la Tabla 3.

PKDD

Start - Id: 40994
class: SqlInjection
POST /IGwd-J6hRWLOSuj/3ebeswrws3c3b1etNRU/rxam56rqadee.aspx? HTTP/1.0
Content-Length: 398
Content-Language: dmmd,AO,eiE4tl
Content-Encoding: identity
Content-MD5: cXRhaWJwNGRvZGxvYm9uMA==
Content-Type: application/x-www-form-urlencoded
Last-Modified: Wed, 21 Dec 05 04:54:17 GMT
Host: 51.166.4.167
Connection: sotgse
Accept: */*
Accept-Charset: iso-8859-8-i;q=0.2, windows-1254, utf-7;q=0.8, windows-1254, x-mac-icelandic;q=0.5
Accept-Encoding: *
Accept-Language: *,q=0.5
Cache-Control: no-cache
Client-ip: 248.121.203.33
Cookie: untrhuslrdea=c085tioe3htser;adctx=ge;THuudkBT@L=wkiusrIb;arhaastuNv=eJLP9PX_F-f9
Cookie2: \$Version="7"
Date: Fri, 26 Dec 08 17:28:38 CET
If-None-Match: *
Max-Forwards: 6285
Authorization: NTLM bG5lYXFheWV0YWZ5dGVpaG1lMHJneHNIYzJzY3duZXdkYXZyZjFubw==
Referer: /iLt0/ntygrC/bsre.tar.gz
Trailer: If-None-Match
User-Agent: Mozilla/2.5 (compatible; Konqueror/2.8; Linux i386; Afo8o; tsmseLe)
UA-Pixels: 892x2342
Via: 6.6 239.119.208.26:7, sru/3.0 www.ona8s.png, 6.4 www.eemi.jpg
Transfer-Encoding: gzip
cbteewjlanitt=ezt.6qJoliNN&aieTsoa=execp_regwrite
'HKEY_LOCAL_MACHINE','SOFTWARE\Microsoft\MSSQLServer\Client\ConnectTo','Nm5xfai','REG_SZ','DBMSSOC
N,hackersip,80'&trAfntrterap=s3A-
gJQzC&cwEux=260&esoch=8287356386&w6shisklht0n=c8kmaxsvunrame&rmosh4ttpaeuub0=731469&rd6=110
4205&ogFt5ERloian=eQh&ih0hseerecitmh=099377&s0uitGriedoo=heerh&nsrht7nxtl1Adr=0018
End - Id: 40994

Tabla 3. Datasets *PKDD*

Se identificó si los *requests* eran legítimos o no en base a un campo de nombre “*class*” que clasifica como “*Valid*” a los *requests* legítimos y para el caso contrario como una de las siguientes siete clases: *Cross-Site Scripting (XSS)*, *SQL Injection*, *LDAP Injection*, *XPATH Injection*, *Path traversal*, *Command execution* y *SSI attacks*.

Transformaciones realizadas:

Se identificó que en algunos casos, parte de los parámetros de las *URIs* se encontraban al final de cada *request*. Esto implicó agregar dicha información a cada *URI* cuando correspondiera.

- En todas las *URIs* se eliminó información adicional que no formaba parte de la misma
- Se realizó la eliminación de registros duplicados y vacíos (*NAs*)
- Creación de la etiqueta, asignando el valor 1 para las *URIs* maliciosas y 0 para las legítimas

Inicialmente se decidió comenzar el estudio para un conjunto reducido de *URIs*, seleccionando en cada *dataset*, únicamente aquellas que estuvieran compuestas por recursos y parámetros.

2.2.4 Bag of words

Es un modelo que sirve para convertir texto arbitrario en vectores de longitud fija en donde se contabiliza la cantidad de veces que cada palabra aparece, sin tener en cuenta la posición de las mismas. Las ventajas que representa dicha representación es su facilidad de implementación y las posibilidades que ofrece al poder extraer características del texto que pueden ser usadas en algoritmos de aprendizaje automático.

En el presente trabajo se construyó un diccionario compuesto por *keys* expertas para la detección de *URIs* maliciosas, tomándose como base la investigación realizada por [7].

En dicha investigación se elaboró un diccionario que fue aplicado a cada *request* para entrenar un modelo *one-class*, constituido por 64 caracteres que fueron seleccionados por expertos en seguridad y que son propensos a ser encontrados en un *request* de ataque.

Consideraciones:

El diccionario construido en el presente trabajo se aplica al parámetro de las *URIs* mientras que el utilizado en [7] se aplica a los *requests*. Además, mientras que en [7] el diccionario está compuesto por 64 *keys* expertas, en este trabajo se adicionaron las siguientes: **.exe, .dll, ', cgi-bin.**

Keys Expertas:

En total se identificaron entonces 68 *keys* expertas que se disponen a continuación en la Tabla 4.

Keys Expertas			
<	<i>#include</i>	<i>document.cookie</i>	>
.. /	<i>select</i>	<i>etc/passwd</i>)
< >	<i>alert</i>	<i>password</i>	<!--
'	<i>exec</i>	<i>bash_history</i>	/
-	<i>union</i>	<i>path/child</i>	/*

* /	<i>order</i>	<i>onmouseover</i>	<i>/</i>
<i>;</i>	<i>winnt</i>	<i>User-Agent:</i>	<i>\$</i>
<i>+</i>	<i>commit</i>	<i>javascript:</i>	<i>0%</i>
<i>=</i>	<i>alter</i>	<i>#¿NOMBRE?</i>	<i>cn=</i>
<i>(</i>	<i>from</i>	<i>between</i>	<i>cmd</i>
<i>:</i>	<i>where</i>	<i>objectclass</i>	<i>or</i>
<i>//</i>	<i>count</i>	<i>Accept:</i>	<i>%0a</i>
<i>*</i>	<i>passwd</i>	<i>upper</i>	<i>url=</i>
<i>"</i>	<i>script</i>	<i>insert</i>	<i>.exe</i>
<i>/</i> <i>c</i>	<i>table</i>	<i>and</i>	<i>.dll</i>
<i>-</i> <i>></i>	<i>shell</i>	<i>href</i>	<i>'</i>
<i>&</i>	<i>into</i>	<i>mail=</i>	<i>cgi-bin</i>

Tabla 4. Keys Expertas

Modelado del bag of words:

Para cada *URI* dentro de cada *dataset* se generó un vector de 68 posiciones que contiene la frecuencia de cada una de las *keys* expertas en la misma posición que les corresponde en el *BoW*. Por ejemplo, si para una *URI* determinada el *BoW* encuentra que hay una ocurrencia de la *key* experta “*select*”, sumará uno en la posición correspondiente a dicha *key* dentro del vector.

Este *BoW* se formó con *keys* expertas que no solo representan una amenaza, debido a que considerar únicamente estas *keys* implicaría quedarse con vectores de ceros con los cuales no tendría sentido entrenar ningún modelo.

Adicionalmente se decidió no eliminar ninguna entrada duplicada del *BoW* (es decir vectores iguales y correspondientes a la misma etiqueta) porque se consideró que estos aportarían mayor peso a tales vectores y *keys* expertas contenidas en ellas.

Para el caso de las *URIs* maliciosas se realizó un control para identificar aquellas en las cuales el *BoW* solo contabilizara ceros en todas las posiciones, de modo tal de determinar potenciales *keys* expertas que pudieran añadirse y que no estuvieran siendo consideradas previamente.

Como resultado de este control es que se añadieron las *keys* expertas adicionales a las de [7].

Limitaciones del bag of words:

- No incorpora información sobre la posición que ocupa cada palabra en la *URI*, sino que solamente informa sobre la frecuencia de esta en el documento.
- Contabiliza la *key* experta “*script*” cada vez que identifica esta palabra incorporada en alguna otra, como por ejemplo “*description*”.

3. Evaluación de los modelos

Se realizó la evaluación del entrenamiento y de la bondad de ajuste de los modelos entrenados aplicando diversas herramientas y métricas para el problema de clasificación tales como la matriz de confusión, las curvas de aprendizaje y múltiples criterios de evaluación.

3.1 Matriz de confusión

Esta herramienta comúnmente utilizada para problemas de clasificación binaria como el de este caso, permite determinar cómo es la distribución del error.

Se utiliza una matriz en donde las filas representan el valor predicho y las columnas el valor real para cada una de las clases, o viceversa. En base a esto se definen los conceptos de *True Positive (TP)*, *True Negative (TN)*, *False Positive (FP)* y *False Negative (FN)*.

Si la instancia es positiva y el modelo la clasifica como tal, el caso es un *TP*, si el modelo la clasifica como negativa, entonces es un *FN*. Análogamente, si la instancia es negativa y el modelo la clasifica como tal, el caso es un *TN*, si el modelo la clasifica como positiva, es un *FP*.

La matriz de confusión puede ser aplicada en los datos de *train*, *test* y *validation*. Este trabajo se enfoca en conocer cómo se distribuye el error en la etapa de evaluación y por ende la misma se aplica sobre los datos de *test*.

A partir de esto se realiza el cálculo y análisis de las siguientes métricas: *accuracy*, *True Positive Rate (TPR)*, *True Negative Rate (TNR)*, *False Positive Rate (FPR)* y *False Negative Rate (FNR)* para comparar la bondad de ajuste de los modelos probados.

En el presente trabajo se analizaron todas las métricas, pero se decidió presentar las siguientes:

- TPR (*TP* sobre el total de positivos) = $TP / (TP + FN)$
- TNR (*TN* sobre el total de negativos) = $TN / (TN + FP)$

En un modelo, una buena *performance* se da cuando el *TPR* y el *TNR* son valores altos, y por lo tanto el *FPR* y el *FNR* son bajos.

3.2 Análisis de Curvas de Aprendizaje

Las curvas de aprendizaje representan una herramienta gráfica que permite analizar cada *step* del proceso de entrenamiento en el tiempo. En el presente trabajo se definieron dos métricas para evaluar el aprendizaje: una enfocada a la optimización del entrenamiento, utilizando la minimización de la función de costo o *loss*, y otra enfocada en la *performance* como la maximización del *accuracy*.

También se decidió evaluar el aprendizaje en los datos de entrenamiento y de validación. En el primer caso se busca analizar qué tan bien está aprendiendo el modelo y en el segundo caso, se evalúa la *performance* en la generalización del mismo ante nuevos datos.

3.3 Definición del threshold y criterios elegidos

Como en el presente trabajo los modelos se basan en redes neuronales con varias capas densas, esto implica que las predicciones obtenidas son probabilidades y por lo tanto se debe definir un umbral previamente para saber si ese caso pertenece a una u otra clase.

A los efectos de determinar la línea de corte sobre las predicciones para clasificar una *URI* como un “ataque” (1) o “no ataque” (0), se utilizaron diferentes criterios que se detallan a continuación:

Umbral por defecto:

En este caso, el punto de corte para clasificar las predicciones es establecido en 0,5.

Criterio 1 - ROC Curve: Construida a partir de la representación gráfica de los *TPR* y los *FPR* en un par de ejes cartesianos, presenta la compensación existente entre el *TPR* y la métrica $1-FPR$, de modo tal que las curvas que más se alejan de la diagonal de 45 grados trazada desde el punto de corte de los ejes (es decir, aquellas en donde

el área que se genera debajo de ellas es mayor) reflejan mejores resultados en la predicción de los modelos. Se buscó el *threshold* óptimo maximizando la diferencia entre el *TPR* y el *FPR*.

Criterio 2 - Precision Recall Curve: Muestra la compensación entre la precisión y la sensibilidad (o *recall*). Grandes áreas debajo de esta curva representan mayores valores de ambas métricas, lo cual es positivo. Se busco el *threshold* óptimo maximizando el *f score*.

Criterio 3: Es una variación del criterio 1. Mientras que en este último el *threshold* óptimo se calcula maximizando la raíz cuadrada de $TPR*(1-FPR)$.

4. Estudio preliminar de modelos

En esta sección se presentan las investigaciones que permitieron un abordaje inicial del problema.

Se decidió comenzar el estudio para un conjunto reducido de *URIs*, seleccionando en cada *dataset*, únicamente aquellas que estuvieran compuestas por recursos y parámetros (*Query*).

El análisis se comenzó a partir del tratamiento de cada *dataset* en forma individual, por lo cual los modelos se entrenaron con distintos datos de entrenamiento.

4.1 Modelo Good/Bad Queries

Los *datasets* empleados en este caso son los de *Good/Bad Queries* descritos en la sección 2.2.1.

En esta instancia se filtró para cada *dataset* el conjunto de *URIs*, seleccionando únicamente aquellas compuestas por recursos y parámetros y separando los unos de los otros en cada una de ellas.

Todos los *datasets* de *Good/Bad Queries* fueron unificados.

El *dataset* resultante contiene separadamente los recursos, los parámetros, los vectores de *BoW* correspondientes a cada *URI* (para parámetros y recursos por separado), los vectores de *BoW* correspondientes a la *URI* completa y la etiqueta.

Del mismo se realiza la siguiente partición de los datos: 50% para *train*, 30% para *test* y un 20% para *validation*. Dicha partición se hizo tomando en cuenta únicamente los vectores de *BoW* correspondientes a los parámetros de las *URIs*.

Esta decisión se tomó considerando que en la parte de los parámetros de una *URI* es más probable encontrar evidencia de ataques que con respecto a la parte de los recursos.

En este caso se generó una red neuronal con una arquitectura inicial para entrenar el modelo con los datos de entrenamiento.

Resultados obtenidos de la evaluación del modelo:

Umbral por Defecto:

TPR: 0.79

TNR: 0.98

Accuracy: 0.90

Criterio 1:

TPR: 0.82

TNR: 0.95

Accuracy: 0.90

Criterio 2:

TPR: 0.82

TNR: 0.95

Accuracy: 0.90

Criterio 3:

TPR: 0.82

TNR: 0.95

Accuracy: 0.90

Conclusiones:

Los criterios 1, 2 y 3 presentaron los mismos resultados, y mientras que en estos casos el *TNR* es inferior al *TNR* obtenido en el umbral por defecto, lo inverso ocurre con el *TPR* en aproximadamente la misma proporción. Por este motivo es difícil determinar si estos criterios son mejores o peores con respecto al umbral por defecto.

En forma paralela se realizó el tratamiento de los *datasets* de *CSIC* y *PKDD* con el objetivo de evaluar el mismo modelo entrenado con los datos de *Good/Bad Queries*, pero con estos otros datos para ver que tan bien generaliza.

Resultados evaluación CSIC:

Umbral por Defecto:

TPR: 0.08

TNR: 0.98

Accuracy: 0.62

Criterio 1:

TPR: 0.10

TNR: 0.96

Accuracy: 0.62

Criterio 2:

TPR: 0.10

TNR: 0.96

Accuracy: 0.62

Criterio 3:

TPR: 0.10

TNR: 0.96

Accuracy: 0.62

Resultados evaluación PKDD:

Umbral por Defecto:

TPR: 0.55

TNR: 0.92

Accuracy: 0.82

Criterio 1:

TPR: 0.82

TNR: 0.95

Accuracy: 0.82

Criterio 2:

TPR: 0.82

TNR: 0.95

Accuracy: 0.82

Criterio 3:

TPR: 0.82

TNR: 0.95

Accuracy: 0.82

Conclusiones:

La evaluación del modelo con los datos de *test* de CSIC generó los peores resultados, especialmente para las métricas de *TPR* y *Accuracy* (que fueron demasiado bajas).

Por otro lado, al ser evaluado con los datos de *test* de *PKDD*, los resultados fueron mejores, pero no tan buenos como en el caso de la evaluación con los propios datos de *Good/Bad Queries*. Además, en ambos casos, la evaluación con los datos de *test* de CSIC y *PKDD* generó las mismas métricas para los criterios 1, 2 y 3, siendo estas superiores a las generadas con el umbral por defecto.

4.2 Modelo CSIC

Los *datasets* empleados en este caso son los de CSIC descritos en la sección 2.2.2.

Nuevamente se filtró para cada *dataset* el conjunto de *URIs*, seleccionando únicamente aquellas compuestas por recursos y parámetros y separando los unos de los otros en cada una de ellas.

Todos los *datasets* de CSIC fueron unificados.

El *dataset* resultante contiene separadamente los recursos, los parámetros, los vectores de *BoW* correspondientes a cada *URI* (para parámetros y recursos por separado), los vectores de *BoW* correspondientes a la *URI* completa y la etiqueta.

A partir de este *dataset* se realiza la siguiente partición de los datos: 50% para *train*, 30% para *test* y un 20% para *validation*. Nuevamente, la partición se hizo tomando en cuenta únicamente los vectores de *BoW* correspondientes a los parámetros de las *URIs*.

De igual modo se mantuvo la misma arquitectura empleada para el *dataset* unificado de *Good/Bad Queries* y la evaluación del modelo se realizó en base a el umbral por defecto y a los tres criterios mencionados en la sección 3.3.

Resultados obtenidos de la evaluación del modelo:

Umbral por Defecto:

TPR: 0.24

TNR: 0.97

Accuracy: 0.66

Criterio 1:

TPR: 0.30

TNR: 0.94

Accuracy: 0.66

Criterio 2:

TPR: 1.00

TNR: 0.00

Accuracy: 0.66

Criterio 3:

TPR: 0.44

TNR: 0.75

Accuracy: 0.66

Conclusiones:

Se observa que tanto en el caso del umbral por defecto como en el de los criterios 1 y 3 el *TPR* es demasiado bajo con respecto al *TNR* y no se encuentran balanceados. Los resultados no fueron positivos en ninguno de los casos y no es posible determinar con exactitud cuál de los criterios resulta más adecuado.

Dado que se utilizó la misma arquitectura que con respecto al modelo entrenado con *Good/Bad Queries* se podría pensar que la distribución de los datos de entrenamiento en este caso no influyó positivamente en los resultados.

4.3 Modelo PKDD

Los *datasets* empleados en este caso son los de *PKDD* descritos en la sección 2.2.3.

Nuevamente se filtró para cada *dataset* el conjunto de *URIs*, seleccionando únicamente aquellas compuestas por recursos y parámetros y separando los unos de los otros en cada una de ellas.

Todos los *datasets* de *PKDD* fueron unificados.

El tratamiento de los datos se hizo análogamente al del modelo entrenado con CSIC descrito anteriormente, motivo por el cual se presentan directamente los resultados obtenidos:

Resultados obtenidos de la evaluación del modelo:

Umbral por Defecto:

TPR: 0.53

TNR: 0.99

Accuracy: 0.86

Criterio 1:

TPR: 0.55

TNR: 0.98

Accuracy: 0.86

Criterio 2:

TPR: 0.55

TNR: 0.98

Accuracy: 0.86

Criterio 3:

TPR: 0.56

TNR: 0.97

Accuracy: 0.86

Conclusiones:

Tanto en el caso del umbral por defecto como en el de los criterios 1, 2 y 3 se obtuvieron resultados similares en las métricas de *TPR* y *TNR* (no balanceadas y con valores muy bajos para el *TPR*) y el mismo valor para el *accuracy*.

A pesar de que las métricas analizadas no son las esperadas, el *accuracy* se constató mayor con respecto al modelo entrenado con los datos de CSIC.

Los diferentes resultados obtenidos en esta etapa preliminar no fueron concluyentes en base al objetivo planteado inicialmente en la introducción de este trabajo, dado que no se pudieron conseguir métricas balanceadas. Se observa que cada modelo de los presentado logró obtener un buen desempeño en el *TPR*, pero no en el *TNR* y

viceversa. Incluso, la estrategia de experimentar con varios criterios de *thresholds* no logra conseguir los resultados deseados.

En la sección siguiente y sobre la base de lo descrito en el párrafo anterior se presenta la nueva estrategia que modifica los datos utilizados en la fase de entrenamiento y evaluación de los modelos. La misma consiste en unificar los *dataset Good/Bad Queries*, *CSIC* y *PKDD*, con el objetivo de disminuir el error de predicción, entrenando un modelo con *URLs* generadas por diferentes dominios.

5. Unificación de datasets

A partir de este punto se decidió unificar los datos de *Good/Bad Queries*, *CSIC* y *PKDD* (con las transformaciones realizadas mediante *scripts* y expresiones regulares) en un único *dataset* en donde cada registro representa una *URI* con su correspondiente etiqueta.

Adicionalmente, a los efectos de mejorar los resultados de las métricas y el *accuracy* se decidió incluir la totalidad de las *URIs*.

A continuación se muestra un ejemplo de la estructura del *dataset* unificado en la Tabla 5.

URI	Etiqueta
/include/phpxd/phpxd.php?appconf[rootpath]=@rfiurl?appconf[rootpath]=@rfiurl?&cmd=id	1
/en-us/account/login.pl?login=ledgersmb_script_code_exec.nasl&script=-e print "content-type: text/plain\x0d\x0a\x0d\x0a";system(id)&action=logout	0
/miAseso8ezikr/jtsgq0ieinwicaPlcnt/.ns2C87vBxmlC/e0.6vKu6@L1BNi4e8Y/n0U782cyx1q1GT/Ha/s_qPF@c875/itashtol5d/mlhtyio04t/eQohwaE6/iMer/PfjgA4s7l.asp	0
/lGwd-J6hRWLOSuJ/3ebeswrws3c3b1etNRU/rxam56rqadee.asmx?cbteewjlanitt=ezt.6qJoliNN&aieTsoa=execxp_regwrite 'HKEY_LOCAL_MACHINE','SOFTWARE\Microsoft\MSSQLServer\Client\ConnectTo','Nm5xfai','REG_SZ','DBMSSOCN,hackersip,80'&trAfntttrerap=s3A-gJQzC&cwEux=260&esoch=8287356386&w6shisklht0n=c8kmaxsvunrame&rmosh4ttpaeuubO=731469&rd6=1104205&ogFt5ERloian=eQh&ih0hseerecitmh=099377&s0uitGriedoo=heerh&nsrht7nxtl1Adr=0018	1

Tabla 5. Dataset unificado

En la Tabla 6 se observa la proporción de *URIs* que corresponden a tráfico normal y anómalo para la totalidad de los datos:

Normal	1.320.055	94.7 %
Anormal	73.556	5.3 %

Tabla 6. Proporción de *URIs* normales y anómalas

La composición del *corpus* presenta un gran desafío a la hora de realizar modelos de clasificación. Independientemente de si se trata de un problema binario o multiclase, un conjunto de datos desbalanceados afecta el entrenamiento del algoritmo, penalizando a la clase o clases que tienen menos cantidad de muestras para entrenar.

5.1 Exploración del corpus con el bag of words de Keys Expertas

5.1.1 Frecuencia de las Keys Expertas en el Corpus

Se analizaron las 20 *keys* expertas más frecuentes en el total del *corpus* para *URIs* de tráfico normal y se concluyó que aproximadamente el 80% más frecuente se distribuye en los caracteres más frecuentes que tienen las *URIs*: El “/” que separa los diferentes niveles o jerarquías dentro del *path*, el “&” que define las diferentes *key-Value* dentro del *Query* y el signo de “=” que separa las variables de los valores.

Se observa la distribución de las *keys* expertas en la Tabla 7.

URIs Normales		
Keys Expertas	Total	Distribución
/	2912801	64,40%
=	413875	9,15%
&	334930	7,41%
-	220503	4,88%
<i>script</i>	197188	4,36%
<i>or</i>	132084	2,92%
<i>/c</i>	100951	2,23%
+	79024	1,75%
:	42688	0,94%
<i>and</i>	17304	0,38%
<i>password</i>	8818	0,19%
<i>mail=</i>	8053	0,18%
<i>insert</i>	5293	0,12%

<i>exec</i>	4160	0,09%
<i>cmd</i>	3395	0,08%
<i>count</i>	3024	0,07%
<i>.dll</i>	2864	0,06%
<i>*</i>	2573	0,06%
<i>select</i>	2531	0,06%
<i>cgi-bin</i>	2381	0,05%

Tabla 7. Distribución de *Keys expertas* más frecuentes en *URIs* normales

Del total de las *keys expertas* consideradas para el análisis se identificaron 13 no presentes en el *corpus*, las cuales se observan en la Tabla 8:

->	'	<i>document.cookie</i>	<i>javascript:</i>	<i>User-Agent:</i>	<i>path/child</i>	<i>%0a</i>
<!--	"	<i>objectclass</i>	<i>Accept:</i>	<i>etc/passwd</i>	<i>#include</i>	

Tabla 8. *Keys* no presentes en el corpus de *URIs* normales

Analizando las *keys expertas* que componen las *URIs* anómalas se observó que también son las más frecuentes las *keys* "/", "=", y "&", sin embargo representan únicamente el 56% del total de *URIs* anómalas. Se observa esto en la Tabla 9.

URIs Anormales		
Key Experta	Total	Distribución
/	257240	20,69%
=	243122	19,56%
&	187724	15,10%
+	183167	14,73%
-	50290	4,05%
<i>script</i>	31570	2,54%
<i>or</i>	29597	2,38%
>	25194	2,03%
../	24357	1,96%
<	23179	1,86%
)	22488	1,81%
(22014	1,77%
;	13556	1,09%

'	11807	0,95%
\$	10684	0,86%
/c	10073	0,81%
:	8512	0,68%
*	8455	0,68%
%0a	7637	0,61%
alert	6092	0,49%

Tabla 9. Distribución de *Keys* expertas más frecuentes en *URIs* anómalas

Por último, se observa en la Tabla 10 que para las *URIs* anómalas las siguientes *keys* expertas no se encuentran representadas en el *corpus*, es decir que no fueron identificadas en ninguna *URI* identificada como ataque:

->	<i>Accept:</i>	<i>User-Agent:</i>
----	----------------	--------------------

Tabla 10. *Keys* expertas no presentes en el corpus de *URIs* anómalas

En el Anexo 2, se encuentra un detalle de los datos presentados para todas las *keys* analizadas en esta sección.

5.1.2 Correlación de las Keys Expertas

Un segundo análisis descriptivo consistió en observar la matriz de correlaciones para las 68 *features*. Primero se buscó determinar la existencia de una **correlación positiva**. En el Gráfico 1 se observa la matriz de correlación, indicando en color verde todas aquellas *features* que presentan una correlación superior al 60%.

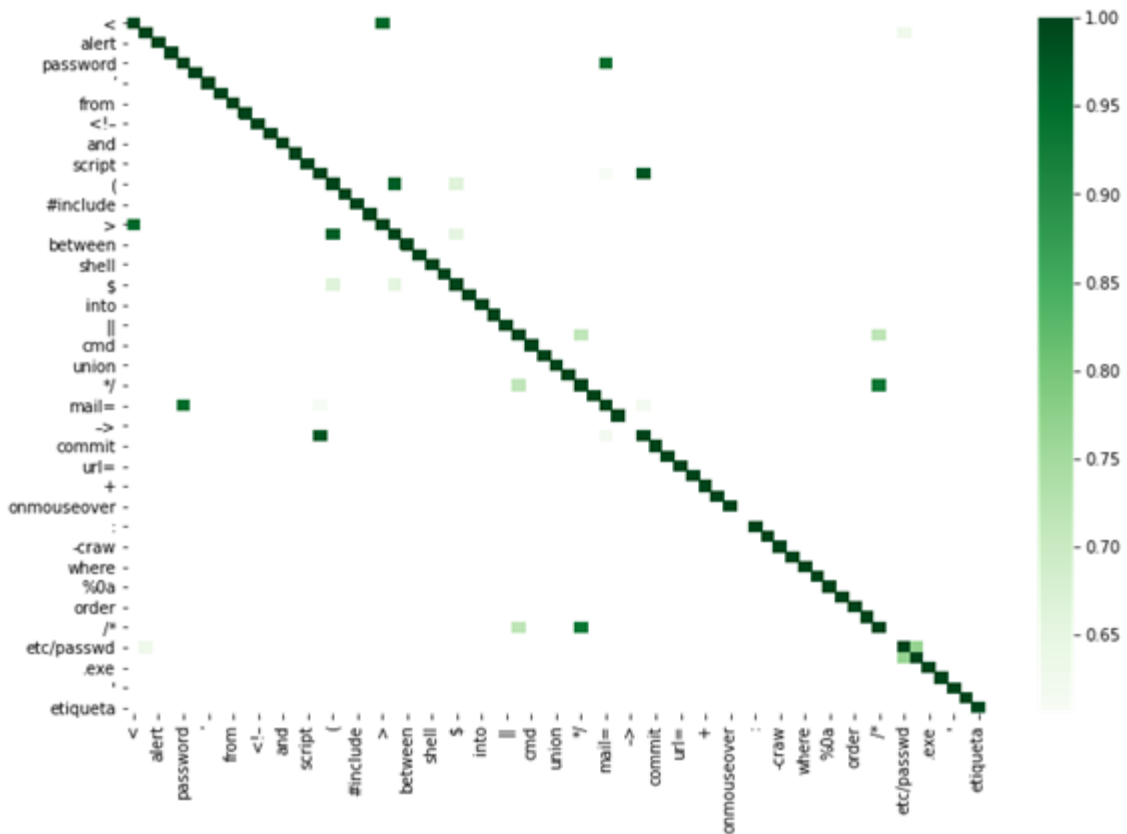


Gráfico 1. Features con correlación superior a 60%

De este análisis, en la Tabla 11 se observa que las *keys* expertas con mayor correlación positiva son las siguientes:

<	>
<i>mail=</i>	<i>password</i>
<i>etc/passwd</i>	<i>../</i>
<i>mail=</i>	=
&	=
)	(
\$	(
\$)
*/	*
/*	*
/*	*/
<i>mail=</i>	&
<i>winnt</i>	<i>passwd</i>

Tabla 11. *Keys* expertas con mayor correlación positiva

Las relaciones positivas identificadas son originadas por tres causas:

La primera es generada por la forma en que se implementó el *BoW* en el *corpus*: la fuerte correlación positiva entre “*/” y “*” es consecuencia de que el *BoW* contabiliza el carácter “*/” como 1 no sólo para la *key* experta “*/”, sino también para la *key* experta “*”.

La segunda causa obedece a relaciones evidentes por cómo se estructura una *URI*. Es el caso de la relación entre “&” y “=”, dado que en el componente de la *query*, cuanto más cantidad de pares de *Key-Values* exista, mayor cantidad de delimitadores necesita la estructura.

Por último, se observaron casos que pueden evidenciar un patrón natural en los datos entre las siguientes *keys* expertas: “winnt” y “passwd”, “etc/passwd” y “../”, “mail=” y “password”.

Finalmente se buscó determinar la existencia de **correlaciones negativas** entre las *features* del *corpus*, y considerando correlaciones menores a -0.60% no se encontró ningún caso para ninguna de las 68 *features*.

6. Desarrollo de modelos sobre datasets unificados

6.1 Modelo Keys Expertas

Este modelo es el primero que utiliza para su entrenamiento el *dataset* formado por todos los conjuntos de datos unificados, y a partir del cual se determinó la arquitectura a emplear para los experimentos posteriores.

Esto quiere decir que es el primer modelo que se toma como línea base para comparar sus resultados con los de los siguientes experimentos, en tanto las métricas no mejoren respecto a él.

6.1.1 Arquitectura del Modelo

La arquitectura que se muestra a continuación es el resultado de múltiples iteraciones de modelos no documentados que se fueron modificando en función de las métricas obtenidas.

Dicha arquitectura es la que presentó los mejores resultados para las pruebas realizadas.

```
model2 = Sequential()  
model2.add(layers.BatchNormalization(input_shape=(inputShape,)))  
model2.add(Dense(850,activation='relu',  
input_shape=(inputShape,),kernel_regularizer=regularizers.l1(0.01) ))  
model2.add(Dense(500, activation='relu'))  
model2.add(Dense(200, activation='relu'))  
model2.add(Dropout(0.20))  
model2.add(Dense(100, activation='relu'))  
model2.add(Dense(32, activation='relu'))  
model2.add(Dense(1, activation='sigmoid'))
```

6.1.2 Análisis de la arquitectura y entrenamiento del modelo

La arquitectura elegida y mencionada en la sección anterior fue entrenada con una partición de *train* de 780.420 URIs y una partición de *validation* de 195.106 URIs. La misma fue evaluada en una partición de *test* de 418.083 URIs.

A continuación se presentan los parámetros de entrenamiento:

```
epochs=70
batch_size=300
optimizadores = Adam(lr=0.00001, beta_1=0.9, beta_2=0.999, epsilon=1e-07,
amsgrad=False)
callbacks = EarlyStopping(monitor='val_loss', patience=5)
model2.compile(loss='binary_crossentropy',
               optimizer=optimizadores,
               metrics=['accuracy'])
history2= model2.fit(x_train, y_train,
                   batch_size=batch_size,
                   epochs=epochs,
                   callbacks=callbacks,
                   class_weight=class_weights,
                   validation_data=(x_validation, y_validation))
```

Esta arquitectura utiliza al final una capa densa con una función de activación “*sigmoid*”. Dicha función es la más adecuada cuando se quiere realizar una clasificación binaria. Adicionalmente se dispusieron previamente varias capas densas, cada una de ellas con una función de activación “*relu*”, con el objetivo de intentar aprender relaciones no lineales entre los datos. A medida se agregaron dichas capas los resultados del modelo fueron mejorando.

En congruencia con la función de activación “*sigmoid*” se utilizó en el “*compile*” una “*loss*” del tipo “*binary_crossentropy*”, para la clasificación de clases binarias.

En el caso del presente modelo el *"input_shape"* fue de 68, ya que había un total de 68 features, es decir, las *keys* expertas del *BoW*. Este parámetro se modifica más adelante en función de la cantidad de features considerados en próximos experimentos que se presentan en la sección 6.2.

Se determinó que a menor *"learning rate"* mejores eran los resultados para el *accuracy* y la *loss* del modelo.

De igual modo, un *"dropout"* de más de 20% generaba mejores resultados que valores inferiores al 10% o superiores al 30% para este parámetro.

Debido a que los vectores del *BoW* poseen un cero en la mayoría de las posiciones, se utilizó un regularizador *L1* que minimizará este hecho al cancelar una gran cantidad de dimensiones en tales vectores.

Una vez entrenado el modelo con un determinado número de *epochs* se analizaron los gráficos de *loss* y de *accuracy* para determinar la posible existencia de *overfitting*.

Luego de entrenar el modelo por primera vez, el gráfico del *accuracy* indicaba picos de la curva de *validation* por sobre la curva de *loss* (se concluyó que un ligero cambio que se hiciera sobre los coeficientes prácticamente no afectaba el *accuracy* en *train*, pero si lo hacía significativamente en el de *validation*). En principio se consideró que esto pudiera tratarse de un sobreajuste, pero luego se concluyó que tan solo era un efecto estacional y que basar una conclusión en dichas curvas no tenía sentido. En otras palabras, se constató que el modelo no había tenido el tiempo suficiente para entrenar completamente.

Este motivo también explica el hecho de que el gráfico de *loss* hubiera mostrado una curva de *train* situada por encima de la de *validation* (de todos modos se consideró que esto podría explicarse también debido a una cuestión de aleatoriedad en la partición de los datos).

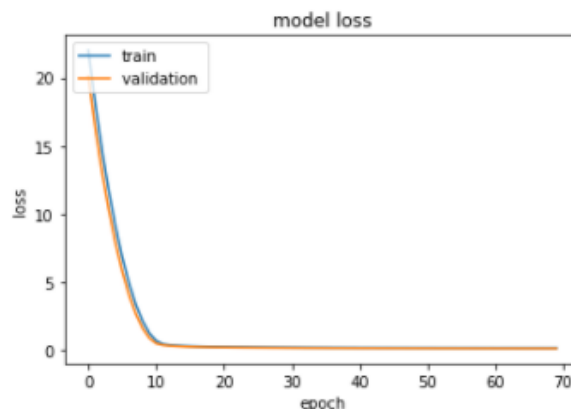
Por este motivo el número de *epochs* se incrementó hasta dejarse en 70 y para evitar potenciales problemas de sobreajuste se utilizó un “*early stopping*” con un rango de paciencia de 5, de modo tal que si la “*validation loss*” no mejora durante cinco iteraciones (o sea si la misma se mantuviera invariante, aumentara o aumentara y luego se redujera pero no por debajo del mejor valor alcanzado), el modelo dejaría de entrenar.

El optimizador Adam sirve para actualizar los pesos de la red en los datos de *train*.

Por último, se utilizó un *class-weight* en el entrenamiento para mitigar el problema del desbalanceo de las clases (ver Tabla 6), afectando la que tiene menos cantidad de muestras. Esta estrategia se centra en penalizar la función de costo de aquellas clases menos frecuentes en el *training data*, en este caso cuando la Etiqueta=0.

Resultados obtenidos:

- **Curvas de aprendizaje:**
 - **Curvas de Loss:**



Gráfica 2. Curva de loss de *Keys Expertas*

Se aprecia en el Gráfico 2 que la curva de *loss* de *validation* se ubica por debajo de la de *train* durante los 10 primeros *epochs* y luego ambas curvas se acoplan.

Viendo la curva de *loss* de *train* ya es posible afirmar que el modelo tiene la capacidad adecuada para aprender sobre la complejidad del

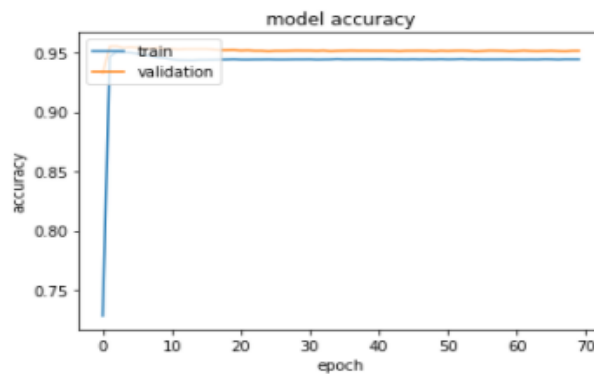
conjunto de datos, dado que no exhibe una línea plana o por el contrario picos relativamente altos.

Adicionalmente, la curva de *loss* de *train* no continúa disminuyendo al final de las iteraciones inputadas, dando a entender que el modelo no es capaz de un mayor aprendizaje que hubiera sido interrumpido.

Por otro lado, la curva de *loss* de *validation* no disminuye hasta cierto punto para luego comenzar a aumentar, lo que daría cuenta de un posible sobreajuste del modelo en dicho punto de inflexión a partir del cual el entrenamiento tendría que haber sido detenido.

Este comportamiento en el que ambas curvas disminuyen hasta un punto de estabilidad con brecha mínima entre ambos valores de *loss* se considera representativo de un buen ajuste del modelo.

- **Curva de Accuracy**



Gráfica 3. Curva de accuracy de *Keys Expertas*

El Gráfico 3 del *accuracy* muestra una curva de *train* situada todo el tiempo por sobre la curva de *validation*. Se aprecia en este resultado que el modelo no da lugar a un crecimiento del *accuracy* prácticamente desde los primeros *epochs* en adelante, por lo cual se asume que en congruencia con el gráfico de *loss*, el mismo ya terminó de entrenar.

- **Métricas:**

Umbral por Defecto:

TPR: 0.94

TNR: 0.95

Accuracy: 0.95

Criterio 1:

TPR: 0.94

TNR: 0.95

Accuracy: 0.95

Criterio 2:

TPR: 0.95

TNR: 0.76

Accuracy: 0.99

Criterio 3:

TPR: 0.94

TNR: 0.95

Accuracy: 0.95

Conclusiones:

Se observó que el Criterio 1 y el Criterio 3 generaron los mismos resultados y que fueron superiores a los obtenidos con el umbral por defecto y con el Criterio 2.

Por este motivo a partir de este punto se decidió considerar únicamente el Criterio 1.

Como consecuencia de esto, dado que la curva de *accuracy* se basa en este criterio, la misma tampoco se continuó utilizando para el análisis.

Si bien consideramos que los resultados obtenidos con este modelo son satisfactorios, se decidió incorporar nuevas técnicas de aprendizaje automático con el objetivo de mejorar los mismos.

6.2 N-Grams

6.2.1 TF-IDF: Term frequency - inverse document frequency

TF-IDF es un modelo de *bag o words* que se utiliza para determinar la relevancia que una palabra tiene en un documento perteneciente a un *corpus*. Se agrupan las palabras más destacadas en el *corpus* sin tener en cuenta el orden de las mismas con el objetivo de realizar algún análisis específico.

TF (term frequency): Refiere a la frecuencia relativa de un término específico en un documento al ser comparado con la totalidad de los términos en el mismo. Esto proporciona información respecto al peso que dicho término tiene en el documento.

La ecuación para obtener el *TF* se observa en Ecuación 1:

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{i,j}}$$

Ecuación 1. *TF - Term Frequency*

En esta ecuación, en el numerador, “ n ” es el número de veces que el término “ i ” aparece en el documento “ j ”. Por otro lado, el denominador es la sumatoria de todos los términos existentes.

IDF (inverse document frequency): Se utiliza como correctivo para restar relevancia a términos muy frecuentes en el *corpus* que aportan poca información, como por ejemplo conectores y artículos. De este modo el *IDF* permite dar mayor relevancia a las palabras realmente significativas en la colección de documentos.

La ecuación para obtener el *IDF* se observa en Ecuación 2:

$$idf(w) = \log\left(\frac{N}{df_t}\right)$$

Ecuación 2. *IDF - Inverse Document Frequency*

En esta ecuación, “ N ” es la cantidad de documentos y “ df ” la cantidad de documentos que contiene el término “ t ”. Es por lo tanto una medida inversa de cuánta relevancia tiene cada término en el *corpus*.

La ecuación para el cálculo del *TF-IDF* se observa en Ecuación 3:

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

Ecuación 3. *TF-IDF*

6.2.1.1 Modelo *TF-IDF* - *Keys Expertas*

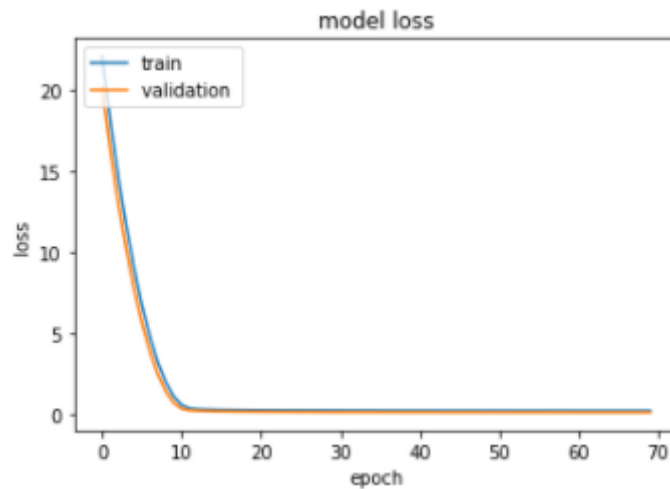
En el presente trabajo la primera aproximación a la utilización del *TF-IDF* fue su aplicación en conjunto con las *keys expertas* que se definieron inicialmente.

Con el objetivo de utilizar un vocabulario propio se definieron manualmente las funciones para el *TF-IDF* en base a las ecuaciones anteriormente expuestas.

Resultados obtenidos de la evaluación del modelo:

- **Curvas de aprendizaje:**

Las curvas de *train* y *validation* obtenidas para el *loss* fueron muy similares a las obtenidas en el Modelo - *Keys Expertas*. Se observa esto en la Tabla 4.



Gráfica 4. Curva de loss TF-IDF *Keys Expertas*

- **Métricas:**

Criterio 1:

TPR: 0.87

TNR: 0.95

Comparación histórica de modelos:

MÉTRICAS	MODELO - KEYS EXPERTAS	TF-IDF - KEYS EXPERTAS
<i>TPR</i>	0.94	0.87
<i>TNR</i>	0.95	0.95

Tabla 12. Comparación histórica Modelo - Keys Expertas y TF-IDF - Keys Expertas

Conclusiones:

En la Tabla 12 se observa que el *TNR* obtenido es el mismo con respecto al modelo anterior pero el *TPR* es inferior, motivo por el cual estas métricas se ven más desbalanceadas.

Por este motivo se decidió continuar con el análisis de *TF-IDF* pero en este caso generando el vocabulario dinámicamente a partir del conjunto de datos.

6.2.1.2 Modelo TF-IDF: Feature Extraction

En este enfoque se experimentó con nuevas modalidades. Por un lado, se levantó la restricción de un vocabulario exógeno como en el caso de *BoW*, de modo tal que sea el modelo el que genere su propio vocabulario a partir de la información que se extrae de la propia *URI* y no en base al conocimiento previo de los expertos en ciberseguridad como en el enfoque del *BoW*.

Asimismo, se buscó implementar el modelo *n-gram* en base al vocabulario creado. El objetivo de incorporar estas modificaciones es intentar mejorar las predicciones. Posteriormente se comparan los resultados obtenidos.

Fue un gran desafío la creación de un vocabulario dada la naturaleza del problema, dado que la sintaxis y los componentes de una *URI* no están tan claramente definidos como es en el caso de un texto. Esto implicó que la implementación del vocabulario se realizará creando la función *getTokens* que usa expresiones regulares para efectuar una división de cada *URI* mediante separadores personalizados, definidos previamente con el objetivo de extraer porciones de esa *URI* o *tokens*.

Se decidió que los separadores empleados también formen parte de los *tokens* porque incluían información relevante para el entrenamiento del modelo. En la Tabla 13 se observan los delimitadores aplicados:

Delimitadores para realizar <i>token</i>															
/..	/	?	=	&	%	+	-	.	_	~	*	<	>	''	:

Tabla 13. Delimitadores para generación de *tokens*

En el proceso se detectó que un pequeño porcentaje de las *URLs* contenían ataques codificados con caracteres en hexadecimal, conteniendo un símbolo % seguido de dos dígitos hexadecimales, por ejemplo %20 que representa el símbolo correspondiente al espacio. Se probaron diferentes técnicas de decodificación pero en todos los casos los resultados generales del modelo se veían degradados. Debido a esto, se tomó la decisión de no decodificar estos casos.

Para el proceso de extracción de features en este enfoque se utilizó el módulo *sklearn.feature_extraction.text* de *scikit-learn*, que permite extraer *features* de un conjunto de datos de texto a un formato que pueda ser soportado por los algoritmos de *machine learning*. El módulo mencionado contiene una función llamada *TFidfVectorizer*.

Cabe señalar que la función *TFidfVectorizer* fue aplicada utilizando el vocabulario personalizado que se creó por medio de la función *getTokens*. Esto se realizó, aplicando como *input getTokens* en el parámetro *tokenizer* de *TFidfVectorizer*, obteniéndose una lista de *tokens*.

El resultado de este procedimiento es un *corpus* de todas las *URLs* que contiene tantas columnas (o *features*) como *tokens* creados por la función *getTokens* aplicada a todas las *URLs*. En cada columna se obtiene calculada la frecuencia *TF-IDF* de cada uno de los *tokens* definidos. Se computa un cero para cada caso en que el *token* no aparezca en la *URI*.

El proceso de *feature selection* fue todo un desafío en este enfoque, dado que la lógica de extracción de *features* (o *tokens*) a través de la creación de *tokens* propios tuvo como resultado una gran cantidad de *features*, implicando que fuera necesario realizar previamente una selección de los *tokens* candidatos para entrenar el modelo. Se tomaron tres consideraciones: la cantidad de *tokens*, el criterio de selección y los recursos disponibles (en especial la *RAM*).

Inicialmente se definió que la cantidad de *tokens* que deberían componer el vocabulario sería de 200, empleando como criterio de selección que fueran los más frecuentes en el *corpus*. Se observó que bajo el criterio de selección empleado, los resultados del modelo mejoraban a medida que la cantidad de *tokens* empleada se incrementaba.

Sin embargo, la gran cantidad de datos y *features* en el *corpus* generaron problemas vinculados al poder de cómputo de la infraestructura utilizada. Debido a esto, fue necesario tener que modificar el modo de trabajo y pasar a una modalidad de procesamiento en bloques (o *batches*).

En el siguiente diagrama se visualiza la lógica empleada para procesar por *batch* el cálculo de *TF-IDF* para el *corpus* y el entrenamiento de la red.

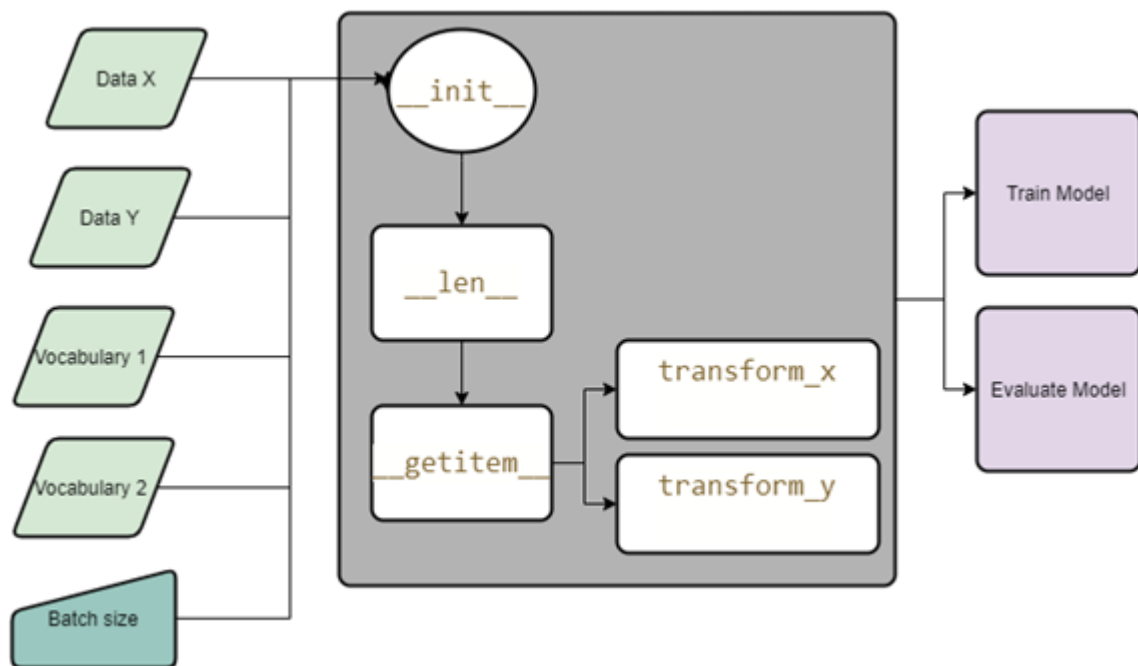


Diagrama 1. Algoritmo para procesar por batch.

La solución propuesta recibe como *inputs*: *Data X*, *Data Y*, *Vocabulary 1*, *Vocabulary 2* y *Batch Size*. *Data X* corresponde a las *URIs* y *Data Y* corresponde a las etiquetas. Mientras que *Vocabulary 1* y *Vocabulary 2* son los vocabularios extraídos de los *tokens* para el modelo de unigrama y bigrama, respectivamente. Por último, *batch size* es el tamaño del lote definido para el procesamiento, en este caso el tamaño se define como la cantidad de *URIs*.

El modelo se alimenta de un “generator” de tipo “DataSequence” que incluye la siguiente secuencia:

init: Inicializa la clase en base a los *inputs* definidos.

len: Calcula la cantidad de ejecuciones (iteraciones) necesarias para el tamaño del *batch* definido.

getitem: Define los *batches* para la *Data X* y *Data Y*.

transform_x: Aplica la lógica de *TF-IDF* directamente sobre la *Data X* (las *URLs*)

transform_y: Conserva la *Data Y* sin aplicarle ninguna transformación.

Por último, el *generator* resultante es utilizado en la fase de entrenamiento del modelo (sobre los datos de *train* y *validation*) y en la fase de evaluación (sobre el *test*)

El cálculo de los vectores de *TF-IDF* se realizó durante el “*fit*” del modelo en bloques de un tamaño predefinido en tiempo de ejecución.

Se observó que a partir de los 500 *tokens* los resultados obtenidos en el modelo no presentaban mejoras significativas.

A su vez, se incorporó en el presente enfoque el modelado de los *tokens* aplicando *n-gram*. Un *n-gram* es un segmento de *n* caracteres de una cadena más larga. Por ejemplo, se puede representar la palabra TEXTO como el siguiente bigrama: _T, TE, EX, XT, T_. El carácter “_” representa un espacio inicial o final.

Los *n-gram* permiten que las palabras similares (en virtud de tener un sufijo diferente o una variación ortográfica) compartan una gran cantidad de *n-gram*. Por ejemplo, en inglés las palabras *retrieve*, *retrieval* y *retrieving*, cuando son representadas por un bigrama, se observa que comparten la mayoría entre sí: *_r*, *re*, *et*, *tr*, *ri*, *ie* y *ev*.

Este trabajo busca aplicar la técnica de *n-gram* a las *URLs* para experimentar si existe cierta similitud entre las mismas, en especial entre aquellas que implican ataques. Esta estrategia puede contribuir a mejorar la *performance* de los modelos entrenados.

Se realizaron numerosos experimentos aplicando esta técnica, pero se decide presentar los resultados de un modelo con unigrama, otro con bigrama y un tercero que es la combinación de ambos (mixto).

Lo descrito en el párrafo anterior se realizó generando diferentes *vectorizer* utilizando la función *TFidfVectorizer*, mediante el parámetro “*ngram_range*”. Dicho parámetro permitió generar *vectorizers* con *tokens* compuestos.

En la Ilustración 3 se puede observar un ejemplo de la representación de una *URI* aplicando un unigrama, un bigrama y una combinación de ambos utilizando la función definida *getokens* y los delimitadores de la Tabla 13.

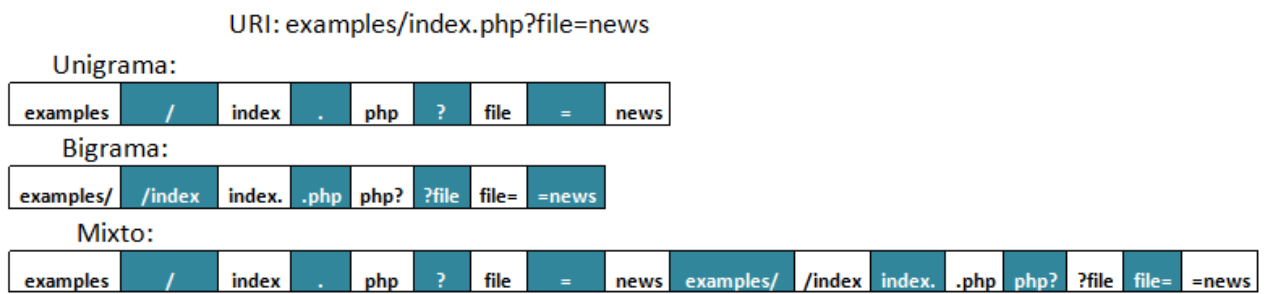


Ilustración 3. Ejemplo de una *URI* representada con Unigrama, Bigrama y Mixto.

Unigrama: Es la opción por defecto y considera un *token* representativo de un único término.

Bigrama: Similar al anterior, pero considera *tokens* que representan conjuntos de dos términos.

Mixto: Es una combinación de los dos anteriores.

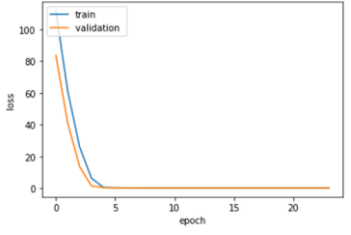
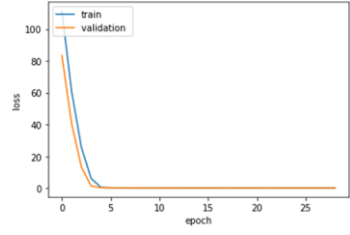
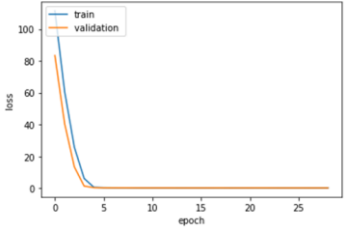
Es preciso señalar que el modelo Mixto se generó de dos modos diferentes. El primero de ellos concatenando los resultados obtenidos con unigrama, con los resultados obtenidos con bigrama. En el segundo caso, se especificó un *ngram_range* de “1,2”, siendo los resultados del mismo superiores respecto a los del primero, motivo por el cual fue descartado.

Además, para el caso del unigrama y bigrama se decidió obtener los 500 *tokens* más frecuentes en el corpus (*max_features=500*), mientras que para el mixto se consideraron 1000 *tokens*.

Es importante enfatizar que partir de esta instancia se comenzó a considerar únicamente el Criterio 1 (ver Sección 3.3) para la evaluación del modelo ya que se consideró que con el mismo se han obtenido los mejores resultados en la mayoría de los casos.

Resultados obtenidos:

Curvas de aprendizaje:

Unigrama	Bigrama	Mixto
 <p data-bbox="236 1281 587 1361">Gráfica 5. Curva de loss Unigrama</p>	 <p data-bbox="619 1281 970 1361">Gráfica 6. Curva de loss Bigrama</p>	 <p data-bbox="1007 1281 1358 1317">Gráfica 7. Curva de loss Mixto</p>

Métricas:

	Criterio 1	
	TPR	TNR
Unigrama	0.96	0.96
Bigrama	0.78	0.80
Mixto: Unigrama + Bigrama	0.96	0.95

Tabla 14. Resultado comparativo de métricas para *TF-IDF: Feature Extraction* Unigrama, Bigrama y Mixto

En la Gráfica 5, 6 y 7 se observa que el gráfico de *loss* es muy similar en los tres casos. Con respecto al modelo de *TF-IDF* con *Keys Expertas*, la única diferencia es que las curvas de *loss* para *train* y *validation* están más separadas entre sí durante las primeras iteraciones.

Comparación histórica de modelos:

Métricas	Modelo - <i>Keys Expertas</i>	TF-IDF Feature Extractions: Unigrama
<i>TPR</i>	0.94	0.96
<i>TNR</i>	0.95	0.96

Tabla 15. Comparación histórica Modelo - *Keys Expertas* y *TF-IDF: Feature Extraction Unigrama*

Conclusiones:

En la Tabla 14 se puede observar que los mejores resultados obtenidos en todas las métricas analizadas fueron los correspondientes al modelo entrenado con unigramas, con un 96% para *TPR* y *TNR*. Se considera que esto es así porque es menos probable que dos *tokens* aparezcan juntos (como en el caso del bigrama/mixto) en diferentes *URIs*.

Considerados de forma individual, los *tokens* adquieren mayor sentido para el tipo de análisis que estamos realizando.

Además, en la Tabla 15 se observa que *TF-IDF: Feature Extraction* con Unigrama mejora las métricas obtenidas con respecto al Modelo - *Keys Expertas*, siendo entonces el mejor modelo obtenido hasta este punto.

6.2.1.3 Ensembles

En esta sección se trabajará con diferentes métodos de *ensemble* con el objetivo de obtener mejores resultados en las predicciones de nuestros modelos. La idea básica de esta técnica es que combinar diferentes clasificadores otorga una mejor *performance* que considerando cada uno aisladamente. Dicho de otro modo, los modelos en *ensemble* mejoran la generalización, dado que minimizan el error esperado con respecto al conjunto de entrenamiento.

Este error se puede descomponer en dos tipos de errores llamados *bias* y *variance*. El primer error corresponde a la buena o mala generalización que tiene un modelo con respecto a una muestra de *test* una vez que el mismo fue entrenado. Mientras que la *variance* caracteriza el error que se produce cuando se intenta entrenar el modelo con diferentes conjuntos de datos.

Es conocido que existe un *tradeoff* entre el *bias* y la *variance* cuando se intenta reducir el error que tiene un modelo, porque los intentos de reducir el *bias* como por ejemplo incrementando la cantidad de datos en el entrenamiento, resulta por lo general en un incremento de la *variance* y viceversa: los esfuerzos por reducir la *variance*, por lo general incrementan el *bias*.

En este contexto, los métodos de *ensemble* usualmente reducen el error producido por la *variance*, dejando casi inalterado el *bias*. En este sentido, parece ser una estrategia razonable crear o seleccionar clasificadores con un bajo *bias* pero con una alta *variance*, dado que se puede utilizar luego alguna técnica de *ensemble* para remover la *variance*. Tal como lo expresa [10] lo ideal en términos de *ensembles* de una red neuronal sería un conjunto de clasificadores o redes que no muestren errores coincidentes. Esto quiere decir, que se debe buscar clasificadores que generalicen bien y que no compartan los mismos errores en el *test*.

Existen varios métodos para crear clasificadores que generalicen de forma diferente, con el fin de ser buenos candidatos para ser utilizados adecuadamente en alguno de los métodos de *ensemble*. Esto se logra modificando diferentes parámetros en la etapa del entrenamiento e incluyen: variar la iniciación aleatoria de los *weights*, variar la topología de la red, variar los algoritmos empleados y variar el conjunto de datos de entrenamiento. Cada uno de estos métodos tiene su ventaja y desventaja pero no es el objetivo de este trabajo abordar esta temática.

En este trabajo se decidió construir modelos de *ensemble* con clasificadores que generalicen diferente utilizando básicamente el último método mencionado en el párrafo anterior: realizar variaciones en el conjunto de entrenamiento. Existen varios enfoques para llevar a cabo esto que se describen en [10]: *Sampling data*, *disjoint training sets*, *Boosting and Adaptive resampling*, *Different data sources* y *Preprocessing*. Este último consiste en utilizar diferentes métodos de procesamiento aplicados al conjunto de datos de entrenamiento o diferentes *features* extraídas.

La literatura sobre el tema abarca diferentes métodos para combinar los diferentes modelos o clasificadores. Básicamente se clasifican en dos grandes grupos: por un lado se encuentran aquellos que combinan clasificadores de forma independiente como por ejemplo *Majority Voting* y *Bagging*. Por otro lado se clasifican los que realizan la combinación de forma coordinada a través de una suma ponderada. Dentro de este grupo los más utilizados son *Stacking* y *Boosting*.

Resumiendo, el presente trabajo optó por construir un modelo de *ensemble* eligiendo clasificadores que generalicen diferente variando los datos de entrenamiento. Para

esto, se utiliza el enfoque *Preprocessing* que busca extraer diferentes *features* para cada clasificador. La estrategia abordada para la combinación de los clasificadores consistió en utilizar dos métodos: *Majority Voting* y *Stacking*.

6.2.1.3.1 Stacking

Stacking es un algoritmo de *machine learning* del tipo *ensemble* que tiene el objetivo general de combinar las predicciones de diferentes modelos que resultan útiles en diferentes aspectos para cubrir las falencias de cada uno de ellos.

Las predicciones de los modelos son utilizadas como el *input* para cada capa secuencial y se combinan formando un nuevo conjunto de predicciones.

En este informe, los modelos seleccionados para realizar el *stacking* fueron los siguientes: Modelo - *Keys Expertas*, Unigrama, Bigrama y Mixto.

A continuación se presenta en el Diagrama 1 una visualización de la combinación de los modelos elegidos en el *stacking*:

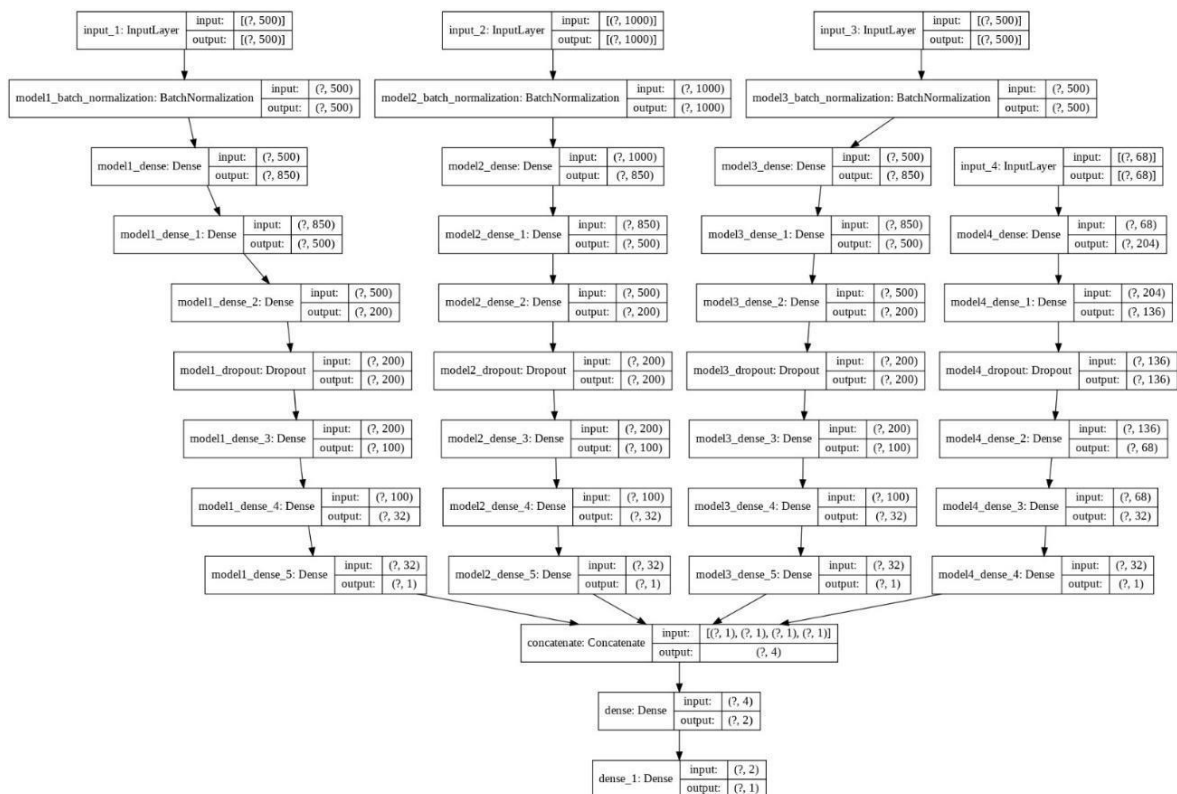


Diagrama 2. Combinación de modelos del *Stacking*

A partir de los modelos seleccionados se realizó una red neuronal cuya arquitectura está compuesta por dos capas de activación: “*relu*” y “*linear*”. El resultado es análogo al de una regresión lineal que toma como *inputs* la combinación de los modelos seleccionados para el *stacking*.

El procesamiento de los datos se continuó efectuando bajo la modalidad *batch*, por lo cual nuevamente se crearon los *generators* para *train*, *test* y *validation*. Ver sección 6.2.1.2.

Resultados obtenidos:

Métricas:

Criterio 1:

TPR: 0.96

TNR: 0.95

Comparación histórica de modelos:

Métricas	<i>TF-IDF</i> Feature extraction: Unigrama	Stacking
<i>TPR</i>	0.96	0.96
<i>TNR</i>	0.96	0.95

Tabla 16. Comparación histórica *TF-IDF: Feature Extraction Unigrama* y *Stacking*

Conclusiones:

Se esperaba que este modelo pudiera superar a los mejores resultados obtenidos hasta el momento (que se corresponden a los del modelo de *TF-IDF: Feature Extraction Unigrama*). Sin embargo, en la Tabla 16 se observa que los mismos fueron ligeramente inferiores. Se cree que los modelos de Bigrama y Mixto para *TF-IDF: Feature Extraction* utilizados en el *Stacking* continúan actuando en detrimento del resultado global.

6.2.1.3.2 Majority Voting

Otro análisis realizado fue el de considerar los tres modelos con mejores resultados obtenidos y realizar una votación por mayoría para todas las *URIs* de forma tal de obtener un nuevo conjunto de predicciones que pudieran ser comparadas con las etiquetas reales.

Los modelos elegidos fueron los siguientes: *TF-IDF: Feature Extraction* Unigrama y Mixto, y Modelo *Keys Expertas*.

La votación se efectuó en forma manual, simplemente considerando el voto mayoritario entre los tres conjuntos de predicciones obtenidas para cada modelo.

Resultados obtenidos:

Métricas:

TPR: 0.96

TNR: 0.95

En este caso no existe ningún criterio elegido para la clasificación de las predicciones porque la misma se efectuó manualmente. Por este motivo presentamos directamente los resultados de la matriz de confusión:

Comparación histórica de modelos:

Métricas	<i>TF-IDF Feature Extraction: Unigrama</i>	<i>Majority Voting</i>
<i>TPR</i>	0.96	0.96
<i>TNR</i>	0.96	0.95

Tabla 17. Comparación histórica *TF-IDF: Feature Extraction Unigrama* y *Majority Voting*

Conclusiones:

Como se puede apreciar en la Tabla 17, el *majority voting* no superó los resultados del modelo *TF-IDF: Feature Extraction* Unigrama. Sin embargo, las métricas de *TPR* fueron iguales, y las de *TNR* muy similares.

6.2.1.4 Comparación global de modelos

	Criterio 1 Test	
	<i>TPR</i>	<i>TNR</i>
Modelo Keys Expertas	0.95	0.95
<i>TF-IDF</i> Keys Expertas	0.87	0.95
Unigramas	0.96	0.96
Bigramas	0.77	0.79
Mixto: Unigrama+Bigrama	0.96	0.95
<i>Stacking</i>	0.96	0.95
<i>Majority Voting</i>	0.96	0.95

Tabla 18. Comparación de métricas global: Modelo - *Keys Expertas*, *TF-IDF: Feature Extraction* (Unigrama, Bigrama y Mixto), *Stacking*, *Majority Voting*

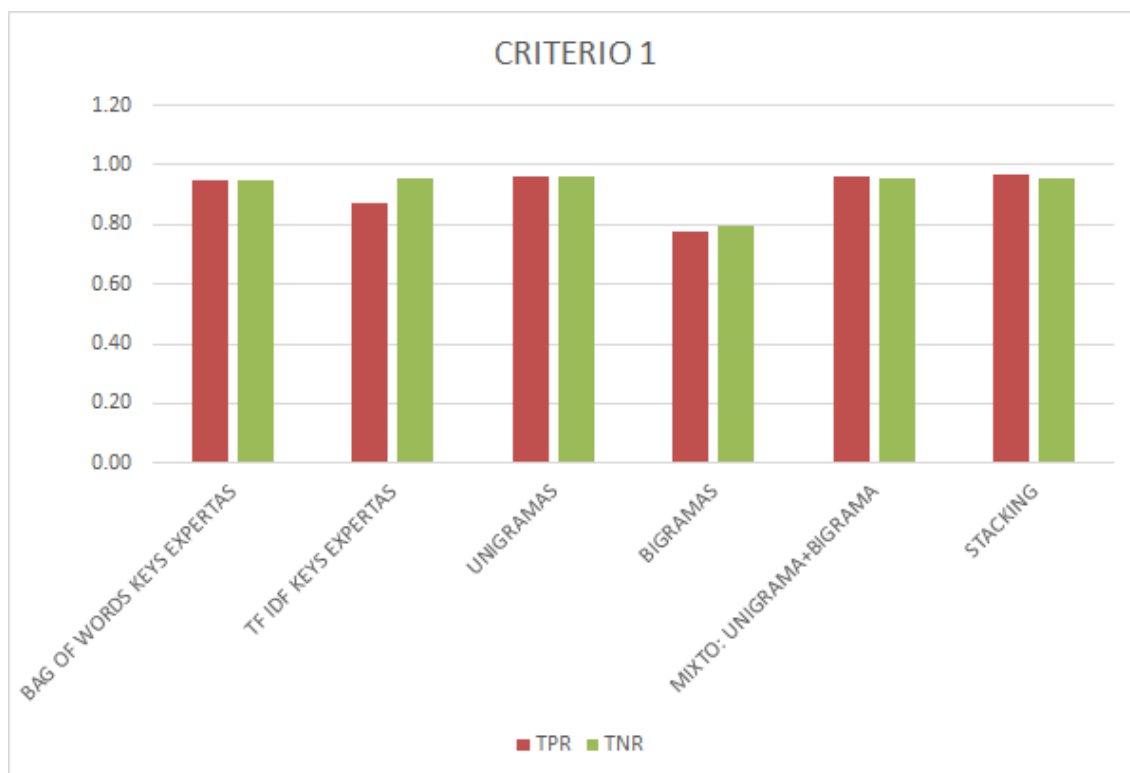


Gráfico 8. Criterio 1 de modelos

Analizando el histórico de los resultados obtenidos en la Tabla 18 y en el Gráfico 8, se puede ver que ninguno de los modelos logró superar los resultados de *TF-IDF: Feature Extraction* Unigrama, utilizando el criterio 1. Los modelos: *Mixto*, *Stacking* y *Majority Voting* obtuvieron las mismas métricas de *TPR* y *TNR*, y esta última jamás superó el 95%.

En cambio, *TF-IDF: Feature Extraction* Unigrama obtuvo 96% en ambas métricas, superando incluso los resultados obtenidos en la investigación tomada como precedente para el presente trabajo.

En el siguiente capítulo se realiza un análisis comparativo entre ambos trabajos.

7. Trabajo de relevamiento

En esta sección se profundiza sobre las diferencias y similitudes de esta investigación con respecto a [7] y de los resultados obtenidos. Como se mencionó en la introducción, [7] realiza un abordaje de los escenarios planteados utilizando *One-Class Classification* y *Anomaly Detection*.

Este trabajo y [7] emplean en la fase de entrenamiento y evaluación de los modelos los dataset CSIC y PKDD. Sin embargo, [7] utiliza también un conjunto de datos privados generados por el tráfico de su propio sitio web (DRUPAL). Al no contar con acceso al mismo, se optó por emplear el dataset *good/bad queries*.

El modelo de *One-Class Classification* definido en [7] consiste en entrenar un modelo con *requests* normales de *HTTP* previamente filtrado el *header* del *request*. Luego utilizan un conjunto de caracteres (*keys*) definidos por un experto en seguridad plausibles de ser encontrados en la sintaxis de un *request* anómalo. Este conjunto de *keys* son utilizadas en el procedimiento de extracción de *features* para modelar la clase válida de su modelo *one-class*.

La representación que realizan de los *requests* previo al entrenamiento consiste en contar el número de veces que es encontrada esa *key* en el *request*.

Como se vió en secciones anteriores este trabajo toma esta idea como punto de partida e incluso utiliza las mismas 64 *keys* utilizadas por [7], incorporando algunas adicionales, detectadas en la etapa de exploración. Además, no solamente se experimentó con el mismo método que se utilizó en el trabajo de [7] para extraer las *features*, sino que también se experimentó con otras técnicas como *TF-IDF*.

En el enfoque de *one class* el clasificador construido en [7] consiste en estimar la función de densidad en el conjunto de entrenamiento utilizando *GMM* (*Gaussian Mixture Models*) y utilizando *EM* (*Expectation Maximization*) para estimar sus parámetros. Una vez obtenida la función de densidad se procede a clasificar el *request* como un “ataque” o “no ataque”, calculando el *threshold* en función de la distancia de *Mahalanobis* entre el centro del *cluster* y el *score* del *request*.

Esta investigación optó por un enfoque completamente diferente, se entrenaron diferentes clasificadores utilizando redes neuronales de tal forma que no solo aprendan que es un *request* normal sino también aquel que corresponde a un ataque. Esta es una gran diferencia con [7] dado que aprende lo que es un *request* válido e identifica el anómalo como una desviación del comportamiento de una *request* normal.

El *threshold* que se definió también difiere del empleado por el trabajo en [7]. En las secciones anteriores se explicó sobre cómo se definieron los modelos trabajados basado en el conocimiento experto, abarcando estos puntos mencionados: algoritmo, *threshold* y criterio de decisión.

El segundo enfoque que se plantea en el artículo que tomamos como referencia experimenta con técnicas como *anomaly detection* usando *n-gram* para incursionar en el modelado de las *signature* de cada atributo de la aplicación *web* para poder identificar posibles ataques. Esto contribuye a poder detectar los ataques *zero-day*. Se incorporó esta idea en esta investigación, pero la implementación se realizó con una estrategia diferente.

La estrategia empleada por [7] consiste en aplicar previamente al entrenamiento dos transformaciones. La primera tiene como objetivo incorporar el hecho de que no todos los *tokens* tienen la misma relevancia, como por ejemplo sucede con las *IPs* que pueden generar una mayor tendencia al *overfitting*. El procedimiento implica primero transformar mayúsculas a minúsculas, remover tildes y reemplazar dígitos por "N". La segunda transformación está vinculada al concepto de "*model field*". El mismo surge de separar el *request* reteniendo únicamente los campos del *request HTTP*, los parámetros y los valores. En este caso la extracción de *features* la realizan utilizando *n-gram* aplicado al *model fields*, con dos formas diferentes: la primera contabiliza la cantidad de ocurrencia del *n-gram* en el *model field* y la segunda contabiliza el número de ocurrencias del *n-gram* dividido el total del *n-gram* que tiene el *model fields*.

La estrategia descrita en el párrafo anterior difiere sensiblemente con este trabajo, tal como se explicó en la sección 6.2.1.2, se aplicó *n-gram* directamente sobre la *URI* y la extracción de las *features* en este enfoque se realizó únicamente aplicando *TF-IDF*.

Otra diferencia se encuentra en la decisión adoptada sobre la distribución definida de los *n-gram* en el conjunto de datos de entrenamiento. Mientras que en [7] se toma cada atributo de los *n-gram* como variables aleatorias independientes, donde los parámetros son aproximados de forma incremental utilizando el algoritmo de *Welford*, este trabajo continúa experimentando con las redes neuronales en la fase de entrenamiento.

Por último, [7] utiliza en el enfoque de *anomaly detection* para el criterio de clasificación de una *request*, la distancia *Mahalanobis* medida entre el *request* y cada *model field*. Es así que se considera una *request* normal si se encuentra entre el valor mínimo y máximo muestreados para cada variable aleatoria de cada tupla formada por el *model field* y los *n-gram* asociados. En esta investigación se decidió mantener el mismo criterio de decisión aplicado en el enfoque de generación de *features* en base al conocimiento experto.

Una de las decisiones más importante de este trabajo implica la utilización de la *URI* para todos los experimentos realizados, a diferencia de muchos trabajos como [7] que utilizan todo o gran parte del *request* de *HTTP*.

Dentro de las métricas para evaluar el desempeño de los modelos de ambos enfoques *One Class* y *Anomaly Detection* [7] utilizó el *TPR* y el *TNR*. En este trabajo se decidió incluir estos indicadores entre otros, para poder analizar y comparar los resultados de ambas investigaciones.

Los resultados presentados por [7] consisten en tres modelos para el enfoque *One Class*: *ModSecurity*, *One-class*: $\lambda = 0, 5$ y una combinación de *ModSecurity* con *One-Class*.

El modelo “*ModSecurity*” definido por [7], considerado como su línea base, no es propiamente un modelo estadístico, dado que corresponde a los datos obtenidos por *ModSecurity* configurado con *OWASP CRS* (versión 2.2.9).

El modelo “*ModSecurity con One-Class*” busca potenciar los resultados de la clasificación incorporando información de ambos expertos (el obtenido por el *WAF* y el de los expertos en la materia).

La combinación consiste en una votación que opera de la siguiente manera: si el *ModSecurity* y el modelo *One-Class* coinciden (ambos dicen que es un ataque o normal) entonces es clasificado como tal. En caso de discrepancia, si el *ModSecurity* lo califica como no ataque y el *One-Class* como ataque, se prioriza el primero. Si el *ModSecurity* lo califica como ataque y *One-Class* como no ataque, se prioriza este último.

Este trabajo implementa de forma más profunda el *voting*, dado que se aplica una votación entre los mejores modelos estadísticos elegidos independientemente de la forma en que se extrajeron las *features*. Incluso se aplicaron técnicas más complejas de *meachine learning* para este propósito como el *stacking*. El desarrollo y los resultados fueron tratados en la sección 6.2.1.3.1.

El segundo enfoque presentado por [7] utiliza *n-gram* de 1 a 5 sobre *Model Field* mientras este trabajo se centra en explorar únicamente *n-gram* de 1 a 2.

A lo largo de esta investigación se utilizaron los resultados del trabajo realizado por [7] como referencia, teniendo presente las diferencias y similitudes en las decisiones de diseño de los experimentos.

A continuación y de modo ilustrativo se presenta una tabla comparativa que resume los métodos y decisiones entre [7] y esta investigación.

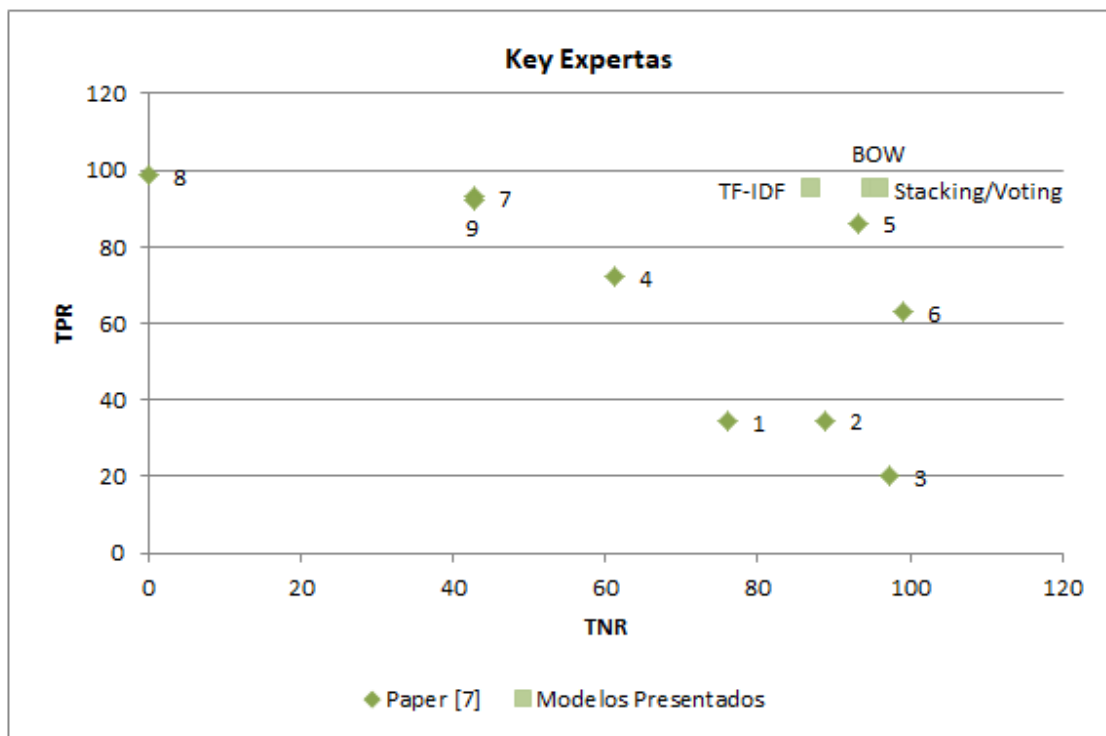
	Paper [7]		Este trabajo	
	<i>One Class</i>	<i>Anomaly Detection</i>	<i>Bag of Word</i>	<i>N - Gram</i>
<i>DataSets</i>	<i>CSIC, PKDD, DRUPAL</i>		<i>CSIC, PKDD, GOOD/BAD QUERIES</i>	
Objeto de Estudio	<i>Requests de HTTP sin IP, protocolo, cookies, proxies del header</i>	<i>Models Field</i>	<i>URIs de HTTP</i>	<i>Tokens de la URIs HTTP aplicando Tabla 13</i>
Composición del DataSet de Entrenamiento	Tráfico Normal	Tráfico Normal	Tráfico Normal y Anormal	
Extracción de Features	Tabla I de [7]	<i>N-Gram sobre Models Field</i>	Tabla I de [7] más .exe, .dll, ', cgi-bin	<i>N-Gram sobre tokens</i>
Representación del Corpus	<i>Count</i>	<ul style="list-style-type: none"> • Cantidad de ocurrencia del <i>n-gram</i> en el <i>model field</i> • Número de ocurrencias del <i>n-gram</i> sobre el total del <i>n-gram</i> que tiene el <i>model fields</i> 	<i>Count y TF - IDF</i>	<i>TF - IDF</i>

Algoritmo	<i>Gaussian Mixture Models (GMM)</i> con <i>Expectation Maximization (EM)</i> para estimar los parámetros del <i>GMM</i>	<i>N-gram</i> son V.A caracterizadas por la tupla $d = (\mu, \sigma, \max, \min, N_d, H_d)$ cuyos parámetros son aproximados por el algoritmo de Welford	Redes neuronales generativas antagónicas (<i>GAN</i>)
Criterio de Decisión	Cantidad de <i>FP</i> aceptable ($\lambda=0,5$)		Criterio 1
Test	<ul style="list-style-type: none"> • CSIC cuando entrenan con <i>PKDD</i> y <i>DRUPAL</i> • <i>PKDD</i> cuando entrenan con CSIC y <i>DRUPAL</i> • <i>DRUPAL</i> cuando entrenan con CSIC y <i>PKDD</i> 		Porcentaje de CSIC <i>PKDD GOOD/BAD QUERIES</i> unificados
Baseline	<i>ModSecurity</i> configurado con <i>OWASP CRS</i> , versión 2.2.9.		Modelo <i>Keys Expertas</i>
Métricas de Evaluación		<i>TPR, TNR</i>	<i>TPR, TNR, FPR, FNR, Accuracy</i>
Combinación	<i>Voting Baseline</i> y <i>One Class</i>		<i>Voting</i> y <i>Stacking: Baseline</i> y todos nuestros modelos.

Tabla 19. Resumen comparativo entre paper [7] y el presente trabajo

En los siguientes gráficos se representa el *TNR* y el *TPR* de todos los modelos de [7] (que se encuentran en la Table III) y de esta investigación. El objetivo es visualizar que tan bien predicen ambas clases todos los clasificadores. En el gráfico 9, se graficaron los datos de los clasificadores que fueron construidos en base a la información obtenida de los expertos en ciberseguridad. Mientras que en el gráfico 10, se representaron aquellos que utilizaron en su entrenamiento *n-gram*.

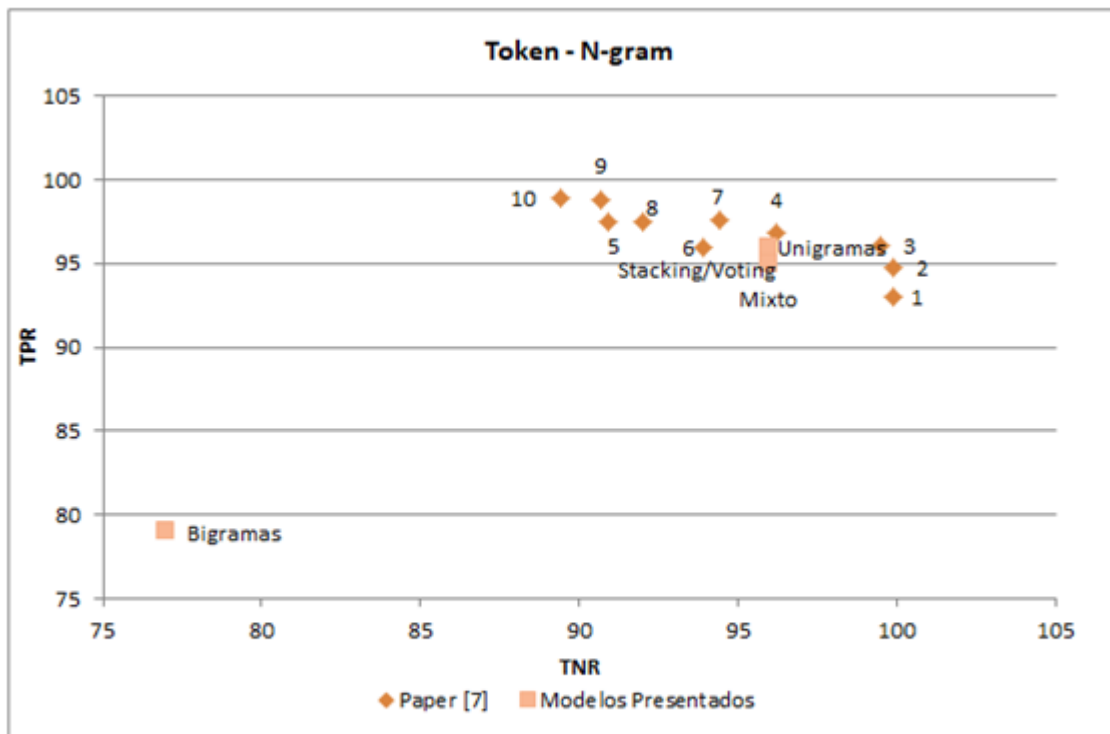
En ambos gráficos se representan los *TPR* y *TNR* de *Stacking* y *Voting* por ser modelos de *ensemble* que consumen clasificadores de ambos enfoques.



1 - ModSecurity, CSIC2010, 2 - One-class: $\lambda = 0.5$, CSIC2010, 3 - Combined OC-MS, CSIC2010, 4 - ModSecurity, DRUPAL, 5 - One-class: $\lambda = 0.5$, DRUPAL, 6 - Combined OC-MS, DRUPAL, 7 - ModSecurity, ECML/PKDD2007, 8 - One-class: $\lambda = 0.5$, ECML/PKDD2007, 9 - Combined OC-MS -ECML/PKDD2007.

Gráfico 9. Keys Expertas

En el Gráfico 9 se observa que los modelos presentados en este trabajo bajo un enfoque de extracción de *features* con *keys* expertas en base al estado del arte, son en general superiores a [7]. En especial, el Modelo-Keys Expertas, consigue muy buenos resultados tanto en el aprendizaje de un ataque (95%) como de una *URI* normal (95%). Los modelos (8), (6) y (3) superan en alguna de las métricas mencionadas al *BoW* presentado, sin embargo, el modelo (8) tiene un sesgo importante para predecir un ataque, pero falla al momento de predecir el tráfico normal. Por otro lado, los modelos (3) y (6) se encuentran sesgados hacia los *true negative* con una *performance* en la predicción de la clase positiva muy por debajo del *BoW*. Además estos modelos requieren información adicional del *WAF* al ser una combinación del modelo *One-Class* con el *ModSecurity*.



1 - n=1 - CSIC2010, 2 - n=2 - CSIC2010, 3 - n=3 - CSIC2010, 4 - n=4 - CSIC2010, 5 - n=5 - CSIC2010, 6 - n=1 - DRUPAL, 7 - n=2 - DRUPAL, 8 - n=3 - DRUPAL, 9 - n=4 - DRUPAL, 10 - n=5 - DRUPAL

Gráfica 10. Token N-Gram

En el Gráfico 10 se representan los resultados del *TPR* y *TNR* para todos los modelos que fueron construidos en base a *tokens* y que fueron entrenados utilizando *n-gram*. En este caso los resultados de este trabajo no son tan claramente superiores como en el enfoque de *Keys Expertas*. Se puede observar que el clasificador entrenado en esta investigación utilizando un modelo bigrama obtiene muy malos resultados a diferencia del presentado por [7]. Incluso su *performance* en términos de los aciertos en la predicción de la clase positiva y negativa es sensiblemente inferior a todos los modelos entrenados en el trabajo [7].

Es importante destacar que los modelos que fueron puestos a prueba utilizando *n-gram* en este trabajo no fueron entrenados. Como se explicó en secciones anteriores, la topología de la red utilizada es exógena ya que no surge del conjunto de entrenamiento, ni tampoco fueron optimizados sus parámetros. A diferencia de [7] que presentan los resultados del modelo *n-gram* 1 a 5 con los demás parámetros optimizados.

8. Conclusiones finales

El presente trabajo mejoró los resultados obtenidos en cuanto a las métricas de *TPR* y *TNR*, en algunos de los experimentos llevados a cabo por [7]. En aquella investigación se trabajó sobre conjuntos de datos considerados en forma individual.

Sin embargo, a diferencia de esto, en este trabajo se utilizó un *dataset* formado por múltiples conjuntos de datos, algunos de los cuales fueron utilizados también por [7], y otros fueron agregados.

Esto representa una contribución significativa al estado del arte, ya que por estas razones la variabilidad de los datos fue mayor que con respecto a la observada al analizar cada conjunto de datos en forma individual, y además, el número de registros fue superior, reduciendo el sesgo.

Otra contribución fue la inclusión de determinadas *keys* al dominio experto en el que se basa la investigación precedente. Las *keys* añadidas son el resultado de un análisis del *corpus* mediante *scripts* y conocimiento experto ajeno a dicha investigación.

La primera dificultad, propia de cuando se trabaja con datos desestructurados, tuvo lugar durante el tratamiento de los datos, al momento de intentar identificar las *keys* expertas en el *corpus* para construir el *BoW*. Por ejemplo, pese al desarrollo de *scripts* y expresiones regulares, no se consiguió que aquellas palabras de las *URIs* que contuvieran “*script*” no se contabilizaran en la posición correspondiente a esta *key* experta en el *BoW*.

Por otro lado, un determinado porcentaje de *URIs* contenía ataques codificados con caracteres en hexadecimal que tampoco pudieron ser tratados sin que los resultados de los modelos no se vieran afectados negativamente.

Adicionalmente, dada la alta sensibilidad de los datos que se encuentra en este tipo de *datasets*, fue particularmente difícil encontrar conjuntos de datos que estuvieran etiquetados.

Debido a la naturaleza del problema que se está abordando, todos los *datasets* que fueron considerados para la presente investigación poseían un *corpus* cuyas clases en el conjunto de datos estaban desbalanceadas. Esto presentó un gran desafío al momento de entrenar las redes neuronales sin que su *performance* se viera afectada de algún modo.

Finalmente, al unificar los datos en un único *dataset*, comenzaron a generarse problemas de cómputo, dado que las herramientas disponibles no contaban con recursos suficientes para realizar el procesamiento. Esto implicó la creación de un algoritmo propio específico para el procesamiento de los datos por bloques.

En cuanto al análisis de los resultados de los modelos, el mejor modelo obtenido fue el de *TF-IDF: Feature Extraction* Unigrama, obteniéndose un 96% para *TPR* y *TNR*.

Si bien en la investigación precedente, al considerar el *dataset* específico *PKDD* y utilizar *One-class classification*, se alcanzó un *TPR* de 98,6%, el *TNR* para dicho caso fue 0.

Por otro lado, cuando se utilizaron técnicas de *n-grams* con unigramas ($n=1$), en la investigación precedente se obtuvo, para el *dataset* de CSIC un *TPR* de 99,9% y un *TNR* de 93%, y para DRUPAL, un *TPR* de 93.9% y un *TNR* de 95.9%. En el primer caso los resultados no se encuentran tan balanceados, y en el segundo caso son inferiores a los obtenidos con el *dataset* unificado en el presente trabajo.

Se concluye que se cumple con el objetivo de presentar un modelo predictivo para la clasificación de URIs como legítimas o anómalas mediante la utilización de datos no privatizados, que sirva como punto de partida para comparar posteriormente contra otros modelos obtenidos a partir de datos privatizados de logs.

Este modelo obtuvo métricas de *TPR* y *TNR* balanceadas y con una buena *performance*, de modo tal que el modelo es capaz de identificar correctamente tanto los *TP* como los *TN*.

9. Trabajo a futuro

Se considera que sería útil extender el presente trabajo al desarrollo de determinadas tareas con el objetivo de mejorar los resultados expuestos:

- Incorporar una mayor cantidad de datos para la fase de entrenamiento de los modelos.
- Considerar mayor información de los *requests* (y no solamente las *URIs*).
- Profundizar el análisis del origen de los errores en los modelos para optimizar las métricas. En especial, considerar los *FP* y *FN*.
- Optimizar la arquitectura de los modelos, tanto en sus parámetros como en su topología.
- Incorporar otras técnicas de extracción de *features* tales como *Topic Modelling* y *NLP*.
- Profundizar en el análisis de la penalización y el impacto de los distintos errores (*FPS* y *FNs*) en relación con los diferentes tipos de ataques.
- Presentar un modelo multiclase que no solo prediga si la *URI* es anómala o no, sino que además determine qué tipo de ataque es. Previendo este tipo de análisis, se almacenaron para algunos *datasets* que disponían de esta información, los tipos de ataque.

10. Referencias Bibliográficas

- [1] A. P. V. Carmen Torrano Giménez and G. Á. Marañón, “*HTTP DATASET CSIC 2010*”, Enero 2012 [Online]. Available: <https://www.isi.csic.es/dataset/>
- [2] “*Analyzing web traffic: Ecml/pkdd 2007 discovery challenge*, Septiembre 2007” [Online]. Available: <http://www.lirmm.fr/pkdd2007-challenge/>.
- [3] *kdnuggets*, “*Machine Learning-driven Firewall*”, Febrero 2007 [Online]. Available: <https://www.kdnuggets.com/2017/02/machine-learning-driven-firewall.html>
Accesed on: Abril 03, 2020
- [4] J. Torres, “*Deep Learning. Introducción Práctica con Keras*,” Tercera ed. Barcelona, ES: Universitat Politècnica de Catalunya, 2018, pp. 111-145. [Online]. Available: <https://www.amazon.es/dp/198312981X>
- [5] “*Uniform Resource Identifier (URI): Generic Syntax*”, IETF RFC 3986, enero 2005
- [6] G. James, D. Witten, T. Hastie y R. Tibshirani, “*An Introduction to Statistical Learning with Applications in R*”, octava ed. NY, USA, 2017, ch. 4, sec.4.4.3, pp. 147–149.
- [7] G. Betarte, E. Giménez, R. Martínez y Á. Pardo “*Machine learning-assisted virtual patching of web applications*” Instituto de Computación, Facultad de Ingeniería Universidad de la República, Uruguay, Marzo 2018
- [8] T. Fawcett, “*ROC Graphs: Notes and Practical Considerations for Researchers*”, HP Laboratories, MS 1143, 1501 Page Mill Road, Palo Alto, CA 94304, Marzo 2004
- [9] J. Ramos, “*Using TF-IDF to Determine Word Relevance in Document Queries*”, Department of Computer Science, Rutgers University, 23515 BPO Way, Piscataway, NJ, 08855, [Online]. Available: <http://citeseerx.ist.psu.edu/index>
- [10] A. Sharkey, *Combining artificial neural nets: ensemble and modular multi-net systems*, in: *Multi-Net Systems*, SpringerVerlag, 1999, sec. 1.1 pp. 5–14.

[11] OWASP Top 10 – 2017 *The Ten Most Critical Web Application Security Risks, 2017*.
[Online]. Available: <https://owasp.org/>

[12] W. B. Cavnar “*Using An N-Gram-Based Document Representation With A Vector Processing Retrieval Model*”, 379 Brookside, Ann Arbor MI 48105, Enero 1994.

Anexo 1

OWASP Top 10:

1. Inyección

Está ligada a los ataques desde aplicaciones *web* hacia bases de datos. Consiste en inyectar sentencias *SQL* con fines maliciosos desde formularios *web* que no estén preparados para no interpretarlos como algo más que simple texto.

2. Pérdida de autenticación

Se relacionan con la suplantación de identidad de un usuario en una aplicación *web*, ya sea por mala gestión de claves, cierres de sesión o expiración de la misma.

3. Exposición a datos sensibles

Se trata de aplicaciones *web* que no tienen un buen manejo de datos sensibles, por ejemplo exponiendo información confidencial que podría llevar al mal uso de la misma.

4. Entradas XML

En este caso se trata de vulnerabilidades relacionadas con el analizador *XML* de una página *web*, por ejemplo para ganar acceso a unidades externas, como puede ser un disco duro, para obtener datos confidenciales.

5. Control de acceso

Este tipo de vulnerabilidades están relacionadas al mal manejo de la autorización dentro de una aplicación, por ejemplo no controlando permisos que podrían llegar a otorgar privilegios de administración a usuarios no autorizados. Un ejemplo sería modificando parte de una url para acceder a un recurso directamente si validar los permisos que se requerirían para su acceso.

6. Mala configuración de la seguridad

Este riesgo está ligado a la mala gestión de cumplimiento de estándares y marcos de seguridad, por ejemplo no siguiendo políticas de buenas prácticas, de aplicación de parches, control de datos confidenciales, trazas de error sin controlar, etc.

7. Secuencia de comandos en sitios cruzados (XSS)

La finalidad de este tipo de ataques es afectar al usuario de la aplicación *web*, ya sea desde tomando control de su equipo hasta la obtención de datos personales.

Esto se realiza colocando código malicioso en el navegador del usuario inyectándose secuencias de comandos maliciosos (XSS) y no en los servidores de dicha aplicación.

8. Deserialización insegura

Esta vulnerabilidad permite la ejecución de código de forma remota en servicios *web*.

9. Uso de componentes con vulnerabilidades conocidas

Esta vulnerabilidad está asociada a que los atacantes puedan aprovecharse de algún componente que forme parte de la aplicación *web* que haya sido comprometido en alguna de sus versiones y no se esté llevando un control de las actualizaciones de los mismos.

10. Insuficiente registro y monitoreo

Muchas veces los ataques demoran semanas o meses en ser detectados, en este periodo y de la mano del poco monitoreo de actividad inusual puede suceder que los atacantes tomen control de componentes de nuestra aplicación sin ser notados.

Anexo 2

<i>Key Experta</i>	<i>Total de URIs</i>	<i>URIs Normales</i>	<i>URIs Anormales</i>	<i>Proporción URIs Anormales en el Total</i>
<i>-></i>	-	-	-	0
<i>User- Agent:</i>	-	-	-	0
<i>Accept:</i>	-	-	-	0
<i>/</i>	3.170.041	2.912.801	257.240	8%
<i>/c</i>	111.024	100.951	10.073	9%
<i>script</i>	228.758	197.188	31.570	14%
<i>order</i>	2.236	1.917	319	14%
<i>alter</i>	357	304	53	15%
<i>and</i>	20.409	17.304	3.105	15%
<i>:</i>	51.200	42.688	8.512	17%
<i>commit</i>	173	144	29	17%

<i>bash_history</i>	136	112	24	18%
<i>or</i>	161.681	132.084	29.597	18%
-	270.793	220.503	50.290	19%
<i>table</i>	1.556	1.237	319	21%
<i>.dll</i>	3.724	2.864	860	23%
<i>into</i>	794	595	199	25%
<i>shell</i>	898	647	251	28%
<i>between</i>	1.644	1.184	460	28%
&	522.654	334.930	187.724	36%
<i>insert</i>	8.320	5.293	3.027	36%
<i>count</i>	4.761	3.024	1.737	36%
<i>cmd</i>	5.358	3.395	1.963	37%
=	656.997	413.875	243.122	37%
<i>password</i>	14.466	8.818	5.648	39%
<i>cgi-bin</i>	3.915	2.381	1.534	39%

<i>mail=</i>	13.469	8.053	5.416	40%
<i>cn=</i>	533	316	217	41%
<i>exec</i>	7.086	4.160	2.926	41%
<i>union</i>	3.826	2.152	1.674	44%
<i>*/</i>	2.023	1.073	950	47%
<i>where</i>	2.696	1.428	1.268	47%
<i>from</i>	4.751	2.372	2.379	50%
<i>winnt</i>	2.485	1.227	1.258	51%
<i>select</i>	5.183	2.531	2.652	51%
<i>url=</i>	603	293	310	51%
<i>/*</i>	2.477	1.130	1.347	54%
<i>upper</i>	259	118	141	54%
<i>0%</i>	273	103	170	62%
<i>/</i>	6.203	2.302	3.901	63%
<i>+</i>	262.191	79.024	183.167	70%

<i>#¿NOMB RE?</i>	38	11	27	71%
*	11.028	2.573	8.455	77%
<i>.exe</i>	5.000	934	4.066	81%
<i>passwd</i>	6.947	1.149	5.798	83%
<i>alert</i>	6.972	880	6.092	87%
\$	12.039	1.355	10.684	89%
(23.473	1.459	22.014	94%
)	23.947	1.459	22.488	94%
<>	52	3	49	94%
'	12.464	657	11.807	95%
//	492	23	469	95%
;	14.186	630	13.556	96%
<i>href</i>	569	23	546	96%
<	23.831	652	23.179	97%
>	25.836	642	25.194	98%

<i>onmouseo ver</i>	199	1	198	99%
<i>../</i>	24.380	23	24.357	100%
<i>%0a</i>	7.637	-	7.637	100%
<i>etc/pass wd</i>	4.154	-	4.154	100%
<i>document .cookie</i>	3.279	-	3.279	100%
<i>javascript :</i>	3.003	-	3.003	100%
<i>objectclas s</i>	509	-	509	100%
<i>path/chil d</i>	110	-	110	100%
<i>#include</i>	62	-	62	100%
<i>'</i>	18	-	18	100%
<i>"</i>	11	-	11	100%
<i><!--</i>	1	-	1	100%