Universidad de Valladolid

Máster universitario

Ingeniería Informática

Trabajo Fin de Máster

`phishingutils`: Estudio sobre herramientas de detección de *phishing* en URLs

Realizado por **Sergio Agudelo Bernal**

✠✠✠

Universidad de Valladolid

20 de septiembre de 2024

Tutor: D. Jesús María Vegas Hernández

Universidad de Valladolid

Máster universitario en Ingeniería Informática

D. Jesús María Vegas Hernández, profesor del departamento de Informática, área de Ciencias de la Computación e Inteligencia Artificial.

Expone:

Que el alumno D. Sergio Agudelo Bernal, ha realizado el Trabajo final de Máster en Ingeniería Informática titulado "PHISHINGUTILS: Estudio sobre herramientas de detección de *phishing* en URLs".

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Valladolid, 20 de septiembre de 2024

V°. B°. del Tutor:

D. Jesús María Vegas Hernández

Resumen

Los ciberataques son cada vez más frecuentes, complejos y pueden tener mayor potencial de causar daños graves en sociedades completas. El *phishing* es uno de ellos, y constituye una de las más grandes vulnerabilidades actuales, siendo objeto de numerosas investigaciones sobre su prevención. La mayoría de ataques ocurren por medio de páginas web, y la identificación de *phishing* por medio de URLs ha demostrado ser una de las técnicas más efectivas, utilizando técnicas de clasificación de *Machine Learning*. Debido a que es requerido realizar la tarea de recopilar un gran volumen de muestras para desarrollar una detección fiable, se evidencia la necesidad de crear herramientas que realicen este proceso automáticamente, además de hacer disponibles conjuntos de datos masivos para investigaciones futuras. Para contribuir con dicha necesidad, se ha creado en este proyecto una biblioteca de Python, que permite gestionar automáticamente el proceso de recopilación y consolidación de conjuntos de datos, a partir de solo algunas configuraciones por parte del usuario. Para demostrar el funcionamiento de esta biblioteca, se realizó una revisión literaria de 30 artículos relacionados y publicados en los últimos 5 años, para obtener el estado del arte de atributos y parámetros para la detección de URLs *phishing*. Como resultados, se cuenta con la versión publicable de la biblioteca desarrollada, un conjunto de datos de 2.500.000 muestras, aproximadamente 20 veces el tamaño de la más grandes fuentes de datos existentes actualmente, también como la documentación de la revisión literaria realizada, propuesta como referencia para investigaciones futuras. Se aplicaron modelos de clasificación al conjunto de datos construido, resultando en valores de precisión de más de 99 %.

Descriptores

framework, *Machine Learning*, página *web*, *phishing*, detección.

Abstract

Cyberattacks are becoming more frequent, complex and can represent a bigger risk of causing great damage in whole societies. Phishing is one type of these attacks, constitutes one of the most important vulnerabilities currently, and is being object of numerous investigations about its prevention. Most attacks are transmitted via web pages, and identifying phishing by its source URL has proved to be one of the most effective detection techniques, accompanied by modeling using Machine Learning classifier algorithms. As it is required to collect - often manually - considerable amount of samples to implement reliable detection, there is an evident demand of tools which handle this process automatically, as well as curated massive datasets that are publicly available for future research. To contribute with this perceived need, this study presents `phishingutils`, a custom-made Python package that allows a way of automatically downloading data sources and creating high-volume datasets for phishing URL detection, by defining model attributes and features using few high level instructions. To demonstrate its applicability, we performed a literature review of 30 articles published in the last 5 years, to document the state of the art in phishing URL detection. As a result, we published the latest stable version of the implemented package, as well as a dataset with 2.500.000 URL samples, approximately more than 20 times the volume of most data sources available currently. We also publish via this document the process and conclusions of the literally review, with the goal of aiding future research in identifying related works. Finally, we tested the resulting dataset with state-of-the-art classifier models, reporting accuracy of more than 99 %.

Keywords

framework, *Machine Learning*, *website*, *phishing*, detection.

Índice general

Índice de figuras

Índice de tablas

Índice de extractos de código

1: Introducción

En un mundo cada vez más dependiente de las tecnologías de la información y comunicaciones, el acontecimiento de ciberataques trae mayores riesgos y posible impacto negativo en las sociedades. Como evidencia, 2023 fue un año en que ocurrieron importantes filtraciones de datos masivas, e incidentes que comprometieron el funcionamiento de entidades de ámbito nacional [27].

Entre las amenazas principales, el *phishing* se ha mantenido como el vector de acceso inicial más común por su inherente relación con la comunicación humana, y por la diversidad de métodos de ataque, los cuales se han vuelto más sofisticados ante la aparición de nuevos avances, como la IA generativa [19]. El medio más usual para transmitir ataques de *phishing* es a través de páginas web, ya que sus URL pueden ser incrustadas en cualquier tipo de mensajería virtual. Por lo anterior, se evidencia la relevancia en contribuir con la investigación de herramientas para la identificación de *phishing* en URLs.

Revisiones literarias recientes indican que cada una de las fuentes necesarias para construir modelos de detección de URLs *phishing* cuenta en su mayoría con máximo 100.000 muestras [48], un tamaño poco representativo respecto al volumen posible de sitios web en todo Internet. Implicaría que de cara a un proyecto de investigación se deban realizar tareas adicionales de integración de diferentes origenes, lo que realmente no debería ser el foco de los esfuerzos. Esto genera una necesidad subyacente en desarrollar y proporcionar herramientas que hagan este tipo de trabajo automáticamente, para disminuir tiempos en tareas posiblemente triviales, y centrarse en el objeto de la investigación.

La propuesta de desarrollo del proyecto nace de la motivación y relevancia en trabajar sobre temas de detección de ataques de *phishing*, el posible impacto que tendría hacer disponibles herramientas que agilicen el proceso de investigación, así mismo de dar visibilidad del estado del arte en este ámbito, como parte de la línea de incentivar la optimización de esfuerzos.

1.1. Objetivos

Se plantea como objetivo global del proyecto desarrollar un corpus y un marco de trabajo para selección y aplicación de diferentes técnicas de *Machine Learning* en el contexto de detección y prevención de *phishing* en páginas web.

Objetivos específicos

- Definir los conjuntos de datos más relevantes en el ámbito académico de la investigación en detección de páginas de *phishing* en funcionamiento actualmente.
- Construir un marco de trabajo (*framework*) para facilitar la recopilación de datos de páginas de *phishing* para uso en investigación.
- Exponer una selección de atributos para aplicar en detección basada en *Machine Learning*, basados en el estado del arte.
- Componer un conjunto de datos curado (*corpus*) mediante uso del *framework* implementado, que incluya las fuentes de datos primarias definidas y la selección de atributos planteada.
- Determinar y caracterizar las principales técnicas de detección de páginas de *phishing* basadas en *Machine Learning*, investigadas en los últimos 5 años.
- Demostrar la integración del conjunto de datos resultante y las técnicas principales de *Machine Learning* mediante casos de uso básicos.

1.2. Estructura del documento

El documento consta de las siguientes partes: en el capítulo 2 se introducen los conceptos relevantes al planteamiento del proyecto, con temáticas como *phishing*, *URLs*, *Machine Learning* y *Python*; en el capítulo 3 se relatan secuencialmente las consideraciones y tareas realizadas a lo largo del desarrollo del proyecto, incluyendo la justificación de decisiones de diseño e implementación; en el capítulo 4 se listan los resultados de la ejecución del proyecto, como el *framework* `phishingutils`, el conjunto de datos, y se discute sobre la percepción del resultado; finalmente en el capítulo 5 se hace una revisión retrospectiva de la experiencia durante el proyecto, concluyendo con temáticas clave tanto positivas como puntos de mejora, adicionalmente se proponen líneas futuras de investigación.

2: Marco teórico y estado del arte

2.1. Acerca del *phishing*

El *phishing* es un ataque de ingeniería social [31], bajo el cual un criminal (atacante) busca apropiarse de la información personal de un individuo (víctima), como su nombre completo, credenciales de usuario, información bancaria; o en ocasiones con el objetivo de diseminar contenido malicioso en una red [21]. En la Figura 2.1 se pueden apreciar que existen diversos tipos de ataques, consolidando al *phishing* como una metodología de ataque amplia, y susceptible a ser transmitida por diferentes medios.

Figura 2.1: Tipos de ataques de *phishing* (adaptado de [37])

Los ataques de *phishing* pueden transmitirse por diferentes medios, por ejemplo:

- SMS;
- Correos electrónicos;

- Páginas web;
- Redes sociales (𝕏, Facebook, etc.).

El tipo más común y estudiado son las páginas web *phishing* [2], esto explicado por la gran dimensión de Internet y porque este método usualmente se superpone con otros, por ejemplo, cuando hay enlaces maliciosos en el cuerpo de un SMS, *tweet* o correo electrónico. Por esta relevancia subyacente se eligió como objeto de estudio las técnicas de detección de URLs *phishing*.

2.2. Metodologías de detección de *phishing* en páginas web

Para inferir si un sitio web tiene contenido de *phishing*, se suele valer de alguno de los siguientes atributos de la página, o de una combinación de ellos [46]:

- URL;
- Contenido de la página: literal, visual, y metadatos (HTML DOM);
- Información sobre el dominio (WHOIS);

Los métodos basados en contenido de la página poseen varias desventajas, como complejidad en accederlo para un volumen elevado de páginas, especialmente en etapas exploratoria y de diseño; también son vulnerables a *exploits* en el navegador [23] u ofuscación de código [1] para ocultar los propósitos maliciosos. Por tanto, se concluye que se encontraría el mejor desempeño y nivel de generalización al seleccionar atributos relacionados con la URL y el dominio de las páginas.

Durante décadas de investigación, varias técnicas de detección de URLs *phishing* han sido propuestas, desde basadas en listas negras o blancas (*blacklist-based* / *whitelist-based*); basadas en heurística (*heuristic-based*); basadas en la naturaleza (*Nature Inspired - NI*); hasta basadas en aprendizaje de máquinas (*Machine Learning based - ML-based*) [6].

Dentro de estas técnicas, las *ML-based* se destacan por su desempeño superior debido a su capacidad de generalizar basado en la información provista al momento del diseño. Eventualmente los ataques se volverían más sofisticados por su capacidad de evadir reconocimiento, y de forma más notoria con la aparición de los modelos de IA generativa (*GenAI*), y su habilidad de producir contenido orgánico a la vista de un ser humano.

La aplicación de estas tecnologías de IA en la detección misma ha hecho frente a estas vulnerabilidades emergentes, y han demostrado una mayor capacidad de detección debido a su habilidad de detectar patrones más complejos, y las funcionalidades que poseen para realizar sintonización fina, siendo adaptables a situaciones específicas [32, 49]. Dado el amplio uso de las técnicas *ML-based* y su evidente relevancia en el ámbito de la investigación, se eligió profundizar en ellas de cara a la ejecución de este proyecto.

2.3. *Machine Learning* (ML)

El aprendizaje máquina, o *Machine Learning* (ML), es un área de la ciencia de la computación, y puede ser definido como el proceso de resolver un problema práctico a partir de la colección de eventos y sus propiedades intrínsecas. Con esta colección de datos existen algoritmos que elaboran un modelo estadístico que es capaz de predecir con cierto grado de confianza, para nuevas observaciones de eventos el resultado esperado.

Para el caso específico de ML aplicado a la detección *phishing* en páginas web, se verá a lo largo de este documento que se utilizan propiedades específicas de las tecnologías web para crear estos modelos estadísticos, que determinan una probabilidad de que el contenido evaluado sea benigno o malicioso.

Aprendizaje supervisado

En el caso del aprendizaje supervisado [13], la colección de datos se compone de **ejemplos etiquetados** $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$, en donde $\mathbf{x}_i$ es el **vector de atributos**, cuya dimensión $j = 1, ..., D$ corresponde al número de atributos distintos en el conjunto de datos, y cada atributo puede ser denotado como $x^{(j)}$, conteniendo siempre el mismo tipo de información para cada vector $\mathbf{x}_i$. La **etiqueta** y_i usualmente expresa sea de forma explícita o implícita, la asociación del vector de atributos a una cierta **clase**, perteneciente a un conjunto $\{1, ..., C\}$ de clases. En casos generales de aprendizaje supervisado suelen aparecer múltiples clases, sin embargo, en el ámbito que se ha planteado sobre *phishing*, constituye un problema binario ya que existen únicamente dos clases: **benigno** y **phishing**.

El objetivo del aprendizaje supervisado es entonces, habiendo tomado una muestra de vectores de atributos etiquetados (conjunto de entrenamiento), producir un *modelo* que permita para una nueva colección de atributos que representen observaciones no antes vistas, predecir el valor de la etiqueta con ciertas métricas de confianza.

En un problema binario como en este proyecto, los indicadores se pueden determinar a partir de los falsos positivos - FP (datos erróneamente clasificados como **phishing**), falsos negativos - FN (datos erróneamente clasificados como **benignos**), verdaderos positivos - VP (datos correctamente clasificados como **phishing**) y verdaderos negativos - VN (datos correctamente clasificados como **benignos**).

Se acostumbra utilizar la **matriz de confusión**, una representación gráfica cartesiana en que se puede apreciar en 4 cuadrantes y usualmente mediante mapas de calor la proporción encontrada de FP, FN, VP y VN, con el resultado esperado de minimizar FP y FN mientras se maximiza VP y VN. Son indicadores principales de rendimiento en el proceso de clasificación [13]:

- **exactitud o *accuracy*** (ratio de predicciones verdaderas respecto al total de predicciones, o $(VN + VP) / (VN + VP + FN + FP)$);
- **precisión o *precision*** (tasa de predicciones verdaderas dentro de las positivas, o $VP/ (VN + VP)$);

- **sensibilidad o *recall*** (tasa de verdaderos positivos, o $VP/\left(VP+FN\right)$).

El ***F1-score*** suele incluirse también, como una aproximación a representar precisión y *recall* bajo el mismo indicador, y es equivalente a:

$$\frac{2\cdot precision\cdot recall}{precision+recall}=\frac{2VP}{2VP+FP+FN}$$

El aprendizaje supervisado es generalmente dividido en dos metodologías [10]: clasificación y regresión. Estas metodologías se diferencian principalmente en que la clasificación es usada para predecir un resultado que represente una variable categórica, mientras que la regresión se aplica a la predicción de un valor numérico real o continuo. En el contexto de este proyecto, como la etiqueta a predecir es claramente categórica, se aplicarán técnicas de **clasificación** en el ámbito del aprendizaje supervisado. Entre los algoritmos tradiciones aplicados a la clasificación se pueden encontrar los árboles de decisión, la regresión logística, las máquinas de vectores de soporte (SVMs), entre otros. También se han desarrollado modelos de aprendizaje en conjunto o *ensemble learning*, en donde se aplican múltiples algoritmos de aprendizaje para obtener mejor rendimiento de lo que se obtendría al usar un algoritmo individual por sí solo.

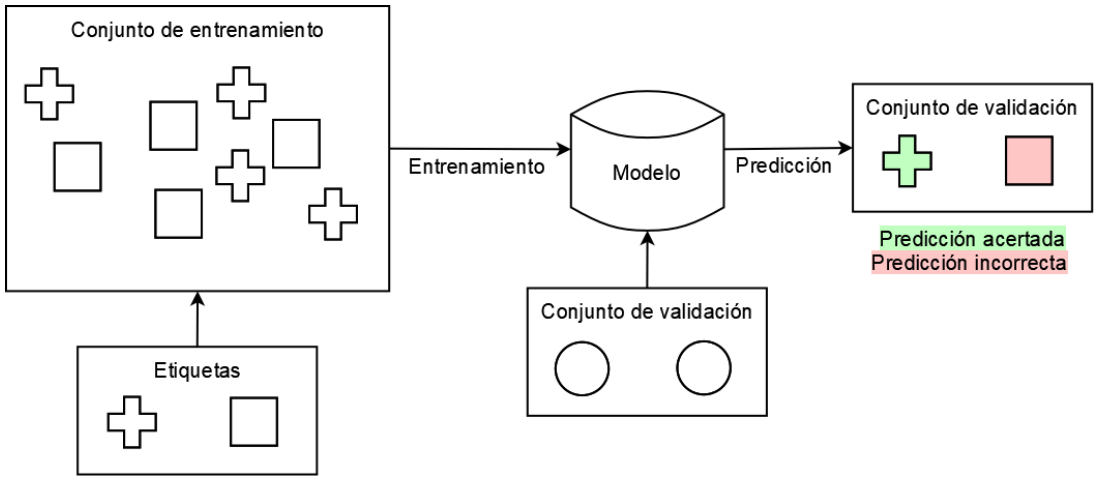

Figura 2.2: Representación esquemática de clasificación en aprendizaje supervisado

Aprendizaje no supervisado

En contraste, bajo el aprendizaje no supervisado [10, 13], la colección de datos se compone de **ejemplos no etiquetados** $\{\mathbf{x}_i\}_{i=1}^{N}$, en donde $\mathbf{x}_i$ continúa representando un **vector de atributos**, que caracterizan a cada observación dentro del conjunto de datos. Al no poseer etiqueta que atribuya a cada vector con su clase correspondiente, el aprendizaje debe estar dirigido a buscar de forma autónoma interrelaciones entre los atributos de la información, e inferir su estructura (extracción de conocimiento).

Las dos técnicas principales de aprendizaje no supervisado son *clustering* y regresión. En el *clustering*, el conjunto de datos se agrupa de acuerdo a similaridades entre los valores de cada uno de sus atributos. En este caso, el algoritmo infiere las posibles clases que podrían categorizar a los datos, con un grado de incertidumbre. La regresión busca encontrar una relación entre una variable dependiente y_i y los vectores de atributos $\mathbf{x}_i$, también llamados variable independiente. La relación encontrada se expresa como:

$$y_i = f(\mathbf{x}_i, \beta) + e_i$$

En donde el el valor de salida y_i se compone del resultado de f, función de la variable independiente con parámetros adicionales β; más un elemento de error e_i inherente al modelo ser una aproximación.

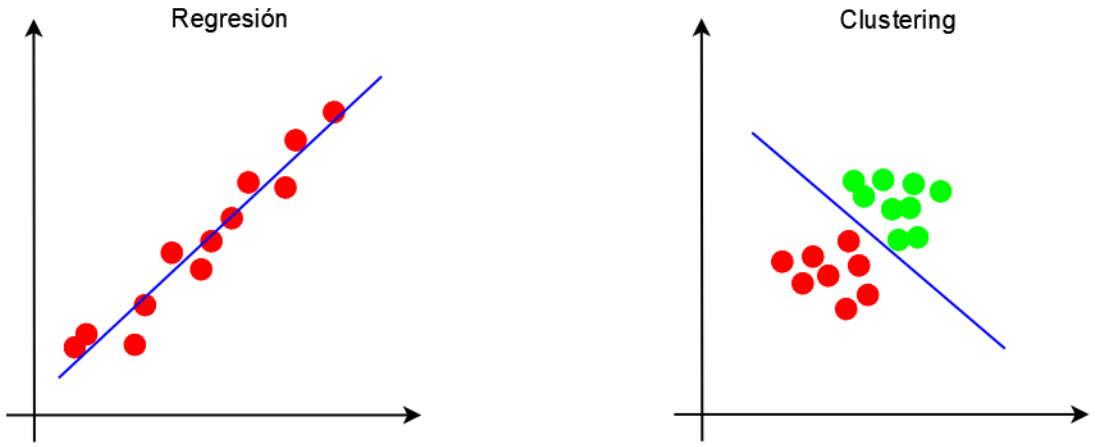

Figura 2.3: Representación gráfica de la regresión y el *clustering* en aprendizaje no supervisado

Redes neuronales y *Deep Learning*

Una red neuronal (NN) [13] es una metodología de *Machine Learning* inspirada en el funcionamiento del cerebro humano. De forma equivalente al aprendizaje no supervisado tradicional, se caracteriza como una función f_{NN} que retorna un valor escalar o vectorial y, a partir de un vector de entrada $\mathbf{x}$ y es una composición de subfunciones $\mathbf{f}_l$. El número l de funciones presentes en esta composición se denominan **capas**, que indican la profundidad de la configuración. Por ejemplo, una red de 4 capas se podría expresar de la siguiente forma:

$$y = f_{NN}(\mathbf{x}) = f_4\left(f_3\left(f_2\left(f_1\left(\mathbf{x}\right)\right)\right)\right)$$

Las funciones de cada capa l se pueden definir como:

$$\mathbf{f}_l(\mathbf{z}) = \mathbf{g}_l\left(\mathbf{W}_l\mathbf{z} + \mathbf{b}_l\right)$$

En donde $\mathbf{z}$ es el vector de entrada de la capa, $\mathbf{g}_l$ se denomina la función de activación, $\mathbf{W}_l$ y $\mathbf{b}_l$ son parámetros dependientes de la tarea a desempeñar en cada capa. Nótese que $\mathbf{z}$ puede ser intepretado como el cambio en dimensionalidad o atributos que va sufriendo $\mathbf{x}$ a lo largo de la red, para conseguir el valor de predicción.

Deep Learning o aprendizaje profundo se refiere al entrenamiento de redes neuronales con más de dos capas que no sean de salida, llamadas capas ocultas. Constituye un acercamiento aún mayor al funcionamiento del cerebro humano, modelando interconexiones más complejas entre neuronas, que pueden ser bidireccionales. Además de las variables presentes en cada capa, introduce la aparición de **hiperparámetros**, cuyo valor afecta al modelo entero en un alto nivel. Algunos hiperparámetros comunes son: tasa de aprendizaje, número de capas ocultas, número de unidades de activación en cada capa, número de iteraciones o épocas, entre otros.

El diseño de modelos usando técnicas de redes neuronales y *Deep Learning* consiste en la selección apropiada de funciones de activación, parámetros y configuración de capas tales que para una aplicación en particular, la predicción del modelo exhiba un desempeño mínimo requerido.

2.4. Algunos conceptos sobre URLs y dominios

Una URL (*Uniform Resource Locator*) [17] es una cadena de texto que se usa para determinar la localización de un recurso único en Internet. Se compone de:

- **Esquema**: Indica el protocolo que el navegador debe usar para solicitar el recurso;
- **Dominio**: Indica cuál servidor Web está siendo solicitado. De derecha a izquierda, consta del dominio de alto nivel (*Top-level Domain - TLD*), Dominio de segundo nivel (*Second-level Domain - SLD*) y de subdominios adicionales. En el ejemplo que se puede apreciar en la Figura 2.4, el TLD sería `.es`, el SLD `uva` y habría un solo subdominio, `inf`, pero en otros ejemplos podría haber más de uno, como en `colab.research.google.com`;
- **Subdirectorio o ruta**: Indica la ruta del recurso específico que se solicita, dentro del servidor Web;
- **Parámetros**: Propiedades adicionales que son provistas al servidor Web para ejecutar la solicitud;
- **Ancla (*anchor*)**: Indica alguna parte específica del documento que se va a acceder, usualmente el navegador se desplaza automáticamente hasta ese lugar, como un atajo para que sea lo primero que el usuario visualice.

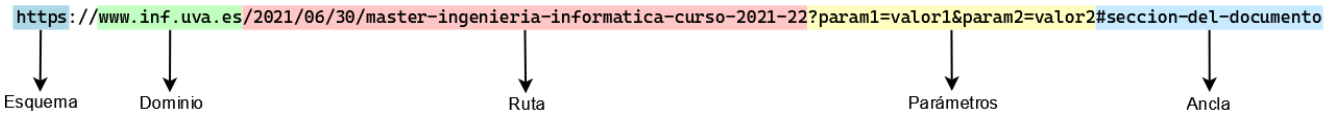

Figura 2.4: Representación gráfica de las partes de una URL

En 2003 fue introducido el estándar RFC 3492, el cual permite establecer una sintaxis de codificación dentro de los nombres de dominio, para representar caracteres que actualmente no son compatibles para uso dentro de un dominio, por ejemplo caracteres del Unicode extendido como letras con acentos, emojis, o caracteres de otros alfabetos o logogramas. Esto permite, por ejemplo, que ahora se pueda digitar en la barra de dirección de un navegador el sitio web `españa.es` y que constituya una dirección válida, a pesar de que el caracter ñ no esté permitido dentro del nombre de un dominio. Esto es debido a que aquella es apenas una representación del dominio real, que sería `xn-espaa-rta.es`. Esta codificación `xn-espaa-rta` es denominada *punycode*, y se ha vuelto parte de las prácticas que los atacantes usan para engañar a los usuarios, haciendo pasar una representación casi idéntica de la página que esperan acceder.

La IANA (*Internet Assigned Numbers Authority*) se encarga de coordinar globalmente, entre muchos otros temas, el registro de nombres de dominio. Provee un servicio llamado WHOIS, que es ampliamente usado para solicitar estas informaciones de dominio, que incluyen datos como fechas de creación y actualización del registro de domino, información de contacto de sus propietarios técnicos y de negocio, direcciones IP asociadas, entre otros aspectos. En el sector privado o en comunidades de desarrolladores, existen varias herramientas *wrapper* o bases de datos consolidadas que permiten tener acceso a una "foto"(*snapshot*) del registro que reportó la IANA en algún momento determinado, para hacer consultas de altos volúmenes de dominios o recurrentes de forma más sencilla.

2.5. Bibliotecas de *Machine Learning*

El lenguaje de programación elegido para desarrollar el proyecto fue Python, debido a la disponibilidad amplia de bibliotecas de manipulación de datos y *Machine Learning*, a pesar de posibles desventajas en rendimiento. Dentro del ecosistema de bibliotecas disponibles para este lenguaje, el enfoque en la elección de la herramienta a utilizar fue sobre cuál ofrecía mayor número de algoritmos, ya que está relacionado a los objetivos del proyecto un primer acercamiento a la comparativa de las principales técnicas de ML en la detección de URLs *phishing*.

En la Tabla 2.1 se resume la comparación realizada, bajo las documentaciones de las versiones más recientes de `scikit-learn` [38] y `MLlib` en `Spark` [9] al momento de elaborar este trabajo de fin de máster. Se puede observar la gran diferencia de algoritmos, en aprendizaje supervisado 22 en `scikit-learn` respecto a 3 en `MLLib`, y en aprendizaje por conjuntos 9 en `scikit-learn` contrastado a 7 en `MLLib`.

Fueron también hechas consideraciones de desempeño, ya que en ese escenario `Mllib` claramente llevaría la ventaja por su aplicación en computación distribuida, sin embargo, en la práctica bastó con implementar los desarrollos en un cómputo individual por medio de las bibliotecas no distribuidas de Python, sin inconveniente alguno.

Categoría	Clases	# de algoritmos disponibles
Aprendizaje supervisado	`sklearn.gaussian_process`	1
	`sklearn.linear_model`	8
	`sklearn.naive_bayes`	5
	`sklearn.neighbors`	2
	`sklearn.svm`	4
	`sklearn.tree`	2
	`pyspark.ml.classification`	3
Aprendizaje por conjuntos	`sklearn.ensemble`	9
	`pyspark.ml.classification`	7

Tabla 2.1: Comparación de algoritmos disponibles para ML supervisado y por conjuntos, basada en `scikit-learn` 1.5.2 y `MLlib` en `Spark` 3.5.2.

En relación a las bibliotecas de *Deep Learning*, dado que el enfoque es introductorio para demostrar la posibilidad de uso del conjunto de datos diseñado, se acogió a `keras` como herramienta de uso, debido a su exposición previa durante el desarrollo del máster, y por la amplia disponibilidad de recursos desarrollados con esta.

2.6. Consideraciones de ámbito general sobre Python

Para evitar limitaciones de recursos, sea en un ordenador personal o en un servidor físico comunitario provisto por la Universidad, se optó por usar el servicio de Colab de Google, que es un servicio alojado de cuadernos de Jupyter, que no requiere configuración previa y permite acceder por demanda a recursos de altas prestaciones como GPUs y TPUs para procesamiento de grandes escalas de datos. Actualmente se observó que tienen una oferta gratuita de acceso a un cómputo básico, pero durante la experiencia se encontró preferencia en adquirir una suscripción, que funciona bajo modo *pay-as-you-go*, y otorga mensualmente una cuota de unidades de cómputo que son gastadas con una tasa fija horaria al acceder a cierto tipo de *runtime*, o configuración de infraestructura, variando la tasa respecto a las prestaciones del *runtime* elegido. Tras prueba y error, los desarrollos implementados se ejecutaron con el **runtime TPU** ofrecido en Colab.

Adicionalmente, como uno de los objetivos señalados fue la implementación de un *framework* para apoyar la creación de modelos predictores, se encontró importante asimilar conceptos de (1) mejores prácticas en elaboración de código, así como (2) documentación

apropiada compatible para visibilizar la forma de uso del *framework* en menús contextuales o cuando el desarrollador ejecute comandos de ayuda sobre algún método en particular.

Sobre el primer numeral, se usó el analizador de código estático **Pylint** [25], para identificar de forma automática errores o divergencias al estándar de elaboración de código, en donde errores notables durante el desarrollo fueron por ejemplo en la nomenclatura de funciones, clases y variables; sentencias sin efecto en la ejecución; o mal uso de variables globales.

Respecto al segundo numeral, se tomaron como referencia el **PEP 257 - Docstring Conventions** [20], el cual hace parte de las *Python Enhancement Proposals* (PEP), iniciativas de mejora para Python, en este caso relacionada con definir convenciones y semántica asociada con las *docstrings* de Python. Estas *docstrings* son el encabezado de módulos, funciones, clases y métodos, los cuales sirven para documentar el propósito del código desarrollado, pero así mismo poblar las referencias mostradas al ejecutar comandos de ayuda sobre uso de elementos de una biblioteca en Python. Complementando la especificación de la PEP, se tomó como inspiración el estilo de documentación del código fuente en el proyecto NumPy [28].

Como guía de creación de bibliotecas empaquetadas y definición de dependencias, se utilizaron las especificaciones **PyPA** (*Python Packaging*) [34]. La revisión de este aspecto permitió la creación de un paquete estándar para publicación a usuarios, como funciona con el resto de módulos estándar o de uso masivo por la comunidad en Python.

3: Desarrollo del proyecto

A partir de la información recopilada en el marco teórico y las consideraciones realizadas a lo largo de su obtención, posteriormente se describe la serie de pasos que se llevaron a cabo para la implementación de los objetivos del proyecto, la justificación de las decisiones tomadas y algunas otras reflexiones adicionales. Vale la pena mencionar que la realización de la revisión literaria en sí debe ser considerado como una parte del desarrollo del proyecto y un resultado como tal por su relación con los objetivos propuestos.

3.1. Revisión literaria

Como los resultados del proyecto planteado tienen una dependencia directa en hacer una revisión estructurada del estado del arte, se partió por encontrar un guía conciso que indicase pasos claros para reunir la información literaria previa a la ejecución. Así, basado en [15], se determinaron las etapas clave para revisión de la literatura. En la Figura 3.5 se expone de forma esquemática el procedimiento y la reducción de alcance que significó. Vale la pena mencionar que la limitación autoimpuesta sobre el volumen de artículos captados fue mediante la duración de la actividad (2 semanas), dado que esta tarea puede ser tan amplia como los recursos (tiempo, personal) lo permitan, y estableciendo un límite temporal permitió dar consecución oportuna a las actividades posteriores dentro del desarrollo del proyecto.

1. **Establecer el procedimiento a usar para seleccionar la el trabajo previo objeto de revisión.**

   Se planteó una revisión del estado del arte con el objetivo de identificar cuáles son las técnicas principales de detección de URLs *phishing*. Esta búsqueda se realizó en las principales publicaciones a las que se tiene acceso público o mediante la Universidad de Valladolid (ACM Digital Library, IEEE Xplore, ScienceDirect y arXiv), de artículos bajo la frase clave *"phishing url detection"*.

2. **Establecer la ventana temporal que cubrirá la revisión.**

   Se estimó que se demarcaría de forma adecuada la actualidad de los trabajos a analizar si fueron publicados con antigüedad no mayor a 5 años. Durante el tiempo dedicado a la búsqueda inicial, se recopilaron cerca de 200 artículos.

3. **Definir los elementos de cada trabajo que serán examinados en la revisión.**

   De acuerdo a la conclusión de la sección 2.2, habiendo reducido el alcance en el ámbito de metodologías de detección a las *ML-based*, se filtró la lista inicial de 200 artículos, mediante revisión de las secciones de abstract y metodología, pues se consideró suficiente y óptimo en cuanto a inversión del tiempo, ya que en esos apartados se obtendría el contexto del artículo y su relevancia al proyecto. Como resultado se documentó un consolidado de 30 trabajos relacionados, cuyo contenido fue verificado que fuese acerca de al menos un tipo de técnicas aplicadas de interés en detección de URLs *phishing*, o alternativamente un artículo de revisión literaria sobre temáticcas de *Machine Learning*.

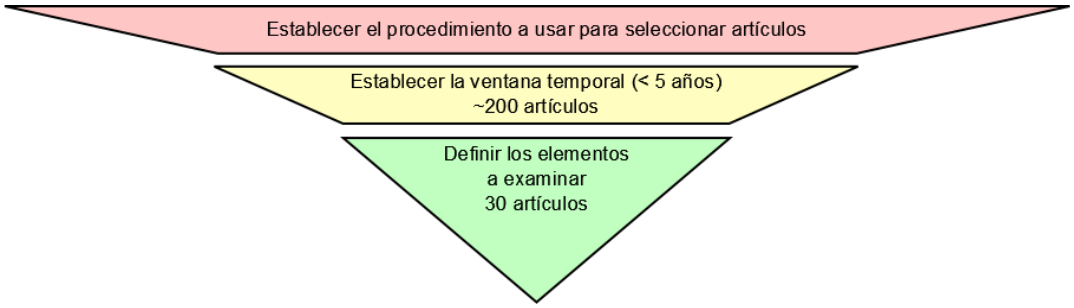

Figura 3.5: Representación esquemática del procedimiento de revisión literaria

Se evidenció un trabajo con alcance similar a este proyecto, y se trata de PhishBench, publicado por Zeng y col. [50], cuyo propósito es dotar al investigador de un *framework* de *benchmarking* en donde se permitiera comparar distintas técnicas de ML, y cargar dinámicamente atributos y métricas.

Se encontraron revisiones sistemáticas como [2, 8, 37, 46, 48] en donde se listan aspectos importantes como los conjuntos de datos principales, las técnicas de clasificación más usadas y una comparativa de desempeño (ver Tabla 3.2). Sobre estos artículos se puede destacar que hay una relevancia continuada en el estudio de sitios web, en específico URLs, y que entre los conjuntos de datos más citados se encuentran PhishTank [33], conjuntos propietarios y de aprendizaje para el caso de URLs maliciosas; y clasificaciones de las páginas web más usadas para el caso de las URLs benignas, como el Alexa Ranking, que actualmente fue descontinuado por Amazon. En cuanto al volumen de datos, se reporta que la mayoría de conjuntos cuentan con entre 1000-99999 muestras. Respecto a las técnicas de detección, la mayor parte aplica técnicas de clasificación sea por árboles de decisión o conjuntos de ellos, SVMs y algunos casos de uso con redes neuronales.

Trabajos como [3–7,11,12,14,18,21,22,29,31,35,40–43,45,47,51] presentan una selección documentada de atributos y de técnicas de aprendizaje supervisado, principalmente clasificadores. La categorización y selección de técnicas y atributos utilizados en el proyecto será propiamente expandida en la sección 3.4, sin embargo, vale la pena mencionar que existen técnicas y atributos en común entre los diferentes artículos, lo que puede implicar que hay ciertas prácticas ‘ estándar en este ámbito. Estos atributos seleccionados se basaron principalmente en características de la URL, pero también se consideraron dimensiones como la información del dominio origen y metainformación contenida en el código fuente en HTML. Estos modelos que se basaron en diferentes dimensiones se denominan *híbridos*, y son ampliamente discutidos debido a su potencial mejor desempeño por cuestiones de redundancia y por hacer frente limitaciones de cada tipo de atributos.

Trabajo relacionado	Algoritmo	Exactitud	Precisión	*Recall*
Alsarhan y col. [8]	*C.45/j48*	94.6 %	94.6 %	94.6 %
	Árbol de decisión	92.1 %	92.1 %	92.1 %
	Naïve Bayes	66.0 %	92.8 %	80.2 %
Safi y col. [37]	*Random Forest*	97.7 %		
	SVM	98.5 %		
Abuadbba y col. [3]	*Regresión logística*	96.1 %	96.7 %	96.3 %
	Árbol de decisión	95.4 %	95.6 %	95.2 %
	KNN	91.6 %	94.3 %	88.4 %
	Naïve Bayes	89.6 %	92.3 %	86.3 %
	Random Forest	94.3 %	93.8 %	93.1 %
	SVM	93.1 %	93.3 %	92.6 %
Adane y col. [4]	*Gradient Boosted Trees*	97.16 %	97.08 %	98.17 %
	Random Forest	96.54 %	96.96 %	95.02 %
Hajizada y col. [3]	*Regresión logística*	92.55 %		
	Random Forest	98.25 %		
	Naïve Bayes	90.4 %		

Tabla 3.2: Resumen comparativo de desempeño de algunos algoritmos de *Machine Learning* aplicados a la detección de URLs con *phishing*, según evidenciado en el estado del arte y con sus métricas reportadas

Finalmente, vale mencionar algunos trabajos que se destacaron por aplicar metodologías novedosas y diferentes a las mapeadas durante la revisión literaria. Kim y col. [23] propone una metodología basada en grafos, construidos a partir de propiedades de URL, dominio y dirección IP de las muestras. Abdelnabi y col. [1] presentan una técnica de detección basada en similaridad visual de páginas web, aplicando un triplete de redes neuronales convolucionales (CNN). Valentim y col. [44] proponen la utilización de redes adversariales generativas (GANs) para apoyar la detección de *phishing squatting*, práctica en la que el

atacante utiliza dominios casi idénticos a unos benignos para dificultar la distinción de su sitio web como malicioso, esto afectando tanto a personas como algoritmos.

3.2. Identificación de las fuentes de datos

En este apartado se describen los origenes de datos principales para extraer listados de URLs benignas y con contenido de *phishing*. Se mencionan particularidades de cada una de estas fuentes, y reflexiones relacionadas con su papel en el proyecto.

URLs con contenido de *phishing*

De acuerdo a los hallazgos sobre conjuntos de datos, documentados en la sección 3.1, la mayoría de trabajos utiliza PhishTank [33], una herramienta con el fin de recopilar de forma colaborativa información sobre páginas *phishing*, e integrarlas en aplicaciones defensivas mediante API, de forma gratuita. El proyecto cuenta con información sobre cerca de 50.000 páginas confirmadas como *phishing*, y atributos relevantes como si la página actualmente está operativa, direcciones IP asociadas, y la entidad de qué empresa busca suplantar, en casos aplicables. Desafortunadamente, en 2020 se deshabilitó en PhishTank la posibilidad de registrar cuentas nuevas tanto para reportar páginas maliciosas como para acceder a la API, por lo que actualmente solo se cuenta con un enlace estático obtenido en su sitio web, para descargar el conjunto de datos actualizado con los reportes más recientes de los usuarios que aún poseen cuenta.

OpenPhish [30] ofrece una solución con cualidades similares, aunque dirigido a ámbitos empresariales, contando con diferentes categorías de suscripciones pagas para acceder a su conjunto de datos. El proyecto permite solicitar acceso gratuito por motivos educativos durante un período limitado, sin embargo, es sujeto a aprobación mediante solicitud directa vía correo electrónico emitido desde el dominio de una empresa o centro educativo. Debido a la posibilidad de un acceso menos restrictivo, aunque reducido a la URL de 500 páginas web confirmadas como maliciosas, se planteó la adopción del conjunto de datos compartido de forma gratuita en el sitio web de OpenPhish.

En tercera instancia, se encuentra Phishing.Database [24], un repositorio de dominios y páginas web *phishing*, sobre el cual se puede contribuir adicionando nuevas páginas maliciosas o solicitando remover sitios que sean confirmados benignos. En la actualidad cuenta con más de 1 millón de páginas *phishing*, representando la mayor parte de muestras encontradas de sitios maliciosos.

URLs con contenido benigno

Para balancear la representación de sitios web *phishing* y poder construir metodologías que permitan discriminarlos de posibles páginas benignas, es necesario obtener un conjunto de datos con un número comparable al mencionado anteriormente, y para ello se puede contar con proyectos como Cisco Umbrella Popularity List [16] o The Majestic Million [26],

los cuales reportan una clasificación de millones de dominios por orden de número de accesos en navegación web. Mediante verificación y supuesto de correlación entre popularidad y fiabilidad, estos conjuntos de datos permiten incluir una muestra significativa de páginas benignas.

Consolidación del conjunto de datos

En resumen, se consideraron los conjuntos de datos listados en la Tabla 3.3, consolidando 2,5 millones entre ambas clases de páginas (debido a limitaciones temporales y a evidencia de duplicados en los datos), pero se destaca que potencialmente se podría configurar un conjunto de datos mayor, con aproximadamente 1,5 millones de páginas *phishing* y 2 millones de páginas benignas. Debido a la evidencia encontrada sobre el volumen alcanzable de las fuentes de datos por separado, y a las restricciones de acceso de cada una de ellas, se nota la utilidad en construir un marco de trabajo que permita una inclusión más transparente y simultánea de la información de URLs, para centrar el esfuerzo no en importarlos sino en analizarlos.

Conjunto de datos	Tipo	Volumen	Licencia de uso
PhishTank	Páginas *phishing*	∼50.000 registros	CC Attribution-ShareAlike 2.5 License
OpenPhish	Páginas *phishing*	∼500 registros	Licencia no estándar
Phishing.Database	Páginas *phishing*	∼1.000.000 registros	MIT License
Cisco Umbrella	Páginas benignas	1.000.000 registros	Licencia no estándar
Majestic Million	Páginas benignas	1.000.000 registros	CC Attribution 3.0 Unported License

Tabla 3.3: Resumen de conjuntos de datos, por tipo, volumen y licenciamiento

3.3. Selección de atributos

Correspondiente a las técnicas supervisadas nombradas en la sección 2.3, se recogieron los 18 atributos relativos a URL más comúnmente usados en los 30 artículos revisados (incluyendo la propia URL, su fuente y extracción de información de dominio, que no son atributos de clasificación como tal, pero se incluyeron para permitir la trazabilidad de cada información a su origen). La selección de atributos se puede encontrar resumida en la Tablas 3.4 y 3.5.

#	Nombre	Descripción	Tipo de dato
1	`url`	URL textual de la muestra	Texto
2	`source`	Fuente de la URL (según Tabla 3.3)	Texto
3	`label`	Etiqueta de la URL, `benigna` o `phishing`	Texto
4	`url_length`	Longitud de la URL en caracteres	Entero
5	`starts_with_ip`	¿La URL inicia con una dirección IP?	Verdadero o Falso
6	`url_entropy`	Entropía de la URL	Decimal
7	`has_punycode`	¿La URL contiene al menos un caracter *punycode*?	Verdadero o Falso
8	`digit_letter_ratio`	Ratio entre dígitos y letras en la URL	Decimal
9	`dot_count`	Conteo de puntos en la URL	Entero
10	`at_count`	Conteo de arrobas (@) en la URL	Entero
11	`dash_count`	Conteo de guiones (-) en la URL	Entero
12	`tld_count`	Conteo de aparición de dominios de alto nivel en la URL	Entero
13	`domain_has_digits`	¿La URL raíz contiene al menos 1 dígito?	Verdadero o Falso
14	`subdomain_count`	Conteo de subdominios en la raíz de la URL	Entero

Tabla 3.4: Selección inicial de atributos de URL, a usar en tareas de creación de modelos de clasificación (atributos 1-14)

#	Nombre	Descripción	Tipo de dato
15	`nan_char_entropy`	Entropía de caracteres no alfanuméricos (NAN)	Decimal
16	`has_internal_links`	¿El subdirectorio de la URL contiene enlaces?	Verdadero o Falso
17	`whois_data`	Registro de información de dominio obtenido en WHOIS	Objeto
18	`domain_age_days`	Edad del dominio en días, de acuerdo con WHOIS	Decimal

Tabla 3.5: Selección inicial de atributos de URL, a usar en tareas de creación de modelos de clasificación (atributos 15-18)

Las entropías de URL [5] y de caracteres no alfanuméricos (NAN) [11] buscan representar la heterogeneidad de caracteres en una URL, aplicando el concepto de entropía en la teoría de la información, también llamado entropía de Shannon.

$$H(X) := -\sum_{x \in X} p(x) log(p(x))$$

En donde X representa la cadena de texto que compone la URL, x cada uno de los caracteres únicos en ella, y $p(x)$ la probabilidad de que el caracter x esté dentro de la cadena. $p(x)$ podría ser expresado como:

$$p(x) = \frac{conteo(x)}{longitud(X)}$$

Siendo $conteo(x)$ el número de veces que x aparece en la cadena X, y $longitud(X)$ la longitud de la URL X en caracteres.

La aplicación de la entropía en este problema de detección se hace bajo el supuesto de que una URL benigna suele ser definida para que sea legible por una persona, por tanto contendría en principio una homogeneidad de caracteres, siendo la mayoría de ellos alfanuméricos. Una URL con contenido de *phishing* busca imitar URLs benignas, pero como los atacantes no tienen acceso a los dominios verificados del contenido que buscan imitar, agregan caracteres especiales con la esperanza de que las víctimas no los determinen; o en otros casos el atacante simplemente tiene acceso a dominios de baja reputación, que acostumbran ser combinaciones aleatorias de caracteres.

El resultado esperado sería que una URL benigna exhibiera bajas entropía de URL y de caracteres NAN.

3.4. Selección de técnicas candidatas de *Machine Learning* a probar

Como fue introducido en la revisión literaria, se categorizaron los trabajos correspondientes a propuestas de aplicación de modelos y selección de atributos, y se obtuvieron los algoritmos que se usaban más frecuentemente en estos trabajos. El resultado se expresa en la Tabla 3.6, en donde se puede apreciar de forma agrupada entre tipos de técnicas principales, los algoritmos en orden de más a menos usado, que aparecían documentados en al menos 2 artículos.

Tipo de técnica	Algoritmo	Trabajos relacionados
Clasificadores	*RandomForest*	[4, 5, 11, 18, 22, 42, 45]
	Gradient Boosting Classifier	[4, 5, 40, 42]
	Naïve Bayes	[8, 11, 22, 40]
	Support Vector Machine (SVM)	[11, 40, 42]
	C.45/j48	[8, 18]
Redes neuronales	*Convolutional Neural Network (CNN)*	[12, 41, 44, 45, 47, 51]
	Long short-term memory (LSTM)	[12, 44, 47]
	Recurrent Neural Network (RNN)	[47, 50]

Tabla 3.6: Listado de metodologías de *Machine Learning* más relevantes en el ámbito de detección de páginas *phishing*, de acuerdo con la revisión del estado del arte

Se destaca de este listado que la mayoría de aplicaciones usan clasificadores a pesar de ser técnicas asentadas en el ámbito del *Machine Learning*, resaltando principalmente los algoritmos de aprendizaje en conjunto. La decisión de nombrar a los módulos `sklearn` y `keras` como las mejores opciones para implementar casos de uso de detección de URLs tuvo que ver en gran medida con que tales módulos tienen implementaciones de todos los algoritmos listados, a diferencia de módulos alternativos. Vale la pena anotar que el algoritmo *C.45/j48* no está implementado como tal en `sklearn`, sin embargo, como mencionan en su documentación [39], el algoritmo que implementan (*CART - Classification and Regression Trees*) es muy similar a *C.45/j48*.

3.5. Implementación del *framework* `phishingutils`

Siguiendo el lineamiento propuesto en los objetivos, se planteó la creación de un **módulo de Python**, o sea, una jerarquía de clases y métodos que permitieran definir la creación de un conjunto de datos mediante alguna expresión de alto nivel. Este módulo posteriormente fue hecho disponible como un **paquete de Python**, colección del módulo, dependencias, instrucciones de instalación y otros guías contenidos en una colección de ficheros comprimidos en un solo archivo, para permitir instalación rápida y replicable en otros ordenadores.

Como ambiente de desarrollo fue usado Visual Studio Code, y como guías de desarrollo del módulo y paquete, fueron tomadas las consideraciones documentadas en la sección 2.6.

Se enumeran también a continuación los módulos de Python estándar y terceros usados, una pequeña descripción de ellos y su utilidad dentro del *framework*.

- `collections`: Módulo estándar que provee tipos de dato adicionales, utilizado para implementar la función de entropía;
- `concurrent`: Módulo estándar que permite ejecutar tareas paralelas, utilizado para hacer frente a limitaciones en la construcción de las columnas `whois_data` y `domain_age_days` del conjunto de datos;
- `dateparser`: Módulo implementado por terceros, permite reconocer fechas dentro de un texto, soporta muchos formatos existentes así como lectura de fechas relativas o en lenguaje natural. Usado para conseguir reconocer los formatos de fecha evidenciados en la fuente WHOIS, para construcción de la columna `domain_age_days`;
- `datetime`: Módulo estándar que permite manipulación de fechas y horas, utilizado para medir tiempos de ejecución, así como para la construcción de la columna `domain_age_days`;
- `itertools`: Módulo estándar que provee herramientas para crear ciclos eficientes, utilizado para hacer gestión de reintentos de procesos que puedan fallar durante la ejecución;
- `math`: Módulo estándar que permite acceso a funciones matemáticas, utilizado para implementar funciones como la de entropía;
- `numpy`: Módulo implementado por terceros, ampliamente usado en temas de análisis de datos y computación numérica, proporcionando implementaciones funciones y tipos de variables para cálculos con énfasis en rendimiento. Usado para obtener compatibilidad con algunos tipos de dato de fecha encontrados en la fuente de WHOIS;
- `os`: Módulo estándar que ofrece interacción con el sistema operativo, utilizado para tareas de lectura y escritura de ficheros fuente y de salida;
- `pandas`: Módulo desarrollado por terceros, considerado uno de los estándares para aplicaciones de análisis y manipulación de datos. Es el módulo principal en este proyecto, y sobre él se implementan las tareas que convierten los datos de entrada en el conjunto curado de salida;
- `regex`: Módulo estándar para operaciones de expresiones regulares o patrones de caracteres dentro de cadenas de texto, utilizado para implementar las columnas que involucraban buscar ciertos patrones dentro del cuerpo de las URL;

- `requests`: Módulo desarrollado por terceros, implementación de biblioteca HTTP que permite gestionar peticiones web. Utilizado para implementar las comunicaciones con los origenes de datos;
- `time`: Módulo estándar de Python que proporciona soporte para operaciones entre tipos de dato temporal. Utilizado para realizar mediciones de ejecución de algunas partes que requerían ser monitorizadas dentro de las pruebas del *framework*;
- `tldextract`: Módulo desarrollado por externos, y proporciona identificación precisa de TLDs, SLDs y subdominios en una URL. Utilizado como forma fiable de extracción de tales informaciones, sin tener que recurrir a una propia implementación, que posiblemente tuviese menor rendimiento;
- `typing`: Biblioteca estándar que provee soporte para indicación expresa de tipos de dato en entrada y salida de funciones (concepto llamado *type hinting*). A pesar de que Python por defecto no requiera esta funcionalidad, es altamente recomendada en el ámbito de construcción de paquetes y módulos como este *framework*, ya que permite hacer indicaciones de los tipos de dato que son esperados en ciertas partes del código, para apoyar a programadores y herramientas de análisis automatizado de código a identificar si la rutina de código funciona como es esperado o no;
- `python-whois`: Módulo desarrollado por un tercero, que proporciona métodos para consultar directamente a servidores WHOIS, e interpretar la información que entreguen en un formato legible en Python. Es otro de los módulos principales en este *framework*, ya que constituye la mejor forma de acceso abierto para integrar los datos requeridos de WHOIS, sin tener que recurrir a un proveedor comercial.

Descrito en un alto nivel, el paquete desarrollado `phishingutils` se compone de una clase llamada `PhishingURLModel`, que reúne los atributos y métodos para creación del objeto del modelo de datos de URLs *phishing*, además de métodos para modificarlo y exportarlo.

Atributo	Tipo de dato	Descripción
`urls_col`	Texto	Nombre deseado de la columna que contendrá las URL
`label`	Texto	Nombre deseado de la columna que contendrá las etiquetas de clase
`sources`	Objeto	Descripción de las fuentes de dato que serán incluidas para cada clase
`max_size`	Entero	Tamaño total máximo del conjunto de datos a devolver
`sample_seed`	Entero	Semilla utilizada en el proceso aleatorio de selección de muestras
`feature_cols`	Objeto	Descripción del listado de atributos que serán agregados al conjunto de datos, que consiste en el nombre de la columna y la función que computa su valor

Tabla 3.7: Clase `PhishingURLModel` y sus atributos

A continuación se ilustran con ejemplos las estructuras de datos para denominar los atributos de la clase `PhishingURLModel`, en cuanto a `sources` se compone de un objeto con dos propiedades, cada uno representa una lista de las fuentes a habilitar para poblar el conjunto de datos de salida.

```
{
    "phishing": [
        "PhishTank",
        "Phishing.Database",
        "OpenPhish-Community"
    ],
    "legitimate": [
        "Cisco-Umbrella",
        "Majestic"
    ]
}
```

Extracto 3.1: Especificación del atributo `sources` de la clase `PhishingURLModel`

Respecto a `feature_cols`, es un arreglo de objetos, cada uno conteniendo las propiedades `name`, nombre de la columna; `func`, función que produce el valor válido del atributo; y `type`, que expresa si la función es tipo *lookup* o (función recibe todo el conjunto de datos y lo devuelve con la columna agregada) o *map* (función escalar que recibe el valor individual que recibe el valor de una URL y computa la propiedad requerida de esta).

```
[
    {"name": "columna1", "func": funcion1, "type": "map"},
    {"name": "columna2", "func": funcion2, "type": "lookup"},
    ...
]
```

Extracto 3.2: Especificación del atributo `feature_cols` de la clase `PhishingURLModel`

Un objeto de la clase `PhishingURLModel` cuenta con los métodos:

- `setup`: Construye el modelo del conjunto de datos, y descarga las fuentes de datos que se hayan habilitado, guardándolas en las rutas `./files/legitimate/` y `./files/phishing/` para fuentes benignas y *phishing* respectivamente;
- `add_map_col` y `add_lookup_col`: Recibe las propiedades de un atributo a adicionar para los tipos de funciones *map* y *lookup* respectivamente, agregándolasal el arreglo `feature_cols` del modelo de datos;
- `export`: Exporta el conjunto de datos resultante en formato `csv.gz`, bajo la ruta `./files/out/out.csv.gz`.

El *framework* `phishingutils`, además del constructor `PhishingURLModel`, proporciona una serie de funciones que se consideraron comunes, para definir los atributos seleccionados y muchos otros atributos que algún eventual investigador quiera implementar. Los métodos están debidamente documentadas para explicar tanto su funcionamiento como las entradas, salidas esperadas y ejemplo de uso. Se puede apreciar como ejemplo la definición de la función `starts_with_ip_addr` a continuación:

```
def starts_with_ip_addr(url: str) -> bool:
"""
Returns True if the base URL of an input URL string is an IP
address, False otherwise

Parameters
----------
url : str
    Input URL string

Returns
-------
f : bool
    True if the base URL of an input URL string is an IP address,
    False otherwise

Examples
--------
>>> import phishingutils.corpus.functions as fn
>>> fn.starts_with_ip_addr("http://192.168.1.1/home")
True

"""
return bool(regex.match(
            r"^((25[0-5]|(2[0-4]|1\d|[1-9]|)\d)\.?\b){4}$",
            tldextract.extract(url).domain))
```

Extracto 3.3: Especificación del atributo `feature_cols` de la clase `PhishingURLModel`

3.6. Recopilación y curado del conjunto de datos

Vale destacar que durante el proceso de implementación de la clase `PhishingURLModel`, se tuvieron que considerar particularidades de cada uno de los origenes de datos como:

- Distinguir las fuentes que traían o no encabezado de columnas;
- Algunos origenes usan redirecciones y limitación en la frecuencia en que es posible descargar datos, se cree que para evitar ataques o uso inapropiado. Esta es la razón por la cual se decidió mantener los archivos en almacenamiento persistente;
- Las fuentes de datos en formato CSV usualmente traen dificultades de interpretación de su esquema si el separador de columnas está contenido en el valor de alguna celda. Por tanto, fue necesario iterar varias veces en la configuración de lectura para conseguir interpretar estos origenes problemáticos, inclusive llegando a descartar partes del conjunto por ser ilegibles.
- Como es esperado que algunas URL se encuentren en más de una fuente de dato de cada clase, se realizó la debida eliminación de duplicados;
- Evidenciado en la sección 3.2, dada la disparidad de volumen de datos disponible entre clases benigna y *phishing*, se debió implementar un procedimiento de balanceo para que se contara con una relación aproximada a 50/50, y con ello evitar cualquier tipo de polarización de los eventuales algoritmos de clasificación sobre una clase o la otra.

Por otro lado, la implementación de las funciones comunes del *framework* fue la que tomó mayores esfuerzos en cuanto a tiempo e iteraciones, pues dentro de las funciones comunes se encuentra la obtención de la información del servicio de WHOIS para cada dominio. Como fue discutido en la sección 2.4, comercialmente se encuentran bases de datos o servicios que permiten hacer consulta de registros WHOIS en masa, sin embargo, en un escenario educativo-investigativo, las opciones pueden ser limitadas. Por tanto, se optó por implementar un procedimiento que permitiera cubrir la necesidad de tener los registros WHOIS necesarios para dotar al conjunto de datos de esta dimensión de dominio, visto que permite crear modelos con mayor rendimiento.

Algunas reflexiones que vale la pena mencionar sobre la experiencia durante esta implementación:

- Fueron probados varios servicios de consulta en masa, como WhoisXMLAPI, sin embargo, con limitaciones como una cuota gratuita de máximo 500 dominios, se volvería inviable construir un conjunto esperado de millones de registros. Por este motivo fue seleccionado `python-whois`, ya que ejecuta sin intermediarios consultas directas a los servidores que cuentan con los registros globales, aunque con la limitación de 1 dominio por petición;

- En la práctica, cada petición era casi instantánea, pero al enviar un volumen elevado de solicitudes en un corto tiempo, se observó que los servidores rechazaban buena parte de ellas. Tras prueba y error, con la meta de reducir los rechazos y los tiempos entre petición, se fijaron ejecuciones paralelas con 5 hilos, cada uno de ellos dando un tiempo muerto de 0.4 segundos entre solicitud, y aplicando 10 reintentos en caso de que la solicitud haya sido rechazada. Con esto se observó un tiempo medio de ejecución global de 0.5 segundos/petición (sumando sobrecostes por persistencia de los datos). Esto significa que la persistencia de cerca de 2.500.000 registros WHOIS de dominios llevó aproximadamente 15 días de ejecución continua, distribuidos a lo largo de 6 semanas que llevó consolidar el conjunto de datos.

```
ERROR:whois.whois:Error trying to connect to socket: closing socket - [Errno -2] Name or service not known
2024-09-09 12:00:02,799 - whois.whois - ERROR - Error trying to connect to socket: closing socket - timed out
ERROR:whois.whois:Error trying to connect to socket: closing socket - timed out
```

Extracto 3.4: Ejemplo de errores de rechazo durante descarga de registros WHOIS

3.7. Creación del paquete `phishingutils-0.1.0`

Siguiendo las instrucciones descritas en la sección 2.6 sobre la creación de paquetes, se construyó el fichero `pyproject.toml`, el cual especifica las instrucciones y parámetros necesarios para empaquetado del módulo implementado.

```
[build-system]
requires = ["setuptools", "wheel"]

[tool.setuptools]
include-package-data = false

[tool.setuptools.packages.find]
include = ["phishingutils*", "README.md"]

[project]
name = "phishingutils"
authors = [
    {name = "Sergio Agudelo Bernal", email = "seagudelobe@unal.edu.co"}
]
maintainers = [
    {name = "Sergio Agudelo Bernal", email = "seagudelobe@unal.edu.co"}
]
description = "A framework for speeding up creation of phishing URL models for detection by means of classifiers."
version = "0.1.0"
license = {text = "MIT License"}
readme = "README.md"
requires-python = ">=3.11.5"
```

```
dependencies = [
    "datefinder==0.7.3",
    "dateparser==1.2.0",
    "numpy==1.26.4",
    "pandas==2.2.2",
    "python-whois==0.9.4",
    "regex==2024.7.24",
    "requests==2.32.3",
    "retry==0.9.2",
    "tldextract==5.1.2",
    "typing==3.7.4.3"
]
```

Extracto 3.5: Ejemplo de contenido de fichero de empaquetado `pyproject.toml`

Se puede apreciar que se incluye la debida documentación del contacto de autoría y manutención del paquete, así como descripción y README elaborado, metainformaciones que deberían poder ser visualizadas bajo demanda del usuario que desee instalarlo. También vale destacar que es recomendable configurar las versiones exactas de las dependencias que hacen que el módulo funcione, que suelen ser las versiones disponibles en el ordenador de autoría, ya que esto permite distribuir un paquete que realmente funcione en las máquinas de los usuarios.

El paquete es creado tras la ejecución en terminal de los comandos `pip install -q build` y `python -m build` en el directorio raíz del paquete (lugar donde también debe constar el fichero `pyproject.toml`). Al finalizar la ejecución de esos dos ítems, en la carpeta `./dist` se crean los ficheros `phishingutils-0.1.0-py3-none-any.whl` y `phishingutils-0.1.0.tar.gz`, que pueden ser distribuibles directamente o a través de `pip`, el instalador de paquetes de Python.

3.8. Aplicación de técnicas de *Machine Learning* al conjunto de datos

Dado que el enfoque principal del proyecto no es ahondar en selección óptima de algoritmos o parámetros de entrenamiento, sino demostrar la aplicabilidad de ellos usando casos de uso básicos, se decidió probar la implementación de los algoritmos más comunes documentados en la sección 3.4, usando los valores por defecto de sus parámetros. Un fragmento ejemplo puede ser visualizado a continuación:

```
# Import dependency modules
import pandas as pd

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

# Get source dataset
DF_FILE_PATH = f"./files/out/out.csv.gz"
phishing_df = pd.read_csv(DF_FILE_PATH)
```

```
10
11 ...
12
13 # Split the data into features (X) and target (y)
14 X = phishing_df.drop('label', axis=1)
15 y = phishing_df['label']
16
17 # Split the data into training and test sets
18 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
19
20 rf = RandomForestClassifier()
21 rf.fit(X_train, y_train)
22 y_pred = rf.predict(X_test)
```

Extracto 3.6: Ejemplo básico de implementación del algoritmo `RandomForestClassifier` usando el conjunto de datos obtenido con `phishingutils`

Fue observado en etapas finales de ejecución del proyecto, que convenía aplicar refinamientos adicionales al conjunto de datos producido, como el tratamiento de valores no definidos en atributos, o que estos estuvieran expresados exclusivamente en formato numérico, ya que los algoritmos de clasificación no reciben valores en otro tipo de dato, siendo necesario hacer un procesamiento adicional del conjunto:

```
1 # Remap label column
2 phishing_df['label'] = phishing_df['label'].map({'phishing': 1, 'legitimate
      ': 0})
3
4 # Remap boolean columns
5 phishing_df['starts_with_ip'] = phishing_df['starts_with_ip'].map({True: 1,
       False: 0})
6 phishing_df['has_punycode'] = phishing_df['has_punycode'].map({True: 1,
     False: 0})
7 phishing_df['domain_has_digits'] = phishing_df['domain_has_digits'].map({
     True: 1, False: 0})
8 phishing_df['has_internal_links'] = phishing_df['has_internal_links'].map({
     True: 1, False: 0})
9
10 # Replace NaN values at column `domain_age_days` with the column mean to
     avoid biasing
11 phishing_df['domain_age_days'] = phishing_df['domain_age_days'].fillna(
     phishing_df['domain_age_days'].mean())
12
13 # Drop rows with `url` value equal to NaN
14 phishing_df = phishing_df[phishing_df['url'].notna()]
```

Extracto 3.7: Adecuaciones adicionales hechas al conjunto de datos luego de ser exportado de `phishingutils`

3.9. Conclusión del capítulo

Se considera que se obtuvo una implementación exitosa del *framework* planeado, resultando en un paquete funcional que puede ser distribuido a otros investigadores. Para expandir en la implementación de los métodos, obtener el módulo, o los cuadernos de ejemplo citados en los extractos de las sección 3.8, puede remitirse al repositorio del proyecto en GitHub, el cual se encuentra disponible al público.

4: Resultados y discusión

Al finalizar la ejecución del desarrollo del proyecto, se obtuvieron tres resultados importantes: (1) la implementación y empaquetado del *framework* `phishingutils`; (2) el conjunto de datos producido a partir de él; y (3) los cuadernos de ejemplo de uso del *framework* y casos de uso introductorios sobre clasificadores entrenados usando el conjunto de datos. Todos los recursos implementados se pueden encontrar en el repositorio del proyecto, alojado en GitHub, el conjunto de datos ha sido adicionalmente publicado en Kaggle para uso público, ya que constituye una plataforma conveniente para divulgación dado su alcance en volumen de usuarios.

A continuación se muestra una representación gráfica de los subdirectorios importantes del proyecto en el repositorio, y una pequeña descripción que se considera que vale la pena ser agregada para sensibilizar sobre contenido de interés dentro de los entregables.

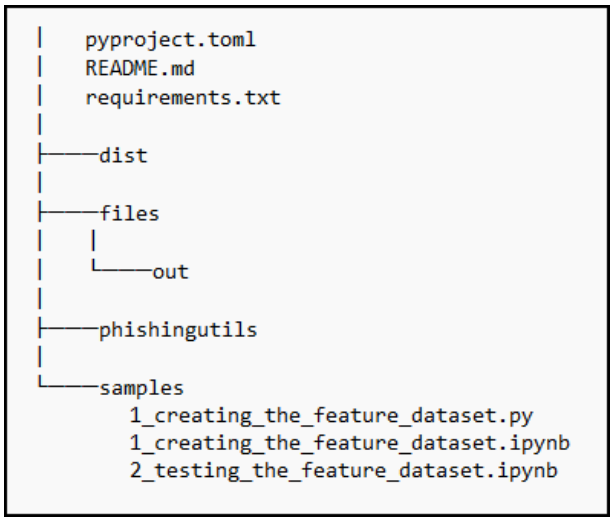

Figura 4.6: Ficheros y subdirectorios pertenecientes al módulo `phishingutils` y utilidades

- En el directorio raíz, `pyproject.toml` y `requirements.txt` contienen las dependencias del paquete y las instrucciones que `pip` debe seguir para construirlo;
- En el directorio `dist` se encuentra la versión más reciente de los paquetes, disponibles para instalación;
- En el directorio `files/out` se encuentran las últimas versiones del conjunto de datos `out.csv.gz` y conjunto de apoyo `whois.csv.gz` con los registros WHOIS asociados al conjunto principal;
- En el directorio `phishingutils` se encuentra el código fuente del *framework* implementado;
- En el directorio `samples` se encuentran los ficheros de ejemplo de uso del *framework* en formato Python (`1_creating_the_feature_dataset.py`) y formato de cuaderno de Jupyter `1_creating_the_feature_dataset.ipynb`. También se encuentra el cuaderno de apoyo con los ejemplos de uso de las técnicas de clasificación usando el conjunto de datos resultante (fichero `2_testing_the_feature_dataset.ipynb`).

4.1. *Framework* `phishingutils` en funcionamiento

El uso del `phishingutils` en el desarrollo de aplicaciones de ML y *phishing* reduce en buena medida los esfuerzos, además basta con apenas unas líneas de código para definir y obtener un conjunto de datos:

```
1  import pandas as pd
2
3  from phishingutils.corpus.base import PhishingURLModel
4  import phishingutils.corpus.functions as fn
5
6  # Creating PhishingURLModel object
7  model = PhishingURLModel(
8      urls_col="url",
9      label_col="label",
10     sources={
11         "phishing": [
12             "PhishTank",
13             "Phishing.Database",
14             "OpenPhish-Community"
15         ],
16         "legitimate": [
17             "Cisco-Umbrella",
18             "Majestic"
19         ]
20     },
21     max_size=1e4,
22     sample_seed=123
23 )
24
```

```
25 model.add_map_col("url_length", fn.char_count)
26 model.add_map_col("starts_with_ip", fn.starts_with_ip_addr)
27 model.add_map_col("url_entropy", fn.url_entropy)
28 model.add_map_col("has_punycode", fn.has_punycode)
29 model.add_map_col("digit_letter_ratio", fn.digit_letter_ratio)
30 model.add_map_col("dot_count", lambda x: fn.expr_counter(url=x, expr=r"\.")
       )
31 model.add_map_col("at_count", lambda x: fn.expr_counter(url=x, expr=r"@"))
32 model.add_map_col("dash_count", lambda x: fn.expr_counter(url=x, expr=r"-")
       )
33 model.add_map_col("tld_count", fn.tld_count)
34 model.add_map_col("domain_has_digits", fn.domain_has_digits)
35 model.add_map_col("subdomain_count", fn.subdomain_count)
36 model.add_map_col("nan_char_entropy", fn.nan_char_entropy)
37 model.add_map_col("has_internal_links", fn.has_internal_links)
38 model.add_lookup_col("whois_data", fn.get_domain_info)
39 model.add_lookup_col("domain_age_days", fn.domain_age)
40
41 # Use this method to create the dataset in runtime
42 df = model.setup()
43 # or alternatively, export a persistent file for later use:
44 # model.export("./files/out/out.csv.gz")
45
46 display(df)
```

Extracto 4.8: Ejemplo de uso del *framework* `phishingutils`

Durante ejecución puede que se observen algunos mensajes de error, derivados de la extracción de registros WHOIS que falten en el conjunto de datos de apoyo `whois.csv.gz`. La salida resultante es un `DataFrame` de `pandas` similar al siguiente:

	url	source	label	url_length	starts_with_ip	url_entropy	has_punycode	d
0	calmhsa.org	Majestic	legitimate	11	False	3.277613	False	
1	homeimprovementvideos.org	Majestic	legitimate	25	False	3.573661	False	
2	warehouse.maloufvip.com	Cisco-Umbrella	legitimate	23	False	3.882045	False	
3	seeyoudirectory.com	Majestic	legitimate	19	False	3.431624	False	
4	21ogkm5th5-dsn.algolia.net	Cisco-Umbrella	legitimate	26	False	4.085055	False	
...	...	...	...	...	...	...	...	
9995	http://pochtarefund-xeq10q.aakmt.xyz/login	Phishing.Database	phishing	42	False	4.594466	False	
9996	http://mobiledevtools.com/https/34.237.113.113...	Phishing.Database	phishing	111	False	4.535333	False	

Figura 4.7: Fragmento del `DataFrame` resultante usando el *framework* `phishingutils`

Tras ejecución usando un tamaño máximo de 2.500.000 registros, las propiedades del `DataFrame` obtenido son las mostradas a continuación. Se puede apreciar que contiene los atributos especificados, y un total de muestras igual al indicado al definir el modelo.

```
>>> display(curated_df.info())

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2500000 entries, 0 to 2499999
Data columns (total 18 columns):
 #   Column               Dtype
---  ------               -----
 0   url                  object
 1   source               object
 2   label                object
 3   url_length           int64
 4   starts_with_ip       bool
 5   url_entropy          float64
 6   has_punycode         bool
 7   digit_letter_ratio   float64
 8   dot_count            int64
 9   at_count             int64
 10  dash_count           int64
 11  tld_count            int64
 12  domain_has_digits    bool
 13  subdomain_count      int64
 14  nan_char_entropy     float64
 15  has_internal_links   bool
 16  whois_data           object
 17  domain_age_days      float64
dtypes: bool(4), float64(4), int64(6), object(4)
memory usage: 276.6+ MB
```

Extracto 4.9: Composición de atributos del conjunto de datos producido

Vale destacar sobre esta tarea de ejecutar el framework para un volumen elevado de datos, que puede ocupar más de 10GB de memoria RAM del ordenador en que se esté ejecutando, y que de haber requerimiento de descargar registros faltantes de WHOIS, se puede tardar aproximadamente 0.5 segundos por valor faltante, como fue estimado durante la construcción del *framework*.

4.2. Descripción detallada del conjunto de datos resultante

Tras consolidación del conjunto de datos final, se caracterizan los atributos de clasificación (es decir, excluyendo las columnas `url`, `source`, `label` y `whois_data`). Se incluyen estadísticas globales de los atributos, así como una representación visual de su distribución.

Atributo `url_length`

- **Tipo:** Entero
- **Valor mínimo:** 4
- **Valor máximo:** 25523
- **Media:** 45.59
- **Desv. estándar:** 74.39
- **Conteo valores faltantes:** 0

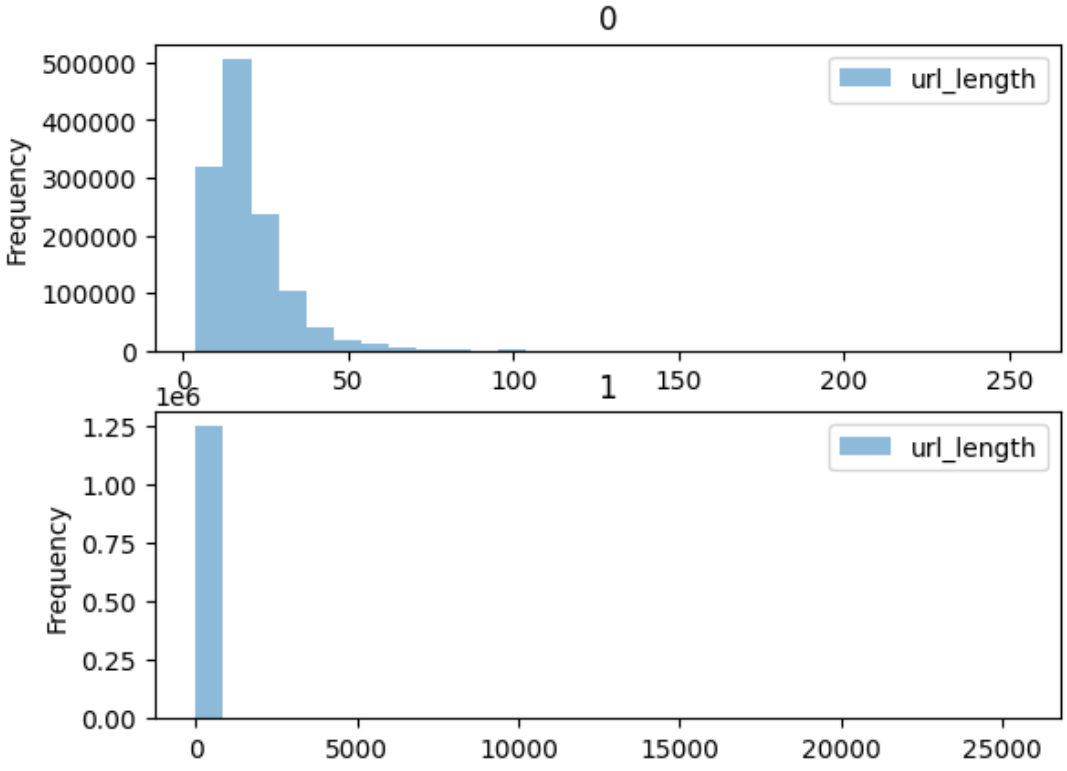

Figura 4.8: Distribución de los valores de la columna `url_length` para URLs benignas (arriba) y *phishing* (abajo)

Atributo `starts_with_ip`

- **Tipo:** Binario
- **Valores posibles:** 0 / 1

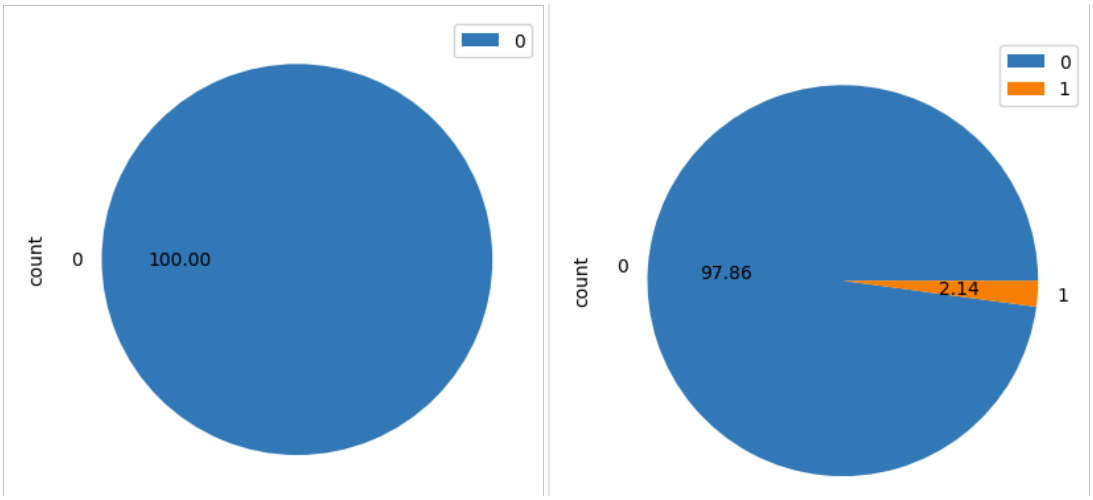

Figura 4.9: Distribución de los valores de la columna `starts_with_ip` para URLs benignas (izquierda) y *phishing* (derecha)

Atributo `url_entropy`

- **Tipo:** Decimal
- **Valor mínimo:** 0
- **Valor máximo:** 6.05
- **Media:** 3.91
- **Desv. estándar:** 0.635
- **Conteo valores faltantes:** 0

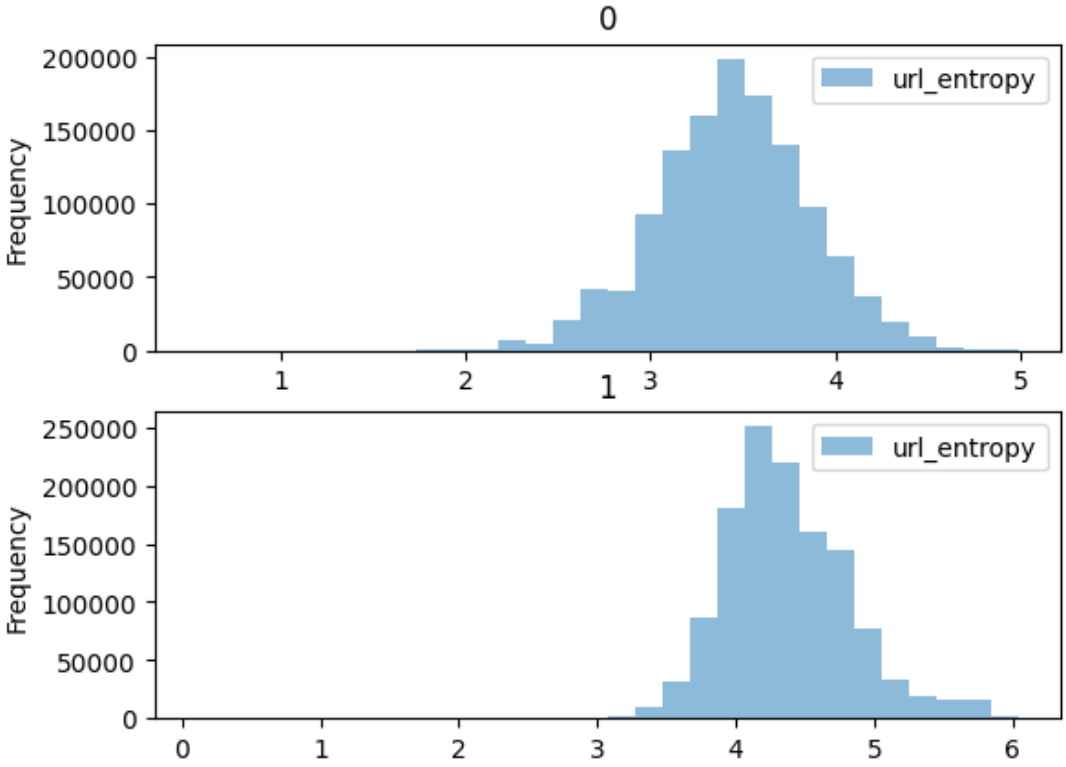

Figura 4.10: Distribución de los valores de la columna `url_entropy` para URLs benignas (arriba) y *phishing* (abajo)

Atributo `has_punycode`

- **Tipo:** Binario
- **Valores posibles:** 0 / 1

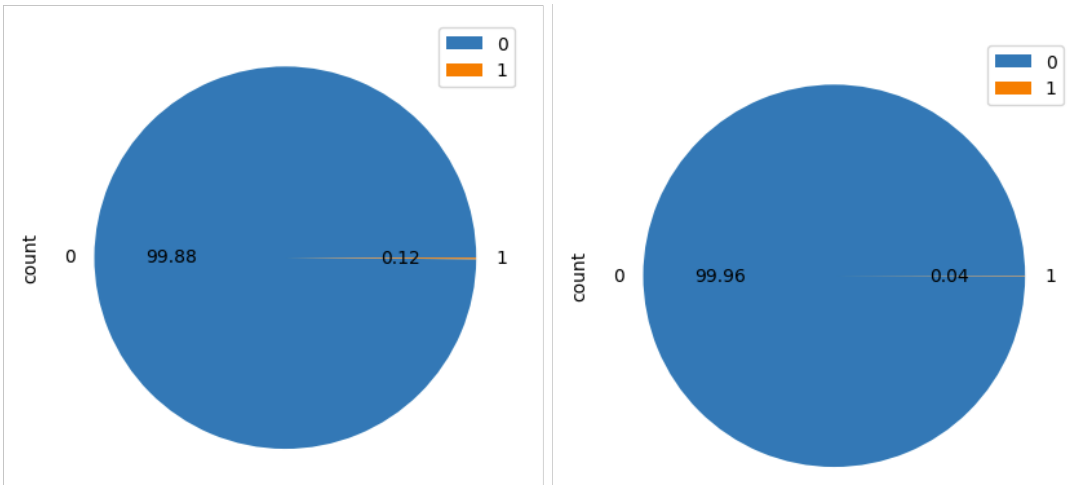

Figura 4.11: Distribución de los valores de la columna `has_punycode` para URLs benignas (izquierda) y *phishing* (derecha)

Atributo `digit_letter_ratio`

- **Tipo:** Decimal
- **Valor mínimo:** 0
- **Valor máximo:** 20.84
- **Media:** 0.117
- **Desv. estándar:** 0.245
- **Conteo valores faltantes:** 1

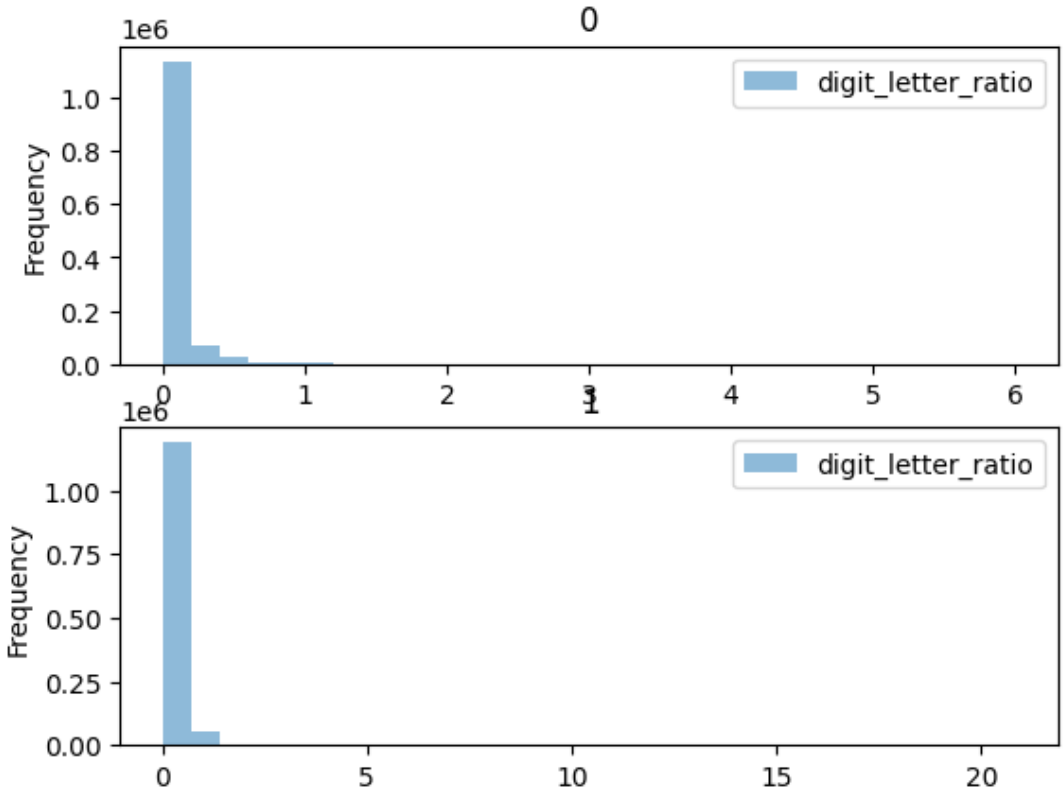

Figura 4.12: Distribución de los valores de la columna `digit_letter_ratio` para URLs benignas (arriba) y *phishing* (abajo)

Atributo `dot_count`

- **Tipo:** Entero
- **Valor mínimo:** 0
- **Valor máximo:** 211
- **Media:** 2.17
- **Desv. estándar:** 1.73
- **Conteo valores faltantes:** 0

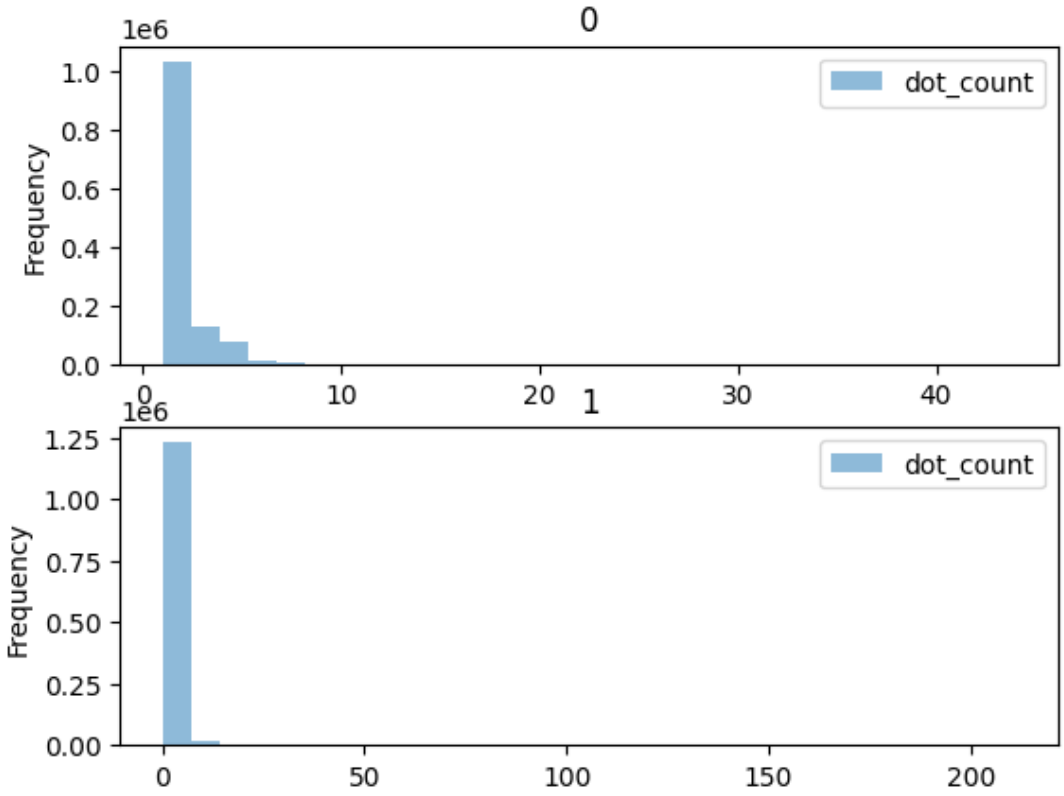

Figura 4.13: Distribución de los valores de la columna `dot_count` para URLs benignas (arriba) y *phishing* (abajo)

Atributo `at_count`

- **Tipo:** Entero
- **Valor mínimo:** 0
- **Valor máximo:** 32
- **Media:** 0.0114
- **Desv. estándar:** 0.14
- **Conteo valores faltantes:** 0

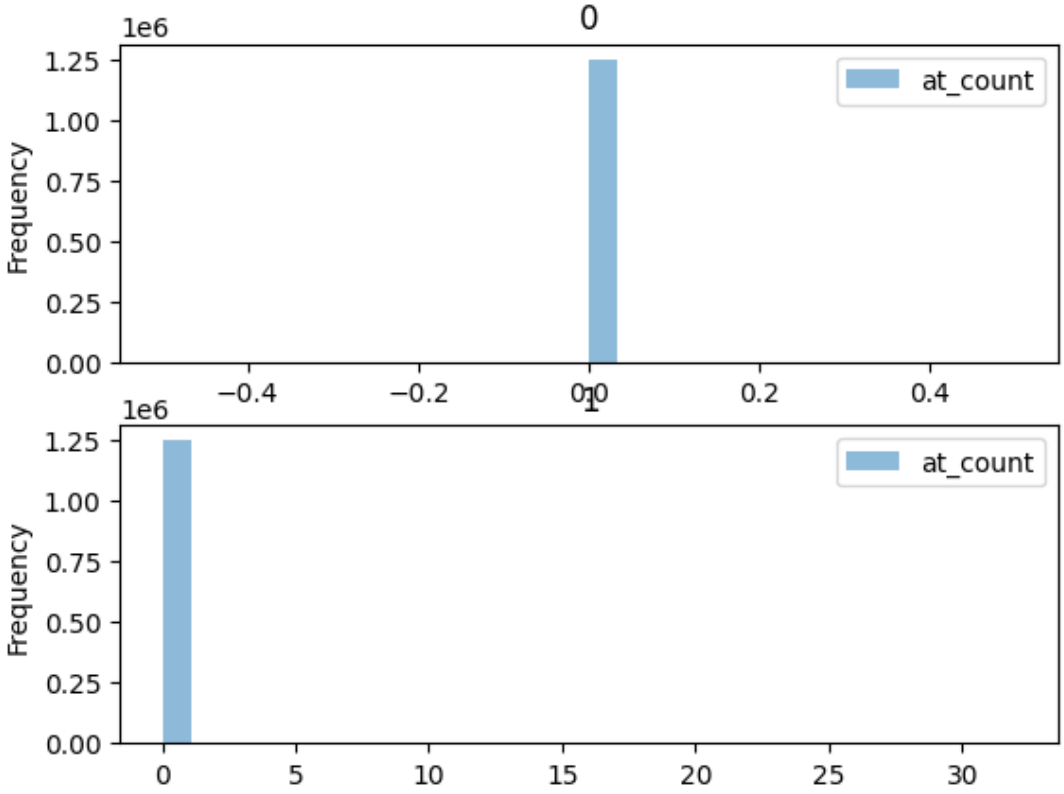

Figura 4.14: Distribución de los valores de la columna `at_count` para URLs benignas (arriba) y *phishing* (abajo)

Atributo `dash_count`

- **Tipo:** Entero
- **Valor mínimo:** 0
- **Valor máximo:** 322
- **Media:** 0.74
- **Desv. estándar:** 1.66
- **Conteo valores faltantes:** 0

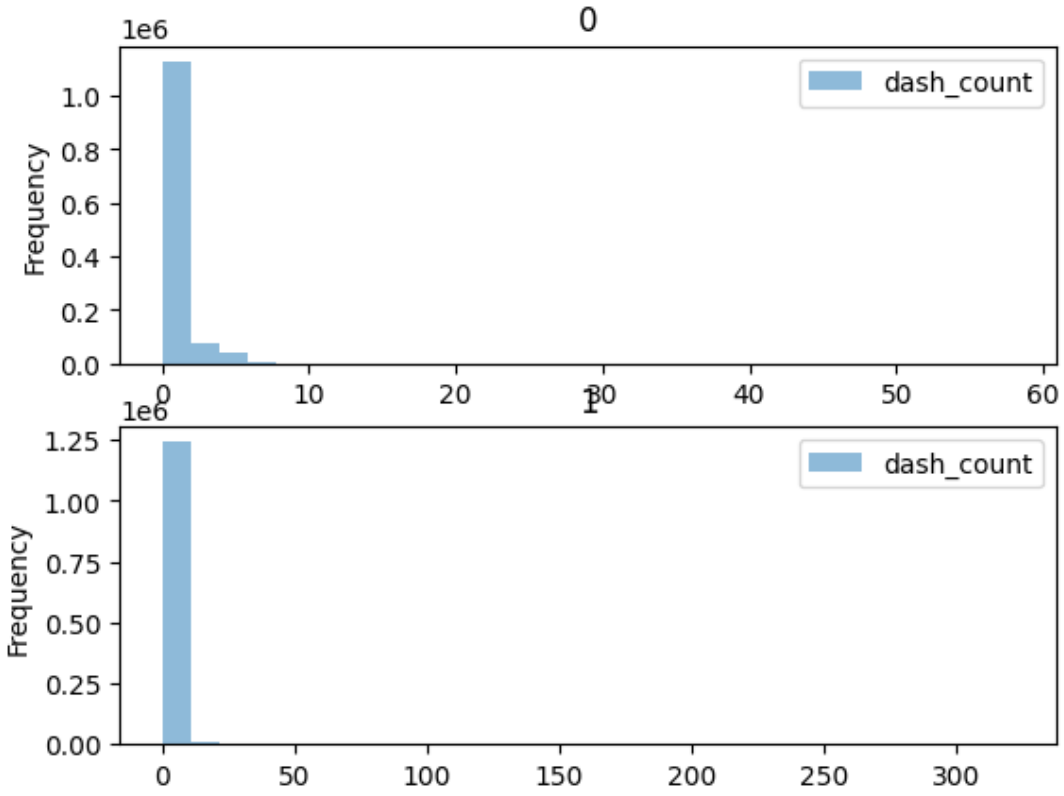

Figura 4.15: Distribución de los valores de la columna `dash_count` para URLs benignas (arriba) y *phishing* (abajo)

Atributo `tld_count`

- **Tipo:** Entero
- **Valor mínimo:** 0
- **Valor máximo:** 65
- **Media:** 0.0392
- **Desv. estándar:** 0.39
- **Conteo valores faltantes:** 0

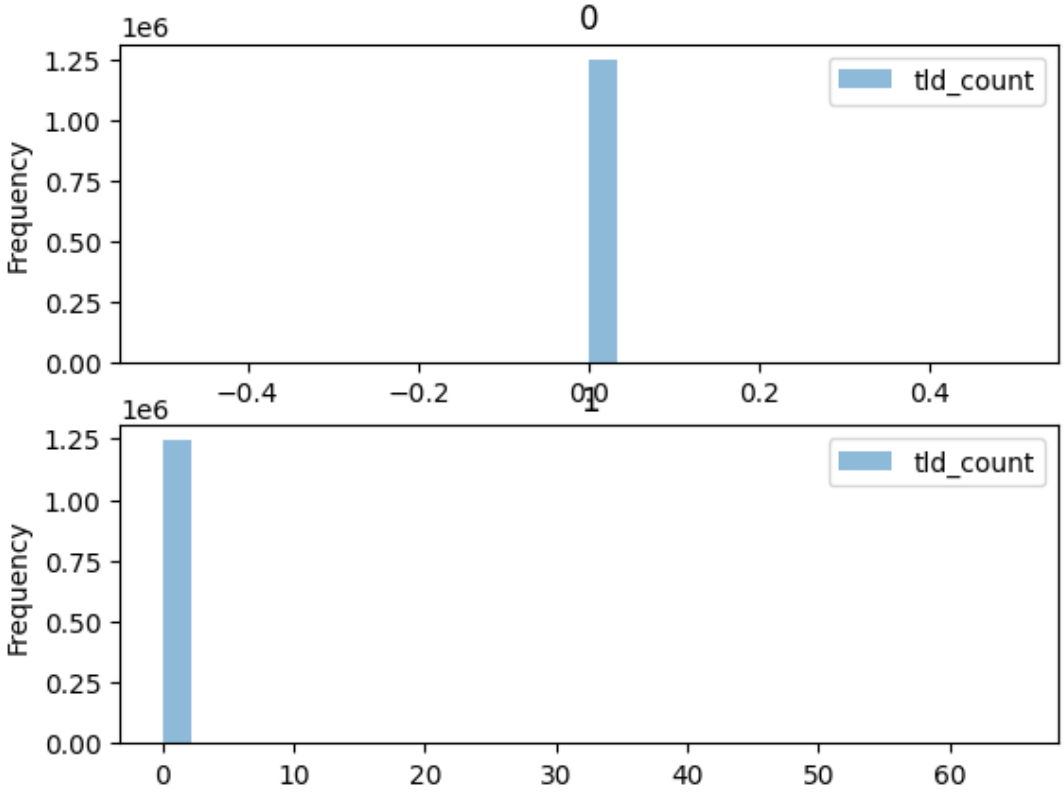

Figura 4.16: Distribución de los valores de la columna `tld_count` para URLs benignas (arriba) y *phishing* (abajo)

Atributo `domain_has_digits`

- **Tipo:** Binario
- **Valores posibles:** 0 / 1

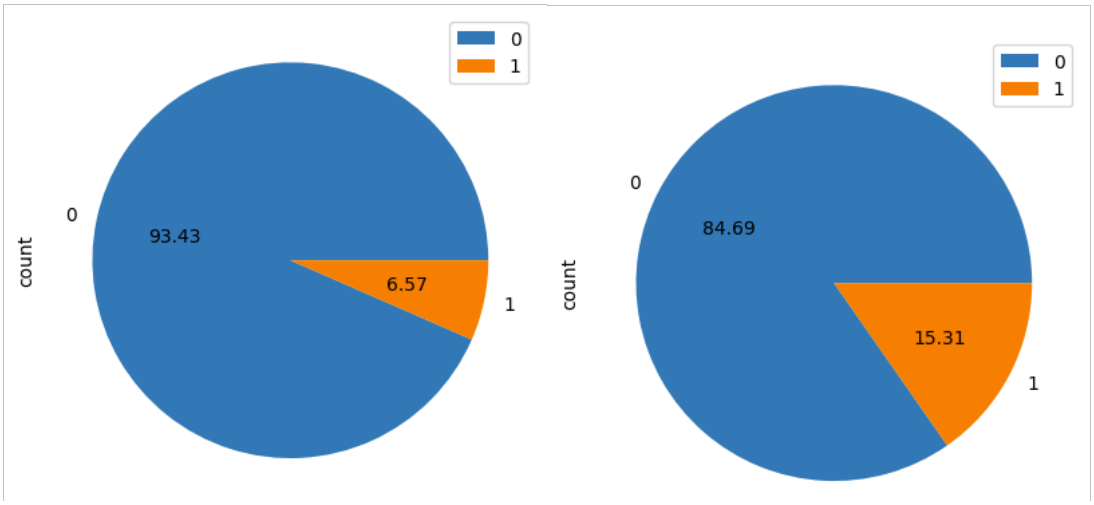

Figura 4.17: Distribución de los valores de la columna `domain_has_digits` para URLs benignas (izquierda) y *phishing* (derecha)

Atributo `subdomain_count`

- **Tipo:** Entero
- **Valor mínimo:** 0
- **Valor máximo:** 43
- **Media:** 0.777
- **Desv. estándar:** 1.10
- **Conteo valores faltantes:** 0

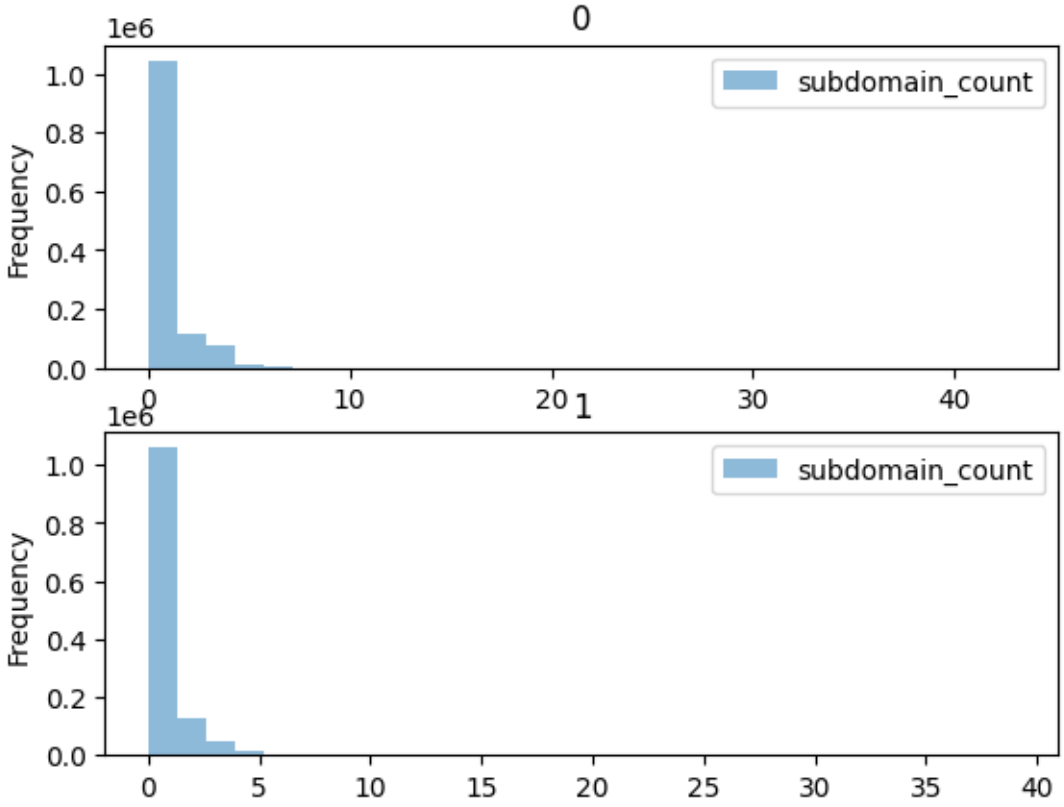

Figura 4.18: Distribución de los valores de la columna `subdomain_count` para URLs benignas (arriba) y *phishing* (abajo)

Atributo `nan_char_entropy`

- **Tipo:** Decimal
- **Valor mínimo:** 0
- **Valor máximo:** 1.901
- **Media:**0.465
- **Desv. estándar:** 0.188
- **Conteo valores faltantes:** 0

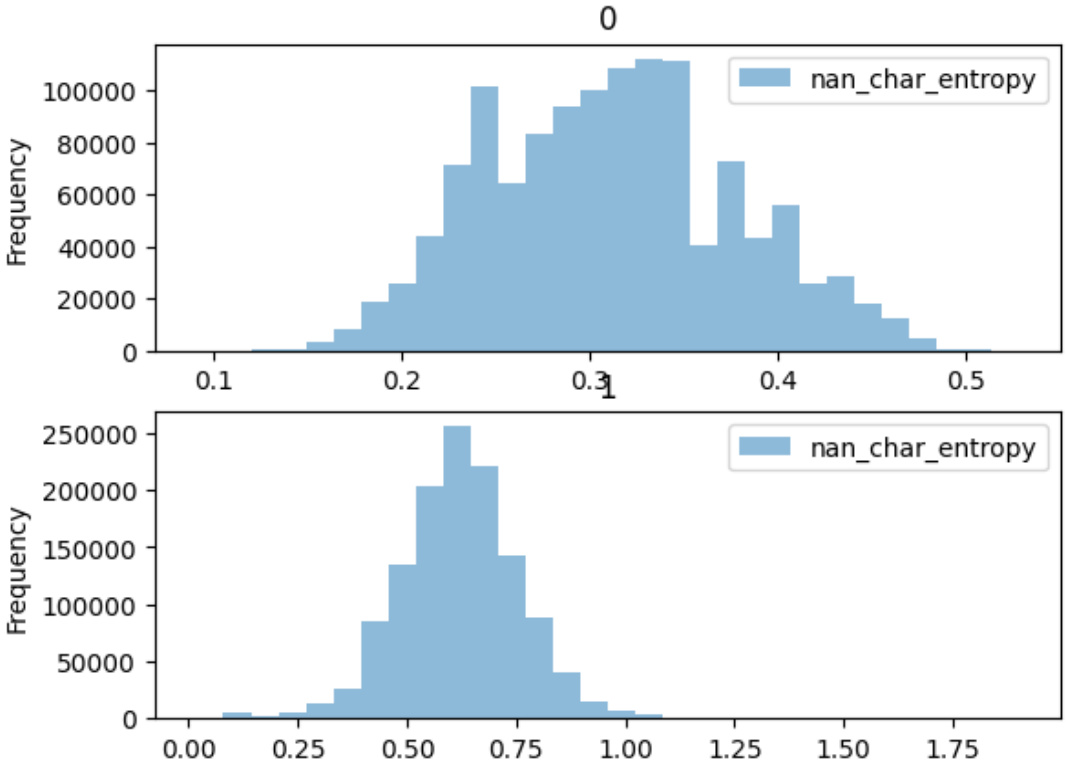

Figura 4.19: Distribución de los valores de la columna `nan_char_entropy` para URLs benignas (arriba) y *phishing* (abajo)

Atributo `has_internal_links`

- **Tipo:** Binario
- **Valores posibles:** 0 / 1

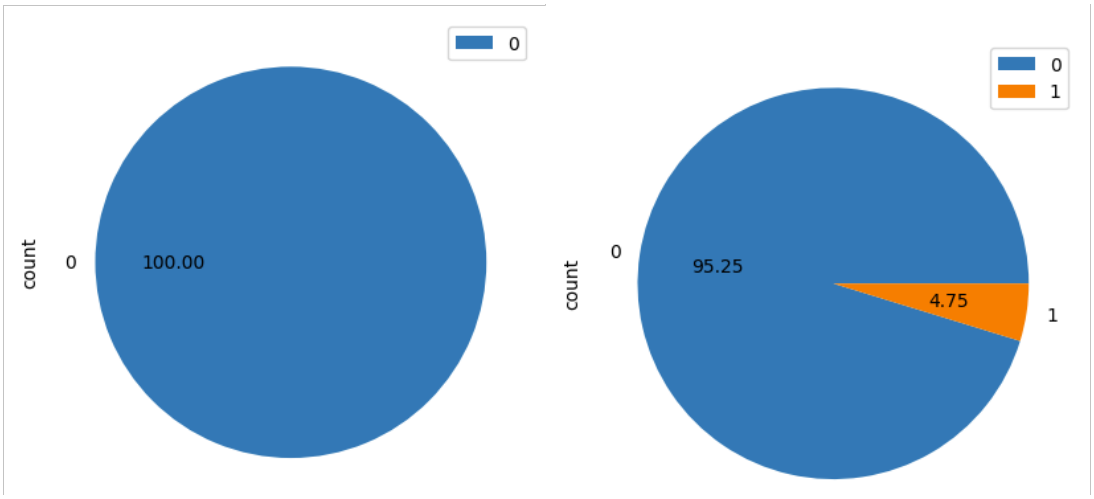

Figura 4.20: Distribución de los valores de la columna `has_internal_links` para URLs benignas (izquierda) y *phishing* (derecha)

Atributo `domain_age_days`

- **Tipo:** Decimal
- **Valor mínimo:** -8.60
- **Valor máximo:** 45541
- **Media:** 4863.09
- **Desv. estándar:** 3345.87
- **Conteo valores faltantes:** 750689

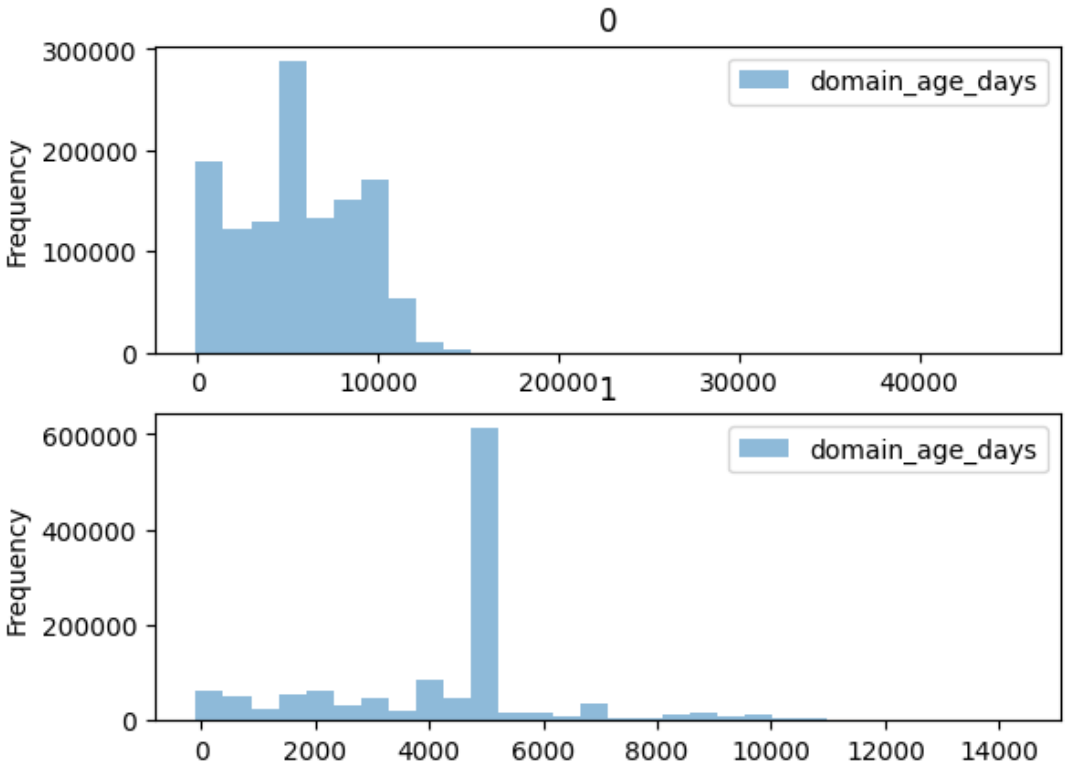

Figura 4.21: Distribución de los valores de la columna `domain_age_days` para URLs benignas (arriba) y *phishing* (abajo)

4.3. Desempeño de las técnicas de *Machine Learning* probadas

Se muestra a continuación un resumen de las métricas de desempeño obtenidas en cada modelo de clasificación probado, así como una representación de la matriz de confusión para notar la distribución de positivos y negativos:

Modelo	Exactitud	Precisión	*Recall*	*F1-score*
`RandomForestClassifier`	**99.999 %**	**100 %**	**99.999 %**	**99.999 %**
`GradientBoostingClassifier`	99.999 %	99.999 %	99.998 %	99.998 %
`GaussianNB`	83.056 %	96.725 %	68.388 %	80.126 %
`LinearSVC`	99.896 %	99.831 %	99.961 %	99.896 %
`DecisionTreeClassifier`	99.999 %	99.998 %	99.999 %	99.999 %

Tabla 4.8: Resultado consolidado de métricas de desempeño por algoritmo de clasificación

4.4. Discusión

Durante el análisis del comportamiento de los atributos, el primer aspecto que salta a la vista son los valores faltantes, para el atributo `digit_letter_ratio` con apenas uno, pero casi 1/3 para el caso de `domain_age_days`, con 750.689 faltantes. Tras inspección del conjunto de datos, esto se debe a que para la gran parte de las URLs, no se encontró registro WHOIS asociado, se asume que es porque el dominio dejó de estar registrado, o más recientemente, debido a la guerra de Rusia y Ucrania, se han impuesto bloqueos de conexión con servidores rusos, afectando entre otros temas, la adquisición de información sobre dominios en ese país [36] (ver ejemplo a continuación). También se podría dudar sobre la fiabilidad de las fuentes relacionadas con *phishing*, ya que muchas URL contienen caracteres aleatorios en sus nombres de dominio, y se juzga que no es posible que tales nombres puedan ser registrados ante la IANA. Para resolver esta falta de datos, como se puede apreciar en el cuaderno `2_testing_the_feature_dataset.ipynb`, al crear los modelos de clasificación se completaron los valores faltantes de edad del dominio con el valor de su media.

```
>>> import whois
>>> whois.whois("elforcer.ru")
Traceback (most recent call last):
...
whois.parser.PywhoisError: No entries found for the selected source(s).
```

Extracto 4.10: Error evidenciado al intentar capturar registros WHOIS del TLD `.ru`, con dominio existente

También se observan algunas tendencias de atributos respecto a la etiqueta de las URL, por ejemplo:

- No existen registros benignos que inicien con una dirección IP ni que contengan enlaces internos, esto se explica naturalmente pues las fuentes de datos (Cisco Umbrella, Majestic Million) son exclusivamente dominios. Por lo que se concluye que no sería un atributo óptimo pues no hay representación en ambas clases;
- La distribución de entropía de URL y caracteres NAN tienen ambas una media mayor en el caso de las URL *phishing*. Se aprecia la utilidad de estos atributos ya que se evidencia correlación con las clases;
- Las URL maliciosas tienden a contener más dígitos en proporcion que las páginas web benignas. Este sería otro atributo apropiado para inferir la clase de los datos;
- Hay un pico pronunciado alrededor de la media para el atributo `domain_age_days`, esto debido a que los valores faltantes se completaron justo con esa magnitud, como fue indicado anteriormente. Se aprecia a posteriori que era más adecuado haber removido los valores faltantes, para no ocasionar ese cambio en la distribución, pues la cantidad de faltantes es considerable.

Respecto a los resultados de desempeño de los clasificadores, se observa valores inusualmente altos de los indicadores, especialmente para el caso de los algoritmos `GradientBoostingClassifier` y `RandomForestClassifier`. Se concluye que aconteció por dos factores principales:

- Las muestras de URLs benignas y *phishing* no corresponden al mismo concepto, las primeras constituyen apenas el nombre de dominio, mientras que las segundas son URLs completas. Esto hace que los atributos que dependan del subdirectorio sean claramente distintos por clase, provocando sobreajuste de los modelos.
- Se comprueba que el aprendizaje en conjunto que ofrece mejor desempeño que las técnicas individuales, como aparece relatado en el marco teórico.

5: Conclusiones y Líneas de trabajo futuras

Como etapa final en el trabajo de fin de máster, se realiza una valoración general del resultado del proyecto, a partir de los elementos de salida obtenidos, posibles mejoras y líneas de trabajo futuras.

5.1. Conclusiones

La implementación y funcionamiento del *framework* `phishingutils` se considera un éxito en cuanto a aprendizaje de las distintas etapas de desarrollo y optimización de una librería, hasta llegar a la fase de publicación y divulgación de un recurso funcional sobre el cual otros desarrolladores puedan construir conocimiento. La construcción del conjunto de datos maestro es un punto de partida principalmente por notable mejora en el volumen de datos y balanceo de clases respecto a otras alternativas, considerándose suficiente para iterar sobre él con el objetivo de conseguir modelos con mejor rendimiento.

Respecto a la selección de atributos y paradigma en sí del uso de clasificadores, dada la naturaleza del problema y los datos disponibles, constituye una área de oportunidad para este proyecto, pues se omitieron consideraciones de diseño que posiblemente resultaran en un desempeño bueno y que al mismo tiempo tuviera aplicación para conjuntos de datos que no fuesen tan claramente diferenciables entre clases.

En relación a la revisión literaria, se considera que se realizó una valoración extensa y por tanto permite evidenciar las tendencias en metodologías de detección de *phishing* que hay en la actualidad, como punto de partida para futuras investigaciones.

En términos globales, se considera que aportar tanto a nivel cualitativo con contextualización del estado del arte, y a nivel cuantitativo de construcción de herramientas técnicas en áreas de ingeniería puede constituir un trabajo notable por contribuir de forma integral en el avance de tecnologías, o en este caso, a hacerle frente a tecnologías usadas con motivos maliciosos.

5.2. Trabajo futuro

Contrastando con los logros percibidos, se nombran a continuación algunos puntos de mejora y áreas posibles de trabajo futuro para contribuir en la utilidad de esta línea de proyecto:

- Se mencionó brevemente en el marco teórico que se evidenció un trabajo similar al realizado, y deliberadamente se omitió realizar un comparativo para no concluir con resultados idénticos. En esta etapa, se considera que ahora puede ser oportuno analizar diferencias entre el proyecto y alternativas afines, con ello identificar puntos de mejora más rápidamente;
- Se debe investigar en formas de normalizar las variables de entrada (en esta caso URL), para eliminar problemas de polarización de atributos debido a falta de fragmentos en la información de cada registro. Para este caso, se podría hacer una búsqueda más profunda de fuentes de datos de gran escala de URLs benignas completas;
- Se requiere redundancia en la obtención de información que bajo una única fuente pueda no estar disponible. Se aprecia la necesidad de utilizar un servicio fiable de información de dominios, ya que es estándar en aplicaciones en la industria de ciberseguridad, así sea por medios comerciales;
- Debido a limitaciones en alcance, se omitió la exploración de algoritmos de *Machine Learning* diferentes a la clasificación. Se encuentra necesario expandir la aplicación a otras técnicas que estén en la frontera de conocimiento, y con ello también se observa que es deseable ampliar las dimensiones actuales de los tipos de datos (ahora URL y dominio) a técnicas híbridas que aprovechen toda la información sobre la que se pueda inferir la categoría de contenido.

Apéndices

Apéndice A

Plan de Proyecto

Este apéndice presenta el plan desarrollado para ejecución del proyecto, y las consideraciones hechas para su concepción.

A.1. Introducción

Se planteó un plan secuencial de tareas a desarrollar, dado que se evidenció una dependencia directa de una fase sobre la siguiente (por ejemplo, no se podría desarrollar el conjunto de datos sin tener completado el *framework*). Cada tarea o fase tiene una duración determinada, estimada para concluir con un entregable bien definido. Los encargados de verificar la conclusión de cada tarea son los gestores del proyecto junto con el desarrollador, para comprobar que cada tarea se haya culminado a plenitud.

A.2. Planificación temporal

A continuación se puede observar una representación gráfica en diagrama de Gantt donde constan las fases secuenciales del proyecto, y además su descripción:

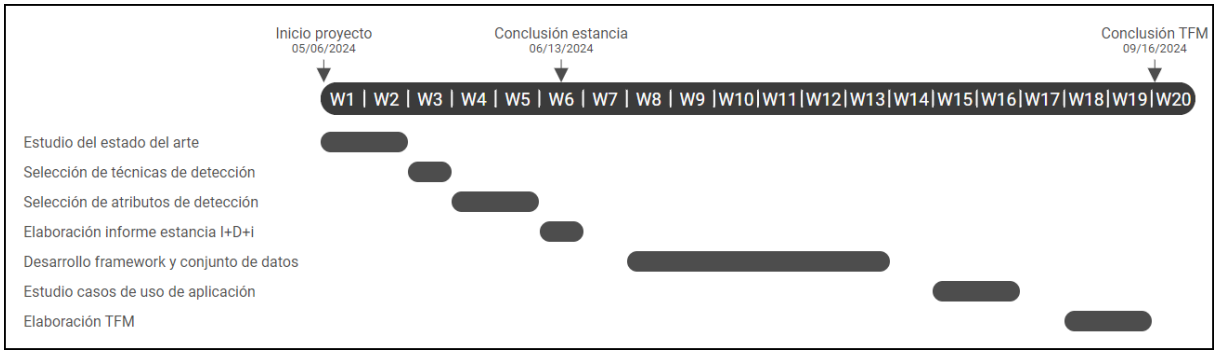

Figura A.1: Distribución temporal de las tareas planeadas para ejecución del proyecto

1. **Estudio del estado del arte (2 semanas)**
   En esta fase el propósito sería identificar tanto los conjuntos de datos más relevantes en cuanto a páginas web de *phishing* como las técnicas de detección existentes en investigaciones recientes. El producto de esta fase sería una caracterización documentada de los conjuntos de datos y las técnicas.

2. **Selección de técnicas de detección (1 semana)** A partir de la caracterización de una serie técnicas, en la segunda fase se esperaría reducir a ciertas metodologías de *Machine Learning* objeto de estudio, que serán tanto relevantes a las investigaciones recientes como fiables en la detección. El ítem de salida en esta etapa sería un listado preliminar de tales técnicas seleccionadas.

3. **Selección de parámetros relevantes a las técnicas de detección (2 semanas)**
   Basado en la selección final de técnicas y en el estudio del estado del arte, el objetivo de esta fase sería determinar el conjunto de características de los datos y transformaciones requeridas para obtener atributos que permitan realizar casos de uso de detección de páginas de *phishing* y establecer un comparativo entre ellos. El entregable de esta fase sería mediante la creación del boceto un *framework* que permita incorporar el conjunto de datos fácilmente, y una caracterización de los parámetros de detección aptos.

4. **Elaboración del informe de estancia I+D+i (1 semana)**
   Si bien se realizaron tareas de documentación a lo largo de todas fases, se propuso tener una tercera etapa dedicada enteramente a elaborar el documento parcial de la estancia, ya que la misma supuso una etapa preliminar que dió consecución a la fase del trabajo final de máster. Así, el entregable de salida de esta fase sería el informe de I+D+i, germen de lo que sería el presente documento.

5. **Conclusión del desarrollo del framework y consolidación del conjunto de datos final (6 semanas)**
   Durante esta etapa ocurrió el esfuerzo principal de desarrollo, el cual fue iterativo debido a que cambios en el conjunto podrían suponer cambios en el *framework*, y viceversa. Esta fase concluyó con la creación del conjunto de datos refinado y el lanzamiento de la versión final del *framework* en forma de paquete.

6. **Estudio de casos de uso de aplicación del *framework* y conjunto de datos (2 semanas)**
   Durante esta etapa se elaborarían los cuadernos de apoyo donde se ejemplifica el uso del *framework* y conjunto de datos, los cuales reportan las métricas de desempeño. Esta fase concluyó con la publicación de los cuadernos, y documentación de la caracterización del conjunto de datos y desempeño de modelos.

7. **Elaboración del trabajo final de máster (2 semanas)**
   De manera similar a la fase 3, se planeó una actividad totalmente dedicada a escribir el documento final, partiendo de bocetos de notas clave obtenidos en anteriores

fases. Esta fase acabaría con la elaboración y revisión conclusiva del trabajo final de máster.

A.3. Estudio de viabilidad

Durante la primera fase del proyecto, la cual sería decisoria para determinar si continuaría su ejecución, se evaluaron criterios de viabilidad en ejecución, económica y legal. Al verse reflejado en las tendencias de vulnerabilidades en ciberseguridad, el ecosistema de herramientas y técnicas, se evidenció una necesidad en contribuir dentro de los esfuerzos de prevención de ataques. Las técnicas de *Machine Learning* han sido la metodología clave para hacer frente a tales amenazas, por lo cual no se encontró aspecto algún que no permitiera la ejecución del proyecto, dado que ya existían y siguen existiendo aplicaciones e investigaciones tanto académicas como corporativas sobre este tema.

Viabilidad económica

Como se trata de un área en la que hay una trayectoria considerable de trabajo previo, no se evidenciaron bloqueos mayores que impidieran el acceso a información y herramientas de forma gratuita. Si bien fue finalmente adquirida una suscripción de Google Colab para permitir la obtención ágil de resultados, no constituyó un requerimiento para la ejecución del proyecto.

Viabilidad legal

Los cuestionamientos sobre licenciamiento de las fuentes de datos usadas se ven evidenciados en la sección de desarrollo del proyecto, y en cuanto a las bibliotecas con que se implementó el *framework* y codificación adicional, se comprobó que pudieran ser utilizados para ese propósito.

Apéndice B

Documentación del Programador

B.1. Introducción

En este apéndice se incluye la documentación necesaria para que los usuarios del *framework* `phishingutils` puedan hacer uso de los entregables implementados en el ámbito de este proyecto. Todos los recursos implementados se pueden encontrar en el repositorio del proyecto, alojado en GitHub, el conjunto de datos ha sido adicionalmente publicado en Kaggle para uso público. Para expandir sobre las instrucciones de esta documentación, se pueden verificar los cuadernos de ejemplo ubicados en la carpeta `./samples` del repositorio.

B.2. Estructura de directorios

A continuación se muestra una representación gráfica de los subdirectorios importantes del proyecto en el repositorio, y una pequeña descripción que se considera que vale la pena ser agregada para sensibilizar sobre contenido de interés dentro de los entregables.

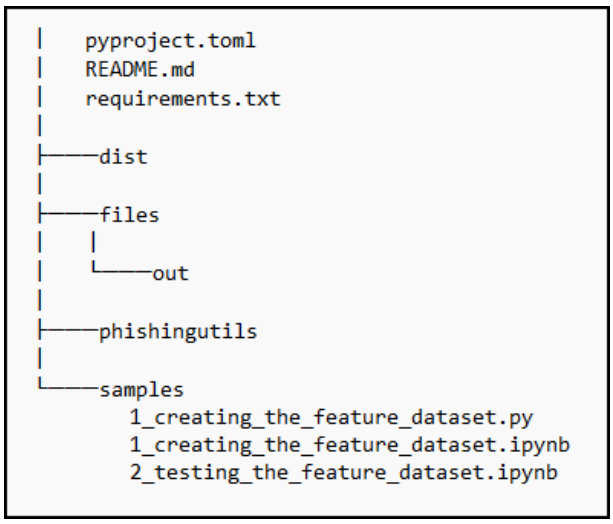

Figura B.1: Ficheros y subdirectorios pertenecientes al módulo `phishingutils` y utilidades

- En el directorio raíz, `pyproject.toml` y `requirements.txt` contienen las dependencias del paquete y las instrucciones que `pip` debe seguir para construirlo;
- En el directorio `dist` se encuentra la versión más reciente de los paquetes, disponibles para instalación;
- En el directorio `files/out` se encuentran las últimas versiones del conjunto de datos `out.csv.gz` y conjunto de apoyo `whois.csv.gz` con los registros WHOIS asociados al conjunto principal;
- En el directorio `phishingutils` se encuentra el código fuente del *framework* implementado;
- En el directorio `samples` se encuentran los ficheros de ejemplo de uso del *framework* en formato Python (`1_creating_the_feature_dataset.py`) y formato de cuaderno de Jupyter `1_creating_the_feature_dataset.ipynb`. También se encuentra el cuaderno de apoyo con los ejemplos de uso de las técnicas de clasificación usando el conjunto de datos resultante (fichero `2_testing_the_feature_dataset.ipynb`).

B.3. Manual del programador

En caso de requerir alguna aclaración sobre el funcionamiento de algun método o clase, se puede utilizar el comando `help()` en Python sobre la entidad requerida, por ejemplo para el paquete:

```
1 >>> import phishingutils
2 >>> help(phishingutils)
3 Help on package phishingutils:
4
```

```
NAME
    phishingutils

DESCRIPTION
    phishingutils
    ============

    Provides
        1. A way of creating datasets (corpus) for phishing URL detection
        via supervised learning, by defining models using high level
        attributes & features.
        2. Includes a collection of common functions deemed useful for
        defining features in a phishing URL detection dataset.
        3. Features dataset label balancing by default, and the
        option of exporting datasets to persistent files.
-- More --
```

B.4. Instalación del *framework* e importación del conjunto de datos

El paquete puede ser importado al clonar el repositorio, y a partir de su directorio raíz instalar usando `pip` las dependencias mediante `pip install -r ./requirements.txt`. Posteriormente crear el fichero `.py` donde se vaya a importar el módulo, usando:

```
from phishingutils.corpus.base import PhishingURLModel
import phishingutils.corpus.functions as fn
```

En caso de preferir la creación e instalación del paquete comprimiedo, puede hacerse tras la ejecución en terminal de los comandos `pip install -q build` y `python -m build` en el directorio raíz del paquete (lugar donde también debe constar el fichero `pyproject.toml`). Al finalizar la ejecución de esos dos ítems, en la carpeta `./dist` se crean los ficheros `phishingutils-0.1.0-py3-none-any.whl` y `phishingutils-0.1.0.tar.gz`, que pueden ser distribuibles directamente o a través de `pip`, el instalador de paquetes de Python, por ejemplo, usando el comando `pip install .`
`dist`
`phishingutils-0.1.0-py3-none-any.whl -user -force-reinstall` (en Windows, instalación forzada y aplicando únicamente a la sesión del usuario que lo ejecuta).

B.5. Pruebas de funcionamiento del *framework*

Tras instalación satisfactoria del paquete de Python, se puede ejecutar el siguiente código de prueba y resultará en la impresión de un dataset conformado por 10.000 URLs y sus propiedades:

```
import pandas as pd

```

```
3  from phishingutils.corpus.base import PhishingURLModel
4  import phishingutils.corpus.functions as fn
5
6  # Creating PhishingURLModel object
7  model = PhishingURLModel(
8      urls_col="url",
9      label_col="label",
10     sources={
11         "phishing": [
12             "PhishTank",
13             "Phishing.Database",
14             "OpenPhish-Community"
15         ],
16         "legitimate": [
17             "Cisco-Umbrella",
18             "Majestic"
19         ]
20     },
21     max_size=1e4,
22     sample_seed=123
23 )
24
25 model.add_map_col("url_length", fn.char_count)
26 model.add_map_col("starts_with_ip", fn.starts_with_ip_addr)
27 model.add_map_col("url_entropy", fn.url_entropy)
28 model.add_map_col("has_punycode", fn.has_punycode)
29 model.add_map_col("digit_letter_ratio", fn.digit_letter_ratio)
30 model.add_map_col("dot_count", lambda x: fn.expr_counter(url=x, expr=r"\.")
       )
31 model.add_map_col("at_count", lambda x: fn.expr_counter(url=x, expr=r"@"))
32 model.add_map_col("dash_count", lambda x: fn.expr_counter(url=x, expr=r"-")
       )
33 model.add_map_col("tld_count", fn.tld_count)
34 model.add_map_col("domain_has_digits", fn.domain_has_digits)
35 model.add_map_col("subdomain_count", fn.subdomain_count)
36 model.add_map_col("nan_char_entropy", fn.nan_char_entropy)
37 model.add_map_col("has_internal_links", fn.has_internal_links)
38 model.add_lookup_col("whois_data", fn.get_domain_info)
39 model.add_lookup_col("domain_age_days", fn.domain_age)
40
41 # Use this method to create the dataset in runtime
42 df = model.setup()
43 # or alternatively, export a persistent file for later use:
44 # model.export("./files/out/out.csv.gz")
45
46 display(df)
```

Extracto B.1: Ejemplo de uso del *framework* `phishingutils`

Bibliografía

[1] Abdelnabi, S., Krombholz, K., and Fritz, M. Visualphishnet: Zero-day phishing website detection by visual similarity. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA, 2020), CCS '20, Association for Computing Machinery, p. 16811698.

[2] Abdillah, R., Shukur, Z., Mohd, M., and Murah, T. M. Z. Phishing classification techniques: A systematic literature review. *IEEE Access 10* (2022), 41574–41591.

[3] Abuadbba, A., Wang, S., Almashor, M., Ahmed, M. E., Gaire, R., Camtepe, S., and Nepal, S. Towards web phishing detection limitations and mitigation, 2022.

[4] Adane, K., Beyene, B., and Abebe, M. Single and hybrid-ensemble learning-based phishing website detection: Examining impacts of varied nature datasets and informative feature selection technique. *Digital Threats 4*, 3 (10 2023).

[5] Adap, V. A. V., Castillo, G. A., Delos Reyes, E. J. M., Ronquillo, E. B., and Vea, L. A. Do not feed the phish: Phishing website detection using url-based features. In *Proceedings of the 2023 5th World Symposium on Software Engineering* (New York, NY, USA, 2023), WSSE '23, Association for Computing Machinery, p. 135141.

[6] Akinyelu, A. A. Machine learning and nature inspired based phishing detection: A literature survey. *International Journal on Artificial Intelligence Tools 28*, 05 (2019), 1930002.

[7] AlSabah, M., Nabeel, M., Boshmaf, Y., and Choo, E. Content-agnostic detection of phishing domains using certificate transparency and passive dns. In *Proceedings of the 25th International Symposium on Research in Attacks, Intrusions and Defenses* (New York, NY, USA, 2022), RAID '22, Association for Computing Machinery, p. 446459.

[8] Alsarhan, A., Igried, B., Bani Saleem, R. M., Alauthman, M., and Aljaidi, M. Enhancing phishing url detection: A comparative study of machine learning algorithms. In *Proceedings of the 2023 Asia Conference on Artificial Intelligence, Machine Learning and Robotics* (New York, NY, USA, 2023), AIMLR '23, Association for Computing Machinery.

[9] Apache Spark. Mllib: Main guide - spark 3.5.2 documentation, 2024. [Internet; accedido a 27/08/2024].

[10] Apache Spark. What is supervised learning? | google cloud, 2024. [Internet; accedido a 14/09/2024].

[11] Aung, E. S., and Yamana, H. Url-based phishing detection using the entropy of non-alphanumeric characters. In *Proceedings of the 21st International Conference on Information Integration and Web-Based Applications & Services* (New York, NY, USA, 2020), iiWAS2019, Association for Computing Machinery, p. 385392.

[12] Aung, E. S., and Yamana, H. Segmentation-based phishing url detection. In *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology* (New York, NY, USA, 2022), WI-IAT '21, Association for Computing Machinery, p. 550556.

[13] Burkov, A. *The Hundred-Page Machine Learning Book*. Andriy Burkov, 2019.

[14] Castaño, F., Fidalgo, E., Alegre, E., Chaves, D., and Sanchez-Paniagua, M. State of the art: Content-based and hybrid phishing detection, 2021.

[15] Chinn, P. L. The traditional literature review. *Nurse Author & Editor 31*, 3-4 (2021), 62–64.

[16] Cisco Umbrella. Cisco Umbrella Popularity List, 2024. [Internet; accedido a 26/04/2024].

[17] Community, M. What is a URL? - Learn web development | MDN, 2024. [Internet; accedido a 15/09/2024].

[18] Cuzzocrea, A., Martinelli, F., and Mercaldo, F. A machine-learning framework for supporting intelligent web-phishing detection and analysis. In *Proceedings of the 23rd International Database Applications & Engineering Symposium* (New York, NY, USA, 2019), IDEAS '19, Association for Computing Machinery.

[19] European Union Agency for Cybersecurity. ENISA Threat Landscape. Report/Study, European Union Agency for Cybersecurity, 2023.

[20] Goodger, D., and van Rossum, G. Pep 257 - docstring conventions | peps.python.org, 2001. [Internet; accedido a 09/09/2024].

[21] Gupta, B. B., Yadav, K., Razzak, I., Psannis, K., Castiglione, A., and Chang, X. A novel approach for phishing urls detection using lexical based machine learning in a real-time environment. *Computer Communications 175* (2021), 47–57.

[22] Hajizada, A., and Jahan, S. Feature selections for phishing urls detection using combination of multiple feature selection methods. In *Proceedings of the 2023 15th International Conference on Machine Learning and Computing* (New York, NY, USA, 2023), ICMLC '23, Association for Computing Machinery, p. 444450.

[23] Kim, T., Park, N., Hong, J., and Kim, S.-W. Phishing url detection: A network-based approach robust to evasion. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA, 2022), CCS '22, Association for Computing Machinery, p. 17691782.

[24] Krog, M. mitchellkrogza/Phishing.Database, June 2024. original-date: 2018-04-20T11:52:18Z.

[25] Logilab and Pylint contributors. Pylint 3.3.0-dev0 documentation, 2024. [Internet; accedido a 16/08/2024].

[26] Majestic. Majestic Million, 2024. [Internet; accedido a 26/04/2024].

[27] Moore, M. Top Cybersecurity Threats in 2023 - University of San Diego Online Degrees, 2024. [Internet; accedido a 26/04/2024].

[28] NumPy team. numpy/numpy: The fundamental package for scientific computing with python., 2024. [Internet; accedido a 09/09/2024].

[29] Ojewumi, T., Ogunleye, G., Oguntunde, B., Folorunsho, O., Fashoto, S., and Ogbu, N. Performance evaluation of machine learning tools for detection of phishing attacks on web pages. *Scientific African 16* (2022), e01165.

[30] OpenPhish. OpenPhish - Phishing Intelligence, 2024. [Internet; accedido a 26/04/2024].

[31] Paliath, S., Qbeitah, M. A., and Aldwairi, M. Phishout: Effective phishing detection using selected features. In *2020 27th International Conference on Telecommunications (ICT)* (2020), pp. 1–5.

[32] Patel, H., Rehman, U., and Iqbal, F. Evaluating the efficacy of large language models in identifying phishing attempts, 2024.

[33] PhishTank. PhishTank | Join the fight against phishing, 2024. [Internet; accedido a 26/04/2024].

[34] PyPA. Pypa specifications - python packaging user guide, 2024. [Internet; accedido a 09/09/2024].

[35] Rashid, F., Doyle, B., Han, S. C., and Seneviratne, S. Phishing url detection generalisation using unsupervised domain adaptation. *Computer Networks 245* (2024), 110398.

[36] richardpenman @ GitHub. Problem with .ru zone ů Issue 207 ů richardpenman/whois, 2024. [Internet; accedido a 24/06/2024].

[37] Safi, A., and Singh, S. A systematic literature review on phishing website detection techniques. *Journal of King Saud University - Computer and Information Sciences 35*, 2 (2023), 590–611.

[38] scikit-learn developers. Api reference - scikit-learn 1.5.2 documentation, 2024. [Internet; accedido a 27/08/2024].

[39] scikit-learn developers. Various decision tree algorithms: 1.10 Decision Trees - scikit-learn stable documentation, 2024. [Internet; accedido a 16/09/2024].

[40] Shirazi, H., Bezawada, B., and Ray, I. "kn0w thy doma1n name": Unbiased phishing detection using domain name based features. In *Proceedings of the 23nd ACM on Symposium on Access Control Models and Technologies* (New York, NY, USA, 2018), SACMAT '18, Association for Computing Machinery, p. 6975.

[41] Siddiq, M. A. A., Arifuzzaman, M., and Islam, M. S. Phishing website detection using deep learning. In *Proceedings of the 2nd International Conference on Computing Advancements* (New York, NY, USA, 2022), ICCA '22, Association for Computing Machinery, p. 8388.

[42] Sirigineedi, S. S., Soni, J., and Upadhyay, H. Learning-based models to detect runtime phishing activities using urls. In *Proceedings of the 2020 4th International Conference on Compute and Data Analysis* (New York, NY, USA, 2020), ICCDA '20, Association for Computing Machinery, p. 102106.

[43] Vajrobol, V., Gupta, B. B., and Gaurav, A. Mutual information based logistic regression for phishing url detection. *Cyber Security and Applications 2* (2024), 100044.

[44] Valentim, R., Drago, I., Trevisan, M., Cerutti, F., and Mellia, M. Augmenting phishing squatting detection with gans. In *Proceedings of the CoNEXT Student Workshop* (New York, NY, USA, 2021), CoNEXT-SW '21, Association for Computing Machinery, p. 34.

[45] van Geest, R., Cascavilla, G., Hulstijn, J., and Zannone, N. The applicability of a hybrid framework for automated phishing detection. *Computers & Security 139* (2024), 103736.

[46] Vijayalakshmi, M., Mercy Shalinie, S., Yang, M. H., and U., R. M. Web phishing detection techniques: a survey on the stateoftheart, taxonomy and future directions. *IET Networks 9*, 5 (9 2020), 235246.

[47] Vo Quang, M., Bui Tan Hai, D., Tran Kim Ngoc, N., Ngo Duc Hoang, S., Nguyen Huu, Q., Phan The, D., and Pham, V.-H. Shark-eyes: A multimodal fusion framework for multi-view-based phishing website detection. In *Proceedings of the 12th International Symposium on Information and Communication Technology* (New York, NY, USA, 2023), SOICT '23, Association for Computing Machinery, p. 793800.

[48] Vrbani, G., Fister, I., and Podgorelec, V. Datasets for phishing websites detection. *Data in Brief 33* (2020), 106438.

[49] Yigit, Y., Buchanan, W. J., Tehrani, M. G., and Maglaras, L. Review of generative ai methods in cybersecurity, 2024.

[50] Zeng, V., Zhou, X., Baki, S., and Verma, R. M. Phishbench 2.0: A versatile and extendable benchmarking framework for phishing. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA, 2020), CCS '20, Association for Computing Machinery, p. 20772079.

[51] Zhu, E., Cheng, K., Zhang, Z., and Wang, H. Pdhf: Effective phishing detection model combining optimal artificial and automatic deep features. *Computers & Security 136* (2024), 103561.