Universidad de Buenos Aires
Facultades de Ciencias Económicas,
Ciencias Exactas y Naturales e Ingeniería

Maestría en Seguridad Informática

Tesis de Maestría

Detección de contenido malicioso mediante técnicas de Machine Learning en las redes sociales

Autora: Ing. Edith Rivero
Tutor: Mg. Juan Devincenzi

Año 2017

Cohorte 2014

Declaración Jurada

Por medio de la presente, el autor manifiesta conocer y aceptar el Reglamento de Tesis vigente y que se hace responsable que la totalidad de los contenidos del presente documento son originales y de su creación exclusiva, o bien pertenecen a terceros u otras fuentes, que han sido adecuadamente referenciados y cuya inclusión no infringe la legislación Nacional e Internacional de Propiedad Intelectual.

FIRMADO
Edith Pamela Rivero Tupac
DNI (Perú): 70303536

Resumen

Desde el inicio de la era de la información, se han generado grandes volúmenes de datos en distintos medios y una de las fuentes más atractivas lo conforman las redes sociales. Entre los usuarios de estas redes, también están los malintencionados que podrían causar perjuicios o materializar delitos informáticos cuando publican contenido sospechoso (descarga de malware, estafas, phishing, pornografía, etc.) [1]. La red social Facebook, que es la más utilizada en Latinoamérica se ha visto explotada por estafas y otro tipo de contenido malicioso [2]. Por parte de Facebook, los esfuerzos de detección de contenido sospechoso iniciaron en el 2011 mediante la implementación de un sistema de detección basado en procesos de descubrimiento de patrones sobre los volúmenes de datos y Machine Learning para hacer predicciones sobre nuevo contenido en base a los patrones conocidos [3].

Sin embargo, a la fecha de esta investigación, se puede observar contenido que evade este sistema debido a la diversidad tanto en idiomas y contextos culturales, y a esto se suma la astucia de los delincuentes que constantemente crean nuevas formas para publicar contenido malicioso.

La presente investigación estará enfocada en la detección de contenido malicioso publicado en Facebook, mediante el desarrollo de un modelo de Machine Learning recopilando datos públicos de páginas y grupos con contenido en español y en el contexto latinoamericano. De tal manera que los usuarios puedan hacer uso de este modelo a través de un sitio web, para recibir la respectiva alerta de detección que permita evitarla vulneración de su seguridad y la posible materialización de delitos informáticos.

Palabras clave: Machine Learning, Facebook, malware, phishing, contenido malicioso.

Índice

Introducción 3
CAPÍTULO I: Contenido malicioso en las Redes Sociales 4
1.1. Contenido malicioso y delitos Informáticos 4
1.2. Redes Sociales 5
1.3. Red Social Facebook 6
1.3.1. Publicación de contenido 6
1.3.2. Contenido malicioso 7
1.3.3. Medidas de protección 9
CAPÍTULO II: Ciencia de los datos 10
2.1. Machine Learning (ML) 11
2.1.1. Aprendizaje No supervisado 12
2.1.2. Aprendizaje Supervisado 12
2.1.2.1. Aplicaciones 14
2.1.2.2. Selección de características 15
2.1.2.3. Support Vector Machine (SVM) 16
2.2. Modelos de ML en la detección de contenido malicioso 17
2.3. Modelos de ML en el sistema de Facebook 19
2.4. Procesamiento de contenido con URLs 21
CAPÍTULO III: Detección de contenido malicioso en Facebook 23
3.1. Estructura de servicios y APIs 23
3.2. Proceso de recolección de datos 24
3.3. Resultados de la distribución de datos 27
3.4. Extracción de características 35
CAPÍTULO IV: Construcción del modelo de detección 39
4.1. Normalización de datos 42

4.2. Etiquetado de datos 45
4.3. Representación gráfica de las características 46
4.4. Características relevantes 51
4.5. Codificación del modelo 52
4.6. Validación cruzada 53
4.7. Implementación del modelo en un sitio web 54
5. Conclusiones 57
6. Anexos 59
7. Bibliografía y Referencias 85

Introducción

La cantidad de datos en las redes sociales está creciendo de forma inconmensurable y existen usuarios que abusan de estos sitios para compartir contenido malicioso. A pesar de que los sistemas de detección implementados están alertando correctamente, aún existe contenido malicioso que evade su detección especialmente en el contexto de Latinoamérica e idioma español. Es así que resulta importante robustecer o constituir nuevos mecanismos de detección en ese contexto. En la presente investigación se presenta un modelo de Machine Learning para la detección de contenido malicioso elaborado con la recolección de datos actuales de la red social Facebook, específicamente contenido cenuros, que puede materializar una intención maliciosa a la simple acción de un clic por parte del usuario.

Se abordará la identificación de contenido que es considerado como malicioso y su relación con los delitos informáticos; seguidamente se revisará las bases de la construcción de un modelo de aprendizaje de Machine Learning. Esto permitirá identificar los elementos necesarios para la recolección de datos y posteriormente mostrar los resultados del modelo y las estadísticas encontradas. Finalmente, se presentará la implementación de un sitio web que acogerá el modelo para recibir consultas de las URLs y respectivamente alertar al usuario sobre contenido malicioso.

CAPÍTULO I: Contenido malicioso en las Redes Sociales

El contenido que todo usuario encuentra mientras accede a la web puede aprovecharse como información valiosa para el usuario, como también puede afectar su experiencia de navegación cuando se ve afectado por aparición de algún malware, redirecccionamientos a otros sitios web no solicitados, solicitudes para instalar complementos no necesarios y peor aún cuando es atacado por robo de identidad o credenciales de cuentas bancarias. Todo este conjunto de eventos perjudiciales al usuario conforma el contenido malicioso que generalmente es propagado en las redes sociales y que cada país intenta controlar o evitar su aparición mediante la implementación de legislaciones.

1.1. Contenido malicioso y delitos Informáticos

El contenido malicioso está incrementando en Latinoamérica. Según el informe del Observatorio de Delitos Informáticos de Latinoamérica (ODILA) [4], el acceso indebido a datos o sistemas, los fraudes y estafas son los más denunciados. Similarmente, de acuerdo con el informe de Symantec [5] también se produjo un incremento de ataques de secuestro de datos conocido como *ramsonware*, los cuales en su mayoría llegan a través mensajes de correo electrónico o durante la navegación en Internet.

Es importante reconocer que cada país tipifica los delitos con sus respectivas penalidades de acuerdo con la consideración del valor de los datos y las consecuencias de pérdida de privacidad, confidencialidad e integridad. La mayor parte de las legislaciones en el campo de la informática se definen después de ocurrido el incidente luego de una lección aprendida del mismo.

A continuación se presenta una lista de los delitos informáticos que se han identificado en Latinoamérica y que pueden producirse durante la navegación en internet:

a) Acceso ilícito: A todo o parte de un sistema informático con vulneración de las medidas de seguridad.

b) Atentado a la integridad de los datos informáticos: Daño y toda alteración (eliminación, deterioro, inaccesibilidad) a los datos informáticos.

c) Atentado a la integridad de los sistemas informáticos: Cuando se inutiliza total o parcialmente un sistema informático, se entorpece o imposibilita su funcionamiento o la prestación de sus servicios.

d) Proposiciones a niños, niñas y adolescentes con fines sexuales: A través de internet o medio análogo contacta con un menor de edad para obtener de éste, material pornográfico o para llevar a cabo actividades sexuales.

e) Tráfico ilegal de datos: El que crea, ingresa o utiliza indebidamente una base de datos para comercializar o traficar.

f) Fraude informático: Procura un provecho ilícito en perjuicio de un tercero la alteración y clonación de datos o sistemas informáticos.

g) Suplantación de identidad: Cuando se reemplaza la identidad de una persona natural o jurídica ocasionando algún perjuicio material o moral.

h) Abuso de mecanismos y dispositivos informáticos: El que fabrica, distribuye u obtiene programas informáticos, contraseñas, códigos de acceso o cualquier otro dato para cometer delitos informáticos.

Tomando en cuenta las legislaciones, se considera contenido malicioso a: Todo contenido que a través de la navegación web permite transgredir estas legislaciones y a todo contenido con presencia de malware y spyware. Se considera el spyware porque actúan silenciosamente enviando información a terceros sin el conocimiento del usuario, aunque explícitamente no causan perjuicio a datos y sistemas.

1.2. Redes Sociales

Una red social en su básico sentido es un conjunto de relaciones entre actores. Estas relaciones pueden representar parentesco, amistad, interés, etc.

Una de las aplicaciones de esta teoría, corresponde a:*Online Social Networks (OSN)*traducido como "redes sociales en línea", que son plataformas web donde las personas que tienen algún interés en común se conectan a través de credenciales y perfiles [6]. Los usuarios tienen al

alcance la información de sus pares relacionados, así como notificaciones de alguna actualización de esta información. Las OSN pueden clasificarse de acuerdo con el tipo de relación que existe, las cuales pueden ser:(1) profesionales, (2) de seguidor a admirado y (3) de amistad entre personas.

1.3. Red Social Facebook

Facebook (www.facebook.com) es una OSN de acceso gratuito, cuyo funcionamiento inicial estuvo enfocado solo a personas, para permitirles crear un perfil de usuario y realizar conexiones con otros. Estas conexiones pueden ser directas, mediante una aceptación de una solicitud de amistad o indirectas cuando los usuarios pertenecen a un grupo en común.

El funcionamiento se amplió para que las organizaciones y empresas puedan definir un perfil particular y realizar publicaciones, organizar eventos e iniciar campañas con algún anuncio. Adicionalmente, proporciona complementos que se integran a los juegos y aplicaciones para compartir información dentro de la red social. Todo esto conforma el negocio publicitario de Facebook.

La popularidad en su utilización y la cantidad de participantes registrados se ha incrementado ponderadamente en el tiempo. A la fecha de esta investigación ha alcanzado los dos mil millones de usuarios. En la Figura 1. se observa una alta inserción en Latinoamérica y con el más alto porcentaje de aumento en Argentina.

1.3.1. Publicación de contenido

Entre los tipos de contenido diferentes al texto plano que se pueden compartir en Facebook son:

- Enlaces: URLs a sitios web externos.
- Evento: Son creados a través de una funcionalidad de Facebook.
- Video, imagen (plana y animada): Subidos por el usuario u organización.
- Emoticonos, etiquetas o stickers animados: Se pueden elegir a través de Facebook.

Latin America Loves Facebook

By some measures, the social network's reach is highest in Argentina

March 2, 2016 | Social Media

SHARE EMAIL PRINT

Facebook is by far the top social site in Latin America, as in many markets around the world.

Facebook Users and Penetration in Latin America, by Country, 2014-2019

	2014	2015	2016	2017	2018	2019
Facebook users (millions)						
Brazil	72.0	79.0	87.0	92.5	94.8	97.0
Mexico	40.0	45.5	51.8	56.9	61.0	64.7
Argentina	18.2	19.7	20.9	21.7	22.4	23.0
Other	63.9	73.2	81.5	87.5	92.9	97.4
Latin America	**194.1**	**217.5**	**241.1**	**258.6**	**271.1**	**282.2**
Facebook user penetration (% of social network users)						
Argentina	94.8%	94.8%	96.3%	96.3%	96.4%	96.5%
Mexico	94.1%	94.5%	94.7%	94.8%	94.9%	95.0%
Brazil	92.2%	91.4%	93.3%	94.5%	94.2%	94.1%
Other	89.6%	88.4%	90.9%	91.1%	91.2%	91.1%
Latin America	**91.9%**	**91.3%**	**93.0%**	**93.5%**	**93.5%**	**93.4%**
Facebook user penetration (% of internet users)						
Mexico	67.3%	70.0%	73.3%	75.3%	75.8%	76.2%
Brazil	66.9%	69.5%	72.6%	75.0%	75.3%	75.5%
Argentina	67.0%	68.0%	70.0%	71.0%	72.0%	72.6%
Other	57.6%	61.0%	64.3%	66.0%	67.5%	68.6%
Latin America	**63.6%**	**66.4%**	**69.5%**	**71.4%**	**72.3%**	**72.9%**

eMarketer last estimated social media usage in Latin America in July 2015. Facebook is set to reach 39.0% of the total population of the region this year—equivalent to 69.5% of internet users and a dramatic 93.0% of social network users.

By some measures, Facebook's reach is highest in Argentina, where 96.3% of social network users and 47.6% of the total population will use the site this year.

Figura 1. Facebook en Latinoamérica. Fuente [7]

De todos estos tipos de contenido, los enlaces a sitios web son los que menos se pueden controlar debido a que su contenido es dinámico y pueden variar en distintos momentos temporales. Esto significa que el contenido mostrado por el sitio web en el instante de la publicación puede ser distinto al contenido observado posteriormente e inclusive convertirse en contenido malicioso.

1.3.2. Contenido malicioso

La popularidad que tiene Facebook también conforma un punto de atracción para los usuarios malintencionados. En el 2011, se dio a conocer un ataque de phishing [8] donde se copió el logo de Facebook y se creó un evento público que permitía a un usuario consultar si fue bloqueado por otros usuarios. En ese evento se adjuntó una URL cuyo destino era un sitio web con la misma vista de Facebook y pedía al usuario completar sus credenciales. Las credenciales fueron robadas y un promedio de diez

millones de usuarios fueron afectados. Otro ataque conocido como *Clickjacking*[1] aprovechó del auge de la reproducción de videos y en Facebook se compartieron enlaces con la imagen de un reproductor de video, en algunos casos similar al reproductor de Youtube. Cuando el usuario hacia clic en el video era redirigido a un sitio web que le solicitaba la instalación de un Plugin (complemento para el explorador) para visualizar el video; sin embargo, se trataba de la instalación de un malware que realizaba publicaciones automáticas en el perfil del usuario [9].

El ataque del malware KoobFace [10] fue el más complicado de contrarrestar, debido al éxito en su propagación en Facebook como en otras redes sociales. La infección de esta *botnet*[2] inicia con una publicación cuyo mensaje incluye un enlace a un video. Dentro del sitio se solicita la descarga de un ejecutable (.exe) que al ejecutarse se muestra como un componente de Adobe Flash. La aceptación de la instalación infecta a la máquina para que sea parte de la botnet. Seguidamente, las instrucciones consisten en reportar al servidor de la botnet todos los sitios web que visita el usuario. Adicionalmente, el malware consulta la red del usuario infectado de Facebook y envía mensajes a todos los usuarios dentro de esta red, compartiendo el enlace malicioso. Otra de las características de este malware es el componente de *captcha*[3] que se instala. Consiste en mostrar un aviso de alerta que exige al usuario resolver un reto captcha para evitar reiniciar su ordenador. La solución que realiza el usuario es enviada como respuesta al nodo servidor de la botnet y después es utilizada por otro nodo esclavo que solicitó resolver un captcha para continuar algún proceso.

Los enlaces compartidos conforman la parte crítica para desencadenar los ataques y depende mucho del usuario si acepta lo que sugiere el sitio. En los eventos reportados, el nivel de confianza de un usuario para hacer clic en la URL y acceder al contenido depende del usuario iniciador de la publicación. Si su relación de amistad es directa la probabilidad de acceso al enlace es

[1]Clickjacking: Es una técnica maliciosa que engaña al usuario web para hacer clic en algo diferente de lo que él percibe.
[2]botnet: Red remotamente controlada donde la máquina o bot infectado recibe mandatos.
[3]captcha: Proceso con fines de distinguir a un humano de una máquina mediante un reto.

más probable, por ese motivo los usuarios maliciosos buscan como objetivo principal a usuarios con muchas relaciones.

1.3.3. Medidas de protección

La red social Facebook tiene definido a través de su portal (https://www.facebook.com/communitystandards) todos los estándares que forman la base de su funcionamiento. Adicionalmente, proporciona un segmento de ayuda (https://www.facebook.com/help/) que permite al usuario reportar o denunciar sobre todo contenido que vaya en contra de sus estándares. Entre las opciones se encuentran:

a) Si un usuario está enviando mensajes no deseados, si utiliza la identidad del usuario sujeto del reporte u otra persona conocida, si está compartiendo publicaciones inapropiadas u ofensivas, si la cuenta es falsa, si está utilizando un nombre diferente al que usa diariamente.
b) Si un usuario necesita ayuda, si puede estar en peligro o lastimándose a sí mismo, cuando es víctima de *bullying*[4] o acoso, cuando un tercero pueda estar utilizando su perfil.
c) Si una publicación o fotografía tiene contenido perturbador, de tipo spam, cuando el usuario aparece en la fotografía y que ésta le cause algún tipo de ofensa o violente su privacidad.
d) Si una publicación, video o fotografía no debería estar en la red social debido a que es grosero, vulgar, sexualmente explícito, causa acoso u odio, presenta amenazas, violencia o términos suicidas.
e) Si existe algún usuario con las características e información de ser menor de 13 años, o cuando el apoderado observa contenido inapropiado del menor.
f) Si existe algún contenido que viole las leyes de propiedad intelectual.

[4] bullying: Uso de la amenaza para dominar o reprimir agresivamente a otros.

CAPÍTULO II: Ciencia de los datos

La gran cantidad de datos que proviene de distintos aplicativos, herramientas, sistemas, equipos, comunicaciones, transacciones, etc., ha sugerido la investigación de métodos que permitan manejar, analizar y obtener información de estos datos. Con el acontecer de los tiempos, han surgido distintas denominaciones como *Knowledge Discovery Database (KDD)*, *Big Data* y *Data Science* para referirse a: "el conocimiento a través de los datos". No obstante, existen algunas diferencias en su aplicación que subyacen en el tipo de datos (estructurados o no estructurados), el volumen o cantidad, los métodos estadísticos empleados y los expertos en la materia requeridos para su comprensión.

La ciencia de los datos es un término utilizado para describir un conjunto interdisciplinario de métodos para la extracción de conocimiento sobre los datos que pueden ser estructurados o no estructurados [11]. El ciclo consta de tareas que incluyen: seleccionar y extraer datos, procesarlos, obtener su valor, visualizarlos y presentarlos.

La intervención humana es necesaria durante la selección de datos porque se requiere un entendimiento del dominio fuente de los datos a ser extraídos, así como el control del tamaño de la muestra. Generalmente el universo de datos coleccionados son complejos, heterogéneos, a gran escala, generados en tiempo real y poseen ruido, es decir que existen datos corruptos o hay pérdida de los mismos. Para contrarrestar este problema es necesaria una etapa de pre procesamiento o limpieza de los datos. En esta etapa, se aplican técnicas de inferencia para otorgar el valor más probable o copiar un dato que pertenezca a un conjunto en común.

Seguidamente a la estructuración de los datos, es posible encontrar datos que no son útiles para el objetivo del resultado a obtener. Por ejemplo, en un estudio realizado por una universidad, se necesitó saber cuál era el perfil de los estudiantes que lograban una inserción laboral temprana [12]; se obtuvieron datos sobre edad, sexo, escuela de origen, promedio de calificación, semestres cursados, entre otros. Fue necesario aplicar las

técnicas de agrupación de conjuntos de Data Mining, que consiste en la identificación de patrones, lo cual permitió descubrir que sólo cuatro variables de todo el conjunto recolectado determinaban la inserción laboral de los alumnos. Posteriormente, este conocimiento les sirvió para aplicar soluciones de mejora en la situación de los estudiantes.

Dependiendo del conjunto de datos, se aplican las técnicas de Data Mining para descubrir patrones anteriormente desconocidos (focalización exploratoria) mientras que las técnicas de Machine Learning se usan para reproducir estos patrones conocidos y hacer predicciones en base a ellos(focalización predictiva).

Es importante aclarar que si bien los patrones útiles permiten hacer predicciones no triviales y más precisas sobre nuevos datos [13], también es posible aplicar algoritmos de Machine Learning para el reconocimiento de los mismos. Ambos, Data Mining y Machine Learning no son procesos exclusivamente consecuentes y tampoco uno es sub conjunto del otro. Por ejemplo, en el proceso de agrupamiento o *clustering* de Data Mining se pueden utilizar algoritmos de Machine Learning. Por otro lado, dependiendo de los datos, se consigue mejorar la precisión en la predicción de un algoritmo de Machine Learning teniendo un reconocimiento de las propiedades de los datos, que se obtienen mediante Data Mining.

2.1. Machine Learning (ML)

Machine Learning se originó en el campo de Inteligencia Artificial. Conforma un conjunto de modelos matemáticos y estadísticos cuyas tareas involucran reconocimiento, diagnostico, control de robots, predicción, etc.

Machine Learning es utilizada en la ciencia de los datos como una herramienta, debido a que produce resultados autónomos sin necesidad de ser explícitamente programados, habiendo pasado por una etapa de entrenamiento [14].

Un ejemplo tradicional de Machine Learning es un sistema de predicción de mails spam, donde la entrada es un conjunto de mails que han sido etiquetados como spam por el humano. El proceso de aprendizaje consiste

en reconocer un conjunto de palabras y características que aparecen en un spam mail. Luego, cuando un nuevo mail es recibido, se revisarán sus características y si encajan en el conjunto de contenido sospechoso, será etiquetado como mail spam. Este sistema, con una base adecuada de datos de entrenamiento, está capacitado para correctamente predecir la etiqueta de todo nuevo mail entrante.

El campo de Machine Learning se ha dividido en algunos sub campos para tratar con los diferentes tipos de tareas de aprendizaje. Por objetivos de la investigación, se hará énfasis en el par: Supervisados y No supervisados.

2.1.1. Aprendizaje No supervisado

En este caso, no hay distinción entre los datos de entrenamiento y de prueba, sólo se necesitan datos de entrada para el algoritmo. El agrupamiento o *clustering* es la aplicación más referenciada del aprendizaje no supervisado [14]. Consiste en la distribución los datos entre sub conjuntos por similitud (*clusters*) y permite descubrir la estructura intrínseca de las relaciones entre los datos. Por ejemplo, cuando tenemos varios artículos de noticias y se necesita organizarlos para obtener aquellos artículos con temas similares.

2.1.2. Aprendizaje Supervisado

En este caso, se utilizan tanto los datos de entrenamiento y los datos de prueba. El primer conjunto de entrenamiento corresponde a la experiencia que contiene información significativa para el algoritmo. El segundo conjunto de prueba permite validar la experiencia adquirida y predecir la información correspondiente.

Esto puede comprenderse como el proceso de transformar una entrada particular en una salida deseada, para ello tiene que haber pasado por el aprendizaje de la obtención de la salida.

Por ejemplo, la detección de eventos de seguridad que ofrece un sistema SIEM (*Security and Event Management*). Este sistema ha sido entrenado por un amplio conjunto de eventos con diferentes características que

posteriormente permite identificar si algún nuevo evento corresponde a un incidente de seguridad y así enviar la alarma respectiva.

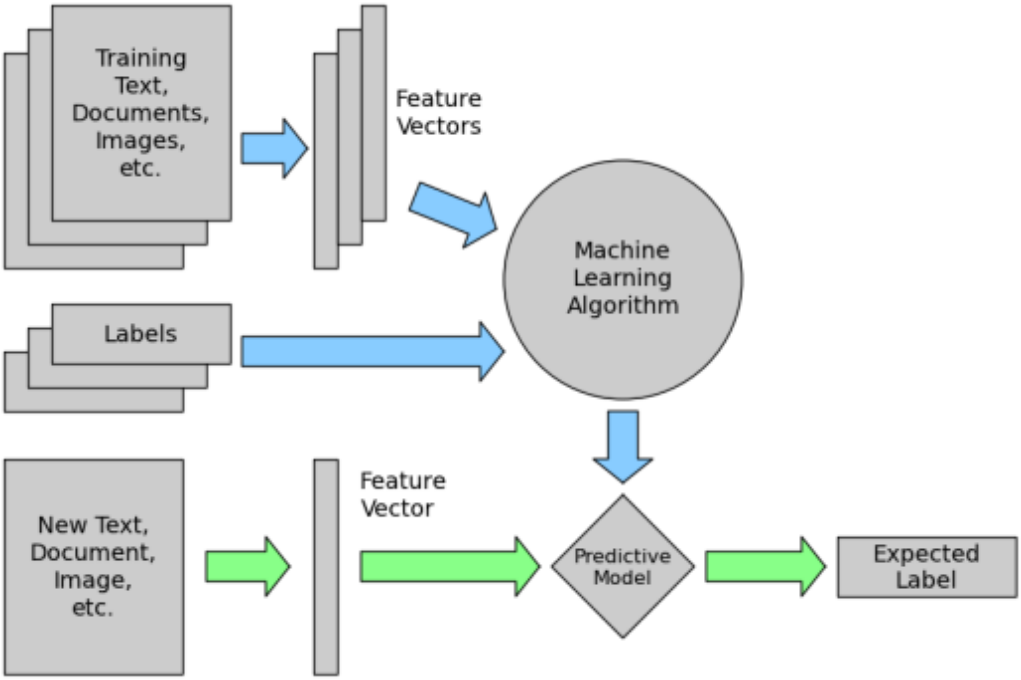


Figura 2. Diagrama del Modelo de aprendizaje supervisado [15]

La Figura 2., describe el flujo del aprendizaje supervisado. Las flechas celestes indican la entrada de los datos de entrenamiento, los cuales pasan por la obtención de características (*feature vectors*), consideradas como las más descriptivas y significativas para los datos. En el ejemplo del sistema SIEM, las características pueden ser cuatro: (1) Cantidad de solicitudes al servidor, (2) horario, (3) si se ha detectado alguna vulnerabilidad y (4) si el servidor está actualizado. Seguidamente, es necesario proporcionarla etiqueta respectiva si corresponde o no a un evento de seguridad. La Tabla 1. describe el ejemplo mencionado.

Características				**Etiqueta**
Cant. Solicitudes	**Horario**	**¿Vulnerabilidad?**	**¿Está actualizado?**	**¿Evento de seguridad?**
170	09:00	NO	NO	NO
10	14:00	SI	SI	SI
200	23:00	NO	NO	SI

Tabla 1. Ejemplo de datos de entrenamiento en el aprendizaje supervisado

Las flechas verdes muestran el proceso de puesta en marcha del modelo (*Predictive Model*), el cual fue previamente entrenado. El modelo inicia con la entrada de un dato nuevo, en el ejemplo del sistema SIEM este dato podría ser el registro de un servicio en una hora pico. A este registro se le extraen las características que el modelo precisa para su evaluación. Seguidamente, el modelo recibe este vector y produce una etiqueta (*Expected label*) que corresponde a la predicción: ¿El evento es de seguridad o no? Este proceso se repite toda vez que recibe un nuevo vector de características, cuya etiqueta es desconocida y se espera conocer.

2.1.2.1. Aplicaciones

El conjunto de algoritmos de Machine Learning es aplicado a numerosos problemas. Por ejemplo, al momento de publicar una fotografía de personas en las redes sociales, se ejecuta el reconocimiento de las características de los rostros para luego sugerir el nombre de cada persona.

En caso de la detección de malware por los sistemas antivirus, se realiza el entrenamiento utilizando la colección de firmas o patrones de comportamiento de los virus. Similarmente, la mitigación de fraude en el comercio electrónico, atraviesa por el entrenamiento de las actividades transaccionales del usuario, localidad, monto, periodicidad y otras variables de compra, las cuales son verificadas por el modelo para alertar una posible transacción fraudulenta.

Otra aplicación es la predicción de embarazo de una mujer en base a los hábitos de su compra, si los pagos los hace con cupones de revistas, si compra por poca cantidad o si deja de comprar algunos productos, etc.

En algunas aplicaciones la cantidad de datos supera los miles, por ejemplo: el análisis de sentimiento sobre blogs de campeonatos de fútbol. El algoritmo requerirá cada mensaje con su respectiva etiqueta: {alegría, decepción, antipatía, neutral}. Esta tarea demandará leer y etiquetar cada mensaje. Para esto son utilizados los servicios de *Crowdsourcing*, que consiste en obtener el trabajo de personas generalmente a través de Internet para desarrollar una tarea.

Es necesario aclarar que tanto la obtención de datos, selección de características y etiquetado de datos de entrenamiento, conforman las tareas más finas y delicadas de realizar para obtener un modelo de predicción bien calificado.

2.1.2.2. Selección de características

También es conocido como selección de variables o selección de atributos. Es el proceso de elegir las características que son las más útiles o relevantes para resolver un determinado problema. Si las características son elegidas de manera que pueda existir características con valores similares para resultados diferentes entonces no importa que tan bueno sea el algoritmo de ML, siempre cometerá errores. Este problema se observa en la Tabla 2.,donde los dos primeros registros presentan valores similares que resultan en una etiqueta diferente, lo cual perturba el aprendizaje del algoritmo.

Características				**Etiqueta**
Cant. Solicitudes	**Horario**	**¿Vulnerabilidad?**	**¿Está actualizado?**	**¿Evento de seguridad?**
202	23:00	NO	NO	NO
200	23:00	NO	NO	SI
50	11:00	NO	SI	SI

Tabla 2. Ejemplo del problema de selección de características

Se distinguen dos tipos de algoritmos de selección de características [16]:

- ***Wrapper***: La evaluación y selección de los atributos lo realiza el algoritmo basado en las estimaciones de exactitud del modelo. Se realizan combinaciones de las características, seleccionando aquellas que obtengan el resultado más preciso para el modelo.
- ***Filter***: Trabaja por separado del algoritmo de aprendizaje y aplica mediciones estadísticas para asignar un puntaje a cada característica. El test Chi cuadrado y la correlación de coeficientes son algunos métodos mayormente utilizados.

2.1.2.3. Support Vector Machine (SVM)

SVM es uno de los algoritmos de clasificación de ML que permite obtener la salida esperada para cada entrada suministrada. Mayormente es aplicado al tipo de clasificación binaria y lineal, aunque puede extenderse a clasificación multiclase. Binaria porque permite distinguir entre dos clases y lineal porque la ecuación que define la clasificación es una línea recta o un espacio bidimensional. Este algoritmo ha sido referenciado por muchos problemas de clasificación debido a su eficiencia computacional y capacidad para manejar espacios de características de alta dimensión, por lo cual es considerado uno de los mejores algoritmos que se puede optar [14].

La particularidad de SVM es que el cálculo de la ecuación solución no tiene en cuenta todos los datos de la muestra, sino solamente se considera un número limitado que tienen propiedades concretas y de singular importancia, denominados **"Vectores de soporte"**. Éstos se observan en la Figura 3., en distintos espacios dimensionales.

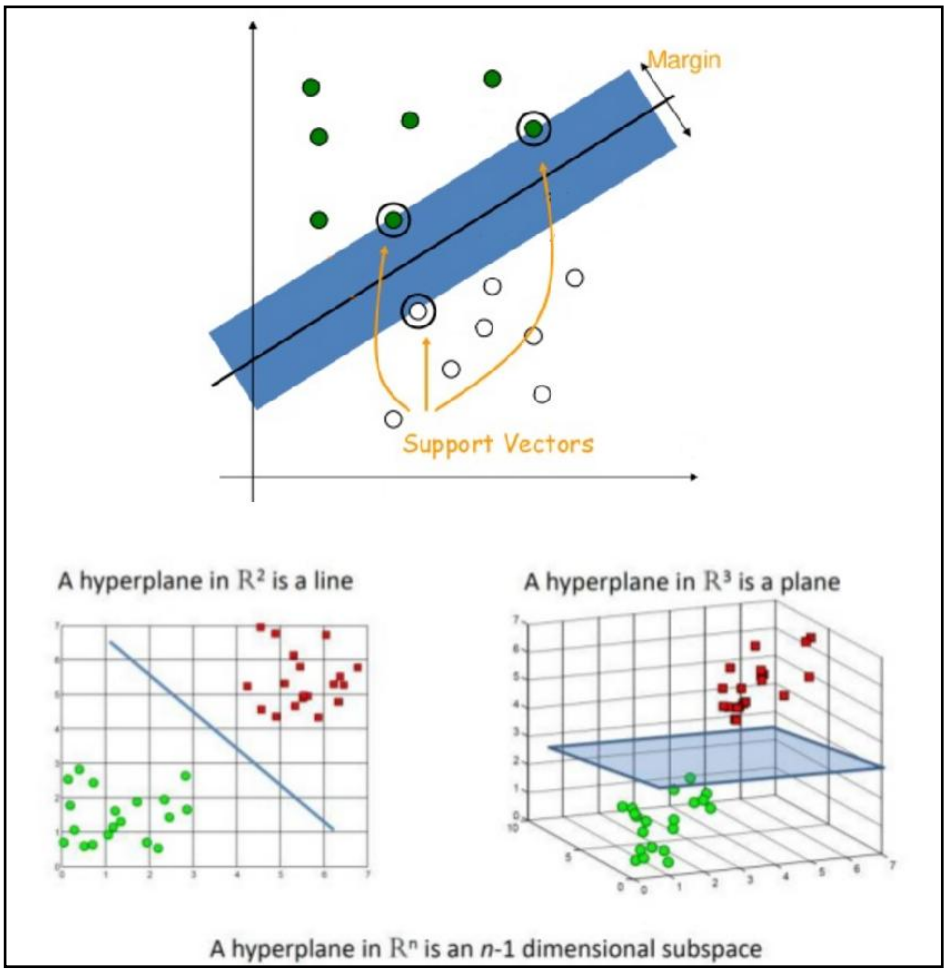


Figura 3. Hiper-planos en SVM. Fuente [17]

Los planos dividen dos conjuntos de color rojo y verde, Por ejemplo, si hablamos de la clasificación de fotografías donde sólo se precise obtenerlas que presentan autos, entonces el conjunto rojo puede representar a las imágenes de contienen al menos un auto y el conjunto verde a aquellas que no contienen ninguno.

2.2. Modelos de ML en la detección de contenido malicioso

Algunas investigaciones se han enfocado en construir modelos de Machine Learning afinando las características de entrenamiento en las bases de datos existentes (recolectadas en años pasados) y otros se han enfocado en recolectar nuevos datos. Todo esto con el objetivo de buscar el modelo adecuado que minimice o anule los falsos positivos en la predicción.

Entre las propuestas, se encuentra la aplicación MyPageKeeper [18] para Facebook, que una vez instalada por el usuario, consumía sus datos de entrada y salida. Estos datos alimentaban al algoritmo SVM y su objetivo era alertar a los usuarios sobre contenido *socware*[5] toda vez que se recibía una publicación en el perfil antes que aparezca públicamente. La aplicación fue instalada por 12 mil usuarios de quienes se observaron las publicaciones en un lapso de cuatro meses.

Otra investigación estuvo enfocada a la detección de cuentas maliciosas [19]; se desarrolló una herramienta llamada COMPA que analizaba cambios en el comportamiento del usuario, por ejemplo: excesivas publicaciones, temas divergentes y cantidad de solicitudes de amistad enviadas. El enfoque propuesto se compone de un modelo estadístico y de detección de anomalías. La construcción del modelo se realizó con bases de datos de Twitter y Facebook. La base de Twitter consistió en mensajes coleccionados desde Mayo de 2011 hasta Agosto de 2011, mientras que la base de Facebook contenía mensajes desde Septiembre de 2007 hasta Julio de 2009. Los resultados fueron positivos puesto que se detectaron cuentas comprometidas con pocos falsos positivos.

[5]Socware: Contenido que propaga malware, promociona ganancias falsas, incluye enlaces que precisen información personal del usuario y/o completar encuestas.

Algunas investigaciones utilizaron datos de fechas pasadas porque hasta el 2009, la red social Facebook permitía obtener información completa de todos los usuarios que compartían la misma ubicación geográfica; la cual fue descontinuada legalmente por temas de privacidad.

Con respecto a los modelos de clasificación basados en las URLs, en [20] se aplicaron técnicas de Text Mining[6] para extraer la identificación de las páginas accedidas con las URLs recolectadas. Se extrajeron en total 17 características obtenidas desde: código fuente, contenido en las etiquetas HTML, referencias a otras URLs, resultados que proporcionaban los motores de búsqueda de la URL y contenido en general de 200 URLs. Éstas fueron colectadas de PhishTank, donde residen URLs tanto de sitios legítimos como de sitios phishing. Se obtuvieron resultados al 99% de exactitud con un modelo hibrido de los algoritmos *Genetic Programming (GP)*[7] y *Classification and Regression Tree (CART)*[8].

Por otro lado, considerando una extensa cantidad de URL cortas utilizadas en las redes sociales. Se apostó por la recolección de *tweets*[9] públicos que contenían por lo menos una URL corta proveniente solamente del servicio Bit.ly (https://bitly.com/). Se utilizó el filtro de Chi-cuadrado para evaluar la precisión de las características, las cuales representaban: cómo fue publicada la URL corta, en qué horario, cuántos duplicados tenía y los patrones de clics que proporciona Bitly. Se obtuvo resultados asertivos [21].

Con respecto a idiomas diferentes al inglés, existe una escases de datos colectados. Por tal motivo, para la detección de spam en español publicado en LinkedIn [22] se realizó la traducción automática al español de las características obtenidas del modelo en inglés. A partir de la traducción se realiza una transferencia de aprendizaje para construir el modelo en español. Se utilizó el modelo *Logistic Regression*[10] para los datos de

[6]Text Mining: Proceso de obtener información de calidad significativa en textos.
[7]GeneticProgramming:Modelo de programación que utiliza las ideas de la evolución biológica para manejar un problema complejo.
[8] CART: Modelo que trabaja a través de la partición recursiva de los datos de entrenamiento para obtener subconjuntos que sean específicos a una determinada clase de destino.
[9]Tweets:Palabra utilizada para referirse a publicaciones realizadas en Twitter.
[10]LogisticRegression:Método estadístico para analizar datos donde hay una o más variables independientes que determinan un resultado.

entrenamiento en el dominio de la traducción automática y el algoritmo de aprendizaje adaptativo: *Kernel Mean Matching (KMM)*[11] para actualizar el modelo en español. Se obtuvieron resultados satisfactorios, aunque se pudo observar ciertas perturbaciones (diferentes distribuciones) en el español obtenido de la traducción automática comparado a la redacción humana.

2.3. Modelos de ML en el sistema de Facebook

En general la recopilación y análisis para la detección tanto de usuarios y contenido malicioso en las redes sociales se da en tres niveles y una combinación de ellos:

- A nivel de una publicación (o post): analizando la relación {post - comentarios} para evaluar la aparición de temas divergentes, cantidad de información del texto, longitud de texto, existencia de URLs, horario de publicación, etc.

- A nivel de usuario: considerando su red de relaciones, sus datos públicos, cantidad de seguidores y amigos, contenido re-compartido por el usuario, contenido que recibe el usuario, solicitudes de amistad enviadas, etc.

- A nivel de campaña, agrupando cuentas de usuarios de acuerdo a las URLs dentro de las publicaciones, donde generalmente los usuarios malintencionados comparten las mismas URLs.

El sistema de detección de Facebook [3] comprende algoritmos de clasificación que pretenden evitar el malware, phishing, spam, cuentas falsas (no representan a una persona) y *creepers* (personas que crean contenido para molestar a otros usuarios). La protección del grafo relacional de los usuarios y objetos tal como eventos, lugares, páginas y aplicaciones resulta un problema complejo, considerando además los requerimientos de escalabilidad y latencia debido a que el sistema realiza verificaciones en tiempo real y clasificaciones en cada acción de lectura y escritura, alcanzando 650 mil revisiones por segundo.

[11] KMM: Algoritmo que produce relaciones de densidad entre los datos de entrenamiento y de prueba al minimizar su discrepancia de media máxima.

El sistema detecta comportamiento anómalo, así como aquellos ataques correspondientes al aprendizaje *Adversarial learning*, que se refiere a patrones creados por el atacante, intencionalmente diseñados para evitar que sean aprendidos y desestabilizar el modelo.

Figura 4. Filtros de Facebook en mensajería y publicaciones.

El flujo de la arquitectura del sistema primero pasa el contenido del usuario por una fuente de políticas de contenido (*policy engine*) que parten del nivel de políticas definidas por Facebook, a nivel de la localidad y a nivel de las configuraciones del usuario. A partir de ello, se define las características que serán evaluadas por el algoritmo de clasificación, que pueden ser: *Random Forests*, *Naive Bayes, Regression, SVM*, entre otros.

Las dos ventajas de este sistema subyacen en la información que recibe; (1) el conocimiento general de los usuarios, su red, publicaciones, lugares visitados, gustos, emociones, eventos, gastos, etc. y (2) la retroalimentación por parte del usuario, que puede ser explícito e implícito, cuando el usuario marca una publicación como spam y cuando elimina una publicación o rechaza una solicitud de usuario, respectivamente. Los resultados de esta detección se observan en la Figura 4., a nivel de mensaje se filtran aquellos cuyo contenido no es solicitado por el usuario o cuando el emisor no pertenece a la red del usuario. A nivel de publicación se evita mostrar los comentarios spam y los que presentan idiomas diferentes.

El amplio espectro que contempla Facebook está enmarcado al grafo relacional de sus elementos. Sin embargo, existe el caso de comentarios cuyo contenido no es marcado como spam y contiene una URL, como de la

Figura 5., donde se comenta sobre una aparente noticia, pero cuyo destino de la URL es una página de contenido sexual. Este tipo de contenido solamente es filtrado mediante la retroalimentación por parte del usuario.

Figura 5.(a) La flecha roja muestra comentario con URL. (b) Página destino de la URL.

2.4. Procesamiento de contenido con URLs

Se han desarrollado investigaciones enfocadas al tratamiento de contenido con URLs, debido a que la acción de un clic por parte del usuario es directamente reconocida como una solicitud de contenido de un sitio web y en caso que éste tenga una intención maliciosa, puede hacerla efectiva.

En la investigación [2], se observó que la proporción del contenido aumenta notablemente durante los eventos que hacen noticia como el caso de la federación de fútbol (copa FIFA 2014) que generó más de 3 billones de publicaciones en 32 días, donde se encontraron URLs cuyos sitios web solicitaban ingresar datos para hacer que su voto cuente en una determinada causa, lo cual podría terminar en una base de emails para

enviar spam o realizar un ataque dirigido con la temática de la causa. La recolección consistió en obtener URLs de las publicaciones mediante expresiones regulares y consultarlas en las listas negras o *blacklists*, tales como: Google Safebrowsing, PhishTank, VirusTotal y WebOfTrust y Surbl.

A partir del conocimiento de las URLs legítimas y maliciosas se hizo el análisis de las entidades (usuarios o páginas) que publicaron tal contenido malicioso, utilizando además las características del tipo de publicación, cantidad de comentarios, cantidad de comparticiones, ubicación, etc.

También se encuentra la propuesta en [23] para la detección de páginas phising en base al código fuente del sitio web y la estructura de la URL. La colección de URLs se obtuvo de PhishTank con la mitad de sitios phishing y mitad de sitios legítimos. A todos ellos se realizó la extracción de características como: (1) los metadatos<META> donde reside el título y la descripción, (2) enlaces foráneos a otros dominios, (3) presencia de scripts, (4) si tiene una dirección IP como dominio, (5) número de puntos en la URL, (6) direccionamiento foráneo de los formularios, (7) si el sitio genera cookies, (8) cuenta con certificado SSL, si la URL aparece en los cinco primeros resultados de los motores de búsqueda, entre otros. Se obtuvo resultados con un 99% de exactitud para el algoritmo Genetic Programming (GP).

CAPÍTULO III: Detección de contenido malicioso en Facebook

Las herramientas enlistadas a continuación fueron utilizadas para recolectar datos actuales que posteriormente sirvieron para alimentar al modelo de clasificación propuesto en esta investigación.

La recolección (o ***web scraping***) se realizó a datos públicos de páginas y grupos de Facebook. A partir de esta fuente de registros, se obtuvieron las URLs y se elaboraron estadísticas informativas.

3.1. Estructura de servicios y APIs

a. **Anaconda Navigator 1.4.3**: Es una interfaz gráfica de usuario que permite un fácil manejo de paquetes en sus diferentes versiones y también en distintos ambientes *(environments)*. Los paquetes contenidos corresponden a Anaconda [24], que es una distribución de Python para procesamiento de datos a gran escala.

b. **Jupyter Notebook 4.3.1**: Es un ambiente de ejecución de código interactivo basado en una aplicación web [25]. Permite editar, guardar y mostrar las entradas y salidas de las computaciones incluyendo la representación de objetos gráficos.

c. **Script Facebook Scraper**: Desde el repositorio de GitHub se pudo acceder a [26]. Este repositorio alberga scripts para obtener las publicaciones y sus comentarios. Fue necesaria su modificación para incluir los identificadores de los permisos que se obtienen a través del sitio para desarrolladores de Facebook.

d. **Facebook Graph API:** Facebook proporciona el modo desarrollador Graph API [27] para realizar consultas sobre los usuarios siempre y cuando se trate de información pública. Esta API permite extraer los comentarios y publicaciones de cualquier usuario sobre una página o grupo público.

   El identificador de acceso (*Access Token*) esta predefinido para una vigencia de una hora que puede ser extendida a más de treinta días.

e. **Beautiful Soup 4.5.6**: Es una librería de Python que permite extraer los datos de etiquetas HTML y XML [28]. Proporciona formas sencillas

de recorrer el árbol de navegación de estas etiquetas, así como editarlas. Se utilizo esta herramienta para ubicar las etiquetas HTML de la pagina web.

f. **Tldextract-2.0.3**: Librería que facilita la extracción del dominio, subdominios y las extensiones de una URL [29].

g. **Cookielib**: Librería que permite obtener automáticamente las cookies HTTP de un sitio web [30].

h. **Whois-0.7**: Librería para obtener la información Whois de los dominios [31]. Esta información es retornada del protocolo Whois que presenta los detalles del cliente registrado como titular del dominio. La consulta se realiza en la base de ICANN (https://whois.icann.org/). También se obtiene la información tal como fechas de duración, creación y expiración, país, etc.

i. **Unshorten.me**: Servicio para obtener las URL que fueron recortadas por los servicios de reducción de URLs. Este servicio está limitado a diez solicitudes por hora [32]. Sin embargo, es posible solicitar el aumento del intervalo de consultas hasta 100 solicitudes por hora.

3.2. Proceso de recolección de datos

El proceso para recolección de los comentarios y publicaciones de las páginas y grupos públicos consistió en realizar una búsqueda de palabras claves que puedan lograr una proliferación masiva del contenido. Si alguna página o grupo se muestra como resultado esta búsqueda, entonces se convierte en un elemento objeto de obtención de su contenido. El resultado se obtiene en archivos de extensión .csv (*Comma Separated Value*). A partir de estos archivos se discrimina aquel contenido que permiten realizar estadísticas y realizar la limpieza de datos para poder alimentar el modelo de clasificación.

A continuación se detallan las actividades realizadas:

- **Palabras claves**

Las palabras u oraciones claves que sirvieron para la búsqueda e identificación de posibles paginas o grupos objetos de recolección fueron:

Funciona, genial pruébalo, truco, es muy fácil, aquí te dejo el link, aplicación, sorteo, gratis, 100% probado, compartir, suscríbete, pack, sígueme, te sigo, regalo, entra aquí, video aquí, clic aquí, encontré la manera de ver, encontré la forma de ver, por fin encontré, visita mi perfil, dejo hack, millón, miles, masiva, gratis, popular, link, aprovéchalo, chat, fais, en un segundo, gana, lo mejor.

Es importante aclarar que los nombres completos de las páginas y grupos estaban acondicionados de caracteres especiales, como los que se muestran en la Figura 6.

Figura 6. Nombres de grupos con caracteres especiales.

❖ Identificadores de páginas y grupos

El script de recolección se ejecuta por cada página o grupo por tanto es necesario proporcionar su identificador. Estos identificadores se obtienen entrando en las herramientas de desarrollador del explorador (generalmente accedido con la tecla F12). Para el caso de una página se debe buscar la etiqueta ***page_id*** y para el caso del grupo ***group_id*** (en algunos casos mediante ***entity_id***).

❖ Recolección de datos

Se realizó la recolección de comentarios y publicaciones en dos etapas. Primero, fue necesario recolectar las publicaciones porque cada una tiene un ***status_id,*** que es el identificador de estado, a partir de cual en la segunda etapa se recolecta los comentarios, proporcionando como entrada

el archivo de las publicaciones. Para realizar una recolección útil, se obtuvieron todas las publicaciones y se hizo una discriminación en los comentarios para obtener solamente aquellos que contengan ***"http"***, una longitud del mensaje mayor a diez o que contengan números de teléfono (este último se obtuvo con intención de mostrar resultados gráficos).

La recolección se realizó en promedio en 31 páginas y 36 grupos. Cada uno de ellos tuvo en promedio 600 registros por día. Que finalmente llegaron a ser más de ciento veinticinco mil URLs (125283) y más de siete mil (7336) números de teléfonos en tres meses de recolección aproximadamente.

- **Limpieza y estructuración de los datos**

En caso de los números telefónicos la tarea de estructuración consistió en identificar el país de cada uno de los números.

En caso de las URLs, la estructuración se realizó en cinco grupos debido a que las URLs de las redes más conocidas representan a páginas legítimas y no resulta necesario consultarlas.

(1) URLs no duplicadas: URLs de diferentes dominios que no se hayan duplicado en algún otro comentario o publicación.

(2) URLs de Facebook: Aquellas URLs de imágenes, videos, gifs, eventos creados desde Facebook.

(3) URLs de WhatsApp: Corresponden a links del tipo: https://chat.whatsapp.com/<Identificador> donde cualquier usuario, después de una aceptación puede ingresar al grupo y participar del contenido.

(4) URLs de otras redes sociales: Las redes sociales en este conjunto corresponden a las más conocidas como Twitter, Google+, Youtube, Instagram, Giphy y Flickr.

(5) URLs cortas: Son las URLs que fueron recortadas por algún servicio. Por ejemplo: bit.ly, goo.gl, adf.ly, short.am, cort.as, etc.

Las tareas siguientes consistieron en (1) una revisión manual del archivo de URLs no duplicadas, eliminando aquellas URLs incompletas y otras con IP locales, que generalmente se usan en juegos en red. (2) Se obtuvo la URL

destino de cada una de las URLs cortas. Respecto a esta tarea se observó URLs cortas de un servicio particular, por ejemplo http://uni.vi/<identificador>que pertenece al canal de televisión Univisión (www.univision.com). Por lo tanto, también se eliminaron este tipo de URLs cortas.

Adicionalmente, mientras se revisaba el archivo de URLs no duplicadas, se observó la existencia de URLs cortas y similarmente, en el archivo de URLs cortas se encontraron URLs no recortadas con dominios de longitud menor a diez (por ejemplo: mia.com). Conociendo esta diferencia se actualizaron y ordenaron los datos y se finalizo la estructuración.

- **Correspondencia de las URLs cortas con su URL destino**

A partir del listado de las URLs cortas, se utilizó UnshortMe [32] para obtener la traducción de estas URLs, ya que retorna resultados de la mayor parte de servicios de reducción de URLs. Sin embargo, no es posible obtener la traducción de aquellos servicios basados en anuncios, donde se tiene que esperar unos segundos antes de acceder al link de destino. Por ejemplo, para consultar las URLs cortas de ***adf.ly***, se utilizó DeAdfly [33]. En otros casos, fue necesario realizar la correspondencia manual debido a que no se encontraron servicios ni páginas que facilitaban la traducción. Es necesario resaltar que los servicios: shirnk.in, abull.me, fas.li y short.am son los más trabajosos para acceder a la URL destino, en el sentido que el botón del enlace no se muestra o aparece deshabilitado y es necesario la resolución de un captcha para habilitarlo; adicionalmente, en el primer clic se abre una ventana emergente con otro anuncio. En caso específico del servicio short.am fue necesario hacer tres clics en el botón del enlace para evadir los anuncios y acceder al sitio destino.

3.3. Resultados de la distribución de datos

Los números telefónicos se obtuvieron de los comentarios cuyo tema de la publicación propone compartir videos, imágenes, audios y credenciales de acceso para películas a quienes comenten su número. El código para la identificación del país se adjunta en el Anexo A. En la Figura 7. se observa

que Perú tiene el porcentaje mayor seguido de México, Colombia y Argentina.

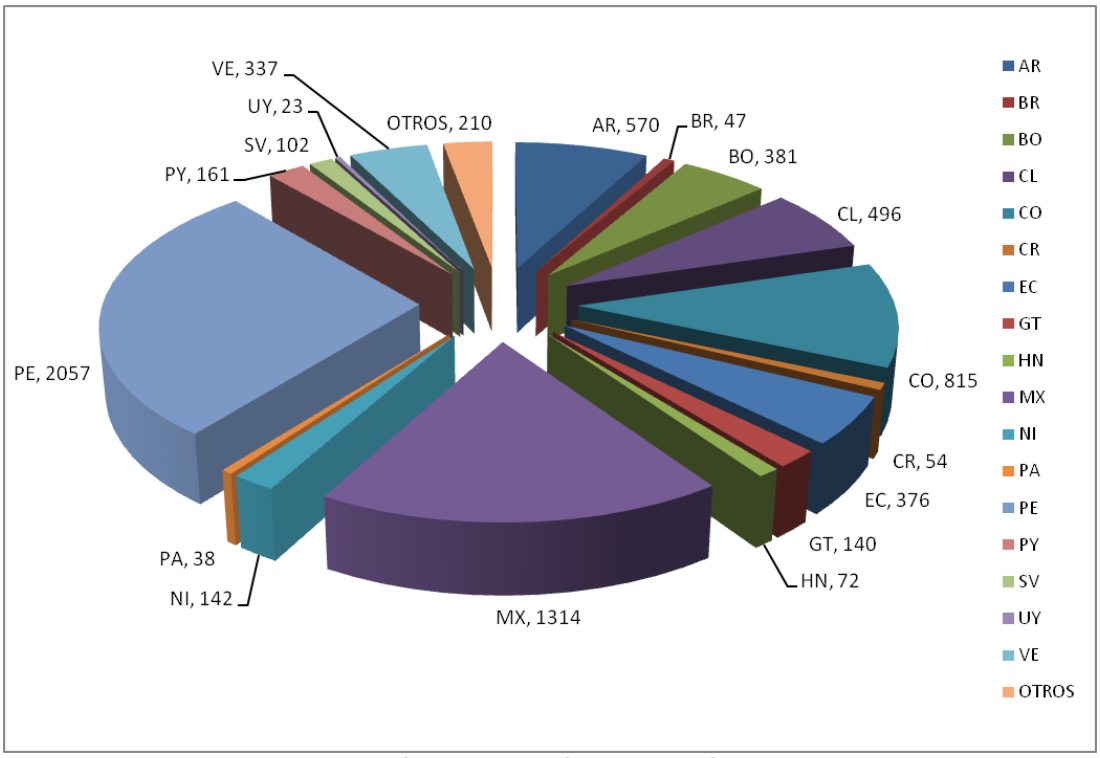


Figura 7. Números de teléfono por país.

La primera fase de estructuración de datos de la Figura 8., cuyo código se adjunta en el Anexo B, muestra que la mayor cantidad de URLs compartidas son enlaces a Facebook. Las URLs de WhatsApp corresponden a un porcentaje menor; sin embargo, anteriormente no se había reportado este tipo de enlace. También se observa una cantidad de 4283 URLs cortas.

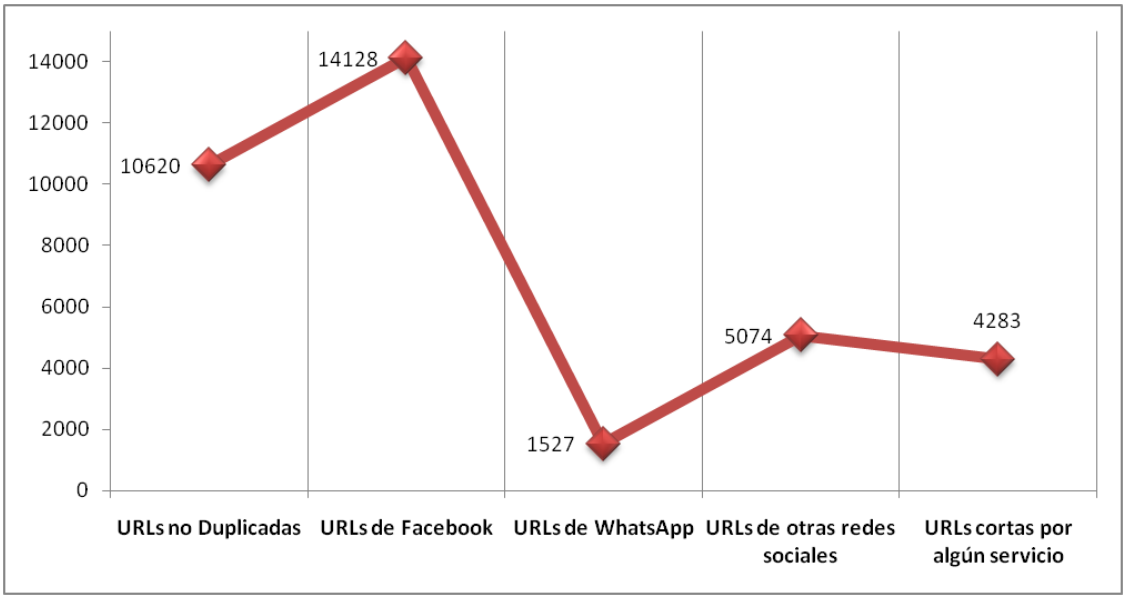


Figura 8. Clasificación URLs recolectadas.

Las URLs cortas en su mayoría corresponden a los servicios generalmente usados como goo.gl, bit.ly, ow.ly. En la Figura 9. se muestra la cantidad por servicio. El dato "Otros (<10 apariciones)", corresponde a los enlaces que aparecieron menos de 10 veces. Este conjunto incluye a:

wn.nr, gsurl.in, xurl.es, klurl.nl, flyt.it, adshort.co, dlvr.it, crop.is, csl.ink, fas.li, ouo.press, bit.do, gol.gl, good.gl, sumo.ly, x.co, hop.bz, microify.com, go2l.ink, cut-urls.com, cor.to, snip.ly, daj.host, cliks.it, wiid.me, skiip.me, link.tl, 1clc.ml, v.ht, tinyium.com

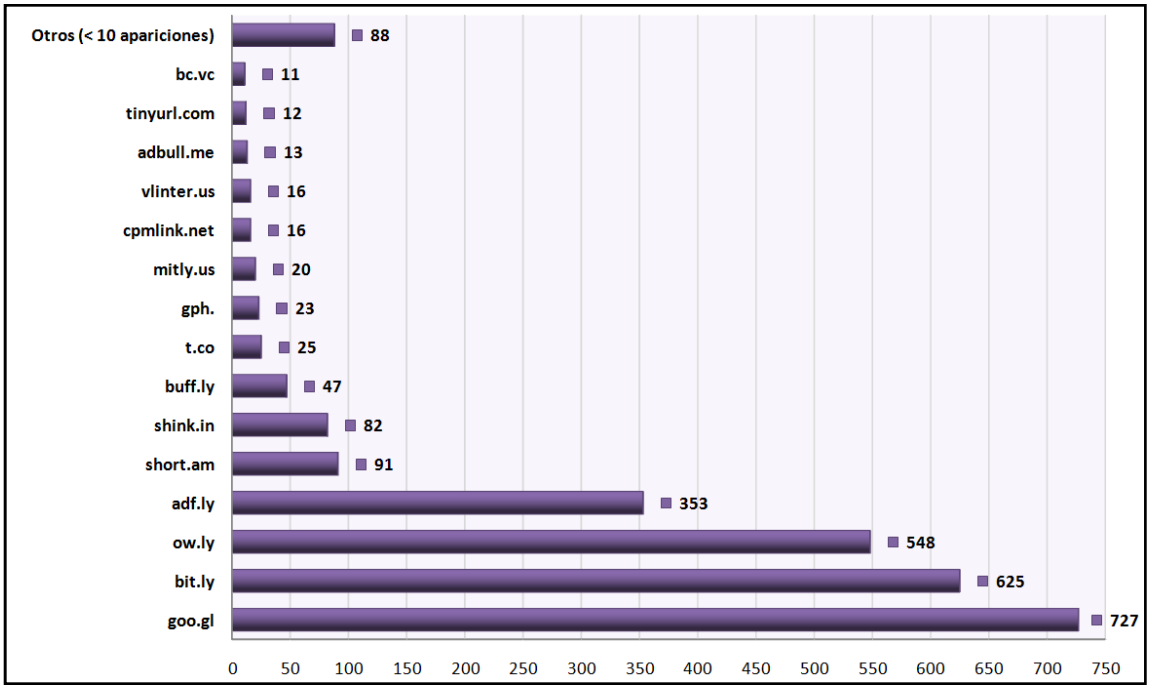


Figura 9. URLs cortas por servicios.

En algunos casos el primer intento de traducción de URLs cortas resultó en otra URL corta por otro servicio y al proceder con la segunda traducción se encontró un escenario similar. El código utilizado para expandir la URL se adjunta en el Anexo C. En la Figura 10. se observa que el quinto nivel de traducción es el máximo que fue necesario consultar para obtener la URL destino. De estos resultados, en la Figura 11., se obtuvo la distribución de los servicios que son mayormente utilizados en varios niveles de traducción.

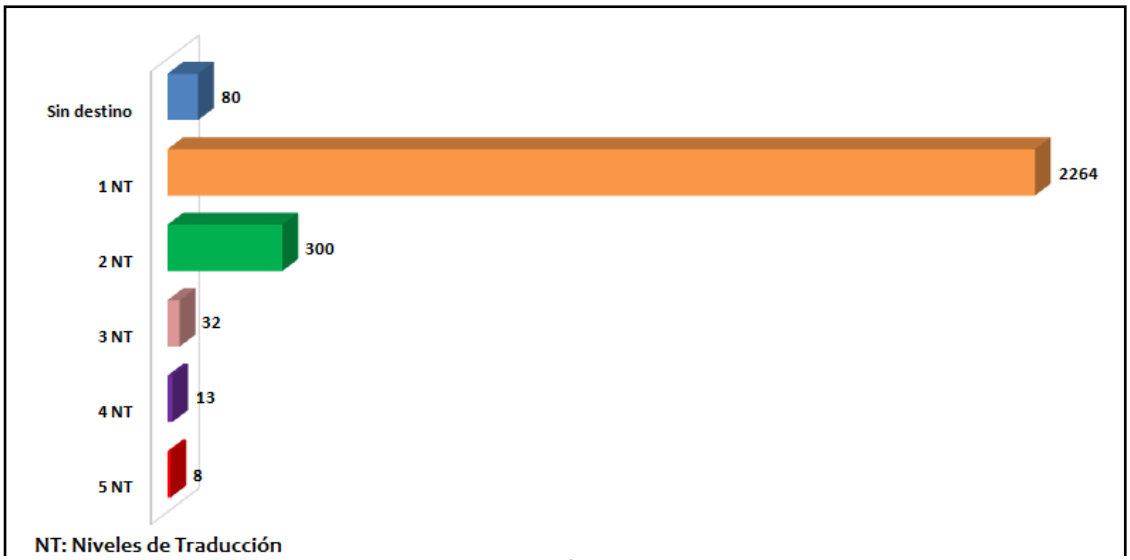


Figura 10. Niveles de traducción (NT) de las URLs cortas.

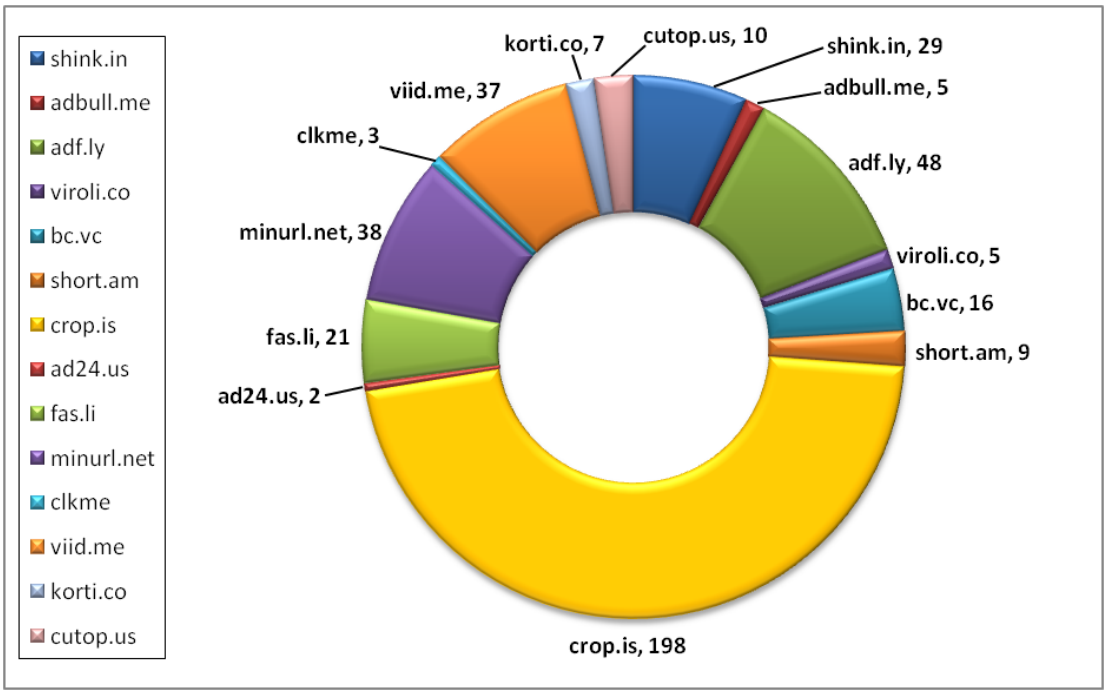


Figura 11. Servicios utilizados en los NT de las URLs cortas.

Es importante notar que todos los servicios de la Figura 11. están basados en anuncios. Esto implica que toda vez que el usuario hace clic en la URL corta, se le muestra una publicidad de aproximadamente 5 segundos que genera una ganancia para el creador del enlace. No es necesario que el usuario haga clic en el anuncio. En general las ganancias fluctúan entre 4 a 5 dólares por cada 1000 clics en la URL corta. El código de la obtención de los NT se adjunta en el Anexo D.

URL destino	1 NT	2 NT	3 NT	4 NT	5 NT
mediafire.com	16	6	0	1	1
nolocreo.com	1	31	0	0	0
mega.nz	274	4	15	11	7
paste2.org	0	11	15	0	0
elfqrin.com	0	2	1	1	0
aldeaviral.com	3	28	0	0	0
viralymedio.com	1	20	0	0	0

Tabla 3. Resultados de URLs destino por NT.

En la Tabla 3. y su respectiva gráfica en la Figura 12. muestra un agrupamiento por las URLs destino más referenciadas en más de un nivel de traducción de las URLs cortas. El código para lograr este agrupamiento se adjunta en el Anexo E. El sitio **mega.nz** que permite compartir archivos es el más referenciado en todos los niveles y tiene la mayor cantidad en el quinto nivel, cuyo contenido se refiere a pornografía, películas de estreno, videojuegos, series y música. Un escenario similar sucede con **mediafire.com** aunque con menores cantidades. Otro contenido que se presta para el negocio de los servicios de acortamiento con anuncios son los sitios con contenido viral que mayormente llaman la atención con frases como *"a que no adivinas que pasa al final"*. Especialmente en el segundo nivel de traducción se observó en mayoría los sitios: **nolocreo.com**, **aldeaviral.com** y **viralymedio.com**. En el caso del sitio**paste2.org** que permite guardar y compartir texto tiene un aumento de aparición en el segundo nivel de traducción, cuyo contenido corresponde a credenciales generalmente para acceder a sitios de entretenimiento. Finalmente, aparece en menor proporción el sitio **elfquin.com**, que permite generar números de tarjetas de crédito.

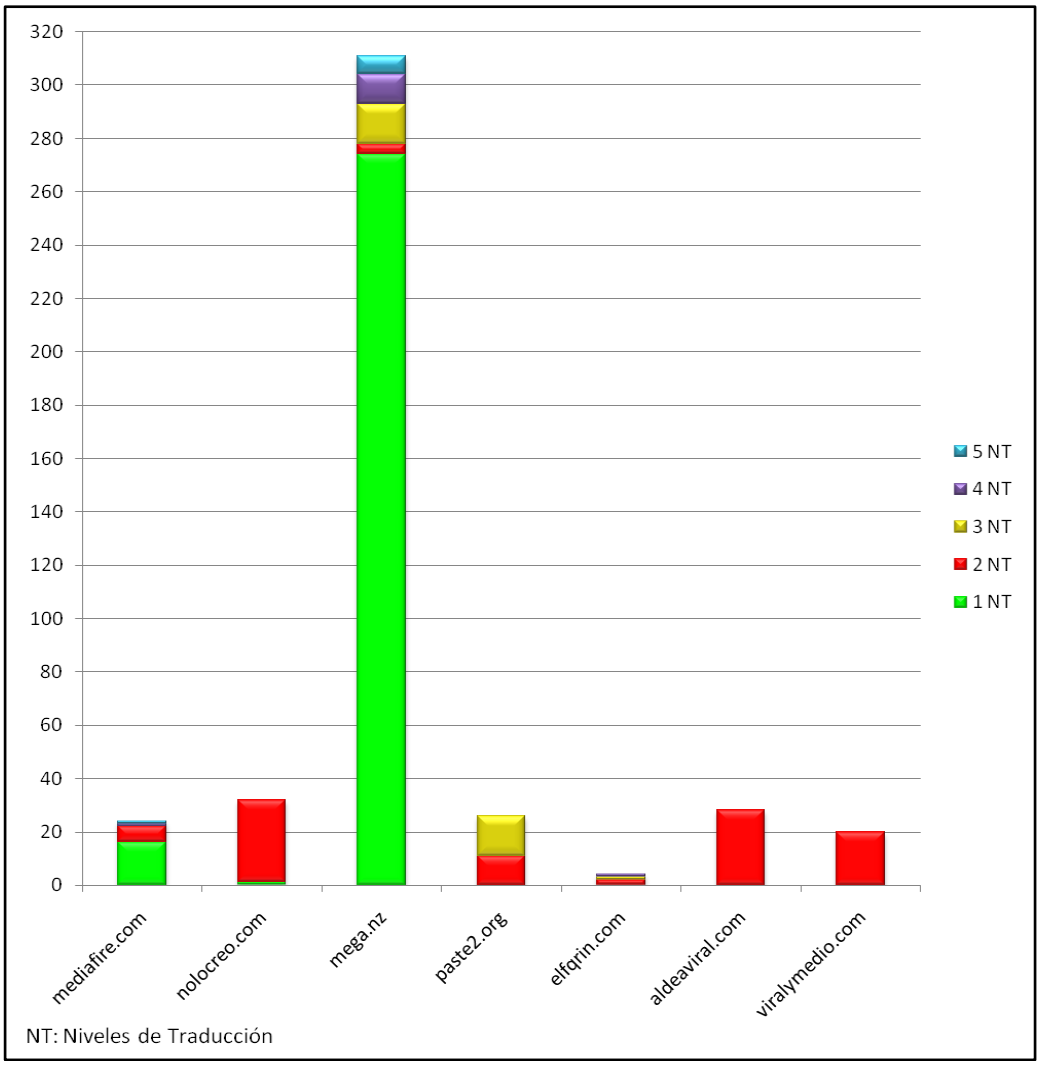


Figura 12. URLs destino por cada NT.

Al finalizar la traducción del conjunto de las URLs cortas se realizó la unión con las URLs del primer conjunto que no requería traducción. Lo cual conformó 12968 URLs en total, evitando URLs repetidas.

Las URLs dentro de esta recopilación contenían un gran número de URLs diferentes, pero con el mismo dominio y con referencia a sitios legítimos como, por ejemplo: canales de televisión, temas de salud, repositorios de archivos, etc. En la Tabla 4., se muestra los dominios de estas URLs con su respectiva cantidad, cuyo agrupamiento fue de acuerdo con el contenido que proporcionan.

SITIOS	CANT.	AGRUPAMIENTO	CANT.
facebook.com	76	Redes sociales	258
twitter.com	3		
instagram.com	1		
google.com	39		
yuotube.com	108		
flickr.com	1		
giphy.com	19		
whatsApp.com	11		
mega.nz	319	Repositorio de archivos	456
mediafire.com	57		
cloudup.com	38		
soundcloud.com	42		
paste2.org	26	Compartir texto	26
telefenoticias.com.ar	2337	Televisión	4516
amoramoramor.pe showdeaaa.com amor-amor-amor.com	1632		
univision.com entretenimiento.univision.com	547		
curiosidadesdelasalud.com curiosidadesdelasalud.net	1059	Temas de salud	1458
remedioscaserosdehoy.com remedioscaserosdehoy.net	267		
baulsaludable.com	66		
recetasnaturales.net	32		
conexionplanetaverde.com	34		
Consejos.pe	106	Curiosidades	790
frasesycuriosidades.com	65		
Noticiasdevida.com	96		
miscuriosidades.info miscuriosidades.club miscuriosidades.es miscuriosidades.com	163		
mitierracuriosa.com	66		
frasesconlavida.com	15		
bigbam.net	129		
lapasionyganasdesentir.blogspot.com.co	150		
psn.cubecode psncube.com pysrencoperu.com psn.freecode.pw psn.mycode.pw	319	Códigos de juegos	331
prizescode.net	12		
Growlandia.com	113	Sobre marihuana	166
lamarihuana.com	41		
cannabislandia.com	12		
gatitoviral.com	57	Temas virales	267
notasvirales.net	34		
viltaitik.com	70		
aldeaviral.com	26		
nolocreo.com	32		
viralymedio.com	19		
tuviralblog.com	29		
		TOTAL	**8268**

Tabla 4. Sitios agrupados por contenido.

La información de la Tabla 4. se observa en la Figura 13. donde la mayor cantidad de 4516 URLs (55%) corresponde a páginas de televisión, seguidamente aparecen los sitios referentes a temas de salud con 1458URLs (18%), el tercer lugar lo conforman los sitios de temas de curiosidades con 790 (10%) y el cuarto lugar con 456 (5%) se refiere a los contenedores de archivos. Es importante notar que 331URLs (4%) hacen referencia a sitios que ofrecen códigos gratis para jugar en PlayStation Network, lo cual es falso y forma parte de los sitios maliciosos [34]. Similarmente, se observa que, en el conjunto de las Redes Sociales, el máximo valor (108) es dirigido a YouTube, el cual es un sitio legítimo para compartir videos; sin embargo, en la descripción de estos videos están adjuntos links que dirigen a la descarga e instalación de herramientas.

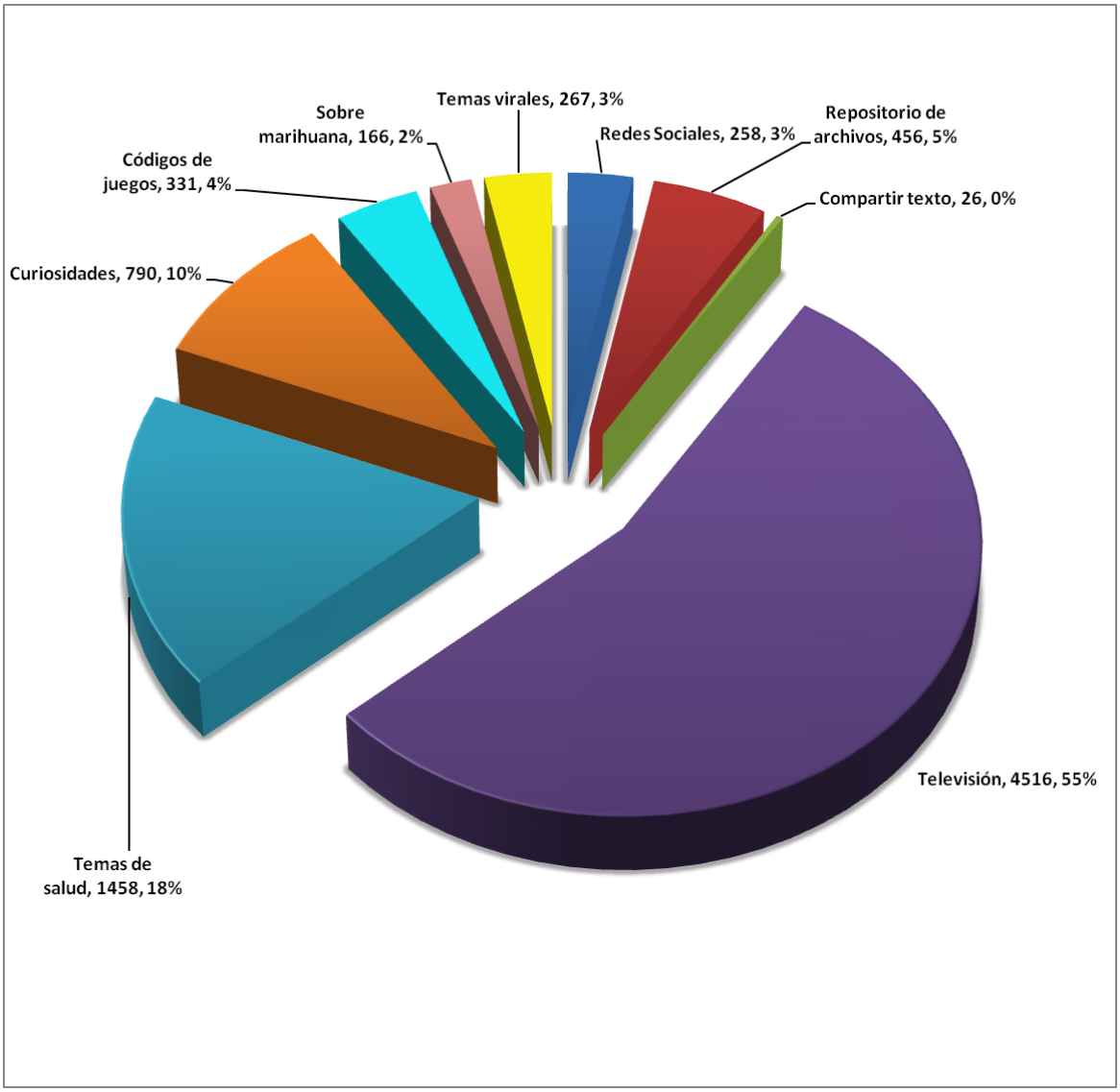


Figura 13. Distribución de URLs agrupadas por tipo de contenido.

Para continuar con el proceso de extracción de características, fue necesario eliminar manualmente las URLs diferentes que tenían el mismo dominio agrupado por contenido de la Tabla 4. Por este motivo, del total de 8268 URLs se mantuvo sólo una URL de muestra por cada dominio desconocido, excluyendo el grupo de: Redes sociales, Repositorio de archivos, Compartir texto y Televisión.

De esta separación quedaron 8247 URLs para ser eliminadas de la colección total de 12968 URLs, que finalmente resultó en **4721 URLs**, las cuales pasan por los procedimientos de extracción de características.

3.4. Extracción de características

La extracción de las características de los sitios web destino de las URLs está basada en la obtención de la identidad del sitio y si éste hace referencia a sitios externos para una redirección o para insertar recursos embebidos. Estos datos se obtuvieron de los elementos indicados a continuación, cuyo código se adjunta en el Anexo F:

- **META Tags**: Son etiquetas presentes en los documentos HTML, se utilizan por los algoritmos de los sitios de búsqueda para decidir los resultados a mostrar para los usuarios. Dentro de esta etiqueta se indica con el atributo name el tipo de dato. Por ejemplo:

  ```
  <meta name="description" content="esta es la descripción">
  ```

  También existe la etiqueta para redireccionar a otra URL después de unos segundos:

  ```
  <meta http-equiv ="refresh" content="4;url="http://ww...">
  ```

- **Title Tag**: Es una de las etiquetas obligatorias dentro de documentos HTML que aparece en la cabecera de la página y permite describir su contenido. Su estructura es:

  ```
  <title>Este es el título de la página</title>
  ```

- **Iframes**: Son etiquetas que permiten incluir un documento HTML en otro. La estructura es: `<iframe src="URL"></iframe>`

  Estas etiquetas son muy utilizadas para incluir videos de Youtube y botones "me gusta" de Facebook; sin embargo, también ha llamado

la atención de sitios maliciosos para incluir contenido que pueda infectar a los visitantes.

Para prevenir que el documento HTML dentro del iframe pueda afectar al sitio que lo alberga se utiliza el comodín "sandbox". Esto evitará que el iframe abra ventanas emergentes, ejecute scripts, lea las cookies del sitio, etc.

- **Script de redirección**: Para redirigir la pagina a otro sitio es posible utilizar la etiqueta script con el código: window.location.href seguido de la URL destino. Por ejemplo:

  <script type="text/javascript">window.location.href="http://example.com"</script>

- **Etiquetas HTML**: Los documentos HTML cuentan con una serie de etiquetas que permiten mostrar imágenes, videos, animaciones, agregar formularios y demás objetos que permiten hacer una navegación en la web más dinámica. Por ejemplo, se pueden incluir imágenes de otros sitios externos, mediante:

  <img src="http://otro_sitio_web/imagen.jpg">

- **Cookies**: Estos son datos enviados de un sitio web para guardarse en la computadora del usuario. Si el sitio tiene subdominios, se puede configurar para compartir las cookies solamente con el dominio principal o con algunos subdominios. Esto a través del encabezado set-cookie.

- **Whois**: Es un servicio que permite averiguar los datos de registro del dominio. La consulta retorna el contacto del registrante del dominio, contacto administrativo, contacto técnico y contacto de abuso. También, retorna las fechas importantes, país de registro, etc.

  Es posible optar por ocultar la información personal del registro de la base pública debido a que cualquiera puede acceder a esta información y es posible recibir mensajería no deseada, anuncios de servicios para desarrollo y diseño web, publicidad, llamadas y marketing en general.

  La protección de privacidad del dominio básica consiste en enmascarar el nombre personal, dirección, teléfono, email, nombre del negocio con identificadores alfanuméricos para posteriormente hacer la correspondencia a los datos originales.

Una de las consecuencias desfavorables de estos servicios es la naturaleza de la privacidad puesto que es posible que el proveedor venda o distribuya esta información. Adicionalmente, la reputación del sitio puede estar sujeto a cuestionamiento, dependiendo del contenido que presenta.

En total se extrajeron 22 características, las cuales son:

(1) **Domain**: Dominio de la URL.

(2) **Title**: Corresponde al texto contenido dentro de la etiqueta <title>

(3) **Description**: Se obtuvo el campo descripción dentro de las etiquetas META.

(4) **Http_Equivalent_Refresh**: Corresponde al campo http-equiv de las etiquetas META. Para conocer si la URL accedida tiene una redirección hacia otra página a través de esta etiqueta.

(5) **Script_Location_Redirect**: Se obtuvo los scripts bajo la acción de location.redirect para conocer si se producían redirecccionamientos desde código script.

(6) **Total_Cookies_Quantity**: Se obtuvo la cantidad total de cookies del sitio.

(7) **Own_Cookies_Quantity**: Se obtuvo la cantidad total de cookies que hacían referencia al mismo dominio. Para hacer la diferencia con las cookies a sitios terceros.

(8) **Country**: Corresponde al país del sitio mediante la consulta Whois.

(9) **Email**: Corresponde al email del sitio mediante la consulta Whois.

(10) **Phone**: Corresponde al teléfono del sitio mediante la consulta Whois.

(11) **Days_Exp_Creation**: Se obtuvo la diferencia de fechas (expiración menos creación) en cantidad de días para obtener el periodo de vida del sitio hasta su expiración.

(12) **Days_Update_Creation**: Diferencia de fechas (actualización menos creación) para conocer el periodo de mantenibilidad del sitio.

(13) **Days_Exp_Update**: Diferencia de fechas (actualización menos creación) para conocer el periodo máximo que se actualiza el sitio.

(14) **Total_Form_Action**: Se obtuvo la cantidad total de etiquetas <form action="">. Esta etiqueta permite pasar datos al servidor y mediante el atributo action se indica la URL donde serán trasladados estos datos.

(15) **Own_Form_Action**: De las etiquetas <form> se discriminó aquellos que se dirigen a alguna URL diferente del dominio origen, para conocer la cantidad de envío de datos a sitios terceros y aquellos que solo hacen referencia a un método del mismo sitio.

(16) **Total_Script_Source**: Se obtuvo la cantidad apariciones de la etiqueta <script> donde el atributo src contenía http, puesto que también puede tratarse de un script interno.

(17) **Own_Script_Source**: Se obtuvo la cantidad de tags <script> cuya fuente hacía referencia al mismo dominio, para conocer la diferencia de referencias a sitios terceros.

(18) **Total_A_References**: Es la cantidad total de referencias. Fue tomada de las etiquetas <a> que permiten incluir referencias a URLs. Generalmente se incluyen en los botones. En el campo href, se indica la URL destino de este elemento.

(19) **Own_A_References**: Se obtuvo las etiquetas <a>cuya URL hacía referencia al mismo dominio.

(20) **Total_Img_Sources**: Se obtuvo la cantidad de etiquetas de imagen: <img> cuya fuente src contenía http dirigido a alguna URL.

(21) **Own_Img_Sources**: Se discriminó la cantidad de etiquetas de imagen cuyo destino URL se trataba del mismo dominio.

(22) **Quantity_iFrames**: Es la cantidad de elementos iFrame que se incluyen en el documento HTML.

Todas estas características fueron extraídas para cada una de las 4721 URLs, que finalmente resultó en 4191 registros debido que se eliminaron los sitios cuyo destino fue inexistente o cuyo dominio ya no está registrado.

CAPÍTULO IV: Construcción del modelo de detección

El inicio para la construcción del modelo de clasificación está denotado por la obtención de datos. En el capítulo anterior se obtuvieron estos datos de tipo texto y número de acuerdo a las características elegidas.

En el conjunto de datos resultante, cuyo ejemplo se muestra en la Figura 14., se encontraron caracteres especiales y algunos registros vacíos. Por estos motivos, fue necesario realizar la limpieza de datos, lo cual requirió en algunos casos acceder nuevamente la URL.

La revisión final de las características sugirió realizar una equivalencia de datos, principalmente para reducir los valores altos; así como, definir nuevas características. Este proceso de normalización de datos también contribuyó a la optimización del modelo en el cálculo y nivel de procesamiento que conlleva su entrenamiento.

url	domain	Title	Descripcion	Http_Equiv	Script_Locat	Total_Cookie	Own_Cookies	Country	Email
http://esnoticia.co/	esnoticia.	El mundo en	"La vacuna cont	0		0	0	United St	ESNOTIC
http://testbr.com/r	testbr.co	Descubriend	Pokemon Go de	0		0	0	PE	alejandr
http://coc-hackonli	coc-hack	HOSTEF	0	http://www.hostef.gr/		0	0	ID	IGOEIJO
http://www.noticia	www.not	YOUTUBER F	LA MAÃ'ANA	0		0	0	FR	contact
http://jjjjnota.com/	jjjjnota.c	La camara qu	0	0		1	1	US	anthony
http://testbr.com/r	testbr.co	Descubriend	Pokemon Go de	0		0	0	PE	alejandr
http://m.memegen	m.memegen.com		0	0		0	0	US	memege
http://www.noticia	www.not	Noticias de Â	Diario de inforn	0		0	0	FR	contact
http://www.tus-not	www.tus-	FALLECE FAN	El dia de ayer	0		0	0	US	TUS-NO

Email	Phone	Days_Exp_Cre	Days_Update	Days_Exp_U	Days_Today	Total_Script	Own_Scr	Total_A_Re	Own_A_R	Total_Img_S	Own_Img_
ESNOTIC	1.4806243	2190	1825	364	1924	5	0	2	0	0	0
alejandr	51.944065	365	-1	365	200	0	0	1	0	7	0
IGOEIJO	62.822775	365	-1	365	143	0	0	1	0	0	0
contact	33.652458	730	274	455	429	1	0	15	10	15	15
anthony	1.3176476	365	6	358	116	3	0	0	0	11	0
alejandr	51.944065	365	-1	365	200	0	0	1	0	7	0
memege	1.4806243	5113	4748	364	4760	0	0	0	0	0	0
contact	33.652458	730	274	455	429	1	0	12	7	0	0
TUS-NO	1.4806243	1826	1456	369	1476	2	0	53	23	11	0

Figura 14. Porción de todos los datos obtenidos.

Por otro lado, se realizó el agrupamiento de las características con valores de tipo texto. De de la característica *title*, se obtuvo la nube de palabras ilustrado en la Figura 15., para conocer cuáles son las palabras más utilizadas en las URLs compartidas. Una nube de palabras muestra con mayor tamaño aquellas palabras cuya frecuencia de aparición es mayor comparado al resto. En la ilustración, las palabras como: "dinero", "ganar", "online", "bitcoin", "clash", son la predominantes, seguidamente: "royale",

"gana", "free", "gratis", "dragon" y en una tercera frecuencia mayoritaria: "hack", "internet", "encuestas", "facebook".

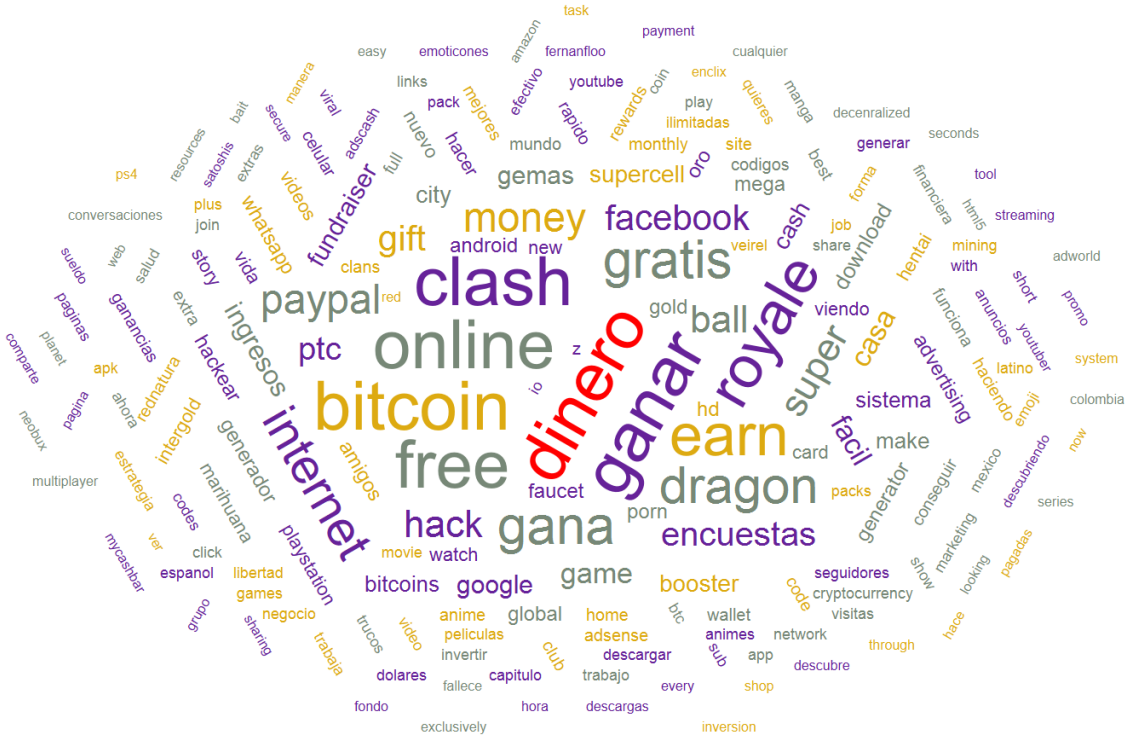


Figura 15. Nube de palabras dentro del campo *title* recolectado.

Entre los datos de tipo texto también se encuentra el país origen del sitio que aloja la URL. Para estos datos se generó la nube de palabras mostrada en la Figura 16.Se identifica que los dominios más referenciados están alojados en países como Panamá, España, Perú, Argentina, Colombia y México.

Figura 16. Nube de países de los dominios recolectados

Con respecto a los dominios, se identificó la presencia de servicios de blogs bajo el dominio: blogspot.com, wordpress.com, wix.com, jimdo.com y zohosites.com, webnode.com. Estos sitios son proporcionados gratuitamente con el propósito de propiciar la comunicación y libertad de expresión. La publicación en la web se genera mediante un subdominio, por ejemplo: miusuario.blogspot.com y su contenido incluye imágenes, videos, URLs, botones "like" de Facebook, etc.

El conjunto de datos obtenidos de estos blogs resultó con un patrón peculiar para las características: *Total_A_References, Own_A_References, Total_Img_Sources* y *Own_Img_Sources*. El patrón observado consistió en cantidades altas para enlaces a distintos dominios diferentes al blog. Esto se obtuvo mediante la diferencia de la cantidad *Total* menos la cantidad *Own* o propia del sitio, lo cual resulta en la cantidad de enlaces foráneos. Es decir, se obtuvieron muchos enlaces a distintos sitios tanto en las URLs que los blogs sugerían acceder y en el origen *src* de las imágenes de los blogs.

Haciendo la comparación con un sitio que no sea un blog, una alta cantidad de enlaces foráneos puede comprometerlo como un sitio malicioso puesto que podría conducir sus acciones maliciosas a través de los otros sitios referenciados y esto genera un punto de sospecha.

Por el contrario, los blogs tienen la peculiaridad de adjuntar varias referencias a otros sitios. En este sentido, si el modelo de clasificación es entrenado con estos datos, podría suceder que un blog se calificado como malicioso. Sin embargo, explícitamente no se conoce cuál es el tipo de contenido al que apuntan esas referencias externas. El contenido de las URLs a ese nivel podrían ser sitios maliciosos o no; por ejemplo, la Figura 17., muestra tres blogs: el 1 y 2 presentan enlaces que parecen sospechosos y el 3 presenta contenido adulto. Estos no pueden ser clasificados como maliciosos porque se desconoce el destino de las URLs referenciadas y por otro lado el contenido de adulto sí está permitido en los blogs, siempre y cuando se indique la categorización durante la creación del blog. Por este motivo el conjunto de registros de blogs fue separado del conjunto de datos.

Figura 17. Sitios de blogs. 1 y 2: Contenido sospechoso - 3: Contenido adulto

4.1. Normalización de datos

La normalización de datos de tipo numérico consistió en la definición de características con valores balanceados en base a las características obtenidas originalmente, cuyo detalle se presenta en la Tabla 5.

Característica	Normalización del valor	Fórmula
Foreign_Cookies	Resta porcentual del total de cookies menos las cookies propias del dominio.	TCQ: Total_Cookies_Quantity OCQ: Own_Cookies_Quantity $\left(1-\frac{\mathrm{OCQ}}{\mathrm{TCQ}}\right)*100$
Days_Exp_Creation_Balanced	Equivalencia de la cantidad de días en razón de años.	$\left(\frac{\mathrm{Days_Exp_Creation}}{365}\right)$
Days_Update_Creation_Balanced	El valor final que puede ser {0,1} se obtiene de condicionales de acuerdo a la recurrencia de la actualización.	DUC: Days_Update_Creation Si la modificación del dominio es en menos de un año: $5<\mathbf{DUC}<365\Longrightarrow 1$ Si la modificación del dominio es a pocos días de su creación: $\mathbf{DUC}<5\Longrightarrow 0$ Si la modificación del dominio es por renovación periódica: $\mathbf{DUC}>364\Longrightarrow 0$
Days_Exp_Update_Balanced	Equivalencia de la cantidad de días en razón de años.	$\left(\frac{\mathrm{Days_Exp_Update}}{365}\right)$

Foreign_FormAction**	Resta porcentual del total de referencias de los form action menos la cantidad de referencias al mismo dominio de la URL. Si:(TFA-OFA)<2 ⟹ **X***100 Si:2≤ (TFA-OFA)<5 ⟹ **X***500 Si: 5≤(TFA-OFA) ⟹ **X***1000	TFA: Total_Form_Action OFA: Own_Form_Action $\boldsymbol{X} = \left(1 - \frac{\text{OFA}}{\text{TFA}}\right)$
Foreign_Script_Source**	Resta porcentual del total de orígenes de los scripts, menos los orígenes al mismo dominio de la URL. Si:(TSS-OSS)<5 ⟹ **X***100 Si:5≤ (TSS-OSS)<25 ⟹ **X***500 Si: 25≤(TSS-OSS) ⟹ **X***1000	TSS: Total_Script_Source OSS: Own_Script_Source $\boldsymbol{X} = \left(1 - \frac{\text{OSS}}{\text{TSS}}\right)$
Foreign_A_References**	Resta porcentual del total de enlaces encontrados menos los enlaces al mismo dominio de la URL. Si:(TAR-OAR)<5 ⟹ **X***100 Si:5≤ (TAR-OAR)<25 ⟹ **X***500 Si: 25≤(TAR-OAR) ⟹ **X***1000	TAR: Total_A_References OAR: Own_A_References $\left(1 - \frac{\text{OAR}}{\text{TAR}}\right)$
Foreign_Img_Sources**	Resta porcentual del total de orígenes de las imágenes menos los orígenes al mismo dominio de la URL. Si:(TIS-OIS)<5 ⟹ **X***100 Si:5≤ (TIS-OIS)<25 ⟹ **X***500 Si: 25≤(TIS-OIS) ⟹ **X***1000	TIS: Total_Img_Sources OIS: Own_ Img_Sources $\left(1 - \frac{\text{OIS}}{\text{TIS}}\right)$
Quantity_iFrames	Mismo valor	----------
** Se valoriza con mayor proporción entre 100, 500 y 1000 mientras sea mayor la cantidad de dominios foráneos referenciados.		

Tabla 5. Normalización de datos numéricos

El balance o equivalencia de los valores y la conjugación de características, se realizó con la finalidad de optimizar el procesamiento del algoritmo de ML. Específicamente porque las cantidades altas (por ejemplo, los días entre las fechas) proporcionan el mismo valor significativo que al dividirlos por un valor promedio (por ejemplo 365días). Con respecto a las fórmulas, se utilizan los valores de enlaces del Total y el Own (propio) para obtener un sólo valor Foreign, que representa a los enlaces foráneos.
Tal como se indica en la Tabla 5., se designó tres valores: 100, 500 y 1000 como factores de multiplicación para los enlaces foráneos. Por ejemplo, si el Total=30 enlaces y los propios Own=28, entonces la diferencia=2 es mínima y el cálculo es ponderado por 100. En cambio, cuando la diferencia es mayor, por ejemplo Total=30 y Own=4, significa que 26 enlaces son foráneos y por tanto es ponderado por 1000. Todo esto bajo la consideración que la mayor cantidad de enlaces foráneos puede tener relación con el contenido malicioso.

4.2. Etiquetado de datos

El etiquetado de datos consistió en asignar manualmente entre la etiqueta "MALICIOUS" y "NOT_MALICIOUS" a cada uno de los registros de las características de las URLs. En esta actividad es importante resaltar que se observaron sitios con imágenes y videos pornográficos, tanto de personas como en animes. Este tipo de páginas fueron retiradas del conjunto de datos, debido a que poseen características de sitios legítimos y el modelo de clasificación proporcionaría la etiqueta de "NOT_MALICIOUS"; sin embargo, estas URLs se obtuvieron de comentarios y publicaciones públicas de Facebook, las cuales menores de edad puedan acceder. En ese contexto, sí podría considerarse como malicioso, lo cual puede depender de la hora de la publicación y en que hilo de comentarios fue publicado, lo cual requiere otro tipo de análisis relacionado a la publicación en sí.

Por otro lado, entre las URLs calificadas como "NOT_MALICIOUS", se observaron registros con similares valores, los cuales también fueron eliminados debido a que no presentaban información sustancial al entrenamiento del modelo de clasificación.

4.3. Representación gráfica de las características

Hasta este punto se tiene el conjunto de datos con sus respectivas etiquetas con un total de 9 características, las cuales son:

{*Foreign_Cookies, Days_Exp_Creation_Balanced, Days_Update_Creation_Balanced, Days_Exp_Update_Balanced, Foreign_FormAction, Foreign_Script_Source, Foreign_A_References, Foreign_Img_Sources, Quantity_iFrames*}.

Estas características multidimensionales presentan la limitación de poder mostrarse en un sólo gráfico porque la visualización comprensible del plano cartesiano se consigue como máximo tres dimensiones. Por tal motivo, generalmente los modelos multidimensionales muestran gráficos en pares de características.

En las Figuras: 18, 19, 20, 21 y 22., se muestran las características en planos bidimensionales, que fueron obtenidas con el código adjuntado en el Anexo G. Es necesario mencionar que en algunos gráficos los valores para el contenido tipo MALICIOUS y NOT_MALICIOUS estarán superpuestos. Sin embargo, la adecuada discriminación y clasificación del tipo de contenido se logra mediante la conjugación con del resto valores del conjunto de características.

El código en el anexo mencionado incluye librerías que también son referenciadas en los códigos desarrollados para la construcción del modelo y para la calificación de las características, motivo por el cual, estas librerías son descritas a continuación.

- **Pandas**: Librería que permite trabajar con estructuras de datos en Python, así como el análisis de datos y está implementado para trabajar con diferentes tipos de datos provenientes de bases de datos, estadísticas, etc.

- **Numpy**: Librería que permite realizar funciones matemáticas y estadísticas de manera eficiente. Básicamente, introduce los objetos

en arreglos multidimensionales, los cuales permiten operaciones por bloques de manera equivalente a operaciones escalares.

- **Sklearn**: Librería que permite acceder y utilizar los algoritmos de Machine Learning en Python. A partir de este repositorio se hace referencia a Support Vector Machines (SVM).

- **Matploitlib**: Librería que permite generar gráficos, histogramas, espectros de potencia, gráficos de barras.

- **Seaborn**: Esta librería está basada en matplotlib. Proporciona una interfaz de alto nivel para gráficos estadísticas atractivas.

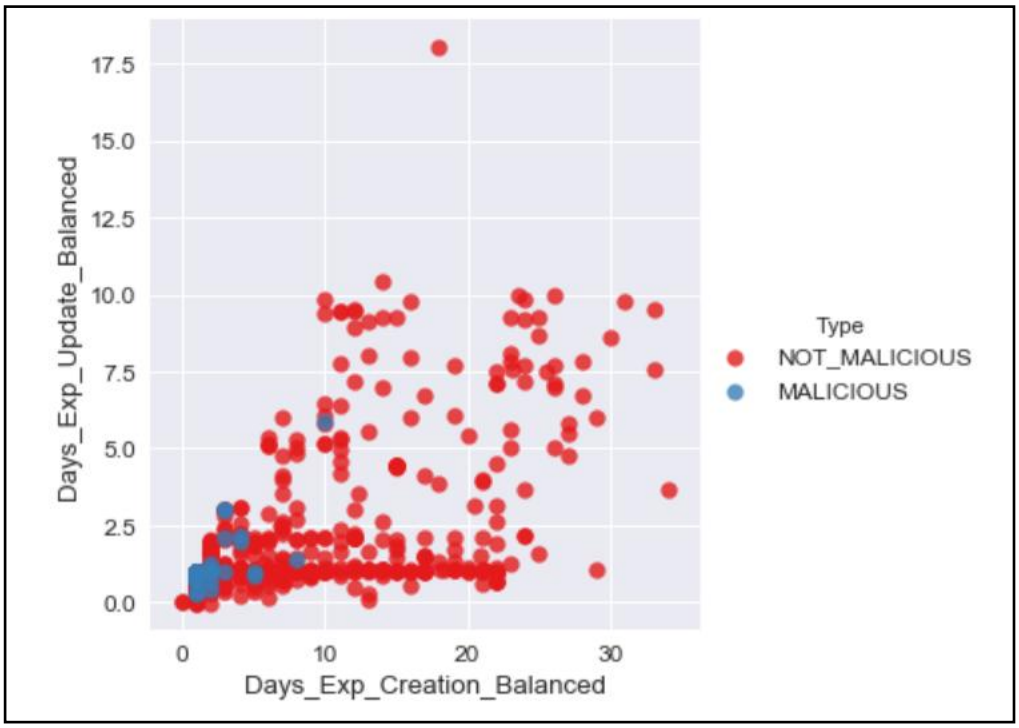


Figura 18. Sitios Days_Exp_Creation_Balanced vs. Days_Exp_Update.

En la Figura 18., se observa que el contenido malicioso está presente en valores bajos de Days_Exp_Creation_Balanced, es decir está albergado en sitios cuyo dominio tiene un tiempo de expiración corta. De igual manera en el caso de Days_Exp_Update presenta valores menores a 2.5 días como periodo de actualización del sitio.

La Figura 19., se puede observar que Foreign_Script_Source toma el valor más alto: 1000 para el contenido malicioso, lo cual indica que las fuentes

"src" de los scripts diferentes al dominio de la URL superan la cantidad de 25. Por otro lado, para el valor de 500 el contenido no malicioso tiene mayor presencia. Esto indica que la cantidad de dominios diferentes es aceptable si está en el rango entre 5 y 25. Para el caso del valor de 100, también es mayor la presencia de contenido no malicioso puesto que la cantidad de dominios diferentes es menor que 5.

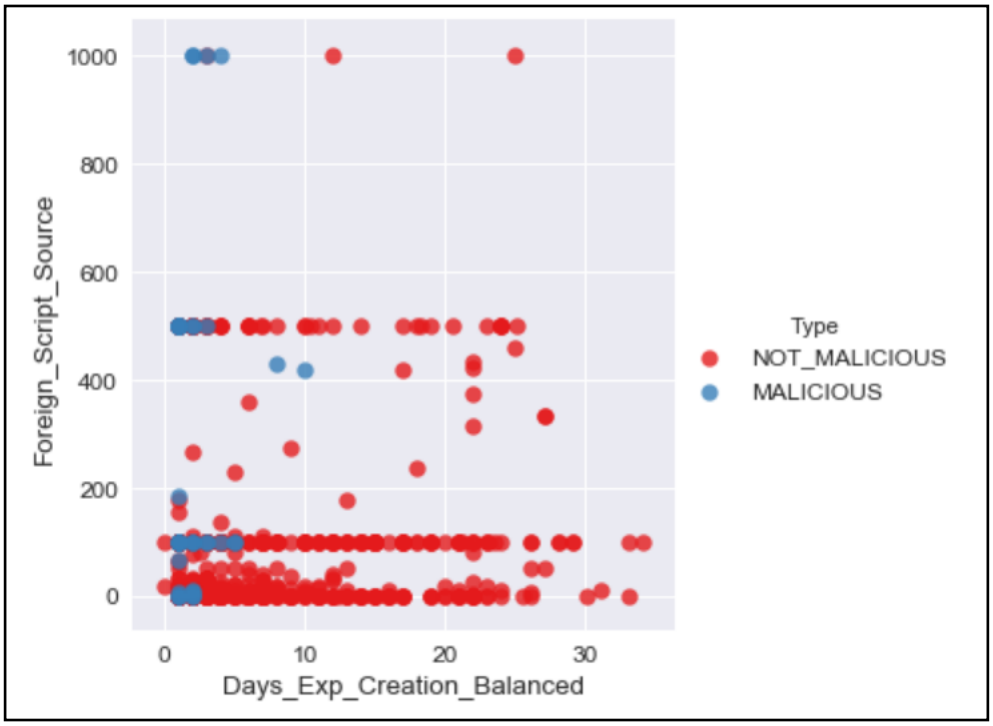


Figura 19. Days_Exp_Creation_Balanced vs. Foreign_Script_Source.

En la Figura 20., el contenido malicioso alcanza el máximo valor para las redirecciones foráneas de los formularios en la característica Foreign_FormAction. Sin embargo, ése es el único pico puesto que el resto de los valores son similares al contenido no malicioso.

La Figura 21., resulta dispersa para ambas coordenadas. En caso de la característica Foreign_A_References, que se refiere a los enlaces incluidos en la URL están presentes en el contenido malicioso y no malicioso, de manera similar ocurre con las fuentes de las imágenes de las URLs en la característica Foreign_Img_Sources.

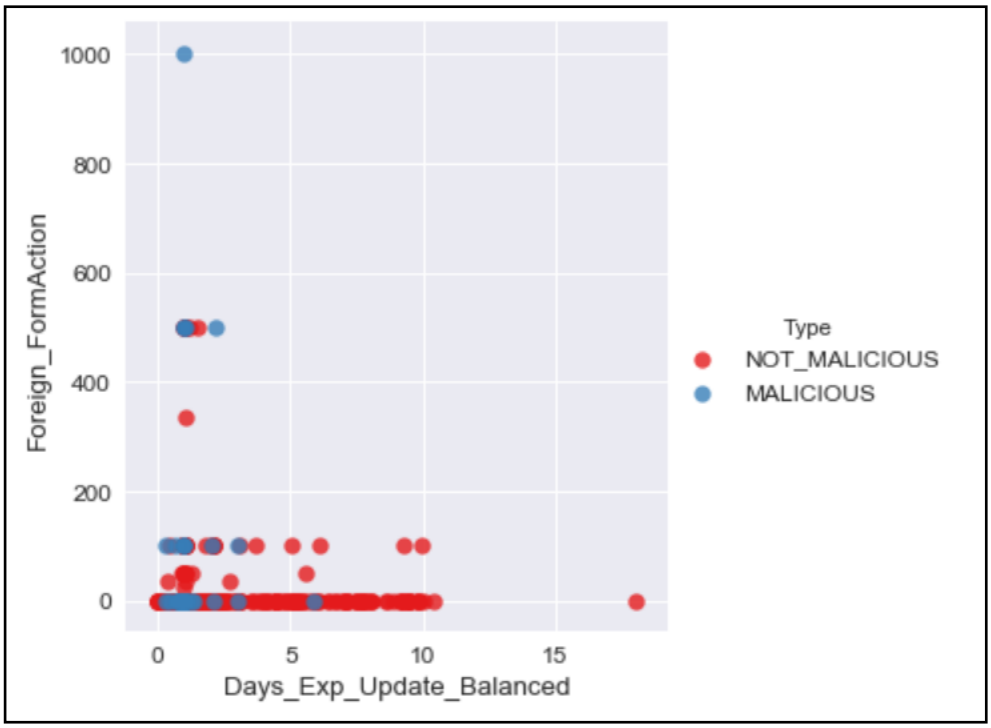


Figura 20. Days_Exp_Update vs. Foreign_FormAction.

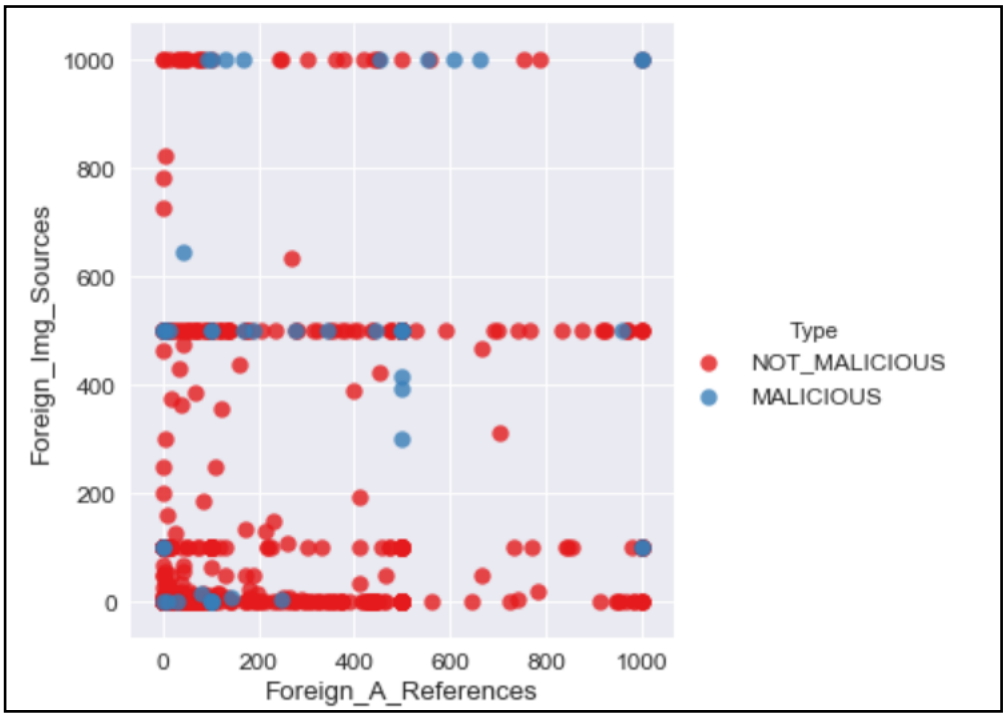


Figura 21. Foreign_A_References vs. Foreign_Img_Sources.

Otra de las características con valores dispersos es la cantidad de iFrames u objetos embebidos bajo la característica Quantity_iFrames que se observa en la Figura 22.

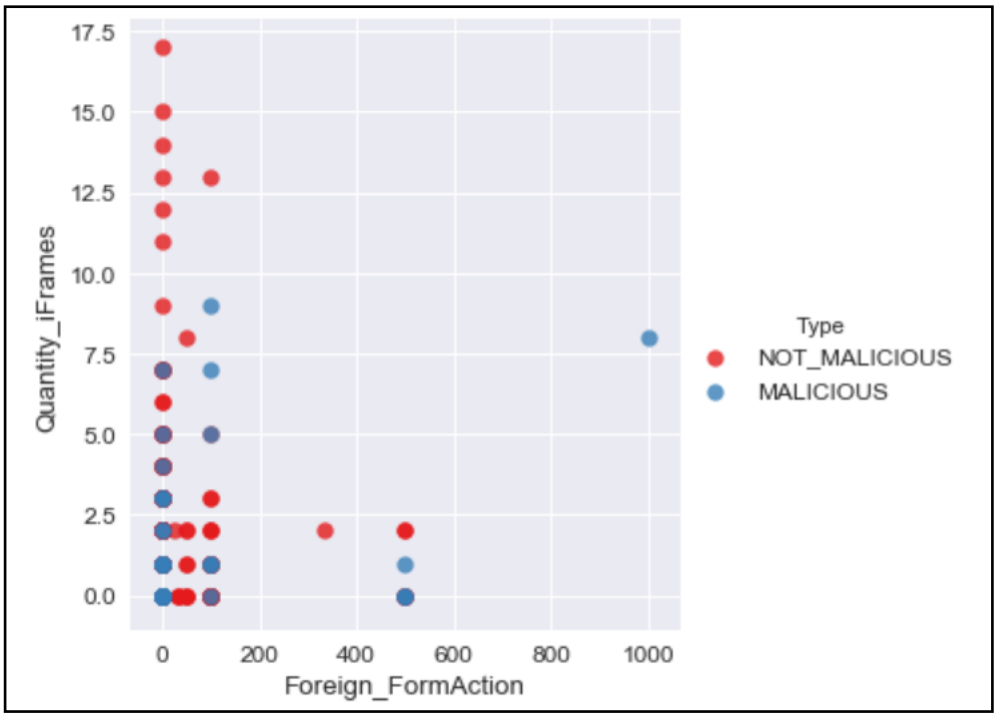


Figura 22. Foreign_Form_Action vs. Quantity_Frames.

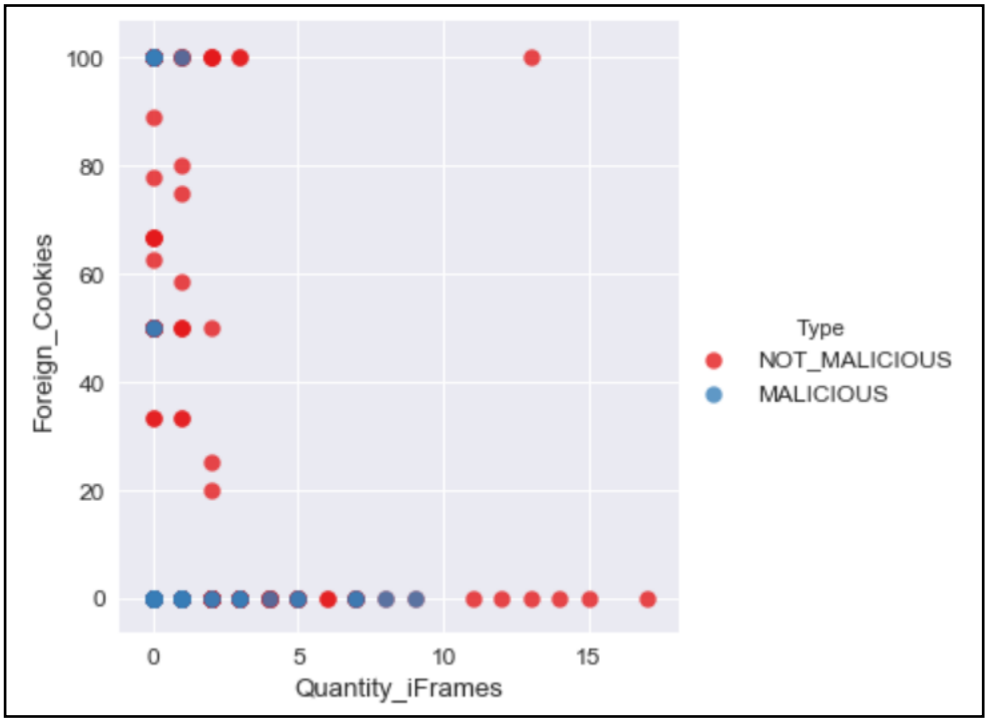


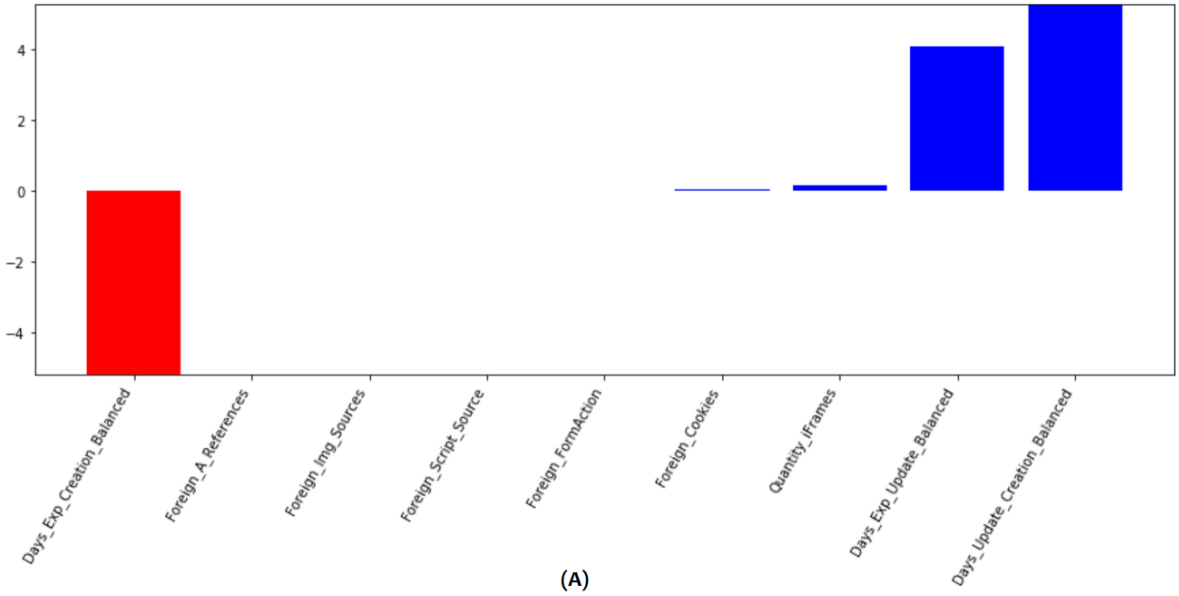
Figura 23. Quantity_Frames vs. Foreign_Cookies.

Finalmente, en la Figura 23., se observa mayor proporción de contenido malicioso en la parte inferior para el caso de cookies foráneas, así como cantidades menores a 10 con respecto a los objetos embebidos.

4.4. Características relevantes

A partir de un modelo lineal SVM es posible determinar el valor informativo o importancia de cada característica de acuerdo al valor de los coeficientes resultantes de los vectores obtenidos del modelo. Esto sólo es posible en modelos SVM lineales o de clasificación binaria [35].

La importancia de cada característica se obtiene a partir del valor absoluto de los coeficientes. Por ese motivo a pesar de obtener (-5.158601) en *Days_Exp_Creation_Balanced,* esta característica es la segunda más informativa para la clasificación del modelo. El código adjuntado en el Anexo H permite obtener los resultados de la Figura 24.



(A)

```
5.286500        Days_Update_Creation_Balanced
5.158601        Days_Exp_Creation_Balanced
4.070493        Days_Exp_Update_Balanced
0.165718        Quantity_iFrames
0.054685        Foreign_Cookies
0.024394        Foreign_FormAction
0.009942        Foreign_Script_Source
0.002848        Foreign_A_References
0.000620        Foreign_Img_Sources
```

(B)

Figura 24. (A) Coeficientes de las características
(B) Valor absoluto y orden de descendente de las características más informativas

Las características con los valores más altos, en este caso las 7 primeras: Days_Update_Creation_Balanced, Days_Exp_Creation_Balanced, Days_Exp_Update_Balanced, Quantity_iFrames, Foreign_Cookies, Foreign_FormAction y Foreign_Script_Source son las más relevantes y su presencia en la construcción del modelo de clasificación son necesarias.

4.5. Codificación del modelo

En el Anexo I, se adjunta el código desarrollado para alimentar el modelo de clasificación SVM. A continuación, en la Tabla 6. se resaltan las líneas de código más importantes:

PARTE 1	`#features_names: Características`
	`features_names=['Foreign_Cookies',`
	`'Days_Exp_Creation_Balanced',`
	`'Days_Update_Creation_Balanced',`
	`'Days_Exp_Update_Balanced',`
	`'Foreign_FormAction',`
	`'Foreign_Script_Source',`
	`'Quantity_iFrames']`
		Se inicializa el conjunto de características más relevantes para el modelo.

PARTE 2	`#features_values: Contenido de valores de las características`
	`features_values=`
	`url_features[features_names].as_matrix()`
	
	`#type_label: Contenido de la etiqueta de datos`
	`type_label=`
	`np.where(url_features['Type']=='MALICIOUS',1,0)`
		De todo el conjunto de datos de 'url_features' se obtiene los valores de las características.
		Si es del tipo MALICIOUS se reemplaza por 1, sino 0.

PARTE 3	`# Instancia del modelo de clasificación SVM`
	`model =svm.SVC(kernel='linear', C=100)`
	`model.fit(features_values,type_label)`
		Se inicializa el modelo con un kernel lineal y con el valor de C en 100. Mientras mayor sea C, mayor es la optimización porque se eligen pequeños márgenes en los hiperplanos.

PARTE 4	`# Método para clasificar si una URL es maliciosa o no`
	`def malicious_or_not(Foreign_Cookies,`
	`Days_Exp_Creation_Balanced,`
	`Days_Update_Creation_Balanced,`
	`Days_Exp_Update_Balanced,`
	`Foreign_FormAction,`
	`Foreign_Script_Source,`
	`Quantity_iFrames):`
	
	`if(model.predict([[Foreign_Cookies,`
	`Days_Exp_Creation_Balanced,`
	`Days_Update_Creation_Balanced,`
	`Days_Exp_Update_Balanced,`
	`Foreign_FormAction,`
	`Foreign_Script_Source,`
	`Quantity_iFrames]]))==0:`
	`print('URL accesible')`
	`else:`
	`print('Esta URL posee contenido con características maliciosas')`
		Se define una función "malicious_or_not" que recibe los valores de las características y retorna la respectiva clasificación en un mensaje.

Tabla 6. Descripción de las líneas de código del modelo de clasificación SVM.

En la Figura 25., se observa la llamada al método de la PARTE 4 de la Tabla 6., donde efectivamente la serie de datos de la última llamada al método corresponde a una URL maliciosa.

Es importante aclarar que toda URL primero debe pasar por un algoritmo que obtiene el valor de estas características y seguidamente pasárselas al método "malicious_or_not" para recibir la respuesta. El detalle de estos procesos está presentado en el numeral 4.7 Implementación del modelo en un sitio web.

```
# Llamada al método
malicious_or_not(100,2.002739726,1,1.04109598,0,0,9)
malicious_or_not(50,11.00821918,0,6.37260274,0,17.39130435,7)
malicious_or_not(50,7.005479452,0,1.082191781,33.33333333,4.651162791,6)
malicious_or_not(50,1,0,1.016438356,500,500,12)

URL accesible
URL accesible
URL accesible
Esta URL posee contenido con características maliciosas
```

Figura 25. Llamada al método de clasificación.

4.6. Validación cruzada

Existe un test de validación cruzada que permite obtener el nivel de precisión del modelo.

En [36], se presenta un código desarrollado para obtener el puntaje de precisión. Este código fue modificado para trabajar con los datos recolectados de esta investigación, el cual se adjunta en el Anexo J.
El puntaje de precisión obtenido resultó: 0.934687953556, que se considera aceptable para el modelo de clasificación implementado.

Como se puede observar en el Anexo J, las características utilizadas no fueron todas, sino solamente las siete características obtenidas en el numeral 4.4. Características relevantes, que corresponden a las más significativas del modelo. Posteriormente y con fines de comparación, en la Figura 26., se muestra el algoritmo alimentado con todas las características, donde el puntaje de precisión resultó: 0.931785195936, que es menor al valor obtenido previamente.

```python
from sklearn.model_selection import LeaveOneOut
from sklearn import datasets
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score
import pandas as pd

#load data
allURLsData = pd.read_csv('Final_With_Types-Copy.csv')

features_names = ['Foreign_Cookies','Days_Exp_Creation_Balanced', 'Days_Update_Creation_Balanced', 'Days_Exp_Update_Balanced',
 'Foreign_FormAction', 'Foreign_Script_Source', 'Foreign_A_References', 'Foreign_Img_Sources','Quantity_iFrames']

X = allURLsData[features_names].as_matrix()
Y = allURLsData['Type'].as_matrix()

#build the model
svm_model = svm.SVC(kernel='linear', C=2**-5, gamma = 'auto',random_state = 0 )

#create the Cross validation object
loo = LeaveOneOut()

#calculate cross validated (leave one out) accuracy score
scores = cross_val_score(svm_model, X,Y, cv = loo, scoring='accuracy')

print( scores.mean() )#Result (the mean accuracy of the 150 folds since we used leave-one-out):
```

```
0.931785195936
```

Figura 26. Validación cruzada de todas las características del modelo.

4.7. Implementación del modelo en un sitio web

Bajo el objetivo de hacer accesible el modelo a los usuarios, se implementó un sitio en PHP[12]. Este lenguaje de programación es extensamente explicado en la web, motivo por el cual sólo se detallará la estructura que corresponde a la implementación del modelo y su respectiva consulta.

Para lograr inicializar el modelo y mantenerlo en memoria fue necesario utilizar Joblib [37]. Éste es un conjunto de herramientas en Python que permite el almacenamiento rápido de los outputs o valores de salida en la caché del disco. Su respuesta es óptima para objetos de alta dimensión como matrices muy grandes de manera que posteriormente sea instanciado en otros procesos. Es recomendado para retener en memoria los modelos de Machine Learning.

[12]PHP: HypertextPreprocessor, es un lenguaje de código abierto especialmente adecuado para el desarrollo web. https://www.w3schools.com/php/

Figura 27. Página para inicializar el modelo

A continuación se detallan las dos páginas desarrolladas, que hacen posible la inicialización y consulta a este modelo:

- **index.php**: El código se adjunta en el Anexo K. Esta página permite cargar el modelo en memoria, de manera que después sea instanciado al momento de ingresar una URL. En la Figura 27. se muestra la interfaz, donde aparece un botón "Inicializar". Una vez que el usuario haga clic en este botón, se llama al código en Python, adjuntado en el Anexo L. Este código carga los datos recolectados e inicializa el modelo, seguidamente se invoca a Joblib para retener el modelo en memoria. Cuando termina este proceso, la página hace una redirección al formulario para ingresar la URL a consultar.

- **formConsultURL.php**: El código se adjunta en el Anexo M. Esta página permite ingresar una URL e indicar si se trata de una URL corta para que internamente se haga la traducción a su respectiva URL original. En la Figura 28., se observa la respuesta que retorna al consultar una URL. Una vez que el usuario haga clic en el botón Procesar, se invoca al código en Python adjuntado en el Anexo N, donde se consulta la URL y se extraen las características que son alimentadas al modelo. El modelo que fue previamente retenido en memoria, es invocado mediante Joblib y finalmente se obtiene el resultado que indica si la URL presenta características maliciosas o no.

Figura 28. Página de consulta de la URL

5. Conclusiones

1. Es importante controlar el acceso a las URLs publicadas en las redes sociales en línea, puesto que existen URLs con contenido malicioso que son propagadas por este medio.
2. La recopilación de datos actuales permitió conocer el contenido malicioso que se está propagando, las estadísticas respectivas y el mecanismo de propagación mediante los servicios de acortamiento de URLs.
3. El aprendizaje supervisado de Machine Learning, en este caso con el algoritmo de Support Vector Machines, resultó óptimo para la detección de contenido malicioso, a través de las características seleccionadas.
4. Entre las características más relevantes del modelo se encuentra que el contenido malicioso presenta menor tiempo de creación y expiración del sitio web y mayor cantidad de referencias externas a otros sitios en las acciones de los formularios y los scripts.
5. La selección de características y el etiquetado de los datos, son las tareas que necesitaron mayor dedicación y en base a éstas se obtuvo 0.93 de puntaje de precisión, el cual puede ser mejorado considerando otras nuevas características y descartando la extracción de las que resultaron poco relevantes para el modelo.
6. El contenido malicioso observado, corresponde a aquello que puede obstaculizar el funcionamiento natural de la navegación web, como la ejecución de scripts, propagación de virus, troyanos y exploitkits. Sin embargo, también se observa la violación de los derechos de autor por la presencia de URLs que apuntan a sitios de descarga de películas, música, videos, juegos, etc.
7. Se observa una gran cantidad de URLs cortas, las cuales deben ser tratadas como una posible amenaza porque se desconoce el destino final de la URL. Por tanto deben ser traducidas previamente a su acceso.
8. La utilización de las URLs por algún servicio de acortamiento que paga por cada enlace accedido es una forma atractiva para propagar

contenido malicioso, porque además de generar una ganancia también logra el acceso al sitio malicioso. Especialmente, estas URLs cortas son compartidas en los temas virales, que resultan atractivos a los usuarios.

9. Existe un conjunto de URLs cortas cuya traducción es otra URL corta pasando por más de dos niveles de traducción antes de obtener la URL destino. Estas URLs en su mayoría dirigen a sitios pornográficos y repositorios de descarga de archivos.
10. Se observó una gran cantidad de URLs con temáticas de ganancia de dinero y minería de bitcoins que precisan la instalación de software de repositorios con dudosa procedencia los cuales deben ser examinados antes de su aceptación.
11. Se observaron URLs que acceden a sitios legítimos como Blogs y videos de Youtube, que a su vez incluían otras URLs con posible contenido malicioso, cuyo acceso debe ser controlado.
12. Es necesaria la implementación de salvaguardas en los navegadores web, específicamente éstos deben contar la función de sandboxing, que muestran mensajes de advertencia sobre los accesos (a la cámara, micrófono, descarga de archivos, etc.) que requiere el sitio web. De esta manera el usuario puede aceptar o denegar los permisos solicitados.

6. Anexos

Anexo A: Código para obtener países de los números telefónicos.

```
from os import listdir
from os.path import isfile, join
import codecs
import csv

defcompactData():
    mypath ='E:\UBA\TTFFM-2017\Notebook-Python\ScrapComments\All-Phones\Results'

    fPhonesname ='collectedPhonesWithCountryLATAM.csv'
with codecs.open(fPhonesname,'wb','utf-8')as fPhone:
        fPhone.write('Phone,Country\n')
    cvsFileName = mypath +'\\'+'collectedPhones.csv'

    cant =0
with open(cvsFileName,'r')as fCsv:
with open(fPhonesname,'ab')as fPhone:
            w = csv.writer(fPhone)
for row in csv.DictReader(fCsv):
                 numberPhone = row['Telefonos ']
if('502'in numberPhone[:4]or'0502'in numberPhone[:5]):
                    w.writerow([numberPhone,'GT'])
elif('503'in numberPhone[:4]):
                    w.writerow([numberPhone,'SV'])
elif('504'in numberPhone[:4]):
                    w.writerow([numberPhone,'HN'])
elif('505'in numberPhone[:4]or'0505'in numberPhone[:5]
                              or'00505'in numberPhone[:6]):
                    w.writerow([numberPhone,'NI'])
elif('506'in numberPhone[:4]or'00506'in numberPhone[:6]):
                    w.writerow([numberPhone,'CR'])
elif('507'in numberPhone[:4]):
                    w.writerow([numberPhone,'PA'])
elif('55'in numberPhone[:3]or'15'
               in numberPhone[:3]or'21'in numberPhone[:3]or'0021'in numberPhone[:5]
               or'021'in numberPhone[:4]):
                    w.writerow([numberPhone,'BR'])
elif('57'in numberPhone[:3]or'057'in numberPhone[:3]
               or'0057'in numberPhone[:5]):
                    w.writerow([numberPhone,'CO'])
elif('51'in numberPhone[:3]or'051'in numberPhone[:4]
or('9'in numberPhone[:2]and len(numberPhone)==10)):
                    w.writerow([numberPhone,'PE'])
elif('54'in numberPhone[:3]):
                    w.writerow([numberPhone,'AR'])
elif('52'in numberPhone[:3]or'052'in numberPhone[:4]):
                    w.writerow([numberPhone,'MX'])
elif('56'in numberPhone[:3]or'18'in numberPhone[:3]
               or'12'in numberPhone[:3]or'17'in numberPhone[:3]):
                    w.writerow([numberPhone,'CL'])
elif('58'in numberPhone[:3]):
                    w.writerow([numberPhone,'VE'])
elif('591'in numberPhone[:4]):
                    w.writerow([numberPhone,'BO'])
elif('593'in numberPhone[:4]):
                    w.writerow([numberPhone,'EC'])
elif('595'in numberPhone[:4]):
                    w.writerow([numberPhone,'PY'])
elif('598'in numberPhone[:4]):
                    w.writerow([numberPhone,'UY'])
else:
                    cant = cant +1
                    w.writerow([numberPhone,'OTHERS'])

print('Numero de telefonos:'+str(cant))
```

Anexo B: Parte 1: Código para separar las URLs por grupos.

```
from os import listdir
from os.path import isfile, join
import codecs
import csv
import tldextract

defcompactData():
    myPath ='E:\UBA\TTFFM-2017\Notebook-Python\ScrapComments\All-URLs'

    fileSpecificURL ='AllUrlsSpecific.csv'
with codecs.open(fileSpecificURL,'wb','utf-8')as fSpecificURL:
        fSpecificURL.write('urls \n')

    fileSpecificShortURL ='AllShortUrlsSpecific.csv'
with codecs.open(fileSpecificShortURL,'wb','utf-8')as fShortSpecificURL:
        fShortSpecificURL.write('urls \n')

    fileFacebookURL ='FacebookUrls.csv'
with codecs.open(fileFacebookURL,'wb','utf-8')as fFbURL:
        fFbURL.write('urls \n')

    fileWhatsAppURL ='WhatsAppUrls.csv'
with codecs.open(fileWhatsAppURL,'wb','utf-8')as fWAURL:
        fWAURL.write('urls \n')

    fileOtherSNURL ='OtherSocialNetworksUrls.csv'
with codecs.open(fileOtherSNURL,'wb','utf-8')as fOSNURL:
        fOSNURL.write('urls \n')

    csv_filename = myPath +"\\"+'NotDuplicatedListOfURLs.csv'

    cant =0
    cantSpecific =0
    cantShortSpecific =0
    cantFb =0
    cantWA =0
    cantONS =0

with open(csv_filename,'r')as fCsv:
for row in csv.DictReader(fCsv):
            webAddress = row['urls ']
            cant = cant +1
if('.facebook.com'in webAddress):
with open(fileFacebookURL,'ab')as fFbURL:
                    w = csv.writer(fFbURL)
                    w.writerow([webAddress])
                    cantFb = cantFb +1
elif('chat.whatsapp.com'in webAddress):
with open(fileWhatsAppURL,'ab')as fWAURL:
                    w = csv.writer(fWAURL)
                    w.writerow([webAddress])
                    cantWA = cantWA +1
elif('.youtube.com'in webAddress or'youtu.be'in webAddress
                or'instagram.com'in webAddress or'twitter.com'in webAddress
                or'.google.'in webAddress or'giphy.com'in webAddress
or'flickr.com'in webAddress):
with open(fileOtherSNURL,'ab')as fOSNURL:
                    w = csv.writer(fOSNURL)
                    w.writerow([webAddress])
                    cantONS = cantONS +1

else:
                ext = tldextract.extract(webAddress)
                shortDomainUrl =''
if(ext.subdomain):
                    shortDomainUrl =ext.subdomain+"."+ ext.domain +"."+ext.suffix
else:
shortDomainUrl=ext.domain+"."+ext.suffix
```

Anexo B: Parte 2: Código para separar las URLs por grupos.

```
if(len(shortDomainUrl)<10and('bit.ly'in shortDomainUrl
                or'goo.gl'in shortDomainUrl or'cort.as'in shortDomainUrl
                or'ow.ly'in shortDomainUrl or'adf.ly'in shortDomainUrl
                or'short.am'in shortDomainUrl or'shink.in'in shortDomainUrl
                or'bc.vc'in shortDomainUrl or'adbull.me'in shortDomainUrl
                or'xurl.es'in shortDomainUrl or'uni.vi'in shortDomainUrl
                or'gph.is'in shortDomainUrl or't.co'in shortDomainUrl
                or'x.co'in shortDomainUrl or'a.co'in shortDomainUrl
                or'fas.li'in shortDomainUrl)):
with open(fileSpecificShortURL,'ab')as fShortSpecificURL:
                            w = csv.writer(fShortSpecificURL)
                            w.writerow([webAddress])
                            cantShortSpecific = cantShortSpecific +1
else:
with open(fileSpecificURL,'ab')as fURL:
                        w = csv.writer(fURL)
                        w.writerow([webAddress])
                        cantSpecific = cantSpecific +1

print('Proceso terminado')

print('Numero de Urls total:'+ str(cant))
print('Numero de Urls especificas:'+ str(cantSpecific))
print('Numero de Urls especificas con link cortos:'+ str(cantShortSpecific))
print('Numero de Urls de Facebook:'+str(cantFb))
print('Numero de Urls de WhatsApp:'+str(cantWA))
print('Numero de Urls de Otras Redes Sociales:'+str(cantONS))
```

Anexo C: Código para expandir las URLs cortas.

```
import codecs
import sys
import logging
from csv import DictReader
import requests
import json

defunshort(url):
"""
    Recibe una URL corta como texto y retorna la URL expandida usando la API.
"""
if url.startswith('https'):
        url = url[8:]
if url.startswith('http'):
        url = url[7:]
    query_url ='https://unshorten.me/json/'+ format(url)
    r = requests.get(query_url)
    output_json = json.loads(r.text)
    end_url = output_json['resolved_url']
if end_url.startswith('http')or end_url.startswith('https'):
return output_json['resolved_url']
else:
return output_json['requested_url'][-2]

defsave_file_proc(path, csv_filename):
    fullPathToSave = path +'\\'+'unshort.csv'
    fullPathToRead = path +'\\'+ csv_filename

with codecs.open(fullPathToSave,'w','utf8')as f:
with open(fullPathToRead,'r')as csv:
for row in DictReader(csv):
                url = row['urls ']
if((len(url)<30)andnot('facebook'in url or'youtube'in url
                     or'youtu.be'in url or'adf.ly'in url or'shink.in'in url)):
print(url)
                   unshortened_url = unshort(url)
ifnot(url.endswith('/')):
if(unshortened_url.endswith('/')):
                            unshortened_url = unshortened_url[:-1]
if(url != unshortened_url):
                       output = url +','+ unshortened_url +'\n'
else:
                       output = url +'\n'
else:
print('no entro')
                   output = url +'\n'
                f.write(output)
print('Proceso finalizado')
```

Anexo D: Parte 1: Código para los niveles de traducción de URLs cortas.

```
from os import listdir
from os.path import isfile, join
import codecs
import csv
import tldextract

defExtractLongURLAndLevels():
    myPath ='E:\UBA\TTFFM-2017\Notebook-Python\ScrapComments\All-URLs\Results'

    fileLongURLFifthLevel ='LongURL-FifthLevel.csv'
with codecs.open(fileLongURLFifthLevel,'wb','utf-8')as fFifth:
        fFifth.write('url\n')

    fileLongURLFourthLevel ='LongURL-FourthLevel.csv'
with codecs.open(fileLongURLFourthLevel,'wb','utf-8')as fFourth:
        fFourth.write('url\n')

    fileLongURLThirdLevel ='LongURL-ThirdLevel.csv'
with codecs.open(fileLongURLThirdLevel,'wb','utf-8')as fThird:
        fThird.write('url\n')

    fileLongURLSecondLevel ='LongURL-SecondLevel.csv'
with codecs.open(fileLongURLFourthLevel,'wb','utf-8')as fSecond:
        fSecond.write('url\n')

    fileLongURLFirstLevel ='LongURL-FistLevel.csv'
with codecs.open(fileLongURLFirstLevel,'wb','utf-8')as fFirst:
        fFirst.write('url\n')

    fileFinalLongURL ='FinalLongURL.csv'
with codecs.open(fileFinalLongURL,'wb','utf-8')as fFinal:
        fFinal.write('url\n')

    fileShortURLsFoundInLevels ='ShortURLsFoundInLevels.csv'
with codecs.open(fileShortURLsFoundInLevels,'wb','utf-8')as fShort:
        fShort.write('url\n')

    csv_filename = myPath +"\\"+'6-AllLongURLsAfterService.csv'
    cantFifth =0
    cantForth =0
    cantThird =0
    cantSecond =0
    cantFirst =0
    cantErrorURL =0#with *
with open(csv_filename,'r')as fCsv:
with open(fileFinalLongURL,'ab')as fFinal:
with open(fileShortURLsFoundInLevels,'ab')as fShort:
                wf = csv.writer(fFinal)
                wt = csv.writer(fShort)
for row in csv.DictReader(fCsv):
if(row['FifthLink']):
                        cantFifth = cantFifth +1
                        wf.writerow([row['FifthLink']])
                        wt.writerow([row['FourthLink']])
                        wt.writerow([row['ThirdLink']])
                        wt.writerow([row['SecondLink']])
                        wt.writerow([row['FirstLink']])
with open(fileLongURLFifthLevel,'ab')as fFifth:
                            wfifth = csv.writer(fFifth)
                            wfifth.writerow([row['FifthLink']])

elif(row['FourthLink']):
                        cantForth = cantForth +1
                        wf.writerow([row['FourthLink']])
                        wt.writerow([row['ThirdLink']])
                        wt.writerow([row['SecondLink']])
                        wt.writerow([row['FirstLink']])
with open(fileLongURLFourthLevel,'ab')as fFourth:
                            wfourth = csv.writer(fFourth)
                            wfourth.writerow([row['FourthLink']])
```

Anexo D: Parte 2: Código para agrupar los niveles de URLs cortas.

```
elif(row['ThirdLink']):
                        cantThird = cantThird +1
                        wf.writerow([row['ThirdLink']])
                        wt.writerow([row['SecondLink']])
                        wt.writerow([row['FirstLink']])
with open(fileLongURLThirdLevel,'ab')as fThird:
                            wthird = csv.writer(fThird)
                            wthird.writerow([row['ThirdLink']])

elif(row['SecondLink']):
                        cantSecond = cantSecond +1
                        wf.writerow([row['SecondLink']])
                        wt.writerow([row['FirstLink']])
with open(fileLongURLSecondLevel,'ab')as fSecond:
                            wsecond = csv.writer(fSecond)
                            wsecond.writerow([row['SecondLink']])
else:
if((row['FirstLink'])and'*'== row['FirstLink']):
                            cantErrorURL = cantErrorURL +1
else:
                            cantFirst = cantFirst +1
                            wf.writerow([row['FirstLink']])
with open(fileLongURLFirstLevel,'ab')as fFirst:
                                wFirst = csv.writer(fFirst)
                                wFirst.writerow([row['FirstLink']])

print('Proceso terminado')

print'cantFifth', cantFifth
print'cantForth',cantForth
print'cantThird', cantThird
print'cantSecond',cantSecond
print'cantFirst',cantFirst
print'cantError',cantErrorURL
```

Anexo E: Código para agrupar los sitios populares de las URLs cortas.

```
from os import listdir
from os.path import isfile, join
import codecs
import csv
import tldextract

defCountByService():
    myPath ='E:\UBA\TTFFM-2017\Notebook-Python\ScrapComments\All-URLs\Results'

    csv_filename = myPath +"\\"+'7-5-LongURL-FifthLevel.csv'
       #Paths: 7-1-LongURL-FistLevel 7-2-LongURL-SecondLevel
       #7-3-LongURL-ThirdLevel 7-4-LongURL-FourthLevel 7-5-LongURL-FifthLevel

    cantMF =0#mediafire.com
    cantNC =0#nolocreo.com
    cantMG =0#mega.nz
    cantP =0#paste2.org
    cantE =0#elfqrin.com
    cantA =0#aldeaviral.com
    cantV =0#viralymedio.com

with open(csv_filename,'r')as fCsv:
for row in csv.DictReader(fCsv):
            webAddress = row['url']
            ext = tldextract.extract(webAddress)
            shortDomainUrl =''
if(ext.subdomain):
                shortDomainUrl = ext.subdomain +"."+ ext.domain +"."+ ext.suffix
else:
                shortDomainUrl = ext.domain +"."+ ext.suffix

if('mediafire.com'in shortDomainUrl):
                cantMF = cantMF +1
elif('nolocreo.com'in shortDomainUrl):
                cantNC = cantNC +1
elif('mega.nz'in shortDomainUrl):
                cantMG = cantMG +1
elif('paste2.org'in shortDomainUrl):
                cantP = cantP +1
elif('elfqrin.com'in shortDomainUrl):
                cantE = cantE +1
elif('aldeaviral.com'in shortDomainUrl):
                cantA = cantA +1
elif('viralymedio.com'in shortDomainUrl):
                cantV = cantV +1

print('Proceso terminado')

print'mediafire.com', cantMF
print'nolocreo.com',cantNC
print'mega.nz',cantMG
print'paste2.org',cantP
print'elfqrin.com',cantE
print'aldeaviral.com',cantA
print'viralymedio.com',cantV
```

Anexo F: Parte 1: Código para obtener las características de las URLs.

```
import sys
import requests
import urllib2
import cookielib
import tldextract
import array
import csv
from csv import DictReader
import pythonwhois
from bs4 import BeautifulSoup
import datetime

headers ={'User-Agent':'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_5) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/50.0.2661.102 Safari/537.36'}

defget_urls_from_csv(csv_filename):
with open(csv_filename,'r')as csvF:
for row in DictReader(csvF):
yield row['urls']

defgetDomainsCookies(urlToQuery, domainUrl, allForeignCookies, allCookies, allOwnCookies):
    qCookies =0    ownCookies =0
try:
        cj = cookielib.CookieJar()
        opener = urllib2.build_opener(urllib2.HTTPCookieProcessor(cj), urllib2.HTTPHandler())
        result = opener.open(urlToQuery).read()
for cookie in cj:
            stringCookie = str(cookie)            pos = stringCookie.index('for')+4
            stringCookie = stringCookie[pos:]     stringCookie = stringCookie[:-2]
            qCookies = qCookies +1
if(domainUrl in stringCookie):
                ownCookies = ownCookies +1
else:
                allForeignCookies.append(stringCookie)
except:
pass
    allCookies.append(qCookies)    allOwnCookies.append(ownCookies)

defgetWhoIsDetails(domainToQueryWhoIs, allWhoIsDetails):
try:
        details = pythonwhois.get_whois(domainToQueryWhoIs)
try:
            allWhoIsDetails.append(details['contacts']['registrant']['country'])
except:
            allWhoIsDetails.append('')
try:
            allWhoIsDetails.append(details['contacts']['registrant']['email'])
except:
            allWhoIsDetails.append('')
try:
            allWhoIsDetails.append(details['contacts']['registrant']['phone'])
except:
            allWhoIsDetails.append('')
try:
            expirationDate =(details['expiration_date'][0])
            creationDate =(details['creation_date'][0])
            updateDate =(details['updated_date'][0])
#Dias entre: expiracion menos creacion
            allWhoIsDetails.append((expirationDate - creationDate).days)
#Dias entre: actualizacion menos creacion
allWhoIsDetails.append((updateDate - creationDate).days)
#Dias entre: expiracion menos actualizacion
allWhoIsDetails.append((expirationDate - updateDate).days)
except:
print('error')
for i in range(6):
                allWhoIsDetails.append('')
except:
for i in range(9):
allWhoIsDetails.append('')
```

Anexo F: Parte 2: Código para obtener las características de las URLs.

```
defgetDescriptionOfURL(soup):
    desc = soup.find(attrs={'name':'Description'})
if desc ==None:
        desc = soup.find(attrs={'name':'description'})
try:
return(desc['content']).encode('utf-8').strip()
except:
return'0'

defgetHTTPequivOfURL(soup):
    refer = soup.find(attrs={'http-equiv':'refresh'})
try:
        urlToRefer = refer['content']
        positionOfEqual = urlToRefer.find('=')+1
return urlToRefer[positionOfEqual:]
except:
return'0'

defgetTitlevOfURL(soup):
    titleTag = soup.find('title')
if(titleTag):
try:
return(titleTag.text.encode('utf-8').strip())
except:
return'0'
else:
return'0'

defgetRedirectionOfURL(soup, locationRedirect):
for scriptTag in soup.find_all('script'):
try:
            completeTag = str(scriptTag)
if('location='in completeTag or'location ='in completeTag
               or'location.href'in completeTag or'location.replace'in completeTag
               or'location.assign'in completeTag):
                 positionStart = completeTag.find('http')
                 positionEnd = completeTag.find("\"", positionStart)
                 URLredirect = completeTag[positionStart:-(len(completeTag)-
                                 positionEnd +1)]
                 locationRedirect.append(URLredirect)
break
except:
pass

defgetFormActionOfURL(soup, domainUrl, frmActionNotEqualToDomain, totalFrmAction,
totalEqDomainFrmAction):
    tFrm =0
    tFrmOfDomain =0
for scriptTag in soup.find_all('form'):
try:
if('http'in scriptTag['action']):
                 tFrm = tFrm +1
if(domainUrl in scriptTag['action']):
                     tFrmOfDomain = tFrmOfDomain +1
else:
                     frmActionNotEqualToDomain.append(scriptTag['action'])
except:
pass
    totalFrmAction.append(tFrm)
totalEqDomainFrmAction.append(tFrmOfDomain)
```

Anexo F: Parte 3: Código para obtener las características de las URLs.

```
defgetScriptSourceOfURL(soup, domainUrl, scriptSrcNotEqualToDomain, totalscriptSrc,
totalEqDomainscriptSrc):
    tScr =0    tScrOfDomain =0
for scriptTag in soup.find_all('script'):
try:
            completeTag = str(scriptTag)
if('src'in completeTag and'http'in scriptTag['src']):
                tScr = tScr +1
if(domainUrl in scriptTag['src']):
                    tScrOfDomain = tScrOfDomain +1
else:
                    scriptSrcNotEqualToDomain.append(scriptTag['src'])
except:
pass
    totalscriptSrc.append(tScr)
    totalEqDomainscriptSrc.append(tScrOfDomain)

defgetReferencesAToLinks(soup, domainUrl, allLinks, total, totalOfDomain):
    t =0    tOfDomain =0
for link in soup.find_all('a'):
try:
            urlHREF = link.get('href')
if(urlHREF and'http'in urlHREF):
                t = t +1
if(domainUrl in urlHREF):
                    tOfDomain = tOfDomain +1
else:
                    allLinks.append(urlHREF)
except:
pass
    total.append(t)
    totalOfDomain.append(tOfDomain)

defgetImageSources(soup, domainUrl, srcNotEqualToDomain, totalImgSrc, totalEqDomainImgSrc):
    tImg =0
    tImgOfDomain =0
for imgTag in soup.find_all('img'):
try:
if('http'in imgTag['src']):
                tImg = tImg +1
if(domainUrl in imgTag['src']):
                    tImgOfDomain = tImgOfDomain +1
else:
                    srcNotEqualToDomain.append(imgTag['src'])
except:
pass
    totalImgSrc.append(tImg)
    totalEqDomainImgSrc.append(tImgOfDomain)

defgetContentOfIframe(children, tagName, listGenericTag):
for genericTag in children.find_all(tagName):
if(tagName =='script'):
try:
                completeTag = str(scriptTag)
if('src'in completeTag and'http'in scriptTag['src']):
                    listGenericTag.append(scriptTag['src'])
except:
pass
elif(tagName =='a'):
try:
                urlHREF = genericTag.get('href')
if(urlHREF and'http'in urlHREF):
                    listGenericTag.append(scriptTag.get('href'))
except:
pass
elif(tagName =='img'):
try:
if('http'in genericTag['src']):
                    listGenericTag.append(genericTag['src'])
except:
pass
```

Anexo F: Parte 4: Código para obtener las características de las URLs.

```
defgetNumberIFramesSocialNetworks(iFrameSRC, iFrameOfFacebook, iFrameOfOtherSocialNetworks,
iFrameOfOtherSites):
    countFb = iFrameOfFacebook[0]
    countOtherSN = iFrameOfOtherSocialNetworks[0]
    countOtherSites = iFrameOfOtherSites[0]
if'facebook.com'in iFrameSRC:
        countFb = countFb +1
elif('youtube.com'in iFrameSRC or'google.com'in iFrameSRC or'player.vimeo'in iFrameSRC):
        countOtherSN = countOtherSN +1
else:
        countOtherSites = countOtherSites +1
    iFrameOfFacebook[0]= countFb
    iFrameOfOtherSocialNetworks[0]= countOtherSN
    iFrameOfOtherSites[0]= countOtherSites

defscrapWebPage(csv_filename):
    fileToSaveScrappedData ='QueryResults0002.csv'
with open(fileToSaveScrappedData,'wb')as f:
        w = csv.writer(f)
        w.writerow(["url","domain","Title","Descripcion","Http_Equivalent_Refresh",
                "Script_Location_Redirect","Total_Cookies_Quantity",
                   "Own_Cookies_Quantity","Country","Email","Phone",
"Days_Exp_Creation","Days_Update_Creation","Days_Exp_Update",
"Days_Today_Creation","Days_Today_Update","Days_Exp_Today",
"Total_Form_Action","Own_Form_Action",
"Total_Script_Source","Own_Script_Source"
"Total_A_References","Own_A_References",
"Total_Img_Sources","Own_Img_Sources",
"Quantity_iFrames","Quantity_iFrames_Facebook",
                "Quantity_iFrames_OSN","Quantity_iFrames_Other_Sites",
"Total_Script_Src_Iframe","Own_Script_Src_IFrame",
"Total_A_Ref_IFrame","Own_A_Ref_IFrame",
"Total_Img_SrcIFrame","Own_Img_Src_IFrame"])
for url in get_urls_from_csv(csv_filename):
print('Url:')
print(url)
try:
                requests.get(url, headers = headers)
except:
print('urlError')
continue

            ext = tldextract.extract(url)
            shortDomainUrl =''
if(ext.subdomain):
                shortDomainUrl = ext.subdomain +"."+ ext.domain +"."+ ext.suffix
else:
                shortDomainUrl = ext.domain +"."+ ext.suffix

#Obtener cookies del dominio
            allForeignCookies =['']
            allCookies = array.array('i')
            allOwnCookies = array.array('i')
            getDomainsCookies(url, shortDomainUrl, allForeignCookies, allCookies,
                         allOwnCookies)

#Obtener el whoIs del dominio
            domainToQueryWhoIs = ext.domain +"."+ ext.suffix
            allWhoIsDetails =['']
            getWhoIsDetails(domainToQueryWhoIs, allWhoIsDetails)
print('domain to query whois:')
print(domainToQueryWhoIs)

            data =''
try:
                r  = requests.get(url, headers = headers)
data =r.text
except:
continue
```

Anexo F: Parte 5: Código para obtener las características de las URLs.

```
#obtener los datos mediante beautifulsoup
soup=BeautifulSoup(data,"html5lib")

#Recopilacion de contenido descriptivo de la url
#Redireccionwindow.location de la url
redirectionPage =['']

#url in form action tag
            frmActionNotEqualToDomain =['']
            totalFrmAction = array.array('i')
            totalEqDomainFrmAction = array.array('i')
            getFormActionOfURL(soup, shortDomainUrl, frmActionNotEqualToDomain, totalFrmAction,
            totalEqDomainFrmAction)

#Source de los scripts src de la URL
scriptSrcNotEqualToDomain =['']
            totalscriptSrc = array.array('i')
            totalEqDomainscriptSrc = array.array('i')
            getScriptSourceOfURL(soup, shortDomainUrl, scriptSrcNotEqualToDomain,
totalscriptSrc, totalEqDomainscriptSrc)

#Recopilacion de referencias a:
#print('A links for url:')
linksNotEqualToDomain=['']
            totalHref = array.array('i')
            totalEqDomainHref = array.array('i')
            getReferencesAToLinks(soup, shortDomainUrl, linksNotEqualToDomain, totalHref,
            totalEqDomainHref)

#Recopilacion de img src:
#print('Img source for url:')
            srcNotEqualToDomain =['']
            totalImgSrc = array.array('i')
            totalEqDomainImgSrc = array.array('i')
            getImageSources(soup, shortDomainUrl, srcNotEqualToDomain, totalImgSrc,
            totalEqDomainImgSrc)

            countIframes =0
            allScriptSRCinIFrames =['']
            allHrefinIFrames =['']
            allImgSRCinIFrames =['']

            iFrameOfFacebook = array.array('i')
            iFrameOfFacebook.append(0)
            iFrameOfOtherSocial = array.array('i')
            iFrameOfOtherSocial.append(0)
            iFrameOfOtherSites = array.array('i')
            iFrameOfOtherSites.append(0)

#Inicializacion de scripts src de iframe
            iFrameScriptSrcNotEqualToDomain =['']
            iFrameTotalscriptSrc = array.array('i')
iFrameTotalEqDomainscriptSrc=array.array('i')

#Inicializacion de href de iframe:
iFramelinksNotEqualToDomain =['']
            iFrametotalHref = array.array('i')
iFrametotalEqDomainHref=array.array('i')

#Inicializacion de imgsrc de iframe:
iFramesrcNotEqualToDomain =['']
            iFrametotalImgSrc = array.array('i')
iFrametotalEqDomainImgSrc=array.array('i')
```

Anexo F: Parte 6: Código para obtener las características de las URLs.

```
            iframexx = soup.find_all('iframe')
for iframe in iframexx:
#print(iframe)
                countIframes = countIframes +1
ifnot('src'in iframe):
continue
                iFrameSRC = iframe.attrs['src']
ifnot(iFrameSRC):
continue

if(shortDomainUrl notin iFrameSRC):
                    getNumberIFramesSocialNetworks(iFrameSRC, iFrameOfFacebook,
iFrameOfOtherSocial, iFrameOfOtherSites)
if(iFrameOfFacebook[0]==0and iFrameOfOtherSocial[0]==0and iFrameOfOtherSites[0]):
                        extIFrame = tldextract.extract(iFrameSRC)
                        shortDomainUrliFrame =''
if(extIFrame.subdomain):
                            shortDomainUrliFrame = extIFrame.subdomain +"."+ extIFrame.domain
+"."+ extIFrame.suffix
else:
                            shortDomainUrliFrame = extIFrame.domain +"."+ extIFrame.suffix

                        response  = requests.get(iFrameSRC, headers = headers)
                        data = response.text
                        children = BeautifulSoup(data,"html5lib")

                        getScriptSourceOfURL(children, shortDomainUrliFrame,
iFrameScriptSrcNotEqualToDomain, iFrameTotalscriptSrc, iFrameTotalEqDomainscriptSrc)

                        getReferencesAToLinks(children, shortDomainUrliFrame,
iFramelinksNotEqualToDomain, iFrametotalHref, iFrametotalEqDomainHref)

                        getImageSources(children, shortDomainUrliFrame,
iFramesrcNotEqualToDomain, iFrametotalImgSrc, iFrametotalEqDomainImgSrc)

if(len(iFrameTotalscriptSrc)==0):
                iFrameTotalscriptSrc.append(0)
if(len(iFrameTotalEqDomainscriptSrc)==0):
                iFrameTotalEqDomainscriptSrc.append(0)
if(len(iFrametotalHref)==0):
                iFrametotalHref.append(0)
if(len(iFrametotalEqDomainHref)==0):
                iFrametotalEqDomainHref.append(0)
if(len(iFrametotalImgSrc)==0):
                iFrametotalImgSrc.append(0)
if(len(iFrametotalEqDomainImgSrc)==0):
                iFrametotalEqDomainImgSrc.append(0)

            w.writerow([url, shortDomainUrl, getTitlevOfURL(soup), getDescriptionOfURL(soup),
                        getHTTPequivOfURL(soup), getRedirectionOfURL(soup, redirectionPage),
                        allCookies[0], allOwnCookies[0],
                        allWhoIsDetails[1], allWhoIsDetails[2], allWhoIsDetails[3],
                        allWhoIsDetails[4], allWhoIsDetails[5], allWhoIsDetails[6],
                        allWhoIsDetails[7], allWhoIsDetails[8], allWhoIsDetails[9],
                        totalFrmAction[0], totalEqDomainFrmAction[0],
                        totalscriptSrc[0], totalEqDomainscriptSrc[0],
                        totalHref[0], totalEqDomainHref[0],
                        totalImgSrc[0], totalEqDomainImgSrc[0],
                        countIframes,
                        iFrameOfFacebook[0], iFrameOfOtherSocial[0], iFrameOfOtherSites[0],
                        iFrameTotalscriptSrc[0], iFrameTotalEqDomainscriptSrc[0],
                        iFrametotalHref[0], iFrametotalEqDomainHref[0],
                        iFrametotalImgSrc[0], iFrametotalEqDomainImgSrc[0]])
print('URL finalizada')
```

Anexo G: Generación y mostrado de gráficos a partir de los datos.

```
import pandas as pd
import numpy as np
from sklearn import svm

import matplotlib.pyplot as plt
import seaborn as sns; sns.set(font_scale=1.2)

%matplotlib inline

url_features = pd.read_csv('Final_With_Types.csv')

sns.lmplot('Days_Exp_Creation_Balanced','Foreign_Cookies',
       data=url_features, hue='Type', palette='Set1',
              fit_reg=False, scatter_kws={"s":70});
```

Anexo H: Código para obtener las características más relevantes del modelo lineal SVM.

```
import pandas as pd
import numpy as np
from sklearn import svm

import matplotlib.pyplot as plt

#Método para mostrar el gráfico de los coeficientes
defplot_coefficients(classifier, feature_names):
    top_features = len(feature_names)
    coef = classifier.coef_.ravel()
    top_coefficients = np.hstack([np.argsort(coef)])

    plt.figure(figsize=(15,5))
    colors =['red'if c <0else'blue'for c in coef[top_coefficients]]
    plt.ylim(ymax = max(coef), ymin =  min(coef))
    plt.bar(np.arange(top_features), coef[top_coefficients], color=colors)

    feature_names = np.array(feature_names)
    plt.xticks(np.arange(0, top_features), feature_names[top_coefficients],
    rotation=60, ha='right')
    plt.show()

#Método para obtener las características más informativas a partir del
#valor absoluto de los coeficientes
defshow_most_informative_features(feature_names, clf, n=20):
    coefs_with_fns = sorted(zip(abs(clf.coef_[0]), feature_names))
    top = zip(coefs_with_fns[:n], coefs_with_fns[:-(n +1):-1])
for(coef_1, fn_1),(coef_2, fn_2)in top:
print"%.6f\t%-15s"%(coef_2, fn_2)


data =  pd.read_csv('Final_With_Types.csv')

#instancia de todas las características
features_names
=['Foreign_Cookies','Days_Exp_Creation_Balanced','Days_Update_Creation_Balan
ced','Days_Exp_Update_Balanced',
'Foreign_FormAction','Foreign_Script_Source','Foreign_A_References','Foreign
_Img_Sources','Quantity_iFrames']

multi_url_features = data[features_names].as_matrix()
type_label = np.where(data['Type']=='MALICIOUS',1,0)

model = svm.SVC(kernel='linear', C=1000)
model.fit(multi_url_features, type_label)

plot_coefficients(model, features_names)
show_most_informative_features(features_names, model)
```

Anexo I: Código del modelo de clasificación SVM de URLs maliciosas.

```
import pandas as pd
import numpy as np
fromsklearnimportsvm

#Carga de los datos
url_features = pd.read_csv('Final_With_Types.csv')

#features_names: Características
features_names
=['Foreign_Cookies','Days_Exp_Creation_Balanced','Days_Update_Creation_Balan
ced','Days_Exp_Update_Balanced',
'Foreign_FormAction','Foreign_Script_Source','Quantity_iFrames']

#features_values: Contenido de valores de las caracteristicas
features_values = url_features[features_names].as_matrix()

#type_label: Contenido de la etiqueta de datos
type_label = np.where(url_features['Type']=='MALICIOUS',1,0)

# Instancia del modelo de clasificación SVM
model = svm.SVC(kernel='linear', C=2**5)
model.fit(features_values, type_label)

# Método para clasificar si una URL es maliciosa o no
defmalicious_or_not(Foreign_Cookies, Days_Exp_Creation_Balanced,
                    Days_Update_Creation_Balanced,
                     Days_Exp_Update_Balanced, Foreign_FormAction,
                    Foreign_Script_Source, Quantity_iFrames):

if(model.predict([[Foreign_Cookies, Days_Exp_Creation_Balanced,
                      Days_Update_Creation_Balanced,
                      Days_Exp_Update_Balanced, Foreign_FormAction,
                    Foreign_Script_Source, Quantity_iFrames]]))==0:
print('URL accesible')
else:
print('Esta URL posee contenido con características maliciosas')

# Llamada al método
malicious_or_not(100,2.002739726,1,1.04109598,0,0,9)
malicious_or_not(50,11.00821918,0,6.37260274,0,17.39130435,7)
malicious_or_not(50,7.005479452,0,1.082191781,33.33333333,4.651162791,6)
malicious_or_not(50,1,0,1.016438356,500,500,12)
```

Anexo J: Código de validación cruzada para calcular el nivel de precisión del modelo.

```
from sklearn.model_selection import LeaveOneOut
from sklearn import datasets
from sklearn import svm
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score
import pandas as pd

#load data
allURLsData = pd.read_csv('Final_With_Types.csv')

features_names
=['Days_Exp_Creation_Balanced','Days_Update_Creation_Balanced','Days_Exp_Upd
ate_Balanced',
'Foreign_FormAction','Foreign_Script_Source','Foreign_A_References','Foreign
_Img_Sources']

X = allURLsData[features_names].as_matrix()
Y = allURLsData['Type'].as_matrix()

#build the model
svm_model = svm.SVC(kernel='linear', C=2**-5, gamma ='auto',random_state =0)

#create the Cross validation object
loo = LeaveOneOut()

#calculate cross validated (leave one out) accuracy score
scores = cross_val_score(svm_model, X,Y, cv = loo, scoring='accuracy')

print(scores.mean())
```

Anexo K: Código de index.php.

```
<htmlclass="no-js"lang="">
	<head>
<metacharset="utf-8">
<metahttp-equiv="X-UA-Compatible"content="IE=edge,chrome=1">
<title></title>
<metaname="description"content="">
<linkrel="stylesheet"href="css/bootstrap.min.css">
<linkrel="stylesheet"href="css/bootstrap-theme.min.css">
<linkrel="stylesheet"href="css/main.css">
</head>
<body>
<divclass="container-fluid">
<h2></h2>
<divid="main">
<divclass="row">
<divclass="col-md-6 col-md-offset-3">
<divclass="panel panel-info panel-blue">
<divclass="panel-body">
				<form
role="form"action=""method="post"enctype="multipart/form-data">
				<divclass="form-group">
					<h1>Inicializar el modelo</h1>
					<divclass="text-center">

	<inputtype="submit"value="Inicializar"class="btn btn-lg btn-info">
					</div>
					<divclass="text-center">
						<label>Cuando el modelo esté
cargado en memoria. Se redigirá la página.</label>
					</div>
				</div>
				</form>
		<divclass="col-md-12">
		<?php
		if(isset($_POST)&&$_SERVER['REQUEST_METHOD']=="POST"){
	$py=shell_exec('C:/ProgramData/Anaconda3/envs/Python2/python
D:/SoftwareDevelopment/PracticePHP/TFM/initializeSVMmodel.py  2>&1');
			if(!(empty($py))){
				if($py=='ERROR.'){
				echo"ERROR. Intentar nuevamente.";
			}
			else{
				header("Location: formConsultURL.php");
		}
			}
		}
		?>
</div>
</div>
</div>
</div>
</div>
</div>
</div>
<scriptsrc="js/vendor/bootstrap.min.js"></script>
<scriptsrc="js/main.js"></script>
</body>
</html>
```

Anexo L: Código de initializeSVMmodel.py.

```
import codecs
import sys
import pandas as pd
import numpy as np
from sklearn import svm
from sklearn.externals import joblib

defmain():
	try:
		#Carga de los datos
		url_features = pd.read_csv('Final_With_Types.csv')

		#features_names: Caracteristicas
		features_names
=['Foreign_Cookies','Days_Exp_Creation_Balanced','Days_Update_Creation_Balan
ced','Days_Exp_Update_Balanced',
'Foreign_FormAction','Foreign_Script_Source','Quantity_iFrames']

		#features_values: Contenido de valores de las caracteristicas
		features_values = url_features[features_names].as_matrix()

		#type_label: Contenido de la etiqueta de datos
		type_label = np.where(url_features['Type']=='MALICIOUS',1,0)

		# Instancia del modelo de clasificacion SVM
		model = svm.SVC(kernel='linear', C=2**5)
		model.fit(features_values, type_label)
		joblib.dump(model,'uploadSVMmodel.pkl', compress=1)
		print("Modelo inicializado")
	except:
		print"ERROR."

if __name__ =='__main__':
    sys.stdout = codecs.getwriter('utf-8')(sys.stdout)
main()
```

Anexo M: Parte 1: Código de formConsultURL.php.

```
<htmlclass="no-js"lang="">
<?php
if(isset($_POST)&&$_SERVER['REQUEST_METHOD']=="POST"){
	if(isset($_POST["url-entrada"])&&isset($_POST["urlType"])){
		$url_input=$_POST["url-entrada"];
		$isShortOrLong=$_POST["urlType"];
		$isShort=1;
		if($isShortOrLong=='isLong'){
			$isShort=0;
		}
		if(!(empty($url_input))){

	$py=shell_exec('C:/ProgramData/Anaconda3/envs/Python2/python
D:/SoftwareDevelopment/PracticePHP/TFM/classify_URL_SVM_model.py
'.$url_input.' '.$isShort.' 2>&1');
		}
	}
	else
		$py='Ingrese la URL y el tipo';
}
?>
	<head>
<metacharset="utf-8">
<metahttp-equiv="X-UA-Compatible"content="IE=edge,chrome=1">
<title></title>
<metaname="description"content="">
<linkrel="stylesheet"href="css/bootstrap.min.css">
<linkrel="stylesheet"href="css/bootstrap-theme.min.css">
<linkrel="stylesheet"href="css/main.css">
</head>
<body>
<divclass="container-fluid">
<h1>Consulta de URL</h1>
<divid="main">
<divclass="row">
<divclass="col-md-6 col-md-offset-3">
<divclass="panel panel-info panel-blue">
<divclass="panel-body">
				<form
role="form"action=""method="post"enctype="multipart/form-data">
				<divclass="form-group">
					<label>URL</label>
					<inputtype="text"class="form-
control"name="url-entrada"placeholder="Ingrese la url">
				</div>
				<divclass="form-group">
					<label>¿Es una URL corta?</label>
					<p/>
					<inputtype = 'Radio'Name
='urlType'value= 'isShort'>Sí
					<inputtype = 'Radio'Name
='urlType'value= 'isLong'>No
				</div>
```

Anexo M: Parte 2: Código de formConsultURL.php.

```
        <divclass="form-group">
                <divclass="col-sm-offset-5 col-sm-10">
                        <inputtype="submit"value="Procesar"class="btn btn-lg
btn-info">
                </div>
        </div>
</form>
        <divclass="form-group"id="divResult">
                <divclass="row">
                        <divclass="form-group col-md-offset-1">
                                <label>

        <?php

        if(isset($_POST)&&$_SERVER['REQUEST_METHOD']=="POST"){
                echo"<label>Resultado:</label>";
        echo"<p/>";
        echo$py;                                                        }
        ?>
                                </label>
                        </div>
                </div>
        </div>
</div>
</div>
</div>
</div>
</div>
</div>
<scriptsrc="js/vendor/bootstrap.min.js"></script>
<scriptsrc="js/main.js"></script>
</body>
</html>
```

Anexo N: Parte1: Código de classify_URL_SVM_model.py.

```
from __future__ import division
from bs4 import BeautifulSoup
import sys
import requests
import codecs
import json
import sys
import logging
from sklearn.externals import joblib
import urllib2
import cookielib
import tldextract
import array
import pythonwhois
import datetime


defunshort(url):
if url.startswith('https'):
        url = url[8:]
if url.startswith('http'):
        url = url[7:]
    query_url ='https://unshorten.me/json/'+ format(url)
    r = requests.get(query_url)
    output_json = json.loads(r.text)
    end_url = output_json['resolved_url']
if end_url.startswith('http')or end_url.startswith('https'):
return output_json['resolved_url']

defgetDomainsCookies(urlToQuery, domainUrl, allForeignCookies, allCookies,
allOwnCookies):
    qCookies =0
    ownCookies =0
try:
        cj = cookielib.CookieJar()
        opener = urllib2.build_opener(urllib2.HTTPCookieProcessor(cj),
urllib2.HTTPHandler())
        result = opener.open(urlToQuery).read()
for cookie in cj:
            stringCookie = str(cookie)
            pos = stringCookie.index('for')+4
            stringCookie = stringCookie[pos:]
            stringCookie = stringCookie[:-2]
            qCookies = qCookies +1
if(domainUrl in stringCookie):
                ownCookies = ownCookies +1
else:
                allForeignCookies.append(stringCookie)
except:
pass
    allCookies.append(qCookies)
allOwnCookies.append(ownCookies)
```

Anexo N: Parte2: Código de classify_URL_SVM_model.py.

```
defgetWhoIsDetails(domainToQueryWhoIs, allWhoIsDetails):
try:
        details = pythonwhois.get_whois(domainToQueryWhoIs)
try:
            expirationDate =(details['expiration_date'][0])
            creationDate =(details['creation_date'][0])
            updateDate =(details['updated_date'][0])
#Dias entre: expiracion menos creacion
            allWhoIsDetails.append((expirationDate - creationDate).days)
#Dias entre: actualizacion menos creacion
allWhoIsDetails.append((updateDate - creationDate).days)
#Dias entre: expiracion menos actualizacion
allWhoIsDetails.append((expirationDate - updateDate).days)
except:
print('error')
for i in range(3):
                allWhoIsDetails.append(0)
except:
for i in range(3):
            allWhoIsDetails.append(0)

defgetFormActionOfURL(soup, domainUrl, frmActionNotEqualToDomain,
totalFrmAction, totalEqDomainFrmAction):
    tFrm =0
    tFrmOfDomain =0
for scriptTag in soup.find_all('form'):
try:
if('http'in scriptTag['action']):
                tFrm = tFrm +1
if(domainUrl in scriptTag['action']):
                    tFrmOfDomain = tFrmOfDomain +1
else:
                    frmActionNotEqualToDomain.append(scriptTag['action'])
except:
pass
    totalFrmAction.append(tFrm)
    totalEqDomainFrmAction.append(tFrmOfDomain)

defgetScriptSourceOfURL(soup, domainUrl, scriptSrcNotEqualToDomain,
totalscriptSrc, totalEqDomainscriptSrc):
    tScr =0
    tScrOfDomain =0
for scriptTag in soup.find_all('script'):
try:
            completeTag = str(scriptTag)
if('src'in completeTag and'http'in scriptTag['src']):
                tScr = tScr +1
if(domainUrl in scriptTag['src']):
                    tScrOfDomain = tScrOfDomain +1
else:
                    scriptSrcNotEqualToDomain.append(scriptTag['src'])
except:
pass
    totalscriptSrc.append(tScr)
totalEqDomainscriptSrc.append(tScrOfDomain)
```

Anexo N: Parte3: Código de classify_URL_SVM_model.py.

```
defgetReferencesAToLinks(soup, domainUrl, allLinks, total, totalOfDomain):
    t =0
    tOfDomain =0
for link in soup.find_all('a'):
try:
            urlHREF = link.get('href')
if(urlHREF and'http'in urlHREF):
                t = t +1
if(domainUrl in urlHREF):
                    tOfDomain = tOfDomain +1
else:
                    allLinks.append(urlHREF)
except:
pass
    total.append(t)
    totalOfDomain.append(tOfDomain)

defgetImageSources(soup, domainUrl, srcNotEqualToDomain, totalImgSrc,
totalEqDomainImgSrc):
    tImg =0
    tImgOfDomain =0
for imgTag in soup.find_all('img'):
try:
if('http'in imgTag['src']):
                tImg = tImg +1
if(domainUrl in imgTag['src']):
                    tImgOfDomain = tImgOfDomain +1
else:
                    srcNotEqualToDomain.append(imgTag['src'])
except:
pass
    totalImgSrc.append(tImg)
    totalEqDomainImgSrc.append(tImgOfDomain)


defscrapWebPage(svm_model, url, isShort):
    headers ={'User-Agent':'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_5)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/50.0.2661.102 Safari/537.36'}
if(isShort ==1):
        url = unshort(url)
try:
        requests.get(url, headers = headers)
except:
print('urlError')

    ext = tldextract.extract(url)
    shortDomainUrl =''
if(ext.subdomain):
        shortDomainUrl = ext.subdomain +"."+ ext.domain +"."+ ext.suffix
else:
        shortDomainUrl = ext.domain +"."+ ext.suffix
    allForeignCookies =['']
    allCookies = array.array('i')
    allOwnCookies = array.array('i')
    getDomainsCookies(url, shortDomainUrl, allForeignCookies, allCookies,
allOwnCookies)
```

Anexo N: Parte4: Código de classify_URL_SVM_model.py.

```
#Obtener el whoIs del dominio
    domainToQueryWhoIs = ext.domain +"."+ ext.suffix
    allWhoIsDetails = array.array('i')
    getWhoIsDetails(domainToQueryWhoIs, allWhoIsDetails)

    data =''
try:
        r  = requests.get(url, headers = headers)
        data = r.text
except:
print('urlError')

#obtener los datos mediante beautifulsoup
soup = BeautifulSoup(data,"html5lib")

#url in form action tag
    frmActionNotEqualToDomain =['']
    totalFrmAction = array.array('i')
    totalEqDomainFrmAction = array.array('i')
    getFormActionOfURL(soup, shortDomainUrl, frmActionNotEqualToDomain,
totalFrmAction, totalEqDomainFrmAction)

#Source de los scripts src de la URL
scriptSrcNotEqualToDomain =['']
    totalscriptSrc = array.array('i')
    totalEqDomainscriptSrc = array.array('i')
    getScriptSourceOfURL(soup, shortDomainUrl, scriptSrcNotEqualToDomain,
totalscriptSrc, totalEqDomainscriptSrc)

#Recopilacion de referencias a:
linksNotEqualToDomain=['']
totalHref = array.array('i')
    totalEqDomainHref = array.array('i')
    getReferencesAToLinks(soup, shortDomainUrl, linksNotEqualToDomain,
totalHref, totalEqDomainHref)

#Recopilacion de imgsrc:
srcNotEqualToDomain=['']
totalImgSrc = array.array('i')
    totalEqDomainImgSrc = array.array('i')
    getImageSources(soup, shortDomainUrl, srcNotEqualToDomain, totalImgSrc,
totalEqDomainImgSrc)

    countIframes =0
    iframexx = soup.find_all('iframe')
for iframe in iframexx:
        countIframes = countIframes +1

    #Valor de cookies
    lastValueCookies =0
if(allCookies[0]!=0):
        lastValueCookies =(1-(allOwnCookies[0]/allCookies[0]))*100

#Valor de Days_Exp_Creation
    lastValueDays_Exp_Creation =0
if(allWhoIsDetails[0]!=0):
lastValueDays_Exp_Creation=allWhoIsDetails[0]/365
```

Anexo N: Parte5: Código de classify_URL_SVM_model.py.

```
#Valor de Days_Update_Creation
    lastValueDays_Update_Creation =0
if(allWhoIsDetails[1]!=0):
        lastValueDays_Update_Creation = allWhoIsDetails[1]/365
#Valor de Days_Exp_Update
    lastValueDays_Exp_Update =0
if(allWhoIsDetails[2]!=0):
        lastValueDays_Exp_Update = allWhoIsDetails[2]/365
#Valor de FormAction
    lastValueFormAction =0
if(totalFrmAction[0]!=0):
        foreignFormAction = totalFrmAction[0]- totalEqDomainFrmAction[0]
if(foreignFormAction <2):
            lastValueFormAction =(1-
(totalEqDomainFrmAction[0]/totalFrmAction[0]))*100
elif(foreignFormAction>=2and foreignFormAction<5):
            lastValueFormAction =(1-
(totalEqDomainFrmAction[0]/totalFrmAction[0]))*500
else:
            lastValueFormAction =(1-
(totalEqDomainFrmAction[0]/totalFrmAction[0]))*1000
#Valor de ScriptSource
    lastValueScriptSource =0
if(totalscriptSrc[0]!=0):
        foreignScriptSrc = totalscriptSrc[0]- totalEqDomainscriptSrc[0]
if(foreignScriptSrc <5):
            lastValueScriptSource =(1-
(totalEqDomainscriptSrc[0]/totalscriptSrc[0]))*100
elif(foreignScriptSrc>=5and foreignScriptSrc<25):
            lastValueScriptSource =(1-
(totalEqDomainscriptSrc[0]/totalscriptSrc[0]))*500
else:
            lastValueScriptSource =(1-
(totalEqDomainscriptSrc[0]/totalscriptSrc[0]))*1000

       if(svm_model.predict([[lastValueCookies, lastValueDays_Exp_Creation,
lastValueDays_Update_Creation, lastValueDays_Exp_Update,
lastValueFormAction, lastValueScriptSource, countIframes]]))==0:
             print('URL accesible')
       else:
             print('Esta URL posee contenido con caracteristicas
maliciosas')

defmain():
try:
             url = sys.argv[1]
             isShort = sys.argv[2]
             svm_model = joblib.load('uploadSVMmodel.pkl')
             scrapWebPage(svm_model, url, isShort)
exceptIndexError:
print("Ingresar una URL")

if __name__ =='__main__':
    sys.stdout = codecs.getwriter('utf-8')(sys.stdout)
main()
```

7. Bibliografía y Referencias

[1] Faghani, M.R. et. al., A Study of Malware Propagation via Online Social Networking, Mining Social Networks and Security Informatics, Lecture Notes in Social Networks, 2013, DOI: 10.1007/978-94-007-6359-3_13

[2] Prateek Dewan, Detecting Malicious Content on Facebook, Social and Information Networks, 2015, url ={http://arxiv.org/abs/1501.00802}

[3] Tao Stein, Facebook Immune System, Proceedings of the 4th Workshop on Social Network Systems, USA, 2012, ISBN: 978-1-4503-0728-4

[4] Observatorio de Delitos Informáticos de Latinoamérica
https://www.odila.org/pdf/Informe_ODILA_2016.pdf
(Consultado el 15/03/2017)

[5] Symantec Informe sobre Seguridad
https://infosertec.com.ar/2017/06/02/symantec-realizo-un-completo-informe-sobre-seguridad-y-tambien-hablo-del-ransomware/
(Consultado el 15/03/2017)

[6] Bhagyashri Toke, "*Spam Detection in Online Social Networks using Integrated Approach*" International Journal of Innovative Research in Computer and Communication Engineering Vol. 4, 2016 (Consultado el 20/02/2017)

[7] LatinAmericaLoves Facebook
https://www.emarketer.com/Article/Latin-America-Loves-Facebook/1013651
(Consultado el 02/02/2017)

[8] Evento falso de Facebook - ScamAlert
http://facecrooks.com/Scam-Watch/SCAM-ALERT-Who-blocked-you-from-his-friend-list-Follow-steps-below.html/
(Consultado el 05/02/2017)

[9] Alertan por virus en Facebook al reproducir un video
http://larepublica.pe/tecnologia/761676-alertan-por-virus-en-facebook-tras-reproducir-un-video(Consultado el 05/02/2017)

[10] The Real Face of KOOBFACE: The Largest Web 2.0 Botnet Explained
https://www.trendmicro.de/cloud-content/us/pdfs/security-intelligence/white-papers/wp_the-real-face-of-koobface.pdf (Consultado el 15/03/2017)

[11] Rodas, J., *"Un ejercicio de análisis utilizando rough sets en un dominio de educación superior mediante el proceso KDD"*, Universidad Politécnica de Cataluña, 2001

[12] GamDias, Data Science
https://www.quora.com/topic/Data-Science (Consultado el 16/03/2017)

[13] Ian H. Witten, et. al., *"Data Mining: Practical Machine Learning Tools and Techniques"*, University of Waikato, ISBN: 978-0128042915

[14] Shai Shalev, *"Understanding Machine Learning From Theory to Algorithms"*, Cambridge University Press, 2014

[15] Supervised Learning Model Diagram
http://blog.kaggle.com/2015/04/08/new-video-series-introduction-to-machine-learning-with-scikit-learn/ (Conultado el 17/03/2017)

[16] Pinar Yildirim, "*Filter Based Feature Selection Methods for Prediction of Risks in Hepatitis Disease"*, International Journal of Machine Learning and Computing, 2015(Consultado el 20/04/2017)

[17] Hyper-planes in Support Vector Machine
http://diggdata.in/search/svm (Consultado el 20/04/2017)

[18] MdSazzadur Rahman, *"Efficient and Scalable Socware Detection in Online Social Networks"*, Proceedings of the 21st USENIX Conference on Security Symposium, 2012

[19] Manuel Egele, et.al., *"COMPA: Detecting Compromised Accounts on Social Networks"*, NDSS Symposium 2013, United States, 2013

[20] Roberto Santana, et.al., *"Text and Data mining to detect phishing websites and spam emails"*, Lecture Notes in Computer Science 8298 Theoretical Computer Science and General Issues, Chennai, India, 2013, ISBN: 978-3-319-03755-4

[21] Cao C., Caverlee J., *"Detecting Spam URLs in Social Media via Behavioral Analysis"*, Advances in Information Retrieval: 37th European Conference on IR Research, ECIR 2015, Vienna, Austria

[22] Qian Sun, et.al., *"Transfer Learning for Bilingual Content Classification"*, Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2015, DOI: 10.1145/2783258.2788575

[23] Mayank Pandey, *"Text and Data Mining to Detect Phishing Websites and Spam Emails"*, Institute for development and research in banking technology Castle Hills, 4th International Conference, SEMCCO 2013, India

[24]Anaconda Navigator
https://docs.continuum.io/anaconda/ (Consultado el 17/02/2017)

[25] Jupyter Notebook
http://jupyter-notebook.readthedocs.io/en/latest/notebook.html
(Consultado el 17/02/2017)

[26] Scripts para obtener los comentarios y posts
https://github.com/minimaxir/facebook-page-post-scraper/
(Consultado el 20/02/2017)

[27] Graph API, Facebook for Developers
https://developers.facebook.com/docs/graph-api (Consultado el 20/02/2016)

[28] Beautiful Soup
https://www.crummy.com/software/BeautifulSoup/bs4/doc/
(Consultado el 22/02/2017)

[29] Librería de extracción de dominio de URL
https://pypi.python.org/pypi/tldextract/2.0.3 (Consultado el 22/02/2017)

[30] Librería para obtener las cookies
https://docs.python.org/2/library/cookielib.html (Consultado el 23/02/2017)

[31] Librería para obtener Whois
https://pypi.python.org/pypi/whois (Consultado el 24/02/2017)

[32] Unshorten Me
https://unshorten.me/api (Consultado el 01/03/2017)

[33] DeAdfLy
http://deadfly.crazytec.net/ (Consultado el 01/03/2017)

[34] Fake free codes scam affects PSN and Steam users - https://blog.avast.com/2014/12/05/fake-free-codes-scam-affects-psn-steam-users/ (Consultado el 06/04/2017)

[35] Yin-Wen Chang, *"Feature Ranking Using Linear SVM"*, JMLR Workshop and Conference Proceedings: Causality and Prediction Challenge (WCCI 2008), Taiwan, 2008

[36] Cross Validation to obtain the accuracy value
https://stackoverflow.com/questions/44939725/support-vector-machine-overfitting-my-data (Consultado el 18/08/2017)

[37] Joblib
https://pythonhosted.org/joblib/ (Consultado el 02/09/2017)