Universitat Oberta de Catalunya

uoc.edu

“Detección de URLs fraudulentas mediante técnicas de aprendizaje automático”

Alumno: Brian Enrique Marin Batista
Plan de estudios: “Máster Universitario en Seguridad de las Tecnologías de la Información y de las comunicaciones”. (MISTIC)
Trabajo de fin de máster. Análisis de datos.

Director del TFM: Helena Rifà Pous
Profesor/a responsable de la asignatura: Enric Hernández Jiménez

Fecha Entrega: 2 de Junio de 2020

Quiero agradecer a todas las personas que día a día me ayudan avanzar hacia mis objetivos y me apoyan constantemente, en especial a mi familia que tiene confianza plena en mi, mi pareja y amigos. También agradecer a todo el profesorado que forma este máster y en todo momento ofrecen su conocimiento y su ayuda. Con total sinceridad … Gracias.

Resumen

Actualmente dentro de los sistemas informáticos los datos representan un activo muy valioso e importante para las empresas, instituciones y gobiernos. Esto produce que cada vez haya más delitos informáticos que intentan robar, suplantar y vender estos datos tan cotizados en el mercado. Por ello, se encuentra que el foco principal de los atacantes y unos de los métodos más efectivos de robo de información son las técnicas de suplantación de identidad (phishing) que implican el uso de ingeniería social.

Estudios de empresas mundialmente conocidas, indican que un tercio de todas las fugas de datos de 2018 incluyen el phishing como el causante y en concreto un aumento del 269% de efectividad de esta técnica es mediante URL en 2018.

Por lo que el siguiente trabajo de fin de máster tiene como objetivo la utilización de técnicas de aprendizaje automático que hacen uso de algoritmos de clasificación, para poder determinar cuando una URL es fraudulenta o legítima. Para ello, se analiza el estado del arte de las aplicaciones de aprendizaje automático existente, los dominios relacionados a los que se aplica y su efectividad en la predicción de este tipo de delitos informáticos.

Dada la complejidad del problema y el propósito de este trabajo, se utilizan conjuntos de datos ya categorizados según una serie de atributos que incorpora una URL, y se utilizarán implementaciones de algoritmos automáticos ya existentes, para proceder a entrenar el modelo predictivo.

Finalmente, se comprueba que un nuevo conjunto de muestras no utilizado para entrenar el modelo y de un escenario realista que se procese por el modelo predictivo, es capaz de determinar cuales de las muestras son legítimas y cuales de ellas son fraudulentas.

Palabras claves: Phishing**,** Aprendizaje automático, Detección, Phishing URL

Abstract

Actually, in computer systems, data represents a very valuable and important asset for companies, institutions and governments. This causes that there are more computer crimes that try to steal, supplant and sell this data as quoted in the market. Therefore, we find that the main focus of the attackers and one of the most effective methods of information theft are phishing techniques that involve the use of social engineering.

Reports of worldwide known companies indicate that one third of all data leaks in 2018 include phishing as the cause and specifically a 269% increase in effectiveness of this technique is through URL in 2018.

Therefore, the following master's degree project aims to use machine learning techniques that make use of classification algorithms, in order to determine when a URL is fraudulent or legitimate. For this, the state of the art of the existing machine learning applications, the related domains to which it is applied and its effectiveness in the prediction of this type of computer crimes are analyzed.

Given the complexity of the problem and the purpose of the project, data sets already categorized according to a series of attributes that incorporate a URL are used, and existing automatic algorithm implementations will be used to proceed to train the predictive model.

Finally, it is verified that a new set of samples not used to train the model and a realistic scenario that is processed by the predictive model, is able to determine which of the samples are legitimate and which of them are fraudulent.

Keywords: Phishing**,** Machine learning, Detection, Phishing URL

Índice

Índice de ilustraciones

1. Introducción

1.1 Contexto y motivación

Dentro de los distintos sectores informáticos que se encuentran en la actualidad y que están en auge son la inteligencia artificial, inteligencia de datos (más conocido por su terminología en inglés big data), la computación en la nube y la ciberseguridad. Esto se debe a que cada vez son más importantes los datos que se utilizan tanto por empresas, gobiernos e instituciones. Por ello, también el campo de la ciberseguridad a adquirido un papel tan importante como los otros sectores, dada esta importancia de los datos surgen más delitos informáticos y nuevas técnicas más sofisticadas por los atacantes.

Un informe realizado por la compañía Verizon del 2019, revela que el 32% de las fugas de datos de 2018 involucran actividades de phishing. Además, añaden que la técnica de phishing estuvo presente en el 78% de los incidentes de ciberespionaje y en la instalación y uso de puertas traseras. Si se entra más en profundidad en el tema, un informe de la compañía de ciberseguridad Trend Micro revela que la técnica de phishing mediante URL a incrementado un 269% en 2018 respecto a 2017.

Este es uno de los motivos por el cual muchas empresas invierten una gran cantidad de recursos en tratar de minimizar el riesgo de sufrir una fuga de datos que comprometa a sus clientes. Ya que diariamente las personas son un foco en el que las técnicas de ingeniería social son más efectivas y sencillas de aplicar que otras técnicas que conllevan ataques más sofisticados y complejos de ejecutar.

Muchas de las soluciones que se aplican en la actualidad se basan en detectar URL fraudulentas mediante la información de la base de datos de antivirus y analizar ciertos patrones en base a los históricos de los ataques que se han ido produciendo. También existen entornos de ejecución (más conocido por su terminología en inglés sandbox) preparados para examinar el comportamiento que se produce al acceder a la URL que se recibe como phishing y determinar si es un contenido potencialmente malicioso o legítimo. No obstante, no dejan de ser contramedidas sustanciales al potencial que tienen los atacantes de elaborar ataques más sofisticados y más complicados de detectar.

El presente trabajo pretende cubrir este foco mediante la aplicación de la inteligencia artificial, más concretamente el campo orientado al aprendizaje automatizado mediante algoritmos de clasificación. Se pretende utilizar este enfoco, ya que es una metodología que cada vez es más utilizada en la elaboración de modelos predictivos, que permiten abordar problemas de distinta índole y para grandes volúmenes de datos.

Para resolver el problema se analiza un conjunto amplio de muestras de URL que han sido analizadas y clasificadas como fraudulentas y legitimas. De este conjunto se examinan los distintos campos y atributos para determinar que información aportan y de que manera serán útiles para el entrenamiento posterior del modelo predictivo. Este modelo predictivo se construye a partir del estudio de los distintos algoritmos de aprendizaje existentes aplicados en el lenguaje de programación escogido. De esta manera se tiene un sistema entrenado al que se introduce un nuevo conjunto de datos no utilizado para el entrenamiento previo y se determina que probabilidad de éxito en la clasificación de URL fraudulenta o legitima se obtiene.

1.2 Objetivos

OBJETIVO GENERAL

El objetivo principal de este trabajo de máster es realizar la detección de URLs (localizadores de recursos uniforme) que se utilizan como herramienta de ingeniería social para la obtención de información confidencial de forma fraudulenta, mediante técnicas de software de aprendizaje automatizado.

OBJETIVOS ESPECÍFICOS

Para satisfacer el objetivo general, los objetivos específicos más relevantes que me planteo conseguir en este trabajo de investigación son los siguientes:

1. Explorar como el cibercriminal utiliza la URL fraudulenta para hacer creer a la víctima que es una persona de confianza y adquirir su información confidencial.

2. Analizar cuales son los parámetros y atributos que contiene una URL fraudulenta para determinar como se utilizan en la suplantación de identidad.

3. Comprender el funcionamiento y la implementación de un sistema de aprendizaje automático.

4. Analizar los distintos algoritmos de aprendizaje automático existentes para determinar cuales pueden ser útiles para el objetivo general a resolver.

5. Aprender a implementar un sistema de aprendizaje automático mediante el uso de un lenguaje de programación.

6. Contrastar a través de los resultados obtenidos con los algoritmos de aprendizaje utilizados si la detección realizada se aproxima o mejora los resultados de otros autores relacionados con el tema del trabajo.

7. Aprender las distintas técnicas de clasificación que utilizan los algoritmos de aprendizaje para comprender mejor como se obtienen los resultados que extraen.

1.3 Metodología

La metodología utilizada para cumplir los objetivos de este proyecto se basa en algunas buenas prácticas que se aplican en el modelo de desarrollo de software iterativo e incremental. Se ha escogido algunas ideas de este modelo ya que permite planificar en las distintas iteraciones, los distintos requisitos a implementar y verificar junto a los hitos de los entregables de este trabajo. De manera que, en cada iteración, se planifican los requisitos que hay que satisfacer en cada etapa del proyecto.

El objetivo de esta metodología es cumplir con las tareas planificadas al inicio del proyecto y tener muy claro en cada iteración cuál es el trabajo que se debe realizar y el tiempo estipulado para cada tarea y sus dependencias. Es importante añadir que es necesario satisfacer adecuadamente las tareas asignadas en una iteración, ya que las tareas asignadas a la siguiente iteración dependerán del trabajo realizado posteriormente.

Por lo tanto, este proyecto tendrá las siguientes iteraciones:

Elaboración del plan de trabajo

En esta iteración inicial se pretende elaborar toda la planificación del proyecto que se lleva a cabo y el listado de tareas necesarias para su resolución. Esta fase consta de 13 días para su elaboración y es una de las fases más importantes, dado que se explica el problema a resolver y se realiza el estudio previo de los requisitos que conlleva el tipo de proyecto.

Análisis y diseño del proyecto

En esta iteración se entra en profundidad en el análisis de los ataques que se efectúan actualmente y que información se extraer de los conjuntos de datos de victimas que han sufrido este tipo de ataques. Para ello también es importante conocer el funcionamiento y la implementación de los sistemas de aprendizaje automático y como actúan los algoritmos que sirven para la elaboración de los modelos predictivos. Además, se definen los casos de usos, arquitectura y requisitos del sistema informático a implementar.

Implementación y pruebas

En esta iteración se pone en práctica todo el análisis previo realizado sobre las tecnologías existentes, y se utiliza el lenguaje de programación escogido para la implementación del análisis de los datos mediante las técnicas de aprendizaje automático. Esto permite posteriormente obtener métricas y realizar una valoración de los resultados obtenidos.

Presentación de resultados y conclusiones

En esta iteración final, como última parte del proyecto, se presentan los resultados obtenidos con los diferentes algoritmos utilizados por los modelos predictivos definidos y que tanto porcentaje de acierto extraen en base al conjunto de datos procesado y que tan efectiva en la solución implementada para su utilización en un escenario real. De manera que pueda ser utilizado como un motor de detección de ataques de este tipo en un entorno corporativo.

1.4 Planificación del trabajo

1.4.1 Listado de tareas

A continuación, se muestran las figuras relacionadas al listado de tareas establecido y agrupado por cada entregable:

TFM Análisis de datos	67%
▼ Entregable 1 - Plan de trabajo	100%
Charla Tutor TFM	100%
Investigación del problema a resolver	100%
Estudio del estado del arte	100%
Describir contexto y motivación del proyecto	100%
Definición de objetivos	100%
Planificación de tareas y metodología	100%
Análisis de riesgos	100%
Revisión del plan de trabajo	100%
Entrega Plan de trabajo	☑
▼ Entregable 2 - Análisis y diseño del proyecto	100%
Estudiar metodología de los ataques de Phishing	100%
Analizar atributos y campos del conjunto de datos	100%
Analizar funcionamiento técnicas aprendizaje automático	100%
Investigar los algoritmos de clasificación	100%
Definir los requisitos y casos de uso del sistema	100%
Diseñar la arquitectura del sistema	100%
Definir los algoritmos a implementar	100%
Establecer herramientas y lenguajes de programación	100%
Revisión del análisis y diseño del proyecto	100%
Entrega Análisis y diseño del proyecto	☑

Ilustración 1: Listado de tareas entregable 1 y 2.

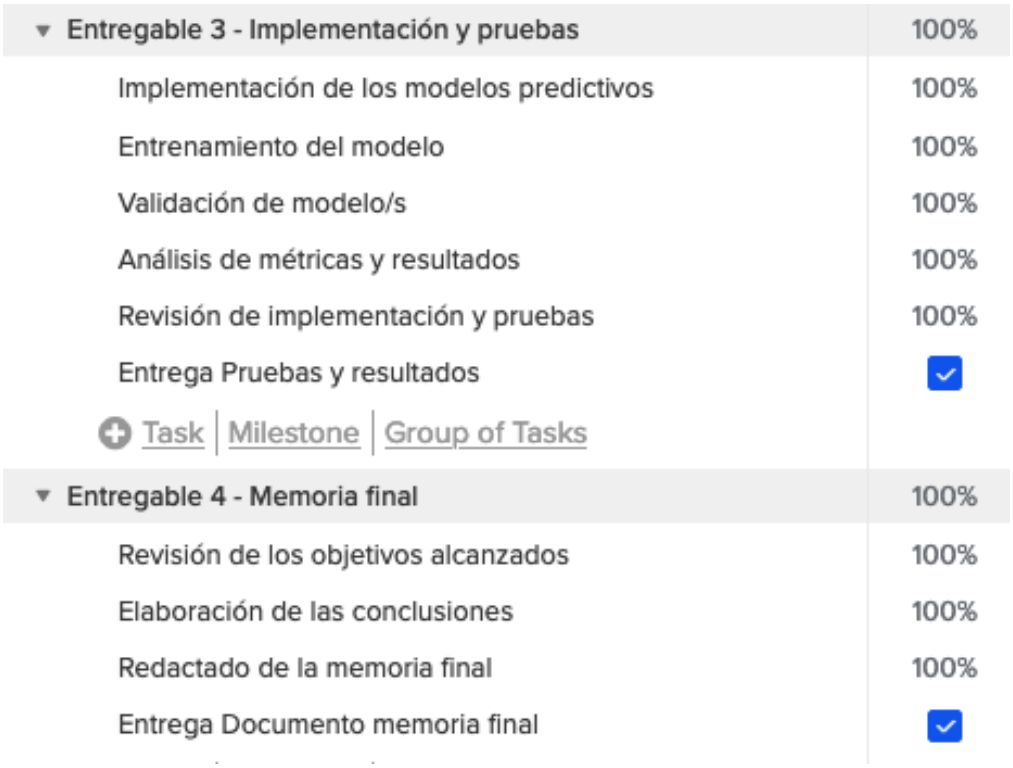


Ilustración 2: Listado de tareas entregable 3 y 4.

1.4.2 Planificación temporal

En este apartado se muestra la relación temporal del listado de tareas y sus dependencias.

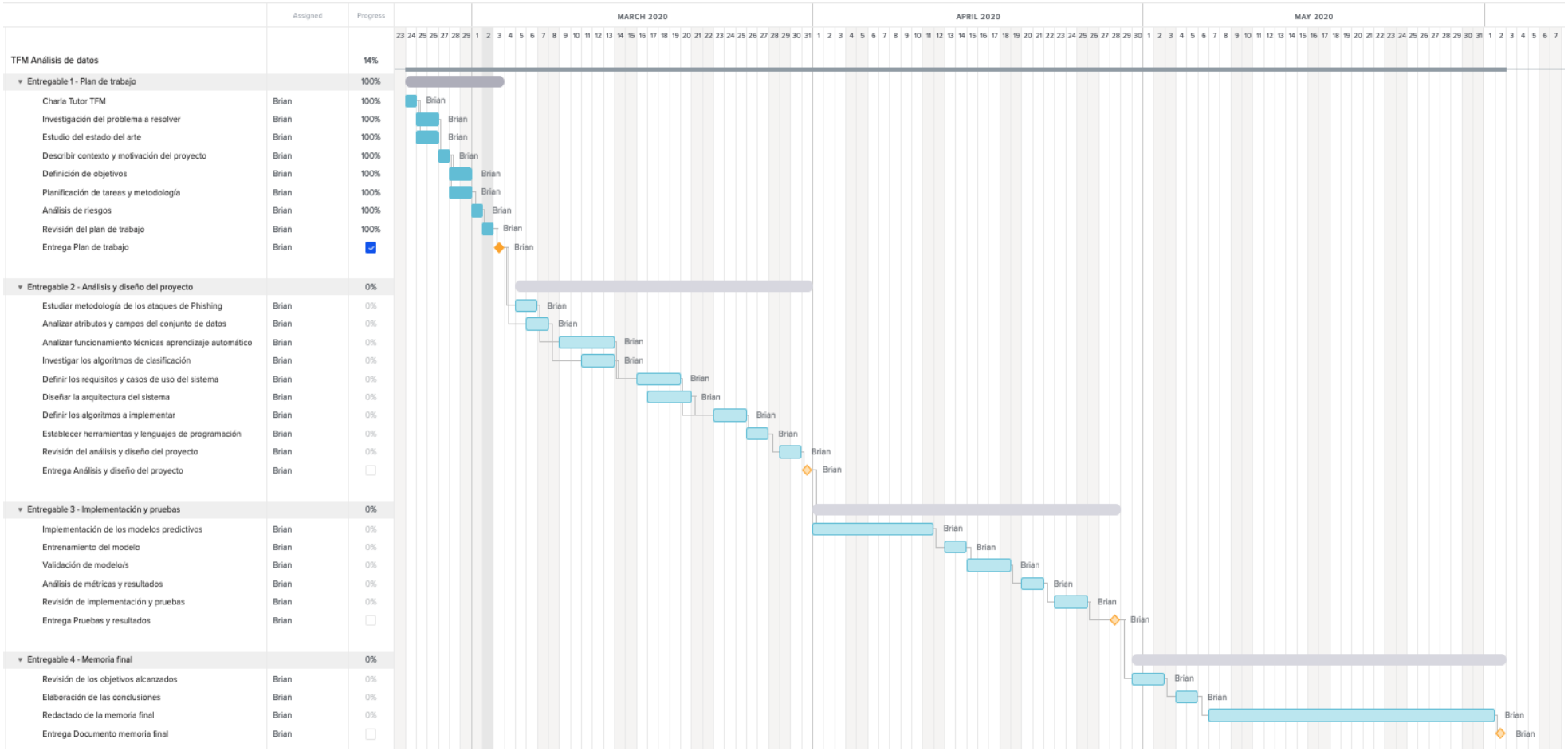


Ilustración 3: Diagrama de Gantt

1.4.3 Recursos necesarios y presupuesto

Dado que es un proyecto que esta enfocado al análisis de datos y al uso de un lenguaje de programación que te permita elaborar los modelos predictivos, se resumen los recursos materiales necesarios en:

- Ordenador personal donde se realiza la implementación. El coste puede variar de un modelo a otro, pero el coste es de 1200€.

- Conexión a internet. Depende el servicio adquirido puede variar el coste, pero en este caso es de unos 30€/mes.

1.5 Análisis de riesgos

Es importante asumir los riesgos principales que en el desarrollo de este proyecto se pueden producir, de manera que se describen y se tienen en cuenta en caso de que puedan afectar a la planificación o a los hitos del proyecto donde se debe realizar las entregas. Para ello se realiza una breve descripción del riesgo, probabilidad e impacto y posibles acciones de mitigación:

- **Riesgo 1: Complejidad en la implementación de los modelos predictivos.** Un riesgo importante se puede producir en poder probar los algoritmos encontrados para elaborar el modelo predictivo. Probabilidad/Impacto (1 - 5): 2 / 5. Entre los muchos lenguajes de programación que existen, se escoge uno que tenga un amplio soporte en librerías que faciliten esta tarea.

- **Riesgo 2: Modelos predictivos ineficientes o ineficaces.** Este riesgo está estrechamente relacionado con el anterior, ya que debido a una mala elección en la manera de implementar el modelo o debido a unos conjuntos de datos no eficientes, puede devolver resultados malos. Probabilidad/Impacto (1 - 5): 3 / 4. Para intentar mitigar este riesgo se realiza un análisis exhaustivo de las técnicas existentes y se comprueba su eficiencia en campos relacionados al domino del proyecto.

- **Riesgo 3: Facilidad de burlar el sistema introduciendo casos sofisticados.** Esto es un riesgo debido a la complejidad del domino escogido por el proyecto, y la construcción de unos modelos predictivos ineficientes.
  Probabilidad/Impacto (1 - 5): 3 / 3. Utilizar un conjunto de dados elevado y consistente en el proceso de elaboración del modelo predictivo.

1.6 Productos obtenidos

Como artefactos que se obtienen en la finalización del trabajo de fin de master, son los que se han ido generando por cada entrega en cada iteración. Se agrupan en los siguientes puntos:

- Entrega 1: Esta primera entrega corresponde al plan de trabajo que se sigue para finalizar el proyecto. Es un documento de texto.

- Entrega 2: Se trata de un documento de texto en cual se define la parte más de análisis y estudio de lo que se implementa. Describe las diferentes partes que corresponde en el sistema y un análisis exhaustivo del ciberataque que se intenta mitigar.

- Entrega 3: En esta entrega se describe la implementación y las pruebas realizadas. Es un documento de texto.

- Entrega 4: Corresponde a la entrega de la memoria final de este proyecto en un documento de texto y archivos de código fuente del código de programación desarrollado.

- Entrega 5: Es un documento multimedia en video que incluye la presentación formal de este trabajo.

2. Estado del arte

Se han encontrado numerosos estudios que intentan resolver este problema aplicando técnicas de aprendizaje automático y en general las métricas de los resultados son bastante óptimas en relación al número de falsos positivos y falsos negativos. Se utilizan principalmente algoritmos de clasificación supervisados donde previamente se tiene un conjunto de datos tratado y etiquetado donde se aporta si la URL es legítima o no. Dicho esto, la pegunta que se debe hacer es por qué a día de hoy los casos de phishing siguen incrementando año tras año y es una de las técnicas más efectiva en la suplantación de identidad. Dichos estudios destacan que sigue siendo un problema complejo de solucionar y que la creatividad de los ciberdelincuentes puede producir burlar el sistema entrenado y quizá sea mejor apostar por una solución que incluya un sistema de aprendizaje profundo y no un sistema de aprendizaje supervisado.

De las diferentes investigaciones que se han analizado se destaca el de Ozgur Koray [1], que utiliza dos tipos de clasificadores como metodología de predicción, las redes neuronales artificiales y la red neuronal en profundidad donde se extraen los componentes de una URL y se seleccionan las características más relevantes para su clasificación. Para ello extraen 16 características que son las que permiten al modelo predictivo llegar a la conclusión de URL fraudulenta o legítima. Este estudio demuestra una solución de un 96% de acierto. No obstante, para este tipo de detecciones es importante la latencia del tiempo de ejecución del algoritmo, por lo que tienen limitaciones en el cálculo de una de las características utilizada en el modelo y es necesario deshabilitar este importante parámetro para la detección.

Otra de las investigaciones centradas en este tema es la de R.Kiruthiga[2] que utiliza los algoritmos de aprendizaje automático como Naïve Bayesian, SVM, árbol de decisión y Random Forest. Además, comenta que proponen un nuevo sistema llamado PhishScore y PhishChecker para añadir precisión a la detección de URL. Los autores del artículo comentan que obtienen un 96% de éxito en este tipo de detecciones, a pesar de ello, describen que las técnicas empleadas por los ciberdelincuentes que utilizan las técnicas de phishing cambian constantemente y creen necesario ir mejorando el sistema añadiendo nuevas características.

En última instancia se analiza el trabajo realizado por Ms. Sophiya Shikalgar[3] que además de utilizar los algoritmos comunes como Naive Bayes, Random Forest y Support vector machine introducen otro llamado XGBoost que produce un modelo de predicción en forma de un conjunto de modelos de predicción débiles, esto quiere decir que construye el modelo de forma escalonada. En esta investigación extraen unas nueve características y entrenan los modelos con un conjunto de datos entorno a las 2900 URLs. Las métricas que obtienen, a pesar de que comentan que son buenas, son inferiores a las otras investigaciones analizadas y la precisión más alta que obtienen es de un 86.3% mediante SVM. Cabe la posibilidad de que haya sido debido al pequeño conjunto de datos utilizado en el entrenamiento de los modelos o las características utilizadas en el proceso.

---

1. https://airccj.org/CSCP/vol8/csit89705.pdf
2. https://www.ijrte.org/wp-content/uploads/papers/v8i2S11/B10180982S1119.pdf
3. https://www.ijert.org/research/detection-of-url-based-phishing-attacks-using-machine-learning-IJERTV8IS110269.pdf

3. Ataques de Phishing

3.1 Introducción

Antes de analizar los distintos métodos de ataque que es posible encontrar en el momento de analizar una URL que se utilice como phishing, es necesario realizar una breve explicación de como se compone una URL (de las siglas en inglés **Uniform Resource Locator**). El objetivo principal de una URL y por lo que se crea es para dirigir de una manera fácil a una página web y no tener que utilizar una dirección IP que es difícil de recordar. La siguiente figura (extraída de internet) muestra la estructura básica y las distintas partes de una típica URL:

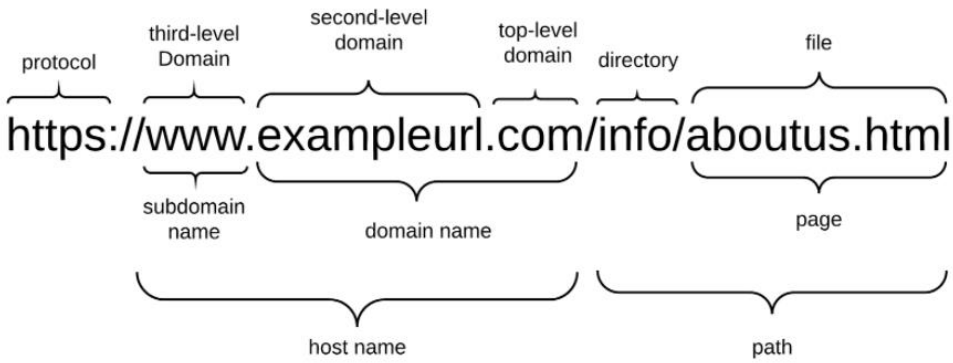


Ilustración 4: Partes de una URL

De la imagen anterior se extrae:

- Protocolo: Protocolo utilizado para acceder al sitio web.
- FQDM (del inglés, Full Qualified Domain Name). Identifica el servidor que aloja la página web. Incluye los siguientes niveles de dominio:
  - Nombre del host o subdominios. Es la parte que aprovechan los atacantes normalmente para engañar a la víctima.
  - Dominio. Esta es la parte en la que es necesario pagar para adquirir en un servicio web.
  - TLD (del inglés, Top Level Domain). Indica el dominio de nivel superior.
- Directorio: Rutas de los recursos que aloja el sitio web.
- Fichero: Indica la página o fichero de contenido que se pretende mostrar al usuario.

Tras analizar la estructura de una URL, se muestra un ejemplo de lo que enviaría un atacante a una victima para el robo de credenciales.

http://rav0n.000webhostapp.com/paypal/paypal/login.html

De la que extraemos lo siguiente:

- **Protocolo:** http://
- **Dominio:** 000webhostapp.com
- **Directorio:** /paypal/paypal/
- **Subdominio:** rav0n.
- **Fichero:** login.html

Por el tipo de directorio que incluye esta URL, se puede identificar que este phishing intenta robar las credenciales del sitio web PayPal.

3.2 Métodos

Métodos que incluyen una URL en el ataque

1. Mezclar URLs legítimas (del sitio web oficial al que se quiere suplantar la identidad) junto al la URL de phishing en el correo que se envía a la víctima.

2. Se basa en la **redirección de URL**. La técnica conocida como "time-bombing" se basa en la creación de una redirección de URL desde la página legítima a la que se pretende hacer phishing. Esta técnica es algo más elaborada, ya que envía la URL legítima a través de correo a la víctima y una vez es recibida (que a evadido los filtros del correo), la página de redirección fraudulenta es creada.

3. Incrustar en el cuerpo del mensaje de correo una imagen en vez de texto, con el contenido que te enviaría la página legítima. De esta manera se incrusta un hipervínculo en la imagen, que al realizar click te redirigirá a la página fraudulenta.

Métodos que se basan en la manipulación de la URL

1. **Uso de sub-dominios:** Este método se basa en la utilización de subdominios de la URL para engañar a la víctima y hacerle creer que está visitando la página legítima. Dado que un usuario común muchas veces desconoce que la jerarquía de dominios va de derecha a izquierda, el atacante invierte el orden y hace creer a la victima que está visitando el dominio legítimo, cuando en realidad esta accediendo al dominio fraudulento.

Ejemplo:
https://www.apple.com.ntemu.cn/mim/55375x21ho779v3619l025cs919f782bq8276481878q70f184.html

2. **Ocultación/acortado de URL**: Una manera común de engañar a la víctima y que no requiere mucha complejidad de ejecución es la de ocultar la URL de la página fraudulenta incluyendo una cadena de texto alternativa como **"Suscríbete"** o **"Haz click aquí"**. De esta manera la víctima no ve la URL extraña que le pueda hacer sospechar y es más susceptible de ser accedida. Otra manera de explotar esta técnica por los atacantes, se basa en el uso de herramientas online como **TinyURL** y **Bitly** que permiten acortar la URL original que se utiliza como phishing.

Ejemplo: https://tinyurl.com/sq87wcd

3. URL mal escritas: En esta técnica de manipulación de URL (también conocida como URL hijacking), el atacante adquiere un dominio similar a la URL fraudulenta a la que quiere suplantar la identidad, pero con errores ortográficos. Hay diferentes variantes de explotar este tipo de ataque, se describen a continuación (*fuente: Wikipedia*):

- Error ortográfico. Ejemplo: ejemple.com.
- Error tipográfico. ejemlop.com
- URL expresada diferente. ejemplos.com
- Utilizar un TLD (dominio de nivel superior) diferente. ejemplo.org
- Utilizar un ccTLD (código de país TLD) diferente. ejemplo.co

4. **Ataque IDN** (del inglés internationalized domain name) homograph. Este método intenta sacar ventaja del uso de caracteres que son visualmente similares. Por ejemplo, la mayúscula i (I) y la minúscula L (l), que son similares.

Otras maneras de prevenir la detección de phishing en URL se basan en mantener una lista negra (incluyen listas de palabras clave, URLs y direcciones IP), mantener una lista blanca, inspeccionando el contenido de la página o realizar un análisis manual del contenido visual de la página.

4. Machine learning y algoritmos de aprendizaje

4.1 Introducción

Según el dominio del problema que se pretende resolver, el conjunto de datos que se utiliza para entrenar el modelo predictivo y que recibe el algoritmo de aprendizaje, viene definido por un conjunto de características extraídas previamente y son datos "etiquetados". Esto quiere decir que cada dato de nuestro conjunto de datos contiene una variable que define si es phishing o no según el problema que se intenta resolver en este trabajo. Esto me dice que el tipo de aprendizaje que se debe aplicar es de tipo supervisado, dado que se intenta resolver un problema de clasificación y lo que se obtiene en la salida del algoritmo es una predicción de que si es o no una URL fraudulenta.

Dentro de los tipos de implementación del aprendizaje automático (rama de la inteligencia artificial) se pueden clasificar en tres categorías:

- Aprendizaje supervisado.
- Aprendizaje no supervisado.
- Aprendizaje de refuerzo (a través de la prueba y error).

Tal y como se comento anteriormente, dentro de las distintas categorías de implementación del aprendizaje automático, el que tiene un mejor enfoque hacia el problema que plantea este proyecto es el de aprendizaje supervisado. Dado que los algoritmos que están dentro de este tipo de aprendizaje, entrenan el modelo a partir de un "histórico" de datos que están etiquetados y nos dicen a que tipo corresponden. Por lo que el algoritmo aprende a asignar una etiqueta de salida en base a este histórico y los distintos casos que encuentra para finalmente predecir a que categoría pertenece.

El aprendizaje supervisado se suele utilizar en:

- Problemas de clasificación (ej. Detección de fraude de identidad). En este tipo de problema se clasifica la variable objetivo de **tipo categórico**.
- Problemas de regresión (ej. Predicciones meteorológicas). En los casos de regresión se clasifica la variable objetivo de **tipo numérico**.

El problema que se intenta resolver podríamos catalogar dentro del tipo de problemas de clasificación, ya que es posible clasificar la variable objetivo como categórica, phishing o no phishing.

Entre los algoritmos más populares utilizados en el aprendizaje automático supervisado para la clasificación de datos encontramos los siguientes:

- Árboles de decisión.
- Clasificación de Naïve Bayes (teorema de bayes).
- Regresión lineal por mínimos cuadrados. No se utilizará en este proyecto.
- Regresión Logística.
- Support Vector Machines (SVM).
- Métodos "Ensemble" (Conjuntos de clasificadores).

De esta lista se han seleccionado los que se utilizan en este proyecto y se explican más en detalle para ver como funcionan internamente sin entrar en profundidad en la base matemática que conllevan.

4.2 Diagrama de flujo elaboración del modelo de aprendizaje automático

Antes de entrar en detalle sobre los distintos algoritmos de aprendizaje automático que se utilizan, se presenta un pequeño diagrama que indica los distintos procesos que se siguen desde la recolección de datos hasta la obtención de métricas del modelo que se ha entrenado. El proceso de entrenamiento del modelo y los procesos consecuentes se repiten a fin de probar los distintos algoritmos que se han investigado.

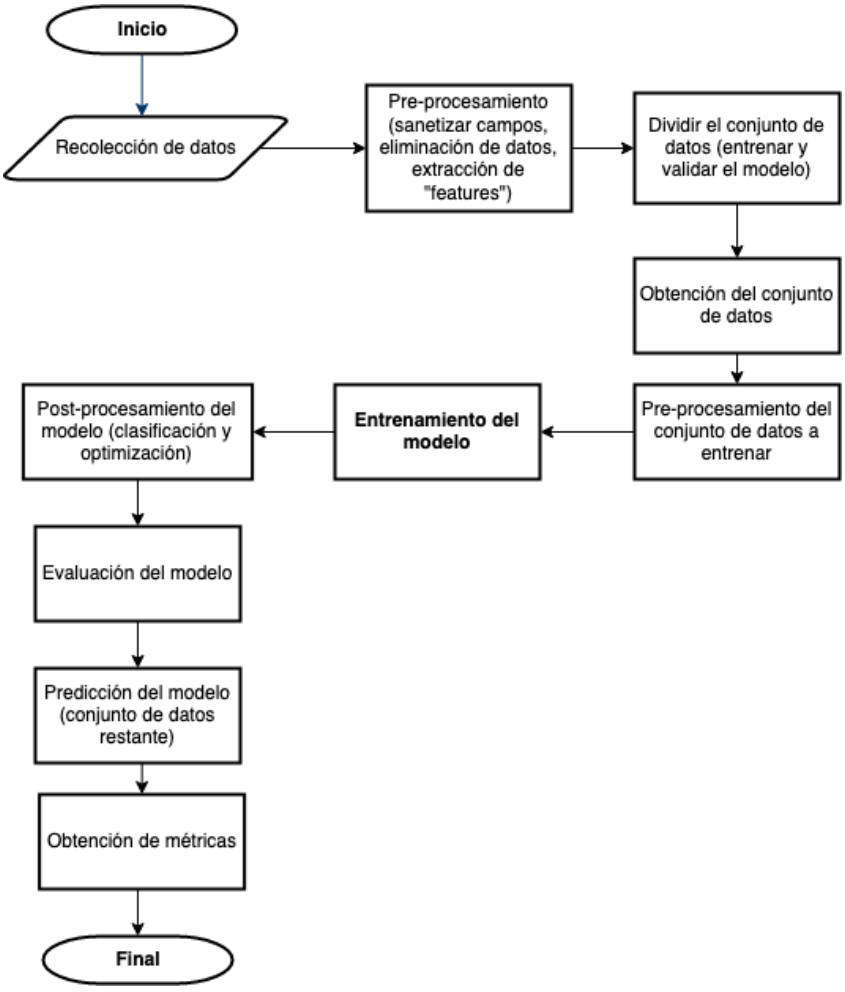


Ilustración 5:Diagrama de flujo aprendizaje automático

4.3 Algoritmos de aprendizaje automático

4.3.1 Regresión Logística

Es una técnica de aprendizaje automático de clasificación binario, en el que se obtiene un valor binario entre 0 y 1. Es una red neuronal con exactamente una neurona que realiza una función logística. La base de este algoritmo se basa en medir la relación entre la variable dependiente (lo que se desea predecir), la etiqueta que nos dice que la URL es phishing o no, en el ámbito de este proyecto y la variable/s independientes (características que utilizamos para medir si una URL es phishing o no). Para ello utiliza una función logística (también llamada función Sigmoide) que determina la probabilidad de la variable dependiente. Se suele utilizar un umbral que determina que si es por encima de 0.5 la afirmación es cierta y por debajo es falsa, pudiéndose gestionar para reducir el número de falsos positivos o falsos negativos.

La función logística (o Sigmoide) se representa en una curva en forma de S en el plano de x, y en el que puede tomar cualquier valor entre 0 y 1 (eje Y). La ecuación que define a la función es la siguiente: $f(x) = 1 / (1 + e^{-x})$. La x de esta función representa el $\sum$ de las características seleccionadas en nuestro conjunto de datos multiplicado por el valor que contienen. Por eso es importante determinar las características adecuadas y el peso que conllevaran para determinar la decisión final. En la siguiente figura se ve una representación gráfica de la función logística.

Ilustración 6: Gráfica regresión logística

Es una técnica sencilla y con los resultados fácilmente interpretables, la desventaja radica que el problema que se intenta resolver debe ser linealmente separable sino la técnica no es eficiente.

4.3.2 Naïve Bayes (NB)

Naïve Bayes es una técnica que utiliza modelos probabilísticos para el aprendizaje automático. El nombre es debido a que se basa en el teorema de Bayes. El teorema de Bayes se utiliza para encontrar la probabilidad de que ocurra A, dado que una variable B ha ocurrido. La ecuación viene definida de la siguiente forma:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Ilustración 7: Fórmula de Bayes

Donde A y B representan eventos, P la probabilidad y P(B) $\neq$ 0.

- P (A | B) es la probabilidad condicional: Define la probabilidad de que ocurra un evento A cuando B es cierto.

- P (B | A) define los mismo que el punto anterior, pero a la inversa.

- P(A) y P(B) indica la probabilidad de A y B de manera independiente uno del otro.

Estos conceptos aplicados al problema que se intenta resolver representarían la variable B con las distintas características que se utilizan en el conjunto de datos seleccionado y la variable A contendría la etiqueta que dictamina si es phishing o no el conjunto de clases de B. Además, este teorema supone que las características son estadísticamente independientes, esto quiere decir que espera que las variables de entrada no tengan relación. Esto aplica al domino al que se aplica, ya que en este trabajo las características que se utilizan no tienen relación directa entre ellas para determinar si una URL es fraudulenta o no.

Dentro de Naïve Bayes se distinguen tres tipos de clasificador:

- **Multinominal Naive Bayes:** Este tipo de clasificador se suele utilizar en la clasificación de documentos, por ejemplo, para determinar que un documento pertenece a una cierta categoría como economía, salud, política, etc. Aquí se utilizaría la frecuencia de las características del conjunto de datos.

- **Bernoulli Naive Bayes:** Este tipo es similar al tipo anterior, con la diferencia que la variable resultante es de tipo booleana. Este tipo es el que especialmente aplica al problema que se intenta resolver, ya que el resultado que se espera de nuestro modelo es una variable que indique si es phishing o no.

- **Gaussian Naive Bayes:** En este tipo la variable resultante es una variable con valores continuos y no discretos. Por el nombre del tipo, aquí se espera que los valores se obtengan de una distribución gaussiana.

4.3.3 Support Vector Machine (SVM)

Máquina de vectores de soporte (en inglés, support vector machine) es otro algoritmo utilizado en el aprendizaje automático, puede ser utilizado en problemas de regresión y clasificación y destaca por el bajo coste computacional que utiliza y los niveles altos de precisión que obtiene. El objetivo principal de SVM es encontrar un híper plano en N-dimensiones espaciales (N representa el número de características) que permita clasificar los puntos de datos. Para separar los datos en dos clases, existen multitud de posible híper planos que escoger, por lo que, SVM intenta encontrar el plano que represente el máximo margen entre las dos, es decir la distancia máxima entre los puntos de datos entre las dos clases. Esto provee que los datos puedan ser clasificados con mayor precisión en una de las clases que ha dictaminado SVM. La siguiente ilustración muestra la representación de los datos a clasificar, donde la línea continua estima el límite entre ambas clases:

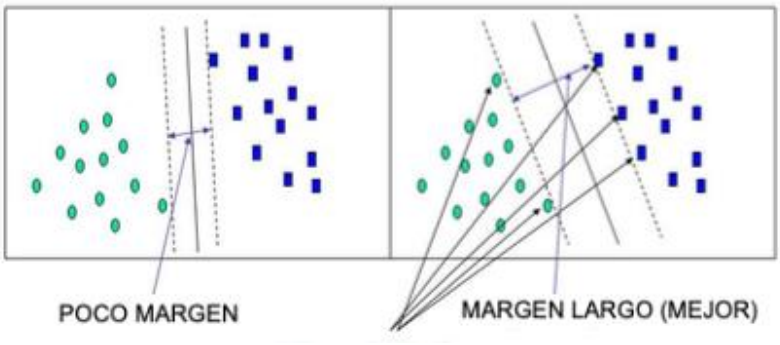


Ilustración 8: SVM

SVM no siempre consigue una separación optima en el plano, esto se conoce como sobreajuste (overfitting). Por ello SVM utiliza un parámetro C que intenta controlar la compensación entre los errores de entrenamiento y los márgenes rígidos, creando un margen blando (soft margin) que se encarga de permitir algunos errores en la clasificación, pero los penaliza.

La ilustración anterior mostrada sería un caso ideal que es posible representar en dos dimensiones, pero no siempre es posible aplicar el algoritmo SVM solo a este caso, por lo que este algoritmo introduce la representación por medio de **funciones Kernel**. Estas funciones se encargan de proyectar la información en un espacio de características de mayor dimensión, que tiene como desventaja el aumento de la capacidad computacional de las máquinas de aprendizaje lineal.

4.3.4 Árbol de decisión

El árbol de decisión es otro de los algoritmos muy utilizado en el aprendizaje automático supervisado para resolver problemas de regresión y clasificación. La base de este algoritmo es sencilla, ya que se basa en una estructura en forma de árbol donde se elige un nodo raíz (la muestra de los datos) y a partir de este se utiliza el resto de atributos para ir descomponiendo el árbol en ramas que indican una condición que puede ser cierta o falsa. De este modo se va bifurcando cada nodo hasta llegar a los nodos finales que no indican la solución al problema que estamos planteando, en el caso de este proyecto si la URL es fraudulenta o no. Este algoritmo se encarga de realizar todas las posibles combinaciones de árboles según los atributos de entrada, para dictaminar cual es el mejor árbol para el problema que se desea resolver.

Para determinar como el árbol se ramifica en subnodos y realizar las predicciones utiliza diversas funciones, de entre las que se destacan las siguientes entre las más populares:

- **Índice Gini**: Calcula un valor en función de la variable objetivo y elementos de la muestra del conjunto de datos, para determinar cual muestra obtiene mayor homogeneidad y realizar la subdivisión de nodos según este criterio.

- **Ganancia de información**: Utiliza los atributos categóricos para intentar estimar la información que aporta cada atributo en base a la entropía (incertidumbre de la información).

En la siguiente ilustración se muestra a grandes rasgos como opera un árbol de decisión:

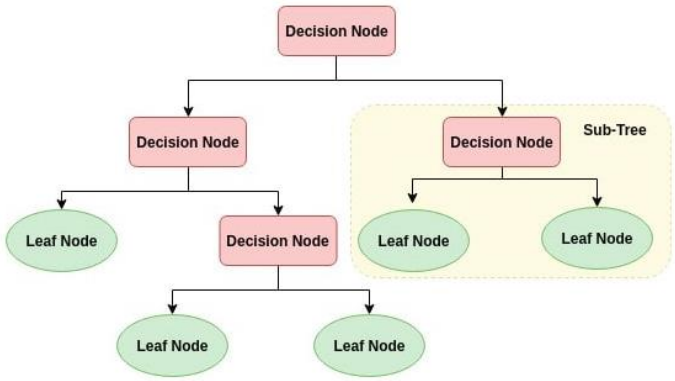


Ilustración 9: Árbol de decisión

Las principales desventajas de esta técnica son que pequeñas variaciones en el conjunto de datos pueden producir arboles de decisiones completamente distintos. Es posible que se creen árboles demasiado complejos y no óptimos para el problema que se intenta resolver. Si el conjunto de datos con el que se entrena el modelo no está equilibrado, puede producir que se creen árboles de decisión sesgados si hay algunas clases que predominan.

4.3.5 Random Forest

Esta técnica se basa en la anterior ya que consiste en utilizar una gran cantidad de arboles de decisión que trabajan en conjunto. Cada árbol dentro del bosque aleatorio se encarga de predecir una clase y la clase con más votos dentro del conjunto de árboles aleatorios se convierte en la predicción del modelo. Los árboles no están correlacionados entre sí, por lo que, si un árbol genera un árbol de decisión erróneo, la suma de los resultados de los otros árboles podrá dictaminar un buen resultado. La principal diferencia con el árbol de decisión es que en bosque aleatorio el proceso de encontrar el nodo raíz (root) y la división de los nodos de características se realiza de manera aleatoria. En la siguiente figura se representa como escoge random forest la predicción del modelo:

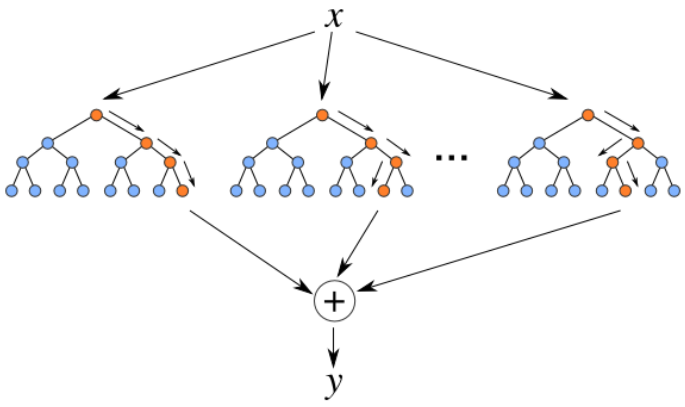


Ilustración 10: Random forest

Para asegurarse de que cada árbol de manera individual no está muy correlacionado con el comportamiento de ninguno de los otros árboles del modelo, el bosque aleatorio utiliza un par de métodos para ello:

- **Agregación de bootstrap**: Se introduce un conjunto de entrenamiento del tamaño de la muestra a cada árbol, pero reemplazando elementos de la muestra.
- **Aleatoriedad de características**: Cada árbol solo puede escoger un subconjunto aleatorio de características. Esto obliga a una variación mayor entre árboles del modelo y una menor correlación entre ellos con una mayor diversificación.

La principal desventaja de esta técnica es la complejidad que supone elaborar este modelo ya que la construcción de los árboles aleatorios consume mucho más tiempo y recursos computacionales que si se utilizara solo la técnica de árbol de decisión.

4.3.6 K vecinos más cercanos (en inglés, k-nearest neighbors)

Por último, dentro de los algoritmos de clasificación que se prueban en este proyecto se encuentra K vecinos más cercanos, que es un algoritmo de aprendizaje automático supervisado como los anteriores. Este algoritmo se basa en clasificar los valores mediante puntos de datos más similares por cercanía, aprendidos en la etapa de entrenamiento del modelo. La K del nombre significa la cantidad de puntos vecinos que tenemos en cuenta para clasificar en phishing o no phishing en el dominio del trabajo. A diferencia de la regresión logística o los árboles de decisión, este algoritmo no aprende explícitamente un modelo, sino que memoriza las instancias en la fase de entrenamiento como base de conocimiento para la predicción.

KNN funciona de la siguiente manera, primero calcula la distancia entre el ítem a clasificar y el resto de elementos del conjunto de datos utilizados en el proceso de entrenamiento. Segundo selecciona los “K” elementos más cercanos (con menor distancia mediante la función que utilice). Por ultimo, realiza un proceso de votación de mayoría entre los K puntos y la etiqueta que predomine será la decisión final. Para medir la cercanía entre puntos se suele utilizar la distancia Euclidiana o la Cosine Similarity que mide el ángulo entre los vectores.

De este algoritmo se destaca su facilidad de uso, su tiempo de cálculo es rápido y no se tiene que hacer suposiciones sobre el conjunto de datos del que se dispone. Como contra tiene que la precisión de la clasificación depende de la calidad de los datos en el entrenamiento, encontrar un valor óptimo de “k” (número de vecinos más cercanos) y puede realizar una clasificación deficiente de los puntos de datos en un límite donde no se pueden clasificar de una forma o de otra.

4.4 Características del conjunto de datos

Para el análisis de los datos y la aplicación de los algoritmos de aprendizaje automatizado para la elaboración de los modelos, se utiliza un conjunto de datos ya sanetizado para evitar tener que buscar en múltiples recursos direcciones de phishing y estandarizar los datos, esto permite centrar los esfuerzos en analizar las características y los algoritmos que mejor se adecuen al tipo de problema que se intenta resolver. A continuación, se muestra una tabla con las características que se utilizan y una breve descripción de que aportan:

No	Característica	Descripción
1	Domain	Indica la URL.
2	Ranking	Métrica que mide la importancia de la web. Mide el número y la calidad de hipervínculos de la página.
3	isIp	Comprueba si la URL contiene una dirección IP.
4	valid	Esta información se obtiene de la API Whois de Google para conocer el estado del registro de la URL.
5	activeDuration	Se obtiene igual que la característica anterior y indica el tiempo transcurrido desde que se registro el dominio.
6	urlLen	Indica el número de caracteres de la URL.
7	is@	Comprueba si la URL incluye arrobas.
8	Isredirect	Comprueba si la URL realiza redirecciones de página.
9	haveDash	Comprueba si la URL incluye “-“.
10	domainLen	Indica el número de caracteres del nombre de dominio.
11	noOfSubdomain	Número de subdominios que incluye la URL.
12	Label	Es la etiqueta que indica si es una URL fraudulento o legítima. 0 indica legítima y 1 indica Phishing.

Además de estas características que se utilizan en el conjunto de datos, se han encontrado otras características que podrían ser igual de útiles para determinar si una URL es legítima o no. Estas características no se utilizan en los modelos que se elaboran, pero es necesario contemplarlas en caso de que los modelos no den unas métricas adecuadas en relación a investigaciones que se han realizado previamente a este proyecto y sea necesario re-entrenar los modelos. Las características son las siguientes:

Característica	Descripción
exeOrZip	Si la URL contiene un archivo a descargar como exe o zip.
noOfDots	Número de puntos que contiene la URL.
specialChar	Contiene símbolos especiales como signo de exclamación, signo del dólar, ampersand, etc.
httpCount	Número de veces que aparece la palabra http.
suspiciousWords	Palabras sospechosas como login, authenticate, profile, resetPassword.
entropy	Mide la incertidumbre de la información donde las palabras menos frecuentes aportan más información.
isShorten	Si la URL está creada con un acortador como bitly.com.

5. Requisitos del sistema

A continuación, se detallan los requisitos funcionales y de calidad que el sistema debe de cumplir.

5.1 Requisitos funcionales

Identificador	Descripción
RF1	El sistema debe de elaborar un modelo de predicción en base a un conjunto de datos y un algoritmo de aprendizaje supervisado.
RF2	El sistema debe de predecir si una URL es legítima o no.
RF3	El sistema debería proporcionar al usuario la capacidad de comprobar si una URL es fraudulenta o no.
RF4	El sistema debe de ser capaz de recibir una URL y devolver una predicción.
RF5	El sistema debe de proporcionar al usuario la capacidad de interactuar con una API Gateway para enviar las peticiones de predicción de URL.
RF6	El sistema debería de permitir la exportación de los modelos predictivos generados.

5.2 Requisitos de calidad

Identificador	Descripción
RC1	El sistema debería de ser capaz de devolver una predicción en un tiempo inferior a 5 segundos.
RC2	El sistema debería de calcular con una precisión superior al 90% si una URL es fraudulenta o no.
RC3	El sistema debería de ser capaz de devolver una respuesta en formato tipo JSON o texto plano de la predicción.
RC4	El sistema debería de implementar los siguientes modelos predictivos: Regresión logística, naïve bayes, SVM, árbol de decisión, random forest y KNN.
RC5	El sistema debe permitir recibir peticiones HTTP GET mediante una aplicación web para realizar consultas de URL.

6. Casos de uso

Este proyecto está enfocado en el análisis de datos y como encontrar y elaborar modelos que permitan tener un porcentaje elevado de acierto en la predicción que realizan. Por ello, el sistema incorpora pocos casos de uso y pocos actores que interactúan con el sistema. En la figura siguiente se muestran los casos de uso del sistema:

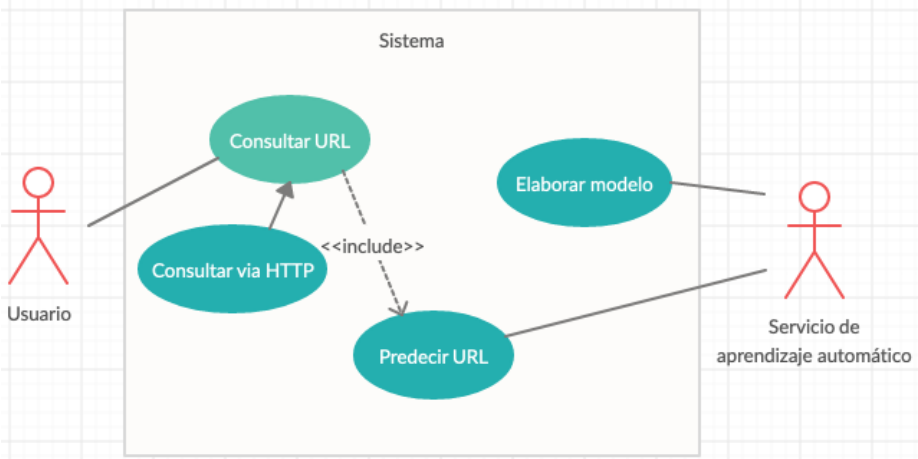


Ilustración 11: Casos de uso del sistema

7. Arquitectura del sistema

En la definición de la arquitectura del sistema se ha optado por una solución sencilla basada en micro servicios, dado que el objetivo de este proyecto no es proporcionar una interfaz compleja de usuario, ni se requiere de una base de datos, sino de una solución donde se puedan aprovechar los modelos creados y enviar peticiones de predicción por parte de usuarios. Por lo que el flujo de la arquitectura del sistema se puede definir en tres pasos:

1. El cliente hace una petición HTTP GET con un cuerpo de mensaje que incluya una URL como campo.

2. Esta petición es recibida por la API Gateway que se encargará de trasmitirla al micro servicio definido.

3. Por último, el micro servicio (creado en Python) se encarga de recibir esta petición, extraer la URL que se pretende predecir y comprobar en los modelos predictivos que tenga definidos cual es el resultado para esa URL. Una vez tenga la predicción, devolverá la respuesta en formato JSON con el resultado.

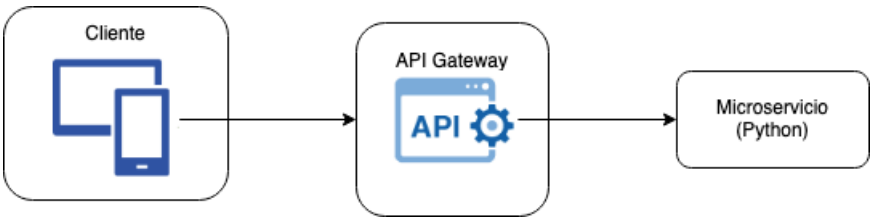


Ilustración 12: Arquitectura del sistema

8. Herramientas y lenguaje de programación

Para la elaboración de este proyecto se han seleccionado las herramientas que están más orientadas a resolver este tipo de tareas y que más facilidad de implementación aportan.

Python (versión 3): Se ha escogido este lenguaje de programación dado que proporciona muchas librerías dedicas al aprendizaje automático y su facilidad de aprendizaje y uso. Se utiliza la versión 3 ya que la versión 2 ha dejado de recibir soporte.

Jupyter Notebook: Es una aplicación web que permite trabajar con el lenguaje Python y se ha escogido ya que permite escribir, documentar y ejecutar código, visualizar datos, realizar cálculos y ver resultados todo desde la misma interfaz de la aplicación. Esto permite la creación de los modelos predictivos y el análisis previo al conjunto de datos y el posterior análisis de métricas y resultados. Es ampliamente utilizado por la comunidad para el aprendizaje automático por ello se ha seleccionado.

Scikit-learn: Es una biblioteca para el aprendizaje automático para el lenguaje que se utiliza Python. Incluye algoritmos de clasificación como máquinas de vectores de soporte o bosques aleatorios.

Numpy/Pandas: Ambos son extensiones para el lenguaje Python y permiten la manipulación y análisis de datos. Son útiles cuando se trata el conjunto de datos para el entrenamiento del modelo.

Seaborn: Igual que las anteriores, Seaborn es una biblioteca para Python dedicada a la generación de gráficos a partir de datos contenidos en listas o vectores. Esta librería facilita la representación gráfica de las métricas que se obtengan en la predicción de los modelos y se utiliza para generar el mapa de calor a partir de la matriz de confusión.

Flask: Es un framework escrito en Python que permite crear aplicaciones web rápidamente y con pocas líneas de código. Se utiliza para crear la parte de los microservicios y permite a los usuarios realizar peticiones al servidor web y realizar consultas de URL. Se ha escogido este framework por su facilidad de uso y lo ligero que es para lo que se pretende utilizar.

Visual Studio Code: Este IDE (entorno de desarrollo integrado) se ha escogido junto a Jupyter Notebook para el desarrollo de la API (interfaz de programación de aplicaciones) que se proporciona a los usuarios. Con esta herramienta se realiza el uso de flask para crear el microservicio que ofrece la posibilidad de realizar consultas de URL.

9. Implementación y pruebas

9.1 Introducción

En la elaboración de los modelos y la obtención de métricas se ha utilizado principalmente la librería de Sklearn, ya que como se ha comentado anteriormente permite fácilmente importar las distintas clases de los distintos modelos que se pretende utilizar. Como la elaboración de los distintos modelos es bastante similar, se ha procedido a documentar en profundidad el proceso en el algoritmo de regresión logística realizado mediante Jupyter Notebooks. No obstante, en cada modelo se incluye los parámetros que acepta la clase de Sklearn que lo implementa y su explicación y la utilidad que proporciona. Distintos modelos aceptan los mismos parámetros, por lo tanto, se ha omitido la explicación redundante de esos mismos parámetros que se hayan comentado anteriormente en otros modelos. Solo se entrará más en detalle si en el modelo en cuestión es realmente necesario modificar este parámetro, de manera que influye en la generación del modelo que se crea.

Después de elaborar los distintos modelos mediante principalmente Sklearn y Jupyter Notebooks, estos modelos se guardarán en disco para que posteriormente puedan ser cargados y utilizados en la API creada mediante el microservicio. Este microservicio esta creado principalmente en Flask y las librerías necesarias para poder realizar el proceso.

9.2 Elaboración de los modelos

Regresión logística (Logistic Regression)

En todos los modelos generados en Jupyer Notebooks es necesario al inicio importar las librerías necesarias para poder generar los modelos. Como se ha comentado anteriormente, se hace uso de pandas para cargar el conjunto de datos, después Sklearn para poder importar la clase que nos permitirá crear el modelo de regresión logística y la clase que nos permite dividir el conjunto de entrenamiento y de test, por último, Seaborn que nos permite visualizar un mapa de calor de la matriz de confusión generada por el modelo predictivo.

```
# Importación de paquetes, funciones y clases.
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
import seaborn as sn
```

Después, se carga el conjunto de datos que utilizamos para entrenar el modelo y en la variable X se almacenan los datos correspondientes a las características que se utilizaran para entrenar el modelo, en esta variable se excluyen las características del dominio y la etiqueta objetivo del entrenamiento. El motivo de que se excluya el dominio que representa la URL que se intenta determinar si es phishing o no es debido a que es una variable que no representa ningún valor significativo en el momento de elaborar el modelo.

```
# Carga del conjunto de datos situado en el mismo directorio.
filename = 'combined_dataset.csv'
data = pd.read_csv(filename, dtype='unicode')
# Se quitan las variables que no formarán parte del entrenamiento.
X = data.drop(['domain', 'label'], axis = 1)
# Se almacena la variable dependiente de predicción.
y = data.label
```

Una vez se tiene el conjunto de datos cargado en memoria, el siguiente paso es dividir este conjunto de datos en la cantidad representativa para el entrenamiento del modelo y la cantidad de muestras para verificar este modelo. Como se muestra en el recuadro de más abajo se observa que se ha dividido el conjunto en un 75% de entrenamiento y un 25% para las pruebas, más adelante en este documento se realiza una comparativa para una división del conjunto diferente a la mostrada.

A continuación, el proceso que continua es la instanciación de la clase que nos permite definir un modelo de regresión logística en este caso. La clase es **LogisticRegression**() y acepta los siguientes parámetros:

- **C**=1.0: Permite aplicar una penalización sobre los valores de los parámetros con el fin de evitar el sobreajuste, que significa minimizar el error entre lo que el modelo predice y la variable dependiente. En el problema que se pretende resolver este valor no influye en la precisión.
- **Class_weight**=None: Permite definir que ciertas características tienen mas peso que las demás. En este caso indica que todas las características tienen el mismo peso.
- **Dual**=False: Es un parámetro orientado a la optimización cuando se utiliza la penalización l2 y el algoritmo liblinear. Como no se utiliza este algoritmo, obtiene el valor de False.
- **Fit_intercept**=True**:** Permite incluir una intercepción en el modelo para la función de decisión. Se ha dejado el valor por defecto ya que no infiere en la precisión del modelo en el aprendizaje automático que se realiza.
- **Intercept_scaling**=0**:** Este parámetro es útil cuando se utiliza el algoritmo de 'liblinear', dado que no se utiliza este algoritmo ya que está mas orientado a conjuntos de datos más pequeños, no se utiliza.
- **L1_ratio**=None: Este parámetro está relacionado con la penalización aplicada, si se utiliza "l2" no aplica.
- **Max_iter**=100: Número máximo de iteraciones para converger a la solución.
- **Multi_class**='auto': Define los datos del tipo de problema que se intenta resolver, si los datos son binarios se escoge "ovr" sino "multinominal" como en este caso al poner "auto".
- **N_jobs**=None: Parámetro utilizado para paralelizar el cálculo del modelo.
- **Penalty**='l2': Este parámetro está relacionado con el algoritmo utilizado e indica la penalización.
- **Random_state**=None: Permite escoger muestras aleatorias del conjunto de datos. No especificado ya que se utiliza en el momento de dividir el conjunto de datos.
- **Solver**='lbfgs': Algoritmo utilizado para optimizar el problema.
- **Tol**=0.000: Este parámetro indica la tolerancia para los criterios de detención, lo que indica a scikit que deje de buscar un mínimo (o máximo) a partir de cierta tolerancia.

- **Verbose**=0**:** Permite mostrar información de registro más detallada por consola. No se recomienda habilitar si no se requiere ya que el algoritmo puede ejecutarse mas lento.
- **Warm_start**=False: Permite reutilizar la solución si se hace la llamada al método fit() en la inicialización.

Cuando es cargado el modelo, se procede al entrenamiento y predicción mediante la división del conjunto de datos realizado.

```
# Se divide el conjunto de los datos para entrenamiento y pruebas (25%).
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)

# Se carga el modelo que se utilizará.
logistic_regression = LogisticRegression(intercept_scaling=0)
# Entrenamiento del modelo.
logistic_regression.fit(X_train, y_train)
# Fase de pruebas con el conjunto de test.
y_pred = logistic_regression.predict(X_test)
```

Cuando se ha realizado el modelo y se le pasa el conjunto de test, se almacena en la variable **y_pred** las métricas obtenidas por el modelo. Mediante esta variable, se procede a elaborar una matriz de confusión que permitirá obtener los siguientes valores:

- Cantidad de verdaderos negativos (no detección, no phishing).
- Cantidad de falso positivos (detección, no phishing).
- Cantidad de falsos negativos (no detección, si phishing).
- Cantidad de verdaderos positivos (detección, si phishing).

```
# Obtención de la matriz de confusión para conocer los VP, VN, FN, FP.
confusion_matrix = pd.crosstab(y_test, y_pred, rownames=['Actual'],
colnames=['Predicción'])
# Se representa el mapa de calor mediante los datos de la matriz de confusión.
sn.heatmap(confusion_matrix, annot=True, fmt="d")
```

Y gracias a la librería de **Seaborn** es posible generar de manera visual el siguiente mapa de calor con los valores correspondiente a la predicción realizada con el conjunto de test y la matriz de confusión. Los valores representados son los expuestos en el listado anterior de arriba abajo y de izquierda a derecha.

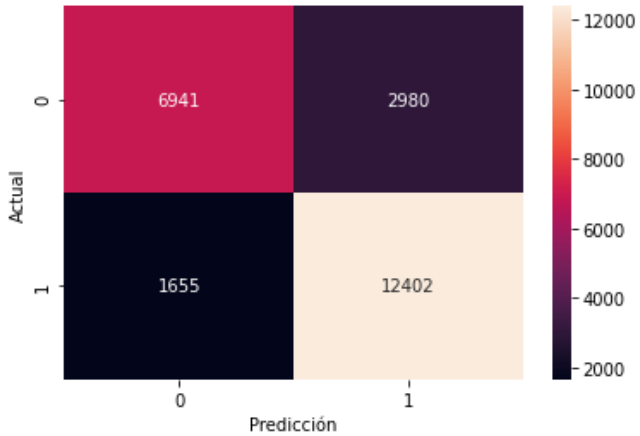


Ilustración 13: Mapa de calor regresión logística

Finalmente se calcula la precisión del modelo mediante el método **score()** que incluye la clase.

```
# Obtención del porcentaje de precisión del modelo elaborado.
print('Accuracy: ', logistic_regression.score(X_test, y_test) * 100)
```

Obteniendo una precisión para este modelo del **80.66%**.

Naïve Bayes

En la elaboración del modelo para Naïve Bayes se ha realizado un proceso similar al anterior, la diferencia está en la clase que se instancia que en esta ocasión es **BernoulliNB()** y que acepta los siguientes parámetros:

- **alpha**=1.0: Parámetro de suavizado. Se ha dejado por defecto ya que no influye en la predicción.
- **binarize**=1.0: Es un umbral que permite asignar boléanos a las características de muestra. Se ha detectado que un mayor umbral perjudica entorno a 5% la predicción.
- **class_prior**=None: Permite definir los mismo que el parámetro **class_weight** comentado anteriormente, pero con probabilidades.
- **fit_prior**=True: Se utiliza para aprender las probabilidades previas de la clase.

Una vez entrenado el modelo y realizado la predicción con el conjunto de datos de test obtenemos los siguientes valores del mapa de calor generado:

- Cantidad de verdaderos negativos: 6553
- Cantidad de falsos positivos: 3368
- Cantidad de falsos negativos: 4410
- Cantidad de verdaderos positivos: 9647

Obteniendo una precisión para este modelo del **67.56%**. De entre los modelos que se han creado para resolver este tipo de problema, Naïve Bayes es el que ha dado un porcentaje de precisión inferior al resto. Esto puede ser debido a que, en el proceso de asignación de porcentajes a las características del conjunto del modelo, no se realiza de manera precisa y no se obtienen buenas estimaciones.

SVM

Para la generación del modelo de SVM se ha instanciado la clase **LinearSVC**() que permite configurar los siguientes parámetros (en general los parámetros que incluye son bastante similares al modelo de regresión logística):

- **C**=1.0
- **Class_weight**=None
- **Dual**=False
- **Fit_intercept**=True
- **Intercept_scaling**=1
- **Loss**='squared_hinge': Especifica la función de pérdida. Los dos valores que permite son 'hinge' y 'squared_hinge' que no influyen en la precisión del modelo.
- **Max_iter**=1000
- **Multi_class**='ovr'

- **Penalty**='l2'
- **Random_state**=None
- **Tol**=0.0001
- **Verbose**=0

Se genera el mapa de calor como anteriormente y se obtienen los siguientes valores:

- Cantidad de verdaderos negativos: 5504
- Cantidad de falsos positivos: 4417
- Cantidad de falsos negativos: 2139
- Cantidad de verdaderos positivos: 11918

Obteniendo una precisión para este modelo del **72.65%**.

Árbol de decisión

En la elaboración del modelo de árbol de decisión se ha hecho uso de la clase **DecisionTreeClassifier()** que incluye unos argumentos bastante variados a los definidos a los modelos anteriores ya que este modelo se basa en la generación de un árbol de decisión (como el nombre del modelo indica). En esta clase podemos encontrar los siguientes parámetros configurables:

- **Ccp_alpha**=0.0: Permite definir una complejidad de coste mínima. Se utiliza como método de poda del árbol para determinar que subárbol tiene una complejidad de coste mayor inferior a este parámetro.
- **Class_weight**=None
- **Criterion**='entropy': Permite seleccionar el criterio en la división de las ramas del árbol. Como se ha documentado anteriormente, es posible escoger entre el índice de Gini o la entropía de la información. Se ha probado ambos valores y la entropía permite obtener un porcentaje algo superior al índice de Gini.
- **Max_depth**=None: Permite definir la profundidad máxima que tendrá el árbol.
- **Max_features**=None: Permite definir el número de características a considerar para la mejor división de las ramas. No se especifica para poder contemplar todas las características disponibles.
- **Max_leaf_nodes**=None: Permite definir un número máximo de nodos hoja en el árbol.
- **Min_impurity_decrease**=0.0: Este parámetro permite controlar el nivel de profundidad del árbol en función de la impureza, que la impureza es el criterio que se utiliza para dividir un nodo mediante el índice de Gini o la entropía. La medida de impureza determina que tan bien se separan las clases.
- **Min_impurity_split**=None: Este parámetro no se utiliza ya que está obsoleto, se utiliza el argumento anterior.
- **Min_samples_leaf**=1: Mínimo de muestra que debe haber para que se cree un nodo hoja. Sirve como punto de división a cualquier profundidad del árbol.
- **Min_samples_split**=2: Indica el número de muestras mínimo para dividir un nodo.
- **Min_weight_fraction_leaf**=0.0: Proporciona una ponderación mínima de la suma total de pesos de las muestras de entrada, para poder estar en un nodo hoja. Cuando el parámetro está a 0 proporciona el mismo peso a todas las muestras.
- **Presort**='deprecated': Este parámetro está obsoleto y no es utilizado.
- **Random_state**=None

- **Splitter=**'best': Permite definir la estrategia utilizada para dividir cada nodo. Admite los parámetros de "best" y "random" donde se ha seleccionado que divida los nodos de la mejor manera.

A partir del mapa de calor generado obtenemos los siguientes valores:

- Cantidad de verdaderos negativos: 9329
- Cantidad de falsos positivos: 592
- Cantidad de falsos negativos: 643
- Cantidad de verdaderos positivos: 13414

Obteniendo una precisión para este modelo del **94.84%**.

En la elaboración de este modelo se pretendía mostrar una imagen del árbol que se genera, dado que genera un árbol de una dimensión elevada y debido a los niveles de profundidad que se crean para llegar a la predicción, no es posible mostrar una imagen del árbol. Por esto motivo, la imagen que genera no permite ver con claridad los distintos niveles que contiene. Se realizó la prueba de añadir un valor al argumento **max_depth** para poder genera una imagen visual del árbol, lo que esto comprometía el modelo generando una precisión entorno al 15% inferior.

Random forest

En la elaboración del modelo de random forest se utiliza la clase **RandomForestClassifier()** que permite unos parámetros bastante similares a los que permite el modelo de decisión en árbol, esto es debido a que este modelo no deja de ser una ampliación del modelo anterior al realizar la predicción en base a un conjunto de árboles de decisión. Se encuentran los siguientes argumentos:

- **Bootstrap**=True: Este valor cuando es 'False' permite indicar al modelo que utilizará todo el conjunto de datos para construir cada árbol. Se ha dejado el valor por defecto ya que da algo de precisión, pero difiere mucho el resultado en ambos valores boléanos.
- **Ccp_alpha**=0.0
- **Class_weight**=None
- **Criterion**='gini': A diferencia del modelo de árbol de decisión, aquí se ha optado a utilizar el criterio de división de las ramas mediante el índice de Gini. Esto es debido a que el índice de Gini permite obtener una precisión algo superior a la entropía de la información.
- **Max_depth**=None
- **Max_features**=None
- **Max_leaf_nodes**=None
- **Max_samples**=None
- **Min_impurity_decrease**=0.0
- **Min_impurity_split**=None
- **Min_samples_leaf**=1
- **Min_samples_split**=2
- **Min_weight_fraction_leaf**=0.0
- **N_estimators**=50: Indica el número de árboles del bosque. Se ha elegido este valor dado que aporta un resultado de precisión bastante similar y óptimo al número de arboles por defecto que son 100.
- **N_jobs**=None

- **Oob_score**=False: Es una técnica de aprendizaje conjunto que permite obtener una mejor aproximación del error del modelo de clasificación generado. Este valor no influye en la precisión del modelo que se calcula.
- **Random_state**=None
- **Verbose**=0
- **Warm_start**=False

De la misma manera que los anteriores casos, se obtienen los siguientes valores del mapa de calor generado:

- Cantidad de verdaderos negativos: 9394
- Cantidad de falsos positivos: 527
- Cantidad de falsos negativos: 498
- Cantidad de verdaderos positivos: 13559

Obteniendo una precisión para este modelo del **95.72%**. De entre todos los modelos generados este es el que ha producido un porcentaje mayor de precisión del modelo.

K vecinos más cercanos

En la elaboración de este modelo aún que el proceso seguido es similar a los modelos implementados con anterioridad, este incluye una ligera diferencia en el proceso de obtención de la precisión del modelo. Ya que este modelo se basa en la elaboración de clústeres de puntos sobre el plano y la distancia entre estos puntos, el uso de la clase de este modelo permite definir un parámetro llamado **n_neighbors** que permite definir la relación entre los puntos para determinar un punto a que clase pertenece. La clase que se ha instanciado para la elaboración de este modelo es **KNeighborsClassifier()** que incluye los siguientes parámetros que difieren ligeramente con los definidos en los otros modelos:

- **Algorithm**=’auto’: Determina el mejor algoritmo automáticamente en base al conjunto de datos de entrenamiento del modelo.
- **Leaf_size**=30: Permite indicar el número de puntos necesarios para converger el algoritmo seleccionado a fuerza bruta. Este valor puede afectar significativamente a la velocidad y memoria requerida en la construcción del árbol que utilizan los algoritmos de ‘ball_tree’ y ‘kd_tree’.
- **Metric**=’minkowski’: Indica la métrica de distancia a utilizar para el árbol. El valor de ‘minkowski’ equivale a la métrica euclidiana estándar.
- **Metric_params**=None: Permite especificar parámetros adicionales a la métrica.
- **N_jobs**=None
- **N_neighbors**=x: Este parámetro es importante en este algoritmo, ya que se da una predicción en base a esta variable, que determina cuantos vecinos es necesario que haya alrededor para especificar que pertenece a una clase o otra. Para comprobar la efectividad del modelo se ha creado una predicción para un rango de enteros.
- **P**=2: Determina si se utiliza la distancia de manhattan (valor = 1) o la euclidiana (valor = 2) para calcular la distancia entre los puntos del plano de coordenadas. En este caso se utiliza la distancia euclidiana pero ambas opciones dan unos valores aproximados.
- **Weights**=’distance’: Determina la función que se utilizará en la predicción. En este caso se utiliza la distancia, que determina que los vecinos más cercanos de un punto tendrán mayor influencia que los lejanos, proporcionando una precisión algo superior.

Como se ha comentado anteriormente, como este modelo puede devolvernos diferentes precisiones del modelo en función del argumento n_neighbors, se ha definido un bucle **for** en un rango de valores que va de K=1 a K=15 y se ha determinado que rango de vecinos devuelve una mejor precisión.

```
k_range = range(1,16)
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=13, weights="distance")
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    print(round(metrics.accuracy_score(y_test, y_pred), 4) * 100)
```

Al ejecutar el bucle del recuadro anterior, se obtienen las siguientes predicciones de modelo:

- K=1 - Precisión: 94.67
- K=2 - 94.45
- K=3 - 95.07
- K=4 - 94.97
- K=5 - 95.07
- K=6 - 95.09
- K=7 - 95.19
- K=8 - 95.26
- K=9 - 95.27
- K=10 - 95.27
- K=11 - 95.28
- K=12 - 95.30
- **K=13 - 95.33** <- Seleccionada.
- K=14 - 95.32
- K=15 - Precisión: 95.28.

Finalmente, como se puede observar, se ha escogido una K correspondiente a **13** que es la precisión superior respecto a las otras Ks. Se ha escogido un limite superior hasta 15 dado que realizando pruebas se ha comprobado que un rango superior es contraproducente produciendo precisiones de modelo inferiores.

Repitiendo una vez más el mismo proceso que en los otros modelos, se obtienen los siguientes valores del mapa de calor:

- Cantidad de verdaderos negativos: 9369
- Cantidad de falsos positivos: 552
- Cantidad de falsos negativos: 567
- Cantidad de verdaderos positivos: 13490

Obteniendo una precisión para este modelo del **95.33%** en función de la K elegida.

9.3 Métricas

A modo de resumen y como comparativa de resultados de los distintos modelos, se ha realizado la predicción de los distintos modelos para una división del conjunto de datos del 90% de entrenamiento y el 10% de test junto al que se había elaborado anteriormente del 75% - 25%. Estos valores corresponden al siguiente número de entradas del conjunto de datos que contiene en total 95910 muestras:

- División 75% - 25%
  - Entrenamiento: 71932 entradas.
  - Test: 23978 entradas.
- División 90% - 10%
  - Entrenamiento: 86319 entradas.
  - Test: 9591 entradas.

Porcentaje de división del conjunto de datos	Algoritmo de clasificación	Matriz de confusión	Precisión del modelo (%)
División 75% - 25% (entrenamiento y test)	Regresión Logística	6941 2980 1655 12402	80.66
	Naïve Bayes	6553 3368 4410 9647	67.56
	SVM	5504 4417 2139 11918	72.65
	Árbol de decisión	9329 592 643 13414	94.84
	Random Forest	9394 527 498 13559	**95.72**
	KNN	9369 552 567 13490	95.33
División 90% - 10% (entrenamiento y test)	Regresión Logística	2231 1739 822 4799	73.29
	Naïve Bayes	2663 1307 1719 3902	68.44
	SVM	2226 1744 818 4803	73.28
	Árbol de decisión	3775 195 236 5385	95.50
	Random Forest	3786 184 193 5428	**96.06**
	KNN	3776 194 199 5422	95.90

Como se puede observar en la tabla (y como se comentó anteriormente) el modelo que da una precisión mayor en ambas divisiones es Random Forest y su predicción basado en un conjunto de árboles. También es posible ver una pérdida de precisión en la regresión logística de aproximadamente el 7% en la división 90% - 10%, por lo que respecta al resto de modelos se produce un incremento entorno al 1% y 2% que no es tan significativo a la pérdida de la regresión logística. Es posible concluir que en general las predicciones de los distintos modelos son elevadas exceptuando Naïve Bayes que es el modelo que peor predicción obtiene seguido por SVM.

Si se comparan los resultados obtenidos con los autores del conjunto de datos, Samuel Marchal, Jérôme François, Radu State y Tomas Engel[1] se observa que los resultados no difieren mucho de lo que ellos obtuvieron en el momento de elaborar el artículo científico. Si bien es cierto que ellos dan un enfoque diferente a su sistema, ya que utilizan una arquitectura basada en Big Data como STORM y utilizan estructuras de datos avanzados basados en un filtro llamado Bloom, también utilizaron los algoritmos de clasificación supervisados como random forest, árboles binarios y SVM entre otros. De entre los algoritmos en los que se coincide ellos obtienen las siguientes métricas en comparativa con las obtenidas:

- Obtuvieron un porcentaje de 95.22% de precisión respecto al 95.72% obtenido en random forest.

- Obtuvieron un porcentaje de 93.81% respecto al 94.84% obtenido en el árbol de decisión.

- Y la gran diferencia más grande es en SVM, ya que ellos obtuvieron una precisión del 86.31% respecto al 72.65% obtenido. Esto puede ser debido a los parámetros configurados en la elaboración del modelo.

Además de estos algoritmos utilizados, en el artículo se describen otros algoritmos que utilizaron y muestran la precisión obtenida. A continuación, se comenta brevemente que aportan estos algoritmos:

- **LMT (del inglés, Logistic Model Tree)**: Es un algoritmo que combina la regresión logística y el árbol de decisión. Indican obtener una precisión del 93.55% con este sistema.

- **C4.5**: Es un algoritmo utilizado para generar un árbol de decisión. Demuestran una precisión del 93.45%. C4.5 presenta las siguientes ventajas respecto a los árboles de decisión:

    - Intenta mitigar el problema del sobreajuste que tienen los árboles de decisión, empleando en el proceso de poda del árbol un solo paso. Dado que crea árboles más generalizados transformando los atributos continuos en nominales.
    - Permite ser utilizando en conjunto de datos discretos y continuos.
    - Es útil en el manejo de datos incompletos.

- **PART**: Es un algoritmo parcial del árbol de decisión basado en listas de decisiones mediante reglas. PART utiliza el algoritmo de C4.5 para construir árboles parciales en cada iteración y así obtener la mejor reglar como hoja. Obtienen una precisión del 90.62%.

- **JRip (RIPPER)**: Es un algoritmo basado en reglas que crea un conjunto de reglas que identifican las clases del conjunto de datos y minimizan la cantidad de errores. El error es definido por la cantidad de ejemplos de entrenamiento mal clasificados por las reglas. Indican una precisión del 89.66%.

---

1. https://hal.inria.fr/hal-01092771/document

9.4 Microservicio de detección de phishing

En esta sección se describe los pasos que se han seguido para implementar el microservicio que proporciona la capacidad de recibir URLs y devolver las predicciones de los distintos modelos implementados anteriormente. En el desarrollo de esta API se ha utilizado principalmente el framework de Python Flask que permite rápidamente construir servicio de manera sencilla.

Previamente a la construcción del servicio, ha sido necesario construir una clase que permite a partir de una URL extraer las características necesarias con las que se han elaborado los modelos, para poder extraer un vector con los valores correspondientes a la URL y poder realizar la predicción posteriormente en los modelos. En la figura siguiente se pueden observar los atributos y los métodos que incorpora:

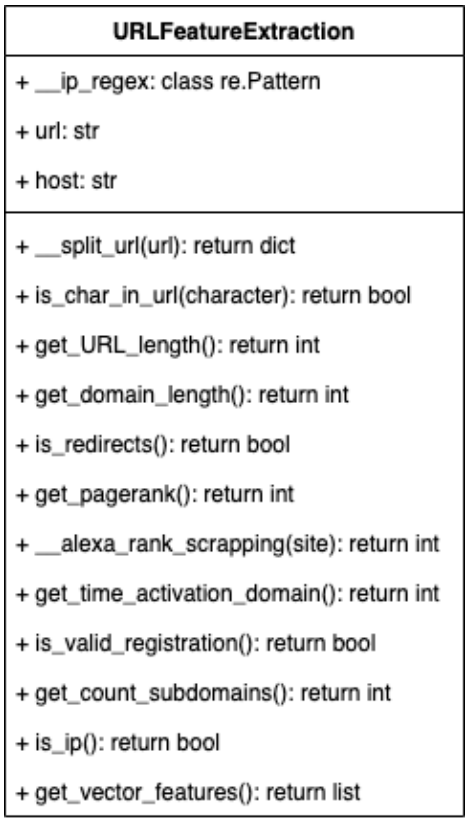


Ilustración 14: Clase de extracción de funcionalidades

Brevemente se explicará que ofrecen los distintos métodos, ya que anteriormente en el apartado 4.4 de este documento se hace referencia a las características necesarias que había que obtener por cada URL para poder realizar los modelos y el nombre de dichos métodos es bastante intuitivo:

- **__split_url(url)**: Es un método privado que permite obtener en un diccionario las distintas partes que componen una URL como el protocolo, host, dominio, etc.

- **Is_char_in_url(character)**: Permite obtener si la URL contiene el carácter pasado por parámetros.

- **Get_URL_length()**: Devuelve el número de caracteres de la URL.

- **Get_domain_length()**: Devuelve el número de caracteres del dominio.

- **Is_redirects()**: Devuelve un booleano especificando si la URL realiza redirecciones.

- **Get_pagerank()**: Este método llama al método privado __alexa_rank_scrapping() para obtener el ranking del dominio de la URL.

- **__alexa_rank_scrapping(site)**: Es un método privado que permite realizar un web scrapping a la página de 'Alexa Rank' para obtener el ranking del dominio en cuestión.

- **Get_time_activation_domain()**: Devuelve un entero con el número de días transcurridos desde la activación del domino.

- **Is_valid_registration()**: Comprueba que la URL es un dominio registrado válido.

- **Get_count_subdoimain()**: Devuelve el número de subdominios que contiene el dominio.

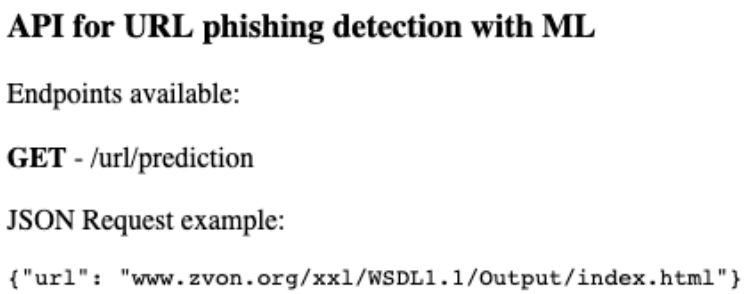
- **Is_ip()**: Devuelve un booleano si la URL incorpora una dirección IP. Utiliza el atributo de __ip_regex que contiene un patrón de regex para la detección de direcciones IP.

- **Get_vector_features()**: Este es el método importante de la clase, ya que permite obtener un vector compuesto con la llamada a los métodos anteriores. Este vector es necesario para poder realizar las predicciones a los distintos modelos ya que es el parámetro que reciben como entrada. El siguiente ejemplo muestra lo que devuelve:

  Ejemplo del vector que genera: [1155, 0, 1, 2000, 85, 0, 0, 1, 20, 2]

Una vez que ha sido creada la clase necesaria para la extracción de características de una URL, se ha creado la API que proporciona dos API endpoints que son los siguientes:

- / o /index: Esta dirección proporciona al usuario una manera sencilla de conocer los recursos disponibles que ofrece el servicio y como interactuar con el servicio.


API for URL phishing detection with ML

Endpoints available:

GET - /url/prediction

JSON Request example:

```
{"url": "www.zvon.org/xxl/WSDL1.1/Output/index.html"}
```

Ilustración 15: Página inicial de la API

- /url/prediction: Es el servicio principal de la aplicación que permite enviar en formato de tipo JSON en el cuerpo del mensaje, una petición de tipo GET de una URL para conocer que probabilidad hay de que sea phishing según los modelos implementados. A continuación, se muestra un ejemplo de la petición y la respuesta que devuelve la API.

  - Ejemplo de petición en formato JSON a la API del microservicio:

```
{
        "url":"http://storicouilcais.it/documenti/Javascript/DropMenuX.js"
}
```

  - Ejemplo de respuesta proporcionado por el servicio una vez a procesado la petición:

```
{
  "predictions": {
    "decision_tree": "phishing",
    "knn": "phishing",
    "logistic_regression": "phishing",
    "naive_bayes": "phishing",
    "random_forest": "phishing",
    "svm": "phishing"
  },
  "scoring": "6/6",
  "url": "http://storicouilcais.it/documenti/javascript/dropmenux.js"
}
```

  - Otro ejemplo de respuesta cuando se envía una petición de una URL que no contiene phishing:

```
{
  "predictions": {
    "decision_tree": "bening",
    "knn": "bening",
    "logistic_regression": "bening",
    "naive_bayes": "bening",
    "random_forest": "bening",
    "svm": "bening"
  },
  "scoring": "0/6",
  "url": https://www.uoc.edu/portal/es/news/actualitat/2020/200-evaluacion-no-presencial-covid19.html
}
```

***Nota**: La implementación del código y las respuestas que devuelven están en inglés con motivo de seguir un estándar al desarrollar código de programación.*

Además de los API endpoints implementados, es necesario destacar dos métodos adicionales que se han desarrollo para poder cargar los ficheros de los modelos generados previamente y realizar las predicciones. Se explican a continuación:

1. Como se muestra en el recuadro de más abajo, la **primera** función (que es llamada antes de arrancar el servicio web por el puerto correspondiente) se utiliza para cargar previamente los ficheros de los modelos del sistema y almacenarlos en un diccionario de Python donde la clave es el nombre del modelo y el valor la instancia del modelo.

2. La **segunda** función, permite a partir de un vector recibido como argumento, realizar una iteración al diccionario correspondiente a los modelos y realizar la predicción correspondiente a cada modelo. Cada predicción a su vez es almacenada en otro diccionario que después será devuelto junto a una puntuación final.

```
IMPLEMENTED_MODELS = {}

# Carga de los modelos del sistema.
def load_models():
    for model in os.listdir('models'):
        if not model.endswith('.sav'):
            continue
        loaded_model = joblib.load('models/' + model)
        model_name = model.split('.')[0]
        IMPLEMENTED_MODELS[model_name] = loaded_model

# Predicciones en base a los modelos cargados previamente.
def make_predictions(vector):
    predictions = {}
    phishing_detection = 0
    for name, model in IMPLEMENTED_MODELS.items():
        prediction = model.predict([vector])[0]
        if prediction == '0':
            predictions[name] = 'bening'
        else:
            predictions[name] = 'phishing'
            phishing_detection += 1
    scoring = str(phishing_detection) + '/' + str(len(IMPLEMENTED_MODELS))
    return predictions, scoring
```

Para finalizar esta sección, se mostrarán algunos ejemplos adicionales donde se han enviado unas URL para comprobar el funcionamiento de la API y la predicción de los modelos generados. Como se comprobará se han seleccionado estas URL ya que el resultado esperado es distinto a lo devuelto por el servicio.

- **Ejemplo 1**: Se envía la URL ‘https://www.ssucbba.org/images/photos/’ -> 1/6 (solo detectado por árbol de decisión) cuando la URL **es phishing**.

- **Ejemplo 2**: Se envía la URL ‘http://www.brandywinedmg.com/dh/DHL/‘ y se obtiene una puntuación de 2/6 (se detecta phishing en árbol de decisión y svm) cuando la URL **es phishing**.

- **Ejemplo 3**: Se envía la URL ‘https://seaborn.pydata.org/examples/scatter_bubbles.html#scatter-bubbles’ y se obtiene una puntuación de 4/6 (solo han devuelto que no es phishing en naïve bayes y svm) cuando la URL **no es phishing.**

- **Ejemplo 4**: Se envía la URL ‘https://www.freecodecamp.org/news/how-i-landed-my-first-developer-job-without-an-application/’ y la API devuelve una puntuación de 4/6 (no phishing en naïve bayes y svm) cuando la URL **no es phishing**.

Estos ejemplos demuestran que, aunque el servicio funcione por lo general de manera adecuada (de las pruebas realizadas), hay situaciones en que los valores que predice no son adecuados y sería necesario realizar una doble validación o revisar el proceso de aprendizaje para dar un servicio más robusto.

10. Conclusiones

En este trabajo se ha definido una solución que permite sin realizar un análisis manual de la URL poder determinar que tan probable es que sea legítima o no. Para ello, como resultado de este proyecto, se ha implementado un servicio que permite enviar una URL y retornar un puntaje según la predicción de los modelos de aprendizaje automatizado que se han elaborado. Este enfoque es muy similar a lo que ofrece el conocido sitio web virustotal, que mediante un listado de motores de antivirus permite determinar que tan probable es que el fichero/dirección sea malicioso. Se ha optado por esta propuesta, ya que aparte de querer ofrecer una solución para la detección de URLs fraudulentas, se creía conveniente (en base a otros estudios que se han analizado) poder ofrecer una solución que pueda ser fácilmente implementado en un entorno real con el impacto menor. Permitiendo obtener que modelos han predicho que el sitio web que se esta enviando es fraudulento y que probabilidad hay de que sea cierto.

Para concluir este proyecto, se realiza un análisis de la planificación que se había estipulado al inicio del trabajo, después se realiza una evaluación de los objetivos, que se ha extraído de ellos y que resultado se ha obtenido. Por último, se realiza una revisión de futuro, donde se revisan algunos puntos donde se podría mejorar, innovar y añadir para que este proyecto pueda aportar una solución que podría ser incluida en entornos reales donde es necesario abarcar este problema tan común en la actualidad.

10.1 Seguimiento de la planificación

En este apartado se realiza un breve repaso a lo que ha sido la planificación del proyecto y sus entregables y tareas adjuntas. A grandes rasgos el cumplimiento de la planificación fue satisfactorio, ya que se pudo entregar todo lo planificado en los plazos establecidos sin ningún retraso. Si que es cierto que la duración de las tareas estipulado no fue muy precisa, dado que había muchas tareas que podían llevarse en menor tiempo y otras requerían de un tiempo superior al estipulado desde el inicio. Opino también que, en mi caso, dado que es un proyecto que contiene una fase orientada al desarrollo y no tan en profundidad a nivel teórico, hubiera sido interesante disponer de mayor tiempo para desarrollar un sistema algo más complejo que pudiera añadir funcionalidades adicionales. En resumen, creo que la planificación ha sido satisfactoria y la mejora de dicha planificación requiere mayor experiencia en nuevos proyectos a lo largo del tiempo.

10.2 Evaluación de objetivos

Para la evaluación de los objetivos, se ha copiado los que se habían especificado en el **apartado 1.2** de este documento, para dar una visión más clara de lo que se ha obtenido por cada punto. A pesar de que esto produce algo de redundancia, aporta de manera más precisa la evaluación de cada objetivo.

OBJETIVO GENERAL

El objetivo principal de este trabajo de máster era realizar la detección de URLs que se utilizan como herramienta de ingeniería social para la obtención de información confidencial de forma fraudulenta, mediante técnicas de software de aprendizaje automatizado.

- El objetivo principal ha sido completado ya que por una parte se ha adquirido el conocimiento de como funcionan los algoritmos de aprendizaje aplicados a problemas de clasificación y por otro lado se ha integrado este conocimiento en la implementación de un sistema que permite la detección de URLs fraudulentas. La herramienta final que se ha construido permite aprovechar los modelos que han sido creados previamente y poder a través de un servicio determinar que probabilidad (según estos modelos) hay de que se este tratando de un caso de phishing. Es cierto que no es un sistema que permite cubrir todos los casos posibles, pero si que da una probabilidad de acierto adecuada. No obstante, hay que remarcar que es un problema complejo de abarcar todas las posibilidades y que requiere ir mejorando cada vez el sistema para obtener precisiones mejores.

OBJETIVOS ESPECÍFICOS

De igual manera que se ha realizado con el objetivo general definido al inicio del proyecto, se realiza una evaluación de cada uno de los objetivos específicos que se habían definido al inicio y su conclusión final.

1. Explorar como el cibercriminal utiliza la URL fraudulenta para hacer creer a la víctima que es una persona de confianza y adquirir su información confidencial.

- Este objetivo junto al siguiente, están estrechamente ligados ya que para comprobar como los ciberdelincuentes actúan es necesario analizar las distintas técnicas que incluyen URL y analizar en profundidad la URL en cuestión. Para ello se aporta información de las distintas técnicas más utilizadas y a la vez efectivas y como son construidas estas URLs para tener mayor porcentaje de éxito en los ataques. La parte negativa de esto es, que los ciberdelincuentes cada día realizan mejoras de sus ataques y buscan más sofisticación y efectividad, para evitar ser detectados por este tipo de sistemas.

2. Analizar cuales son los parámetros y atributos que contiene una URL fraudulenta para determinar como se utilizan en la suplantación de identidad.

- Se han consultado muchos ejemplos de URLs fraudulentas y se han analizado todos los parámetros que están involucrados y cuales son los más comunes utilizados por los cibercriminales.

3. Comprender el funcionamiento y la implementación de un sistema de aprendizaje automático.

- Este objetivo se ha completado, aportando información teórica de como funciona el sistema de aprendizaje escogido. No obstante, no se ha entrado en profundidad a ver a nivel de cada capa de como se diseña un algoritmo desde 0.

4. Analizar los distintos algoritmos de aprendizaje automático existentes para determinar cuales pueden ser útiles para el objetivo general a resolver.

- Este trabajo se ha centrado principalmente a analizar los algoritmos de aprendizaje automático de clasificación supervisado, que son los que están más orientados a resolver el tipo de problema que se presentaba al inicio del proyecto. Si tenemos en cuenta este hecho, se han analizado la mayor parte de los algoritmos supervisados tanto de clasificación como de regresión.

5. Aprender a implementar un sistema de aprendizaje automático mediante el uso de un lenguaje de programación.

- Si bien es cierto que la resolución de este objetivo ha sido facilitada mayormente a las librerías disponibles que proporciona Sklearn, ha sido necesario estudiar el funcionamiento de esta librería y como interactuar con ella. Además, implementar un sistema de aprendizaje automático desde cero no era el objetivo de este proyecto y es algo que requeriría mayor tiempo al asignado a este trabajo.

6. Contrastar a través de los resultados obtenidos con los algoritmos de aprendizaje utilizados si la detección realizada se aproxima o mejora los resultados de otros autores relacionados con el tema del trabajo.

- En visón global se cumple con la detección esperada en relación a otros documentos científicos analizados previamente. Si que es cierto que se esperaba una precisión mayor en los casos de Naïve Bayes y en SVM, no tienen como resultado una baja precisión ya que es superior al 60% pero en el caso de SVM hay otros artículos que muestran una precisión superior al 85%. Hay que tener en cuenta que las funcionalidades extraídas del conjunto de datos difieren de otros artículos a este trabajo y eso puede ser la causa principal. Como añadido, es necesario remarcar la precisión elevada mediante el uso del algoritmo de KNN ya que es superior a otros estudios.

7. Aprender las distintas técnicas de clasificación que utilizan los algoritmos de aprendizaje para comprender mejor como se obtienen los resultados que extraen.

- Este objetivo se a completado con creces, ya que se ha dado tanto una visión teórica de como funciona cada algoritmo que se utiliza, incluyendo las funciones matemáticas (si utilizan). Por otro parte cuando se han implementado a nivel de código, se ha dado en detalle cada parámetro que permite configurar y que utilidad presentan.

10.3 Líneas futuras

Como líneas futuras se cree conveniente mejorar el sistema existente con algunas ideas que sería muy interesante probar y de que manera impactarían en el rendimiento del sistema y en la predictibilidad del servicio. Si bien es cierto que en las conclusiones se ha transmitido que los objetivos han sido cumplidos y que el sistema implementado permite predecir si una URL es legítima o fraudulenta, hay casos como los que se han demostrado anteriormente en los cuales la respuesta emitida por el sistema no presenta una buena estimación y según que vectores pueden vulnerar los modelos predictivos entrenados. Para ello, se proponen una serie de ideas de futuro que sería interesante probar en el sistema:

- Una de las primeras propuestas y que sería interesante añadir al sistema, es la posibilidad de disponer de un motor de base de datos en el que se vayan almacenando las URLs que se hayan ido consultado y poder elaborar un sistema inteligente que pueda ir mejorando en base al conjunto de URLs que se hayan ido procesando. Una base de datos no relacional podría encajar en este tipo de problema, ya que permiten gran rendimiento en grandes volúmenes de datos.

- Otra de las ideas que se han planteado a lo largo de este proyecto ha sido la de aplicar otro tipo de aprendizaje automático, en el que no sea necesario entrenar un modelo previamente, como las redes neuronales y el aprendizaje profundo (en inglés, deep learning). Se escogieron los algoritmos de aprendizaje clasificados supervisado por el tipo de problema que se intentaba resolver, pero sería interesante ver que tal funcionan otro tipo de algoritmos de aprendizaje. Quizá estos algoritmos permitan lidiar mejor con los ataques adversarios, que dichos ataques son definidos de manera específica para burlar los sistemas que utilizan máquinas de aprendizaje.

- También se cree necesario mejorar el funcionamiento que tiene actualmente la API que permite realizar peticiones, permitiendo poder recibir un conjunto de URLs a la vez o un fichero el cual contenga un conjunto de direcciones de sitios web y que permita devolver los resultados para cada una de ellas. Sería muy útil además poder añadir una pequeña interfaz gráfica en la que se puedan realizar peticiones de una manera más amigable para el usuario.

11. Bibliografía

[1] ¿Cómo escribir los objetivos de tu trabajo de investigación? [en línea]. Universidades.cr Blog. <https://www.universidadescr.com/blog/como-escribir-objetivos/>

[2] 20 Phishing Statics to Keep You from Getting Hooked in 2019 [en línea]. Hashedout by The SSL store. <https://www.thesslstore.com/blog/20-phishing-statistics-to-keep-you-from-getting-hooked-in-2019/>

[3] 5 Common Phishing Techniques [en línea]. VadeSecure. <https://www.vadesecure.com/en/5-common-phishing-techniques/>

[4] Phishing Tools & Techniques [en línea]. Infosec. <https://resources.infosecinstitute.com/category/enterprise/phishing/phishing-tools-techniques/>

[5] PhishTank. Join the fight against phishing [en línea]. Phishtank. <https://www.phishtank.com/>

[6] OpenPhish. Timely. Accurate. Relevant Threat Intelligence [en línea]. OpenPhish. <https://openphish.com/index.html>

[7] A Survey of URL-based Phishing Detection [en línea]. Eint Sandi Aung, Chaw Thet Zan and Hayato Yamana. [Paper] <https://db-event.jpn.org/deim2019/post/papers/201.pdf>

[8] Tipos de aprendizaje en Machine Learning: supervisado y no supervisado [en línea]. LUCA Telefonica Data Unit. <https://empresas.blogthinkbig.com/que-algoritmo-elegir-en-ml-aprendizaje/>

[9] Phishing Detection From URLs By Using Neural Networks [en línea]. Ozgur Koray Sahingoz, Saide Isilay Baykal and Deniz Bulut. [Paper] <https://airccj.org/CSCP/vol8/csit89705.pdf>

[10] Phishing URL Detection with ML [en línea]. Towards data science. <https://towardsdatascience.com/phishing-domain-detection-with-ml-5be9c99293e5>

[11] Detection of URL based Phishing Attacks using Machine Learning [en línea]. Sophiya Shikalgar, S.D. Sawarkar and Swati Narwane. [Paper] <https://www.ijert.org/research/detection-of-url-based-phishing-attacks-using-machine-learning-IJERTV8IS110269.pdf>

[12] Phishing Websites Detection Using Machine Learning [en línea]. R. Kiruthiga and D. Akila. [Paper] <https://www.ijrte.org/wp-content/uploads/papers/v8i2S11/B10180982S1119.pdf>

[13] Malicious URL Detection using Machine Learning: A Survey [en línea]. Doyen Sahoo, Chenghao Liu and Steven C.H. HOI. [Paper] <https://arxiv.org/pdf/1701.07179.pdf>

[14] Phishing Website Detection using Machine Learning Algorithms [en línea]. ResearchGate. [Paper] <https://www.researchgate.net/publication/328541785_Phishing_Website_Detection_using_Machine_Learning_Algorithms>

[15] Detection of phishing URLs using machine learning techniques [en línea]. ResearchGate. [Paper] <https://www.researchgate.net/publication/269032183_Detection_of_phishing_URLs_using_machine_learning_techniques>

[16] La regression logística [en línea]. Analytics Lane. <https://www.analyticslane.com/2018/07/23/la-regresion-logistica/>

[17] Naïve Bayes Classifiers [en línea]. GeeksforGeeks. <https://www.geeksforgeeks.org/naive-bayes-classifiers/>

[18] Máquina de vectores de soporte [en línea]. Wikipedia. <https://es.wikipedia.org/wiki/M%C3%A1quinas_de_vectores_de_soporte>

[19] Support Vector Machine – Introduction to Machine Learning Algorithms [en línea]. Towars data science. <https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47>

[20] Decision Trees in Machine Learning [en línea]. Towars data science. <https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052>

[21] Understanding Random Forest [en línea]. Towars data science. <https://towardsdatascience.com/understanding-random-forest-58381e0602d2>

[22] Classification Algorithms - Random Forest [en línea]. Tutorialspoint. <https://www.tutorialspoint.com/machine_learning_with_python/machine_learning_with_python_classification_algorithms_random_forest.htm>

[23] Clasificar con K-Nearest-Neighbor ejemplo en Python [en línea]. Aprende Machine Learning. <https://www.aprendemachinelearning.com/clasificar-con-k-nearest-neighbor-ejemplo-en-python/>

[24] User Guide. Supervised learning [en línea]. Scikit Learn. <https://scikit-learn.org/stable/user_guide.html>

[25] PhishingAndLegitimateURLs [en línea]. Kaggle. [Dataset] <https://www.kaggle.com/kunal4892/phishingandlegitimateurls>