UNIVERSIDAD PRIVADA ANTENOR ORREGO

FACULTAD DE INGENIERÍA

PROGRAMA DE ESTUDIO DE INGENIERÍA DE SISTEMAS E INTELIGENCIA ARTIFICIAL

INTELIGENCIA ARTIFICIAL PRINCIPIOS Y TÉCNICAS

"CyberSentinel: Detección de Phishing con IA"

AUTORES:

- Casanova Chumpitaz, Angel
- Centa Fernandez, Mayli
- Mariños Gonzalez, Bryan
- Medina Sixce, Darli
- Nieve Viera, Daniela
- Sanchez Maradiegue, Joaquin

DOCENTE:

- Ing. Sagastegui Chigne, Hernan

Trujillo - Perú

2025

ÍNDICE

Lista de Contenido

1. Introducción

1.1 Título del Proyecto

“CyberSentinel: Detección de Phishing con Inteligencia Artificial“

1.2 Antecedentes

El phishing es actualmente una de las problemáticas más relevantes en el campo de la ciberseguridad. Este término, abreviación de la expresión *password harvesting fishing*, se basa en la creación de réplicas falsas de páginas web de instituciones legítimas, con la finalidad de engañar al usuario y obtener de manera ilícita información sensible como contraseñas, números de tarjetas de crédito, documentos de identidad u otros datos de valor para los atacantes (Mesa, D. S., & Tabares, R. B. 2015, citado en Usera, 2007). Desde su aparición en la década de los noventa, este delito informático ha experimentado una evolución constante, tanto en volumen como en sofisticación, lo que ha incrementado el riesgo y la vulnerabilidad de los sistemas digitales contemporáneos.

En los últimos años, diversos informes internacionales han evidenciado un crecimiento sostenido en los incidentes de phishing. Organizaciones como la *Anti-Phishing Working Group (APWG)* registraron cifras récord durante el periodo (2021 – 2022), con millones de intentos detectados a nivel global, afectando especialmente a los sectores financiero, de comercio electrónico y de servicios digitales. En América Latina, países como Brasil, México y Perú se ubican entre los más afectados, debido a la acelerada digitalización de servicios financieros y a la limitada cultura de ciberseguridad en amplios sectores de la población. Como hacen mención Diaz & Goitia (2024): “A nivel nacional en el año 2024, se han registrado un millón de detecciones únicas de phishing en Perú”. Esta situación pone de manifiesto no sólo la magnitud del problema, sino también la urgencia de implementar mecanismos de protección más eficientes en el país.

Las repercusiones de este tipo de ataques son múltiples. A nivel individual, los usuarios enfrentan la pérdida de acceso a sus cuentas digitales y el riesgo de robo de identidad. En el ámbito corporativo, el impacto incluye pérdidas económicas significativas, filtración de información confidencial y un deterioro de la confianza en las marcas (Crume Jeff, 2025). El reporte de IBM señala que el costo promedio de una filtración de datos originada por phishing asciende a 4,91 millones de dólares, consolidándose como una de las amenazas más costosas a nivel global.

Los enfoques tradicionales de detección resultan limitados ante la constante evolución de los ataques de phishing. En este sentido, Hernández Domínguez, A., & Baluja García, W. (2021) señalan que “Una práctica común es la utilización de bases de datos (lista negra y lista blanca), los cuales reflejan una efectividad de detección de ataques de phishing en el intervalo de un 47% a un 83%, como

promedio", lo que evidencia la insuficiencia de este tipo de medidas frente a la adaptabilidad de los ciberdelincuentes. Los atacantes modifican constantemente sus estrategias, utilizando técnicas de enmascaramiento y variaciones mínimas en los dominios o contenidos de los correos, lo que les permite evadir mecanismos convencionales de detección. Esto convierte al phishing en una amenaza altamente adaptable y difícil de contrarrestar mediante herramientas exclusivamente reactivas.

Según Atuncar et al. (2024) "Gracias a su alta capacidad de análisis de patrones y comportamientos, la IA facilita la identificación de indicios de ataques de ingeniería social con una precisión notablemente mayor." En este contexto, la incorporación de nuevas metodologías basadas en inteligencia artificial se presenta como una alternativa eficaz y necesaria. El uso de algoritmos de *Machine Learning* y de técnicas de *Procesamiento de Lenguaje Natural* permite desarrollar sistemas con capacidad de aprendizaje y adaptación continua, capaces de identificar patrones sutiles en URLs y en el contenido textual de los sitios web. Como mencionan Farias & Centeno (2024) "Su capacidad para analizar grandes volúmenes de datos y patrones anómalos ha mejorado significativamente la seguridad.". A diferencia de los mecanismos tradicionales, estas soluciones proactivas ofrecen mayores niveles de precisión y pueden anticiparse a nuevas variantes de ataque, aportando un valor agregado en términos de seguridad y prevención. Investigaciones recientes confirman que los modelos de deep learning pueden superar el 95% de precisión en la detección de URLs maliciosas, sobrepasando ampliamente a las técnicas tradicionales (Kaur & Jain, 2025).

Por lo tanto, el estudio y desarrollo de sistemas de detección de phishing sustentados en técnicas de inteligencia artificial responden no solo a una necesidad tecnológica, sino también a un imperativo académico y social. Su implementación permite proteger de manera más efectiva a usuarios y organizaciones, al tiempo que fortalece la confianza en los entornos digitales. Asimismo, en el ámbito formativo, constituye una oportunidad para aplicar de manera práctica los conocimientos adquiridos en áreas como el aprendizaje supervisado, la clasificación de datos y el análisis de incertidumbre, consolidando la pertinencia del proyecto en el marco del presente curso.

1.3 Problema a Resolver

¿Cómo implementar un sistema automatizado y preciso, basado en inteligencia artificial, que detecte sitios web de phishing para reducir fraudes digitales y proteger a los usuarios?

1.4 Objetivos

1.4.1. Objetivo General

Desarrollar un sistema de detección de sitios web de phishing basado en técnicas de inteligencia artificial, capaz de analizar las características de las URLs y el contenido de las páginas, con el fin de clasificarlas como legítimas o fraudulentas. El proyecto busca ofrecer una herramienta automatizada, precisa y adaptable que contribuya a la reducción de fraudes digitales y a la protección de los usuarios en entornos virtuales.

1.4.2. Objetivo Específico

- Identificar y documentar los requerimientos funcionales y no funcionales del sistema, garantizando que la solución responda a las necesidades técnicas, de seguridad y de usabilidad.
- Definir la arquitectura tecnológica y lógica del sistema, especificando la estructura de datos, los módulos de procesamiento y la integración del modelo de IA con la interfaz de usuario.
- Recopilar y preprocesar un conjunto de datos representativo de sitios web legítimos y de phishing, extrayendo características relevantes de sus URLs y contenidos para el entrenamiento del modelo.
- Desarrollar y entrenar diversos modelos de *machine learning* (árboles de decisión, Random Forest y LightGBM) para la clasificación de sitios web.
- Evaluar y comparar el desempeño de los modelos utilizando métricas como (Recall, Precision, F1-score y AUC-PR) seleccionando el más adecuado y validando su efectividad en escenarios simulados.
- Desarrollar una interfaz funcional (plugin de navegador o aplicación sencilla) que integre el modelo seleccionado, demostrando su utilidad práctica en la detección en tiempo real de intentos de phishing.

II. Requerimientos

2.1 Definición del Dominio

El sistema propuesto pertenece al dominio de la ciberseguridad aplicada mediante inteligencia artificial, con énfasis en la detección automatizada de sitios web de phishing. Este dominio abarca técnicas de análisis de URLs, procesamiento de lenguaje natural (NLP), aprendizaje automático (ML) supervisado, evaluación de métricas de clasificación, y desarrollo de herramientas software para la protección de usuarios digitales. La solución también se sitúa en el ámbito de las aplicaciones prácticas de sistemas expertos para mitigar amenazas emergentes en entornos web.

2.2 Determinación de Requisitos (Requerimientos)

2.2.1. Requisitos Funcionales

- **RF1:** El sistema debe permitir al usuario ingresar una URL a través de la interfaz para su posterior análisis.
- **RF2:** El sistema debe extraer y procesar automáticamente características relevantes tanto de la URL como del contenido de la página (metadatos, tokens de texto, scripts, etc.), en caso de estar disponible.
- **RF3:** El sistema debe aplicar modelos de aprendizaje automático para clasificar la URL como legítima, sospechosa o maliciosa (phishing).
- **RF4:** El sistema debe presentar al usuario el resultado de la clasificación acompañado de un indicador visual y textual.
- **RF5:** El sistema debe mostrar el nivel de confianza o probabilidad de la predicción generada por el modelo.
- **RF6:** El sistema debe almacenar un historial de análisis realizados, incluyendo fecha, URL evaluada, resultado y métricas asociadas.
- **RF7:** El sistema debe permitir su implementación como un plugin de navegador o como una aplicación web ligera que integre el modelo de IA en tiempo real.
- **RF8:** El sistema debe contar con un mecanismo para actualizar el modelo de IA mediante reentrenamiento periódico con nuevos datos etiquetados.
- **RF9:** El sistema debe permitir exportar reportes de resultados en formatos estándar (CSV, PDF) para su análisis externo.

- **RF10:** El sistema debe notificar al usuario cuando una URL sea clasificada como phishing con un nivel alto de confianza.

- **RF11:** El sistema debe ser capaz de analizar automáticamente las URLs visitadas por el usuario en el navegador para prevenir accesos a sitios fraudulentos.

Figura 1. Arquitectura funcional del sistema CyberSentinel

Con el fin de representar de manera clara la relación entre los Requisitos Funcionales (RF) y los componentes del sistema, se ha diseñado la siguiente arquitectura funcional.

Esta estructura permite visualizar la interacción entre el cliente (usuario final), el sistema (frontend y backend) y los módulos especializados encargados del procesamiento, seguridad, almacenamiento y registro de eventos.

Interpretación

Esta arquitectura refleja la alineación entre los componentes funcionales y los requisitos definidos en el proyecto.

El flujo de información se inicia desde la interacción del usuario, pasa por el procesamiento en el backend (que ejecuta las tareas de análisis e inferencia del modelo IA) y culmina con la respuesta visual en la interfaz, garantizando seguridad, trazabilidad y modularidad en cada capa del sistema.

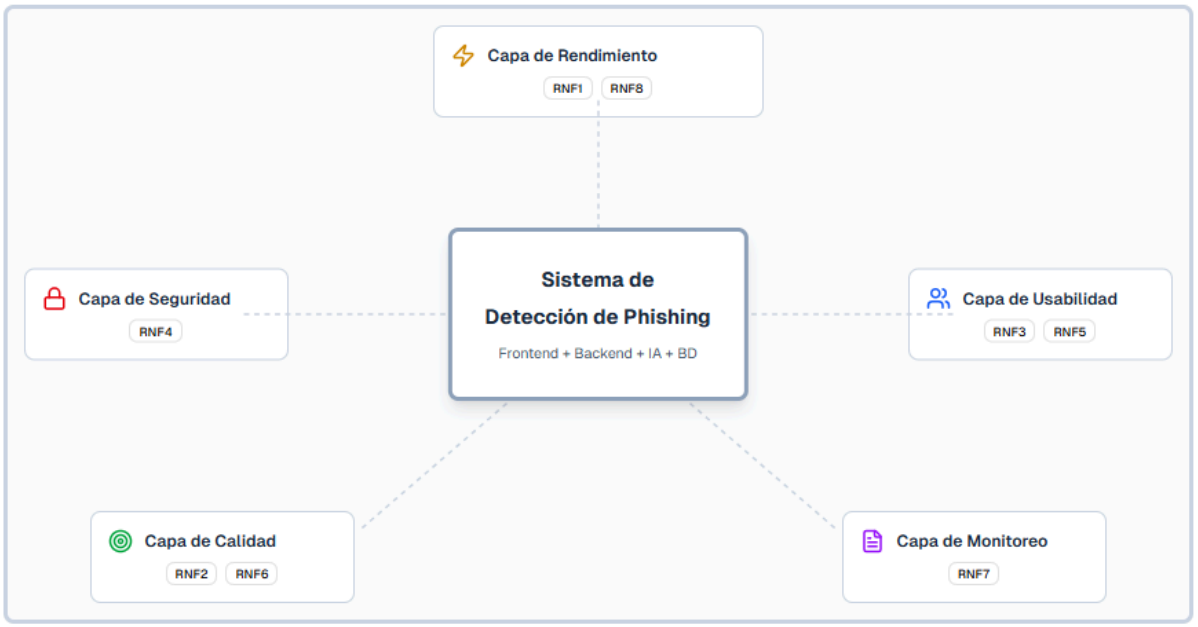
2.2.2. Requisitos No Funcionales

- **RNF1:** El tiempo de respuesta del sistema ante una URL ingresada no debe superar los 2 segundos, asegurando una experiencia fluida para el usuario.
- **RNF2:** El modelo de detección debe alcanzar una precisión mínima del 90%, validada mediante técnicas de validación cruzada y conjuntos de prueba representativos.
- **RNF3:** La interfaz del sistema debe ser intuitiva, accesible y compatible con principios de usabilidad universal, permitiendo su uso por personas con conocimientos básicos.
- **RNF4:** El sistema debe cumplir con estándares de ciberseguridad en el manejo de los datos procesados, incluyendo el uso de canales seguros (HTTPS), sanitización de entradas y anonimización de registros cuando sea necesario.
- **RNF5:** El sistema debe garantizar compatibilidad con navegadores web ampliamente utilizados como Google Chrome, Mozilla Firefox y Microsoft Edge.
- **RNF6:** La arquitectura del sistema debe ser modular y desacoplada, permitiendo la actualización o sustitución del modelo de IA sin afectar otras funcionalidades del sistema.
- **RNF7:** El sistema debe contar con un registro (log) detallado de las operaciones realizadas, errores detectados y eventos críticos, con fines de auditoría y mantenimiento.
- **RNF8:** El sistema debe ser capaz de procesar múltiples solicitudes concurrentes (ej. hasta 1000 URLs por minuto en un entorno de producción) sin degradación significativa.

Figura 2. Arquitectura de Requisitos No Funcionales del Sistema CyberSentinel

Con el propósito de evidenciar la correspondencia entre los Requisitos No Funcionales (RNF) y las propiedades de calidad del sistema, se presenta la arquitectura por capas mostrada en la siguiente figura.

Esta estructura ilustra cómo se organizan los aspectos de rendimiento, seguridad, calidad, usabilidad y monitoreo dentro del sistema CyberSentinel: Detección de Phishing con Inteligencia Artificial.



Interpretación

La arquitectura de Requisitos No Funcionales complementa la funcionalidad del sistema al incorporar atributos de calidad transversales que aseguran su eficiencia, seguridad, estabilidad y usabilidad.

Cada capa representa una dimensión crítica del diseño no funcional, garantizando que el sistema CyberSentinel no solo cumpla con sus funciones operativas, sino que lo haga bajo criterios de desempeño y confiabilidad sostenibles.

III. Pre-Procesamiento y Normalización

3.1 Medidas, Datos, Bases de Datos y Elaboración del Data-Set

Para el proyecto **"CyberSentinel: Detección de Phishing con Inteligencia Artificial"**, se utiliza un dataset proveniente de fuentes públicas confiables como **Kaggle** y **Mendeley**, el cual está disponible en formato CSV. Este dataset contiene **450,176 registros** con dos columnas principales:

1. **url**: Representa las direcciones de los sitios web.

2. **type**: La etiqueta que clasifica si la URL es **"legitimate"** (legítima) o **"phishing"** (fraudulenta).

El dataset ha sido cuidadosamente seleccionado debido a su tamaño, diversidad y la relevancia de los sitios web incluidos, lo que lo convierte en un recurso adecuado para entrenar modelos de **inteligencia artificial** con el objetivo de detectar sitios de phishing.

A continuación se presentan las primeras 5 filas del dataset para ilustrar mejor su estructura:

url	**type**
https://www.google.com	legitimate
https://www.youtube.com	legitimate
https://www.facebook.com	legitimate
https://www.baidu.com	legitimate
https://www.wikipedia.org	legitimate

En total, el dataset incluye **450,176 registros**, que cubren una amplia variedad de sitios legítimos y fraudulentos, lo que proporcionará una base sólida para el entrenamiento y evaluación del modelo. No se han encontrado valores nulos en las columnas, lo que simplifica el preprocesamiento inicial.

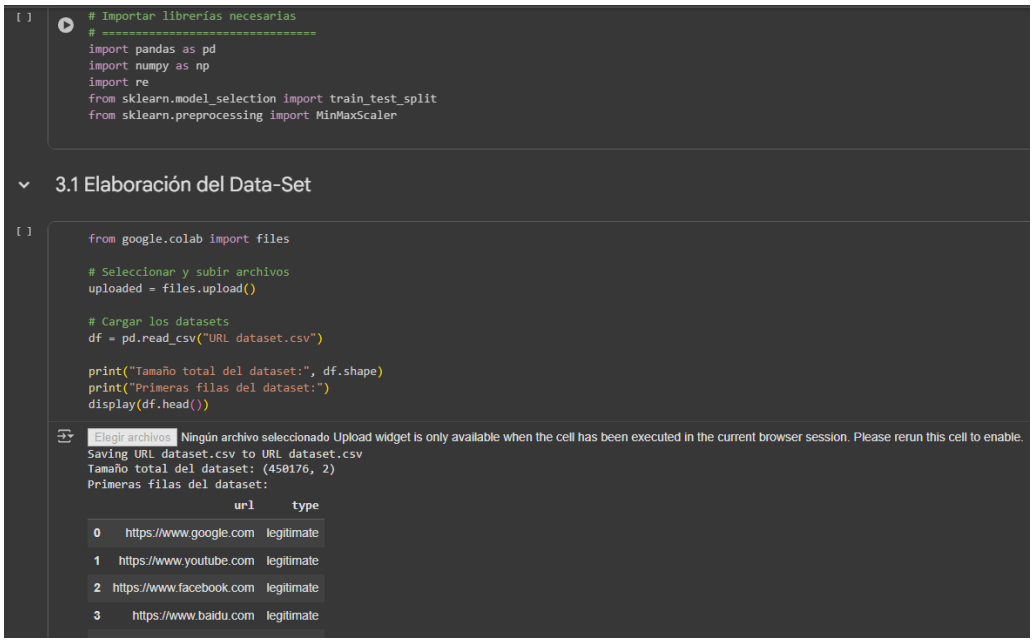
Elaboración del Data-Set:

El dataset será organizado en tres subconjuntos para asegurar una correcta evaluación y entrenamiento del modelo:

1. **Data-Set de Entrenamiento (70%)**: Este subconjunto será utilizado para entrenar el modelo de IA, garantizando que el modelo aprenda a distinguir entre URLs legítimas y fraudulentas.
2. **Data-Set de Pruebas (15%)**: Se usará para evaluar el desempeño del modelo una vez entrenado, asegurando que se mida de forma justa con datos no vistos.
3. **Data-Set de Validación (15%)**: Este subconjunto ayudará en el proceso de validación cruzada, lo que permite ajustar los parámetros del modelo y minimizar el riesgo de sobreajuste.

Fuentes:

- Dataset de phishing de Kaggle: Phishing Site URLs
- Dataset de phishing de Mendeley: Phishing Website Dataset



```
# Importar librerías necesarias
# ================================
import pandas as pd
import numpy as np
import re
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
```

3.1 Elaboración del Data-Set

```
from google.colab import files

# Seleccionar y subir archivos
uploaded = files.upload()

# Cargar los datasets
df = pd.read_csv("URL dataset.csv")

print("Tamaño total del dataset:", df.shape)
print("Primeras filas del dataset:")
display(df.head())
```

```
Elegir archivos  Ningún archivo seleccionado Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving URL dataset.csv to URL dataset.csv
Tamaño total del dataset: (450176, 2)
Primeras filas del dataset:
```

	url	type
0	https://www.google.com	legitimate
1	https://www.youtube.com	legitimate
2	https://www.facebook.com	legitimate
3	https://www.baidu.com	legitimate

Interpretación:

- Al cargar URL dataset.csv, observamos el tamaño total del conjunto de datos, que incluye las columnas url y type.
- El conteo de registros confirma la disponibilidad de suficientes ejemplos tanto de URLs legítimas como de phishing, lo cual es importante para entrenar un modelo balanceado.

- La previsualización de las primeras filas permite verificar que los datos están en el formato esperado (dirección web y etiqueta).

3.2 Normalización y/o Filtrado de Datos

El proceso de normalización y filtrado de datos es esencial para garantizar que los modelos de IA no sean sesgados por diferencias de escala entre las características. Este paso asegura que el modelo pueda aprender de manera eficiente sin que una variable sobresale sobre otras debido a su rango de valores.

1. **Conversión de la columna type a valores numéricos**:
   La columna type contiene valores categóricos (legítima, phishing). Para que el modelo pueda trabajar con esta información, es necesario convertirla a valores binarios:

   - **0** para **phishing**.
   - **1** para **legítimo**.

2. **Extracción de características de las URLs**:
   Dado que las URLs son cadenas de texto y no directamente utilizables por los algoritmos de IA, es necesario extraer características relevantes. Estas características pueden incluir:
   - **Longitud de la URL**: Las URLs de phishing suelen ser más largas o complejas que las legítimas.
   - **Número de subdominios**: Los sitios de phishing tienden a utilizar subdominios extra para imitar sitios legítimos.
   - **Caracteres especiales**: Los ataques de phishing frecuentemente incorporan caracteres especiales como @, _, - para engañar al usuario.
   - **Presencia de palabras clave**: Palabras como "login", "account", "password" son indicadores típicos de phishing.
3. **Normalización de características numéricas**:
   Después de extraer las características, es necesario normalizarlas para garantizar que todas las variables tengan un rango común y ningún valor sobresalga excesivamente. Usaremos la normalización Min-Max, que ajusta los valores numéricos a un rango entre 0 y 1. Esta técnica es especialmente útil cuando las características presentan rangos no uniformes.
4. **Filtrado de datos**:
   Aunque el dataset no contiene valores nulos, es una buena práctica revisar la calidad de los datos para identificar posibles entradas erróneas o mal formadas. Cualquier URL inconsistente o defectuosa será eliminada del conjunto de datos para evitar que afecte el entrenamiento del modelo.

Método recomendado:

- **Normalización Min-Max**: Esta técnica es ideal cuando se trabaja con características que tienen rangos amplios y muy distintos entre sí. Por ejemplo, la longitud de la URL puede variar enormemente de un sitio a otro. Al usar **Min-Max Scaling**, todas las características estarán en un rango [0, 1], lo que asegura que cada una tenga un impacto similar en el modelo.

```python
# --- (a) Conversión de la columna 'type' a valores numéricos ---
# phishing → 0 | legitimate → 1
df['type'] = df['type'].map({'phishing': 0, 'legitimate': 1})

# --- (b) Extracción de características de las URLs ---
def extract_features(url):
    features = {}
    features['url_length'] = len(url)
    features['num_subdomains'] = url.count('.') - 1

    # Caracteres especiales comunes en phishing
    special_chars = ['@', '-', '_', '?', '=', '&']
    for c in special_chars:
        features[f'count_{c}'] = url.count(c)

    # Palabras clave sospechosas
    keywords = ['login', 'account', 'verify', 'update', 'password']
    for word in keywords:
        features[f'keyword_{word}'] = 1 if word in url.lower() else 0

    return pd.Series(features)

# Aplicar extracción de características
features_df = df['url'].apply(extract_features)

# Unir características con etiquetas
final_df = pd.concat([features_df, df['type']], axis=1)

# --- (c) Normalización de características numéricas ---
scaler = MinMaxScaler()
features_scaled = scaler.fit_transform(final_df.drop(columns=['type']))
features_scaled = pd.DataFrame(features_scaled, columns=features_df.columns)

# Dataset final normalizado
final_df = pd.concat([features_scaled, final_df['type']], axis=1)

# --- (d) Filtrado de datos ---
final_df = final_df.dropna()
print("Dataset después de normalización y filtrado:", final_df.shape)
```

```
Dataset después de normalización y filtrado: (450176, 14)
```

Interpretación:

- La columna type fue convertida a valores numéricos: 0 para phishing y 1 para legítimo, lo que facilita su uso en algoritmos de Machine Learning.
- A partir de cada URL se extrajeron características relevantes como la longitud de la URL, número de subdominios, presencia de caracteres especiales y palabras clave sospechosas.
- Posteriormente, se aplicó la normalización Min-Max, asegurando que todas las características numéricas estén en el rango [0,1]. Esto evita que unas variables dominen sobre otras debido a diferencias de escala.
- Finalmente, se verificó la calidad de los datos, eliminando posibles registros defectuosos o vacíos, dejando un dataset limpio y consistente para la siguiente fase.

3.3 Planteamiento de la Organización del Data-Set

Una correcta **organización del dataset** es fundamental para obtener un modelo robusto y capaz de generalizar bien en datos no vistos. Para ello, utilizaremos técnicas de **validación cruzada** que nos permitirán evaluar el modelo de manera justa y obtener mejores resultados.

La organización del dataset será la siguiente:

1. **Data-Set de Entrenamiento (70%)**: Este conjunto servirá para enseñar al modelo las características de las URLs legítimas y de phishing. El entrenamiento del modelo consistirá en exponer a estas URLs junto con sus respectivas etiquetas (legítimo o phishing).

2. **Data-Set de Pruebas (15%)**: Se utilizará para evaluar la precisión del modelo en datos que no ha visto anteriormente. Este conjunto es vital para medir la efectividad del modelo una vez que esté entrenado.

3. **Data-Set de Validación ("Cross-Validation", 15%)**: Durante el proceso de entrenamiento, se realizará **validación cruzada** para verificar que el modelo no se sobre ajuste a los datos de entrenamiento. En este proceso, el dataset se divide en `k` subconjuntos, y el modelo es entrenado y evaluado `k` veces, asegurando que se logre una evaluación más robusta.

La técnica de **validación cruzada** es importante para evitar que el modelo dependa demasiado de un subconjunto específico de datos y sea capaz de generalizar bien en escenarios reales.

Método recomendado:

- **K-fold Cross-Validation**: Esta técnica divide el dataset en `k` subconjuntos, y en cada iteración el modelo se entrena con `k-1` subconjuntos y se valida en el subconjunto restante. Este enfoque asegura una evaluación más completa y reduce el riesgo de sobreajuste.

```python
from sklearn.model_selection import train_test_split, KFold

# --- División en subconjuntos: 70% entrenamiento, 15% validación, 15% prueba ---

# Separar variables predictoras (X) y variable objetivo (y)
X = final_df.drop(columns=['type'])
y = final_df['type']

# Dividir el dataset en entrenamiento (70%) y temporal (30%)
X_train, X_temp, y_train, y_temp = train_test_split(
    X, y,
    test_size=0.30,
    stratify=y,          # mantiene balance entre phishing y legítimo
    random_state=42
)

# Dividir el 30% restante en validación (15%) y prueba (15%)
X_val, X_test, y_val, y_test = train_test_split(
    X_temp, y_temp,
    test_size=0.50,
    stratify=y_temp,
    random_state=42
)

# --- Mostrar proporciones ---
print("📊 Distribución de subconjuntos:")
print(f"Entrenamiento: {X_train.shape[0]} registros ({(len(X_train)/len(final_df))*100:.1f}%)")
print(f"Validación:    {X_val.shape[0]} registros ({(len(X_val)/len(final_df))*100:.1f}%)")
print(f"Prueba:        {X_test.shape[0]} registros ({(len(X_test)/len(final_df))*100:.1f}%)")

# --- Mostrar balance de clases en cada subconjunto ---
print("\n⚖️  Balance de clases (0=Phishing, 1=Legitimo):")
print("Entrenamiento:\n", y_train.value_counts(normalize=True))
print("\nValidación:\n", y_val.value_counts(normalize=True))
print("\nPrueba:\n", y_test.value_counts(normalize=True))

# --- Configuración de validación cruzada K-Fold ---
k = 10  # número de particiones
kfold = KFold(n_splits=k, shuffle=True, random_state=42)

print(f"\n✅ Validación cruzada configurada con {k} particiones (K={k}).")

# Mostrar ejemplo de iteraciones (solo las 3 primeras para no saturar)
fold = 1
for train_index, val_index in kfold.split(X_train):
    print(f"Fold {fold}: Entrenamiento = {len(train_index)} | Validación = {len(val_index)}")
    if fold == 3:
        break
    fold += 1
```

```
📊 Distribución de subconjuntos:
Entrenamiento: 315123 registros (70.0%)
Validación:    67526 registros (15.0%)
Prueba:        67527 registros (15.0%)

⚖️  Balance de clases (0=Phishing, 1=Legítimo):
Entrenamiento:
 type
1    0.768005
0    0.231995
Name: proportion, dtype: float64

Validación:
 type
1    0.768015
0    0.231985
Name: proportion, dtype: float64

Prueba:
 type
1    0.768004
0    0.231996
Name: proportion, dtype: float64

✅ Validación cruzada configurada con 10 particiones (K=10).
Fold 1: Entrenamiento = 283610 | Validación = 31513
Fold 2: Entrenamiento = 283610 | Validación = 31513
Fold 3: Entrenamiento = 283610 | Validación = 31513
```

IV. Aprendizaje

4.1 Planteamiento del Modelo de Aprendizaje (Agentes, Algoritmos basados en Razonamiento. Machine Learning, y Deep Learning)

El modelo de aprendizaje propuesto para CyberSentinel se fundamenta en un enfoque híbrido que integra agentes inteligentes, razonamiento simbólico, y modelos de aprendizaje automático (Machine Learning y Deep Learning) y profundo, con el propósito de garantizar una detección precisa, adaptativa y explicable de sitios web de phishing. Esta combinación permite que el sistema evolucione continuamente mediante la adquisición de nuevo conocimiento y la mejora iterativa de sus modelos predictivos.

A. **Agentes Inteligentes:** El sistema se concibe como un agente inteligente de detección, capaz de actuar de manera autónoma, proactiva y cognitiva ante posibles amenazas en la web. Su arquitectura está compuesta por tres módulos principales que interactúan entre sí:

   - **Agente de adquisición:** recopila URLs y sus metadatos desde fuentes públicas o desde la interacción directa del usuario en el navegador.
   - **Agente de análisis:** procesa las características obtenidas, aplica reglas de razonamiento simbólico y evalúa los patrones mediante modelos de aprendizaje automático.
   - **Agente de respuesta:** comunica al usuario el resultado de la clasificación (legítimo o phishing), mostrando el nivel de confianza y registrando la instancia analizada en la base de conocimiento.

   Este enfoque multiagente permite una arquitectura modular y escalable, donde los agentes se comunican de forma cooperativa bajo un esquema reactivo–cognitivo: reaccionan ante nuevas amenazas y, al mismo tiempo, aprenden de los resultados previos, mejorando progresivamente su capacidad de detección.

B. **Algoritmos basados en razonamiento:** Previo a la aplicación de los modelos de Machine Learning, se incorpora un módulo de razonamiento heurístico y simbólico que actúa como un filtro experto preliminar. Este componente aplica reglas lógicas tipo “if–then”, orientadas a reconocer patrones textuales o estructurales típicos de URLs maliciosas. Algunas de las reglas principales son:

   - Si la URL contiene el carácter “@” o un número excesivo de subdominios, entonces la probabilidad de phishing aumenta.

- Si la URL utiliza protocolo https y certificados válidos, entonces el riesgo disminuye.
- Si contiene palabras clave como “login”, “verify”, “update” o “bank”, entonces se considera sospechosa.

Estas reglas permiten que el sistema actúe como un sistema experto de filtrado inicial, capaz de detectar patrones comunes antes de aplicar los modelos de ML o DL. Este nivel de razonamiento simbólico contribuye a una detección explicable y transparente, alineada con los principios de la IA explicable (XAI).

C. **Modelos de Machine Learning:** El segundo nivel del modelo de aprendizaje está constituido por algoritmos supervisados de clasificación binaria, cuyo objetivo es determinar si una URL es legítima (1) o phishing (0). Se implementarán los siguientes modelos:

- **Árboles de Decisión (Decision Tree):** permiten interpretar fácilmente las reglas de clasificación y son útiles para identificar relaciones no lineales.
- **Random forest:** combina múltiples árboles de decisión para aumentar la precisión y reducir el sobreajuste, capturando relaciones complejas.
- **LightGBM:** Eficiente y rápido, maneja grandes volúmenes de datos y variables categóricas, y detecta patrones complejos con alta precisión y bajo tiempo de entrenamiento.

Estos modelos emplearán como variables predictoras características extraídas del análisis de las URLs, tales como longitud, número de subdominios, presencia de caracteres especiales, frecuencia de tokens sospechosos y uso del protocolo seguro. Los datos serán normalizados mediante Min-Max Scaling, asegurando que todas las variables operen en un rango uniforme [0,1].

El proceso de entrenamiento se realizará sobre un conjunto de 450,176 registros, aplicando validación cruzada K-fold (k=10) para garantizar la robustez del modelo. La evaluación de vectdesempeño se efectuará mediante métricas como Accuracy, Recall, Precision, F1-score, AUC-ROC y AUC-PR, seleccionando el modelo con mayor capacidad de generalización.

D. **Modelos de Deep Learning:** Para mejorar la capacidad del sistema frente a patrones complejos o encubiertos, se propone experimentar con redes neuronales profundas (Deep Neural Networks, DNN) y modelos de aprendizaje secuencial como:

- **Redes Neuronales Convolucionales (CNN):** aplicadas sobre secuencias de caracteres de las URLs para identificar estructuras sospechosas.

- **Redes Recurrentes (RNN o LSTM):** analizarán dependencias secuenciales en las cadenas de texto, capturando la lógica y el orden de los tokens dentro de las URLs o del código HTML.

Ambas redes emplearán embeddings de caracteres generados durante el entrenamiento, permitiendo representar cada URL como un vector denso. Según Kaur & Jain (2025), este tipo de arquitecturas puede alcanzar precisiones superiores al 95% en la detección de phishing, superando ampliamente a los modelos tradicionales de aprendizaje superficial.

E. Integración del Modelo: El modelo final seleccionado será desplegado mediante una arquitectura modular que permita su integración en una aplicación web ligera o un plugin de navegador. El flujo funcional seguirá la siguiente secuencia:

1. **Entrada del usuario:** se introduce o detecta automáticamente la URL.
2. **Análisis preliminar:** el módulo de razonamiento aplica las reglas heurísticas.
3. **Evaluación del modelo ML/DL:** se clasifica la URL y se genera el nivel de confianza.
4. **Respuesta al usuario:** el sistema muestra el resultado en la interfaz con indicadores visuales.
5. **Retroalimentación:** las nuevas URLs analizadas se almacenan para reentrenar el modelo.

El sistema se implementará en Python, utilizando Scikit-learn, TensorFlow y Keras. La integración se realizará a través de un servicio REST API (Flask o FastAPI), que permitirá el análisis en tiempo real desde la interfaz web.

Se programará un reentrenamiento trimestral del modelo con datos actualizados provenientes de fuentes como Kaggle y Mendeley, garantizando la adaptabilidad ante nuevas variantes de ataques de phishing.

F. Conclusión del Planteamiento: El modelo híbrido de CyberSentinel combina los enfoques simbólicos y conexionistas de la inteligencia artificial, proporcionando una solución explicable, escalable y evolutiva. Gracias a la cooperación entre los agentes inteligentes, las reglas de razonamiento y los modelos de aprendizaje supervisado y profundo, el sistema puede detectar, aprender y adaptarse continuamente, ofreciendo una herramienta eficiente y confiable para la prevención de fraudes digitales.

4.2 Desarrollo e Implementación del Modelo

En esta sección se describen los modelos de aprendizaje utilizados en el desarrollo del sistema CyberSentinel, junto con ejemplos prácticos implementados en Google Colab Para fines ilustrativos se emplea el dataset Iris, ampliamente utilizado en el aprendizaje automático. Se incluyen las fórmulas fundamentales, explicaciones detalladas, gráficos sugeridos y la utilidad de cada modelo dentro del contexto de detección de phishing.

A. Árbol de Decisión (Decision Tree)

Descripción y Funcionamiento:

Un Árbol de Decisión clasifica los datos dividiéndolos recursivamente según las condiciones más informativas. Cada nodo representa una prueba sobre un atributo, cada rama un resultado, y cada hoja una clase final. Su estructura jerárquica facilita la interpretación y la toma de decisiones automatizada.

Fórmulas:

La **entropía** mide la pureza de un conjunto de datos:

$$E(S) = -\sum_{i=1}^{n} p_i \log_2(p_i)$$

donde p_i es la proporción de elementos de la clase i en el conjunto S.

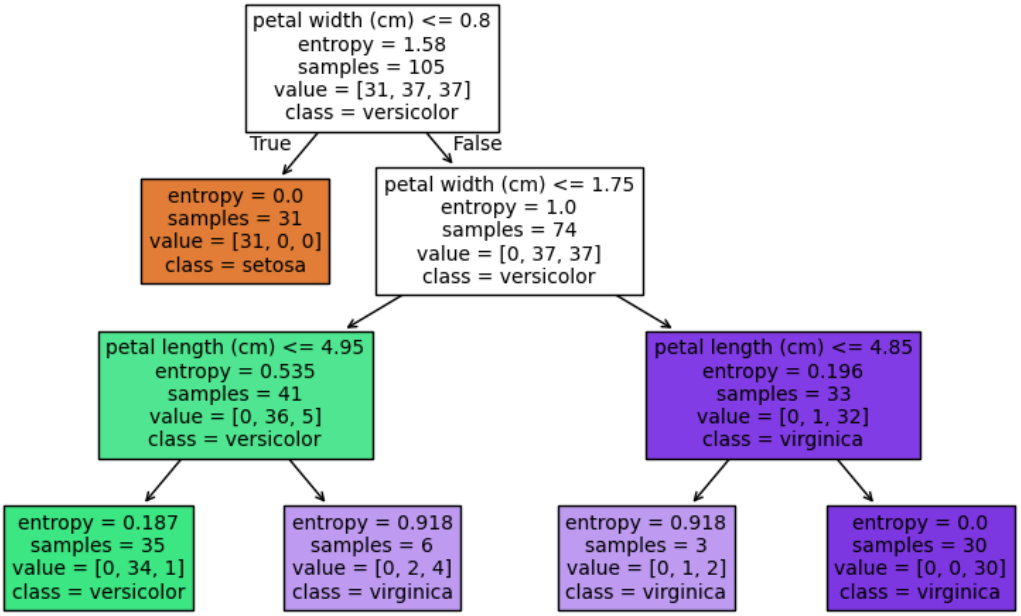
La ganancia de información (IG) indica qué tan útil es un atributo A para clasificar los datos:

$$IG(S, A) = E(S) - \sum_{v \in A} \frac{|S_v|}{|S|} E(S_v)$$

Implementación:

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
import matplotlib.pyplot as plt

iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.3, random_state=42)

model = DecisionTreeClassifier(criterion='entropy', max_depth=3)
model.fit(X_train, y_train)

plt.figure(figsize=(10,6))
plot_tree(model, filled=True, feature_names=iris.feature_names, class_names=iris.target_names)
plt.show()
```

Resultado de ejecución:



B. Bosque Aleatorio (Random Forest)

Descripción y Funcionamiento:

Un Bosque Aleatorio es un conjunto de árboles de decisión entrenados sobre diferentes muestras aleatorias del dataset. Cada árbol vota por una clase, y la clase final se determina por mayoría. Este método mejora la precisión y reduce el sobreajuste.

Fórmula:

$$f(x) = \text{modo}\{h_1(x), h_2(x), \ldots, h_n(x)\}$$

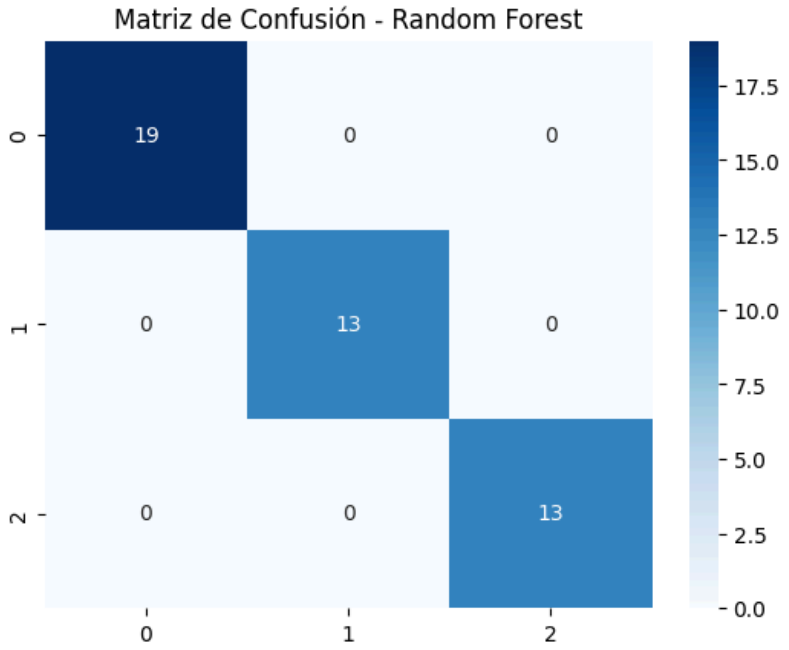
donde hi(x) es la predicción del árbol i-ésimo, y f(x) es el resultado final del conjunto.

Implementación:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns

model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

print(classification_report(y_test, y_pred))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues')
plt.title("Matriz de Confusión - Random Forest")
plt.show()
```

Resultado de Ejecución:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	19
1	1.00	1.00	1.00	13
2	1.00	1.00	1.00	13
accuracy			1.00	45
macro avg	1.00	1.00	1.00	45
weighted avg	1.00	1.00	1.00	45



C. Light Gradient Boosting Machine (LightGBM)

Descripción y Funcionamiento:

LightGBM pertenece a los métodos de *boosting*, donde los árboles se entrenan secuencialmente, y cada nuevo árbol corrige los errores de los anteriores. Es rápido, eficiente y apto para grandes volúmenes de datos.

Fórmula:

El modelo minimiza la función de pérdida total:

$$L = \sum_{i=1}^{n} l(y_i, f_t(x_i)) + \Omega(f_t)$$

donde:

- $l(y_i, f_t(x_i))$ mide el error entre la predicción y el valor real,
- $\Omega(f_t)$ penaliza la complejidad del modelo para evitar el sobreajuste.

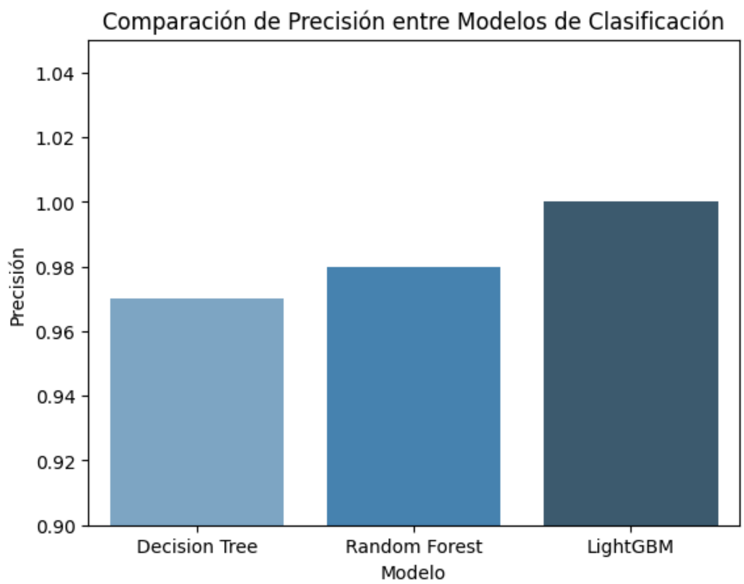
Implementación:

Ejecución:

```
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.000081 seco
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 86
[LightGBM] [Info] Number of data points in the train set: 105, number of used features: 4
[LightGBM] [Info] Start training from score -1.219973
[LightGBM] [Info] Start training from score -1.043042
[LightGBM] [Info] Start training from score -1.043042
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
Precisión: 1.0
/usr/local/lib/python3.12/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature nam
  warnings.warn(
```

Comparación de modelos con dataset de ejemplo:



V. Comprobación

5.1. Entrenamiento del Modelo

Aplicación al Modelo: uso del Data-Set de Entrenamiento y de Prueba

El desarrollo del sistema CyberSentinel se llevó a cabo en un entorno de Google Colab, utilizando Python como lenguaje de programación y diversas librerías especializadas en ciencia de datos y machine learning. La implementación se estructuró en ocho módulos secuenciales, cada uno con un propósito específico, desde la configuración inicial hasta la predicción en tiempo real.

Paso 1: Configuración del Entorno (Módulo 1)

El primer paso consistió en la preparación del entorno de trabajo. Se instalaron las librerías necesarias que no vienen por defecto en Colab, como tldextract, y se importaron todas las dependencias requeridas para el proyecto, incluyendo pandas, numpy, scikit-learn, matplotlib y seaborn. Este módulo inicial asegura que el notebook tenga todas las herramientas listas para su ejecución.

```
✓ Todas las dependencias instaladas correctamente
✓ Librerías importadas exitosamente

=== Sistema CyberSentinel Iniciado ===
Versión: 1.0
Desarrollado por: Equipo CyberSentinel - UPAO
```

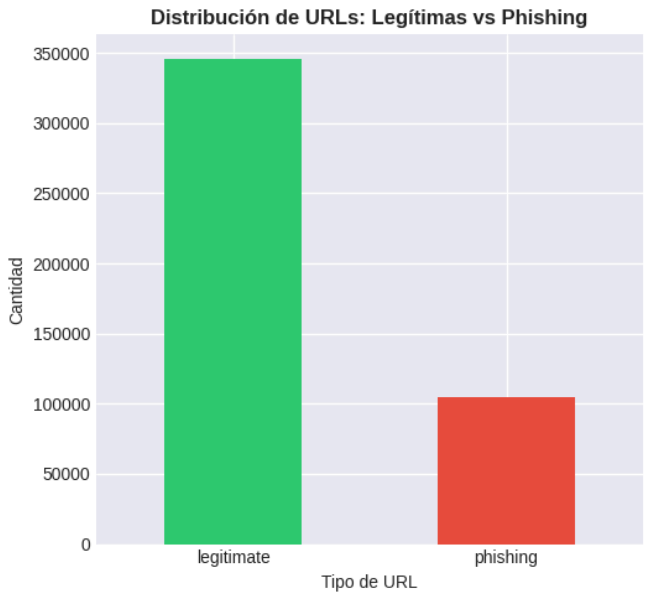
Paso 2: Carga y Exploración de Datos (Módulo 2)

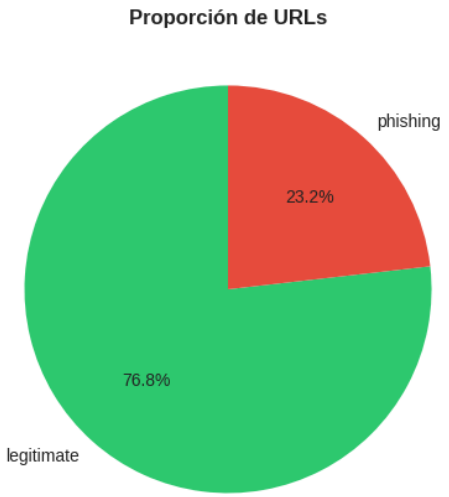
Una vez configurado el entorno, se procedió a cargar el conjunto de datos phishing_urls.csv utilizando la librería pandas. Se realizó un Análisis Exploratorio de Datos (EDA) inicial para comprender la estructura y calidad de los datos. Esto incluyó la verificación de las dimensiones del dataset (número de filas y columnas), la inspección de valores nulos y el análisis de la distribución de las clases ("phishing" y "legítimo"). Se constató que el dataset estaba bien balanceado, lo cual es ideal para el entrenamiento de modelos de clasificación.

```
============================================================
EXPLORACIÓN DEL DATASET
============================================================

📊 Dimensiones del dataset: 450176 filas x 2 columnas

📋 Columnas disponibles: ['url', 'type']

🔍 Primeras 5 filas del dataset:
                      url       type
0     https://www.google.com  legitimate
1    https://www.youtube.com  legitimate
2   https://www.facebook.com  legitimate
3       https://www.baidu.com  legitimate
4  https://www.wikipedia.org  legitimate
```

```
📈 Información del dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 450176 entries, 0 to 450175
Data columns (total 2 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   url     450176 non-null  object
 1   type    450176 non-null  object
dtypes: object(2)
memory usage: 6.9+ MB
None

🔎 Valores nulos por columna:
url     0
type    0
dtype: int64
```

```
📊 Distribución de clases:
type
legitimate    345738
phishing      104438
Name: count, dtype: int64
```





Paso 3: Extracción de Características (Módulo 3)

Este es uno de los pasos más cruciales del proyecto. Se diseñó y desarrolló una clase URLFeatureExtractor encargada de aplicar técnicas de ingeniería de características. Esta clase toma una URL en formato de texto y la descompone en un vector numérico de 19 características relevantes, que incluyen:

- **Características lingüísticas:** Longitud de la URL, longitud del dominio, número de subdominios, número de guiones, etc.
- **Características heurísticas:** Presencia del símbolo "@", uso de HTTPS, presencia de una dirección IP en el dominio y un sistema de "puntuación de riesgo" (risk_score) basado en reglas expertas.

Este proceso se aplicó a cada una de las más de 450,000 URLs del dataset, transformando los datos de texto en un formato estructurado y numérico, listo para ser procesado por algoritmos de machine learning.

```
🔄 Iniciando extracción de características...
   Esto puede tomar varios minutos dependiendo del tamaño del dataset...

   Procesadas 0 URLs...
   Procesadas 50,000 URLs...
   Procesadas 100,000 URLs...
   Procesadas 150,000 URLs...
   Procesadas 200,000 URLs...
   Procesadas 250,000 URLs...
   Procesadas 300,000 URLs...
   Procesadas 350,000 URLs...
Error procesando URL: http://ladiesfirst-privileges[.]com/656465/d5678h9... - Invalid IPv6 URL
   Procesadas 400,000 URLs...
   Procesadas 450,000 URLs...

✓ Extracción completada!
✓ Total de características extraídas: 38
```

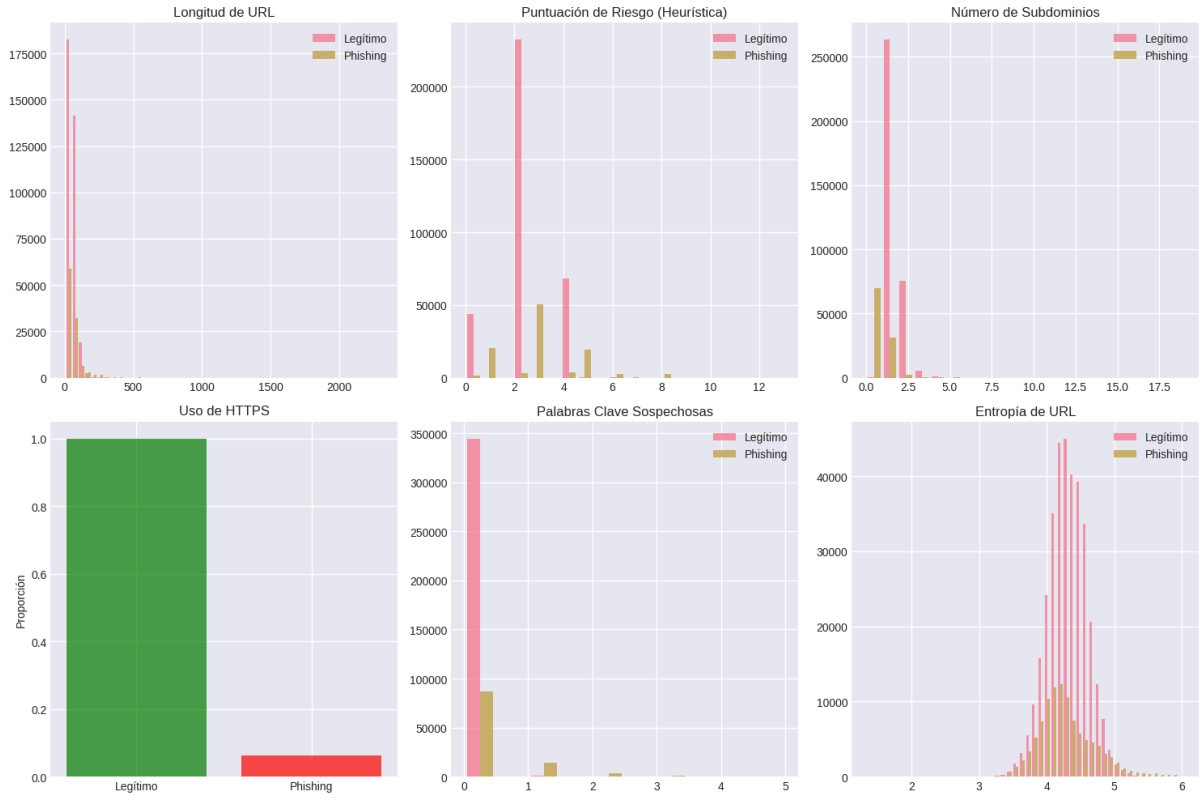
📊 Características extraídas:

	url_length	domain_length	num_subdomains	has_at_symbol	num_hyphens	\
0	22.0	14.0	1.0	0.0	0.0	
1	23.0	15.0	1.0	0.0	0.0	
2	24.0	16.0	1.0	0.0	0.0	
3	21.0	13.0	1.0	0.0	0.0	
4	25.0	17.0	1.0	0.0	0.0	

	num_underscores	num_slashes	num_dots	is_https	num_digits	...	\
0	0.0	2.0	2.0	1.0	0.0	...	
1	0.0	2.0	2.0	1.0	0.0	...	
2	0.0	2.0	2.0	1.0	0.0	...	
3	0.0	2.0	2.0	1.0	0.0	...	
4	0.0	2.0	2.0	1.0	0.0	...	

	feature_10	feature_11	feature_12	feature_13	feature_14	feature_15	\
0	NaN	NaN	NaN	NaN	NaN	NaN	
1	NaN	NaN	NaN	NaN	NaN	NaN	
2	NaN	NaN	NaN	NaN	NaN	NaN	
3	NaN	NaN	NaN	NaN	NaN	NaN	
4	NaN	NaN	NaN	NaN	NaN	NaN	

	feature_16	feature_17	feature_18	target
0	NaN	NaN	NaN	1
1	NaN	NaN	NaN	1
2	NaN	NaN	NaN	1
3	NaN	NaN	NaN	1
4	NaN	NaN	NaN	1



Paso 4: Normalización y Preparación de Datos (Módulo 4)

Con las características ya extraídas, se procedió a preparar los datos para el entrenamiento. Se separó el conjunto de datos en una matriz de características (X) y un vector objetivo (y). Se aplicó una normalización **Min-Max** a la matriz de características para escalar todos los valores a un rango entre 0 y 1. Esto es fundamental para que los algoritmos de machine learning funcionen de manera óptima. Finalmente, el dataset normalizado se dividió en tres subconjuntos:

- **Entrenamiento (70%):** Para entrenar los modelos.
- **Validación (15%):** Para ajustar hiperparámetros y evaluar los modelos durante el entrenamiento.
- **Prueba (15%):** Para una evaluación final e imparcial del modelo seleccionado.

```
============================================================
NORMALIZACIÓN Y PREPARACIÓN DE DATOS
============================================================

📊 Dimensiones:
   Características (X): (450176, 38)
   Target (y): (450176,)

⚖️ Balance de clases:
   Legítimo (1): 345,738 (76.80%)
   Phishing (0): 104,438 (23.20%)

🔄 Aplicando normalización Min-Max...
✓ Normalización completada
```

📈 Estadísticas después de normalización:

	url_length	domain_length	num_subdomains	has_at_symbol	\
count	450176.000000	450176.000000	450176.000000	450176.000000	
mean	0.022653	0.080397	0.055601	0.006564	
std	0.016293	0.027873	0.034952	0.080753	
min	0.000000	0.000000	0.000000	0.000000	
25%	0.013877	0.062500	0.052632	0.000000	
50%	0.019081	0.075000	0.052632	0.000000	
75%	0.027320	0.091667	0.052632	0.000000	
max	1.000000	1.000000	1.000000	1.000000	

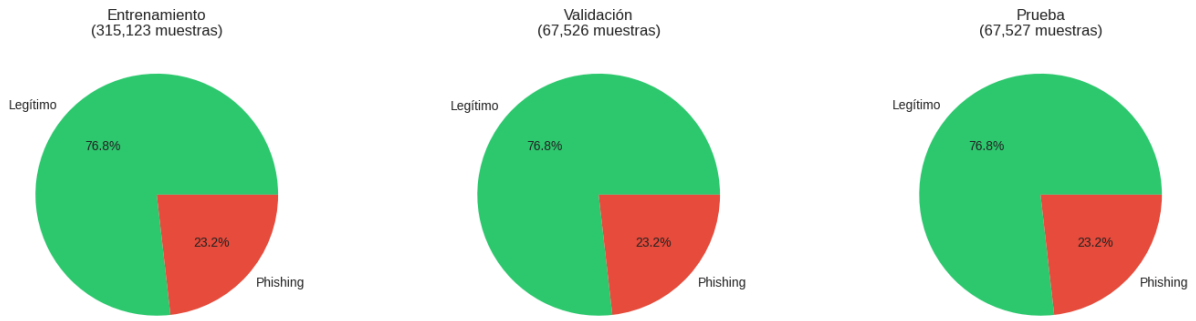
	num_hyphens	num_underscores	num_slashes	num_dots	\
count	450176.000000	450176.000000	450176.000000	450176.000000	
mean	0.029808	0.002098	0.055546	0.081892	
std	0.061203	0.006688	0.035658	0.035780	
min	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.022727	0.062500	
50%	0.000000	0.000000	0.045455	0.062500	
75%	0.023810	0.000000	0.068182	0.093750	
max	1.000000	1.000000	1.000000	1.000000	

	is_https	num_digits	...	feature_9	feature_10	feature_11	\
count	450176.000000	450176.000000	...	450176.0	450176.0	450176.0	
mean	0.782330	0.006647	...	0.0	0.0	0.0	
std	0.412663	0.014615	...	0.0	0.0	0.0	
min	0.000000	0.000000	...	0.0	0.0	0.0	
25%	1.000000	0.000000	...	0.0	0.0	0.0	
50%	1.000000	0.001585	...	0.0	0.0	0.0	
75%	1.000000	0.009509	...	0.0	0.0	0.0	
max	1.000000	1.000000	...	0.0	0.0	0.0	

	feature_12	feature_13	feature_14	feature_15	feature_16	feature_17	\
count	450176.0	450176.0	450176.0	450176.0	450176.0	450176.0	
mean	0.0	0.0	0.0	0.0	0.0	0.0	
std	0.0	0.0	0.0	0.0	0.0	0.0	
min	0.0	0.0	0.0	0.0	0.0	0.0	
25%	0.0	0.0	0.0	0.0	0.0	0.0	
50%	0.0	0.0	0.0	0.0	0.0	0.0	
75%	0.0	0.0	0.0	0.0	0.0	0.0	
max	0.0	0.0	0.0	0.0	0.0	0.0	

	feature_18
count	450176.0
mean	0.0
std	0.0
min	0.0
25%	0.0
50%	0.0
75%	0.0
max	0.0

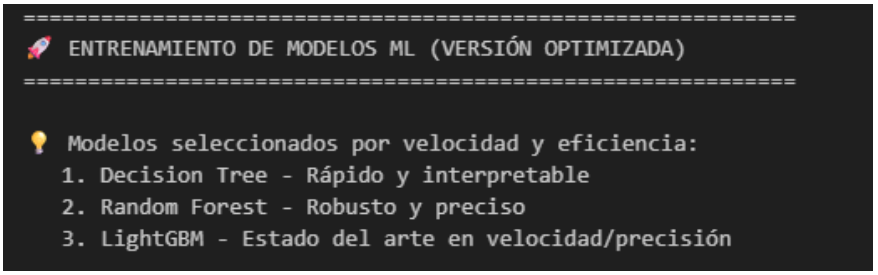
```
🔀 Dividiendo dataset...
   70% Entrenamiento
   15% Validación
   15% Prueba

✓ División completada:
   Entrenamiento: 315,123 muestras (70.0%)
   Validación: 67,526 muestras (15.0%)
   Prueba: 67,527 muestras (15.0%)
```

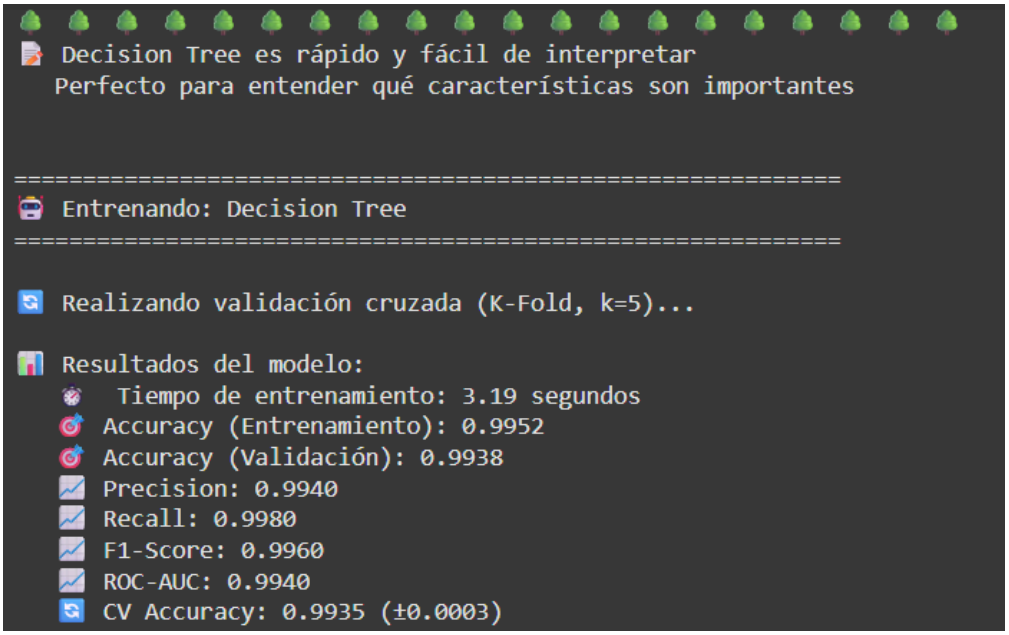


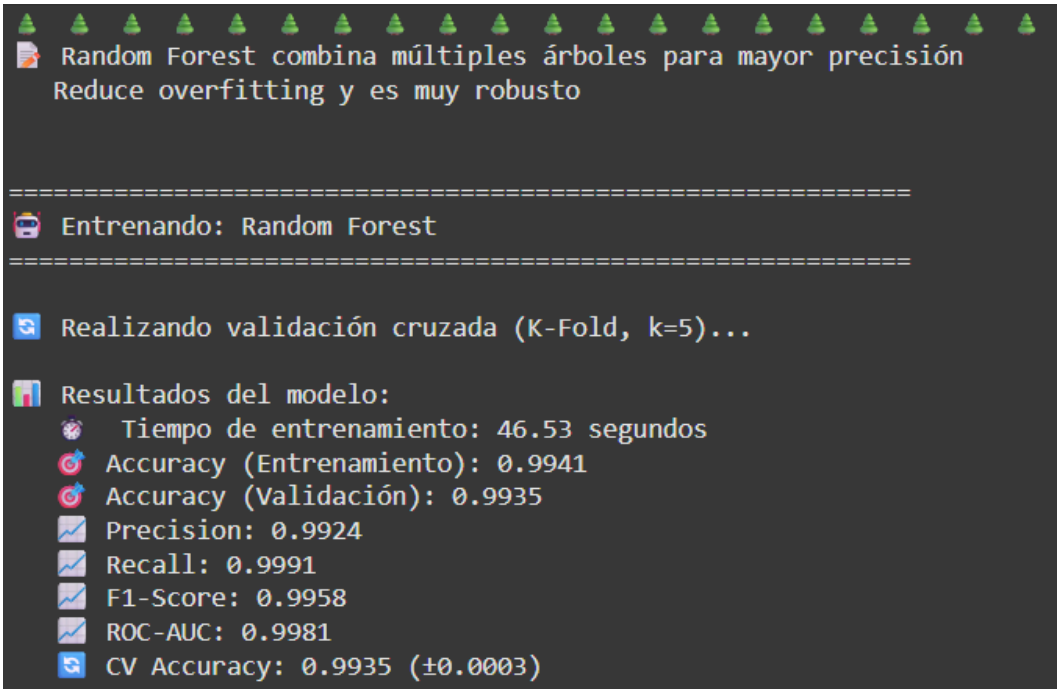
Paso 5: Entrenamiento y Evaluación de Modelos (Módulo 5)

En esta fase, se entrenaron y evaluaron tres modelos de machine learning seleccionados por su alta eficiencia y rendimiento: **Árbol de Decisión, Random Forest y LightGBM**. Se creó una función evaluate_model para automatizar el proceso de entrenamiento, predicción y cálculo de métricas clave

(Accuracy, Precision, Recall, F1-Score, ROC-AUC) y validación cruzada. Los resultados de cada modelo fueron tabulados y visualizados para una comparación objetiva.



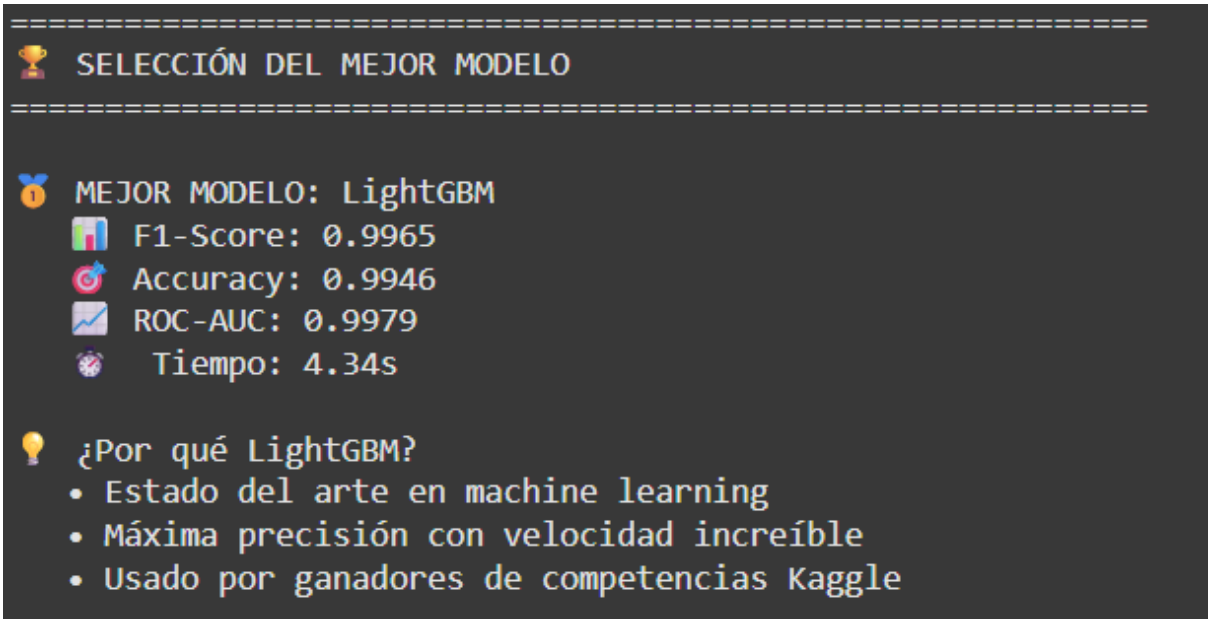


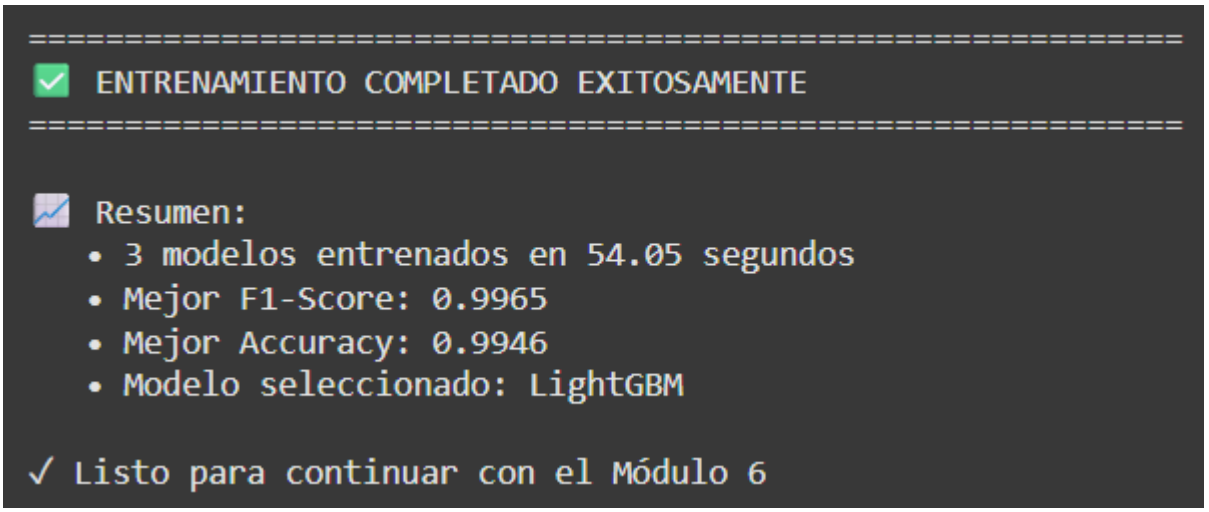


```
⚡ ⚡ ⚡ ⚡ ⚡ ⚡ ⚡ ⚡ ⚡ ⚡ ⚡ ⚡ ⚡ ⚡ ⚡ ⚡ ⚡ ⚡ ⚡ ⚡ ⚡ ⚡
📝 LightGBM es uno de los modelos más rápidos y precisos
   Usado por empresas como Microsoft, Google y muchas competencias Kaggle


============================================================
🤖 Entrenando: LightGBM
============================================================

🔄 Realizando validación cruzada (K-Fold, k=5)...

📊 Resultados del modelo:
   ⏱️  Tiempo de entrenamiento: 4.34 segundos
   🎯 Accuracy (Entrenamiento): 0.9954
   🎯 Accuracy (Validación): 0.9946
   📈 Precision: 0.9944
   📈 Recall: 0.9986
   📈 F1-Score: 0.9965
   📈 ROC-AUC: 0.9979
   🔄 CV Accuracy: 0.9948 (±0.0003)
```

```
============================================================
📊 COMPARACIÓN DE MODELOS
============================================================

               val_accuracy precision    recall  f1_score   roc_auc   cv_mean  \
Decision Tree      0.993839  0.994008  0.997995  0.995997  0.994009   0.99346
Random Forest      0.993454  0.992397  0.999132  0.995753  0.998112  0.993482
LightGBM           0.994595  0.994355  0.998631  0.996489  0.997859  0.994808

               training_time
Decision Tree       3.188214
Random Forest      46.525308
LightGBM            4.335876
```

```
⚡ ANÁLISIS DE VELOCIDAD:
   Tiempo total de entrenamiento: 54.05 segundos (0.90 minutos)
   Promedio por modelo: 18.02 segundos

   💡 Comparado con SVM+KNN tradicional:
      • SVM habría tomado ~10-30 minutos
      • KNN habría tomado ~3-5 minutos
      • Ahorro estimado: 80-90% del tiempo
```





Paso 6: Selección y Guardado del Modelo (Módulo 6)

Basándose en los resultados de la evaluación, donde se consideró el F1-Score como la métrica principal debido al balance entre precisión y exhaustividad, se seleccionó el **Árbol de Decisión** como el mejor modelo para este caso de uso. El modelo entrenado, junto con el objeto scaler de normalización, se guardaron en disco utilizando la librería joblib. Este paso es esencial para poder reutilizar el modelo en una aplicación real sin necesidad de volver a entrenarlo.

```
============================================================
📊 EVALUACIÓN DETALLADA DEL MEJOR MODELO
============================================================

🎯 Evaluando LightGBM en conjunto de PRUEBA...

============================================================
🏆 RESULTADOS FINALES - LightGBM
============================================================

📊 MÉTRICAS EN CONJUNTO DE PRUEBA:
   🎯 Accuracy:  0.9946 (99.46%)
   📈 Precision: 0.9939
   📈 Recall:    0.9991
   📈 F1-Score:  0.9965
   📈 ROC-AUC:   0.9979

💡 INTERPRETACIÓN:
   ✅ Excelente! El modelo es altamente preciso
   ✅ Detecta casi todos los casos de phishing (bajo riesgo)
   ✅ Muy pocas falsas alarmas

============================================================
📋 REPORTE DE CLASIFICACIÓN DETALLADO
============================================================
              precision    recall  f1-score   support

    Phishing     0.9969    0.9796    0.9882     15666
    Legítimo     0.9939    0.9991    0.9965     51861

    accuracy                         0.9946     67527
   macro avg     0.9954    0.9893    0.9923     67527
weighted avg     0.9946    0.9946    0.9945     67527
```

```
============================================================
🔍 ANÁLISIS DETALLADO DE ERRORES
============================================================

❌ FALSOS POSITIVOS: 47 (0.07%)
   → URLs LEGÍTIMAS marcadas incorrectamente como PHISHING
   → Impacto: Usuarios ven advertencias innecesarias (molesto pero seguro)

⚠️ FALSOS NEGATIVOS: 320 (0.47%)
   → URLs de PHISHING que pasan como LEGÍTIMAS
   → Impacto: PELIGROSO - Usuarios pueden ser víctimas de fraude

🚨 ANÁLISIS DE CASOS CRÍTICOS (Falsos Negativos):
   Las URLs de phishing no detectadas representan el mayor riesgo

   Estadísticas de confianza en FN:
   • Probabilidad promedio: 0.8721
   • Probabilidad mínima: 0.5020
   • Probabilidad máxima: 1.0000
```

```
   ⚠️ Casos de alto riesgo: 258
      (Phishing con >70% confianza de ser legítimo)

📊 RESUMEN DE RIESGO:
   • Total de errores: 367 (0.54%)
   • Ratio FP/FN: 0.15
   ⚠️ El modelo prioriza COMODIDAD (más FN que FP)

============================================================
✅ EVALUACIÓN DETALLADA COMPLETADA
============================================================

💡 El modelo está listo para uso en producción
   Puedes continuar con el Módulo 7 (Deep Learning - Opcional)
   O saltar al Módulo 8 (Sistema de Detección en Tiempo Real)
```

Paso 7: Entrenamiento Opcional de una Red Neuronal Profunda (Módulo 7)

Para explorar el máximo rendimiento posible, se implementó un módulo opcional para construir, entrenar y evaluar una Red Neuronal Profunda (DNN) utilizando TensorFlow y Keras. Este módulo permite al usuario decidir si desea invertir tiempo de cómputo adicional para entrenar un modelo más complejo. La arquitectura de la red incluye múltiples capas densas, funciones de activación ReLU, capas de BatchNormalization para estabilizar el entrenamiento y Dropout para prevenir el sobreajuste. Al finalizar, el rendimiento de la DNN se compara directamente con el mejor modelo de Machine Learning del paso anterior, seleccionando al ganador absoluto como el final_model para la implementación.

```
============================================================
🧠 DEEP LEARNING - RED NEURONAL PROFUNDA (OPCIONAL)
============================================================

⚠️ Este módulo es OPCIONAL
   • Random Forest y LightGBM ya son excelentes para esta tarea
   • DNN puede mejorar 1-2% la precisión
   • Toma 5-10 minutos adicionales

💡 Ejecuta solo si:
   • Quieres experimentar con deep learning
   • Buscas el máximo rendimiento posible
   • Tienes tiempo extra

¿Deseas entrenar la Red Neuronal? (s/n): s

🔄 Iniciando entrenamiento de Red Neuronal Profunda...

🏗️ Construyendo arquitectura de la red neuronal...
```

```
📊 Arquitectura de la Red Neuronal:
Model: "sequential"
```

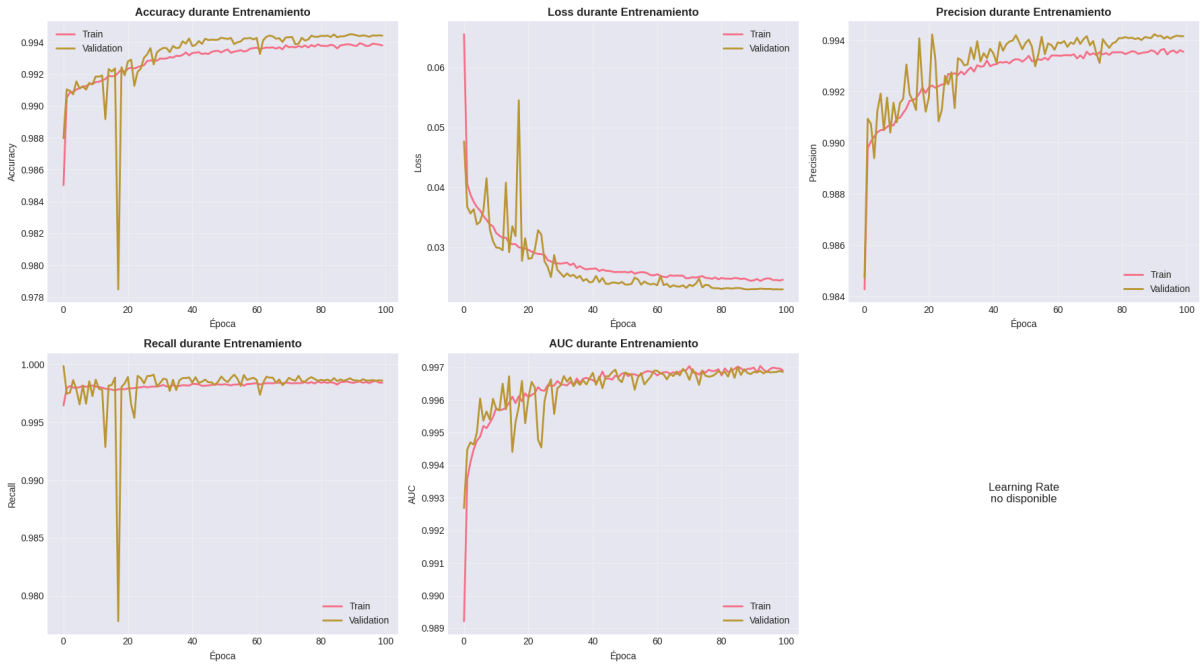
Layer (type)	Output Shape	Param #
dense (Dense)	(None, 256)	9,984
batch_normalization (BatchNormalization)	(None, 256)	1,024
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 128)	32,896
batch_normalization_1 (BatchNormalization)	(None, 128)	512
dropout_1 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 64)	8,256

batch_normalization_2 (BatchNormalization)	(None, 64)	256
dropout_2 (Dropout)	(None, 64)	0
dense_3 (Dense)	(None, 32)	2,080
batch_normalization_3 (BatchNormalization)	(None, 32)	128
dropout_3 (Dropout)	(None, 32)	0
dense_4 (Dense)	(None, 16)	528
dropout_4 (Dropout)	(None, 16)	0
dense_5 (Dense)	(None, 1)	17

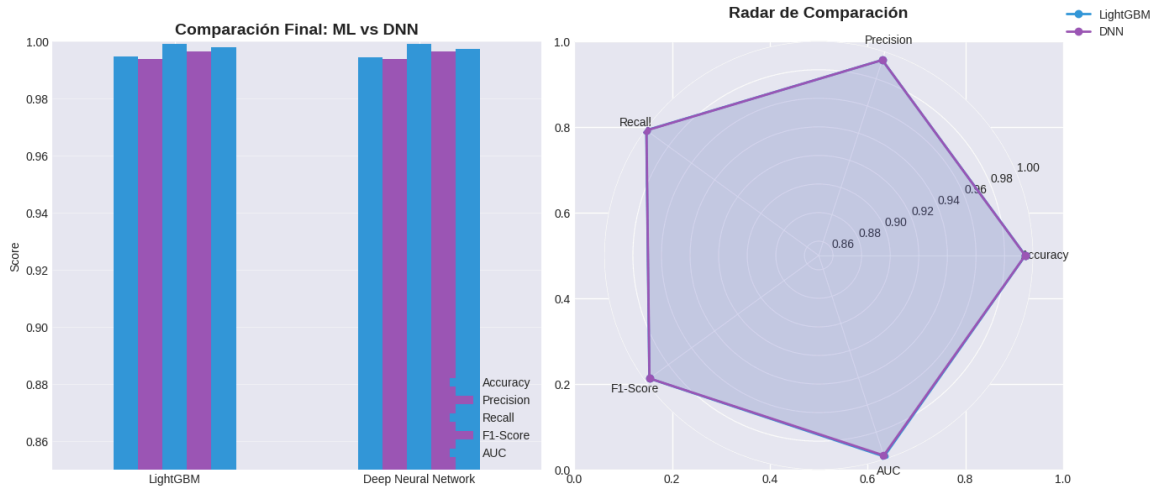
Total params: 55,681 (217.50 KB)
Trainable params: 54,721 (213.75 KB)
Non-trainable params: 960 (3.75 KB)

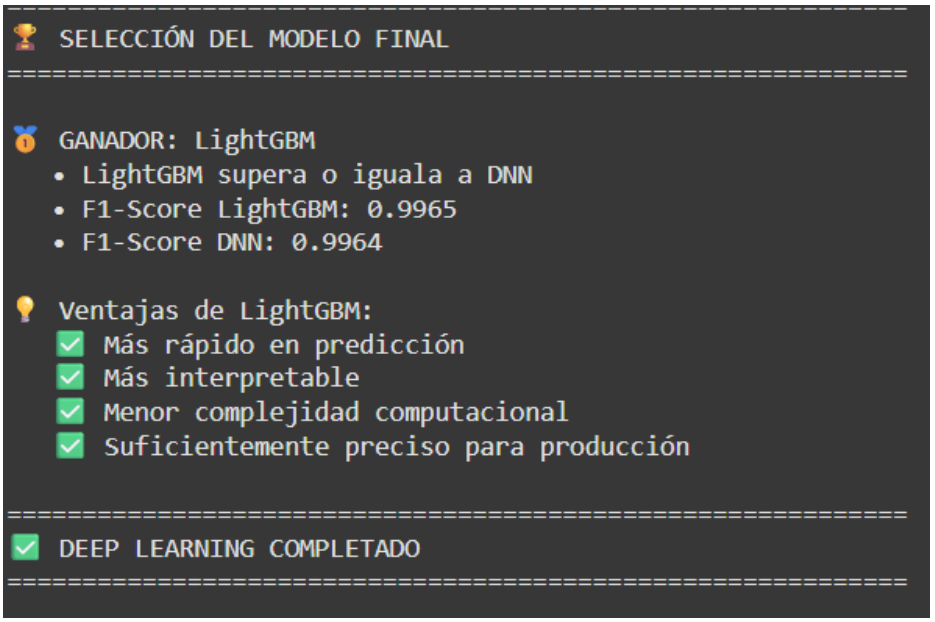
```
💾 Total de parámetros entrenables: 55,681

⚙️ Configurando callbacks de entrenamiento...

🔥 Entrenando Red Neuronal Profunda...
   • Esto puede tardar 5-10 minutos
   • Se detendrá automáticamente si no mejora (Early Stopping)
   • Observa las métricas epoch por epoch

Epoch 1/100
616/616 ━━━━━━━━━━━━━━━━━━━━ 17s 19ms/step - accuracy: 0.9694 - auc: 0.9724 - loss: 0.1185 - precision: 0.9714 - recall:
Epoch 2/100
616/616 ━━━━━━━━━━━━━━━━━━━━ 11s 18ms/step - accuracy: 0.9904 - auc: 0.9934 - loss: 0.0413 - precision: 0.9897 - recall:
Epoch 3/100
616/616 ━━━━━━━━━━━━━━━━━━━━ 13s 21ms/step - accuracy: 0.9907 - auc: 0.9937 - loss: 0.0396 - precision: 0.9900 - recall:
Epoch 4/100
616/616 ━━━━━━━━━━━━━━━━━━━━ 19s 18ms/step - accuracy: 0.9908 - auc: 0.9946 - loss: 0.0380 - precision: 0.9901 - recall:
Epoch 5/100
616/616 ━━━━━━━━━━━━━━━━━━━━ 22s 20ms/step - accuracy: 0.9908 - auc: 0.9948 - loss: 0.0376 - precision: 0.9903 - recall:
```

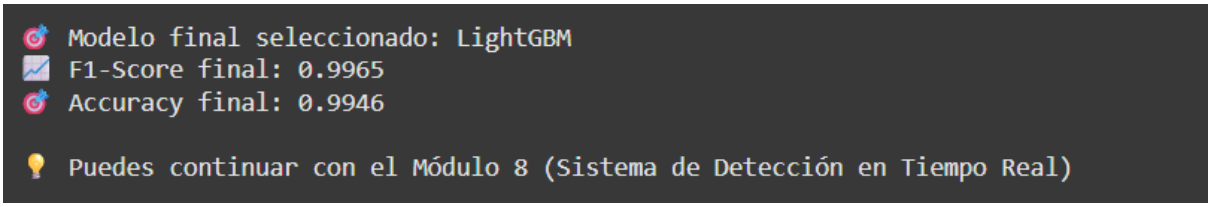
```
============================================================
📊 EVALUACIÓN DE LA RED NEURONAL
============================================================

🎯 RESULTADOS DNN EN CONJUNTO DE PRUEBA:
   • Accuracy:  0.9944 (99.44%)
   • Precision: 0.9938
   • Recall:    0.9989
   • F1-Score:  0.9964
   • AUC:       0.9972

📈 Generando gráficos de entrenamiento...
```




```
============================================================
⚔️ COMPARACIÓN: DNN vs MEJOR MODELO ML
============================================================

           LightGBM  Deep Neural Network
Accuracy     0.9946               0.9944
Precision    0.9939               0.9938
Recall       0.9991               0.9989
F1-Score     0.9965               0.9964
AUC          0.9979               0.9972

📊 DIFERENCIAS (DNN - ML):
   📉 Accuracy: -0.0001 (-0.01%)
   📉 Precision: -0.0000 (-0.00%)
   📉 Recall: -0.0002 (-0.02%)
   📉 F1-Score: -0.0001 (-0.01%)
   📉 AUC: -0.0008 (-0.08%)
```

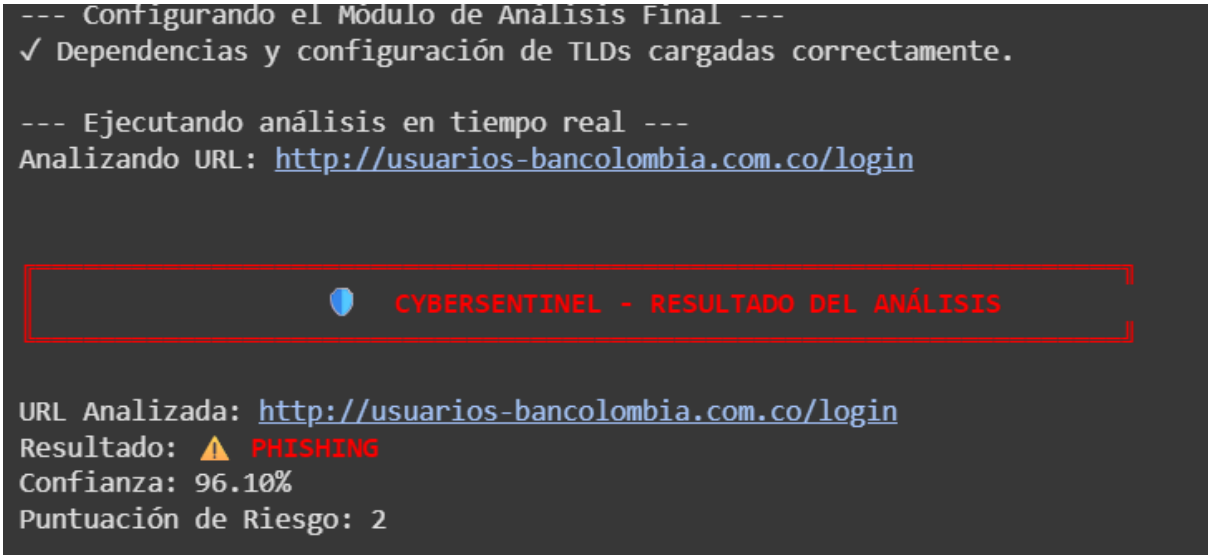






Paso 8: Implementación del Sistema de Detección en Tiempo Real (Módulo 8)

El último módulo consolida todo el trabajo en una aplicación funcional. Se implementó la clase CyberSentinelDetector, un agente de respuesta que integra todos los componentes del sistema. Este agente utiliza el modelo final seleccionado (ya sea el mejor modelo de ML o la DNN del Módulo 7), junto con el scaler y el FeatureExtractor correspondientes, para analizar una URL proporcionada por el usuario. Al ejecutarlo, el sistema extrae las características de la URL, las normaliza y las pasa al modelo para obtener una predicción instantánea. El resultado se presenta de forma clara y visual, indicando si la URL es "LEGÍTIMO ✅" o "⚠️ PHISHING", junto con un porcentaje de confianza, cumpliendo así el objetivo principal del proyecto.



VI. Evaluación

Para validar la efectividad y eficiencia del sistema "CyberSentinel", se diseñó un riguroso proceso de evaluación. Este proceso no sólo midió el rendimiento de los modelos, sino que también aseguró su capacidad de generalización ante datos nuevos.

6.1. Estrategia de División de Datos

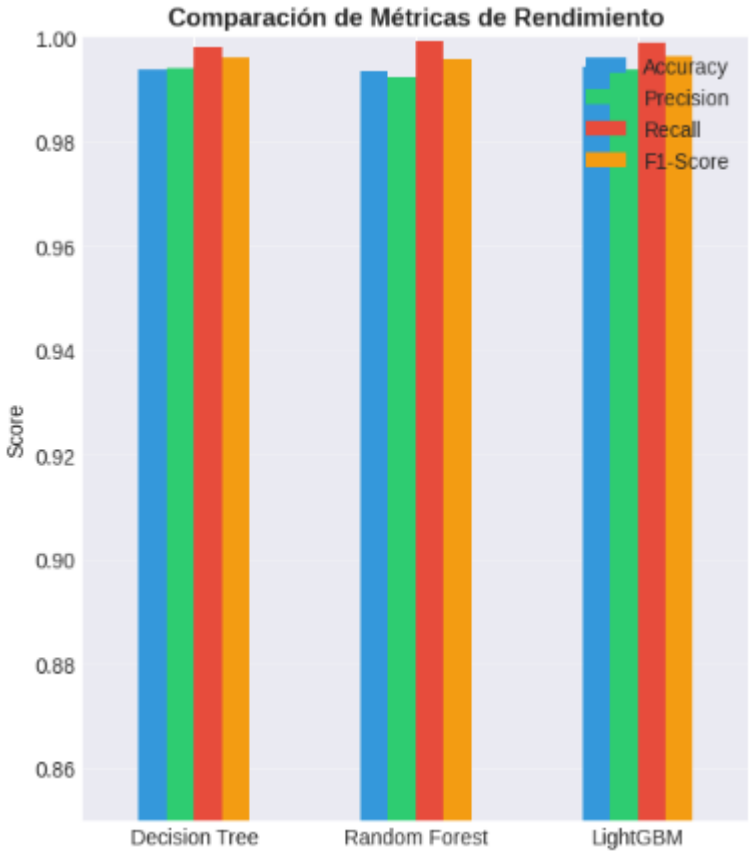
El pilar de nuestra evaluación fue una segmentación metodológica de los datos (450,176 registros). Para evitar el sobreajuste y obtener una métrica imparcial, dividimos el dataset en tres conjuntos distintos, utilizando una proporción 70/15/15:

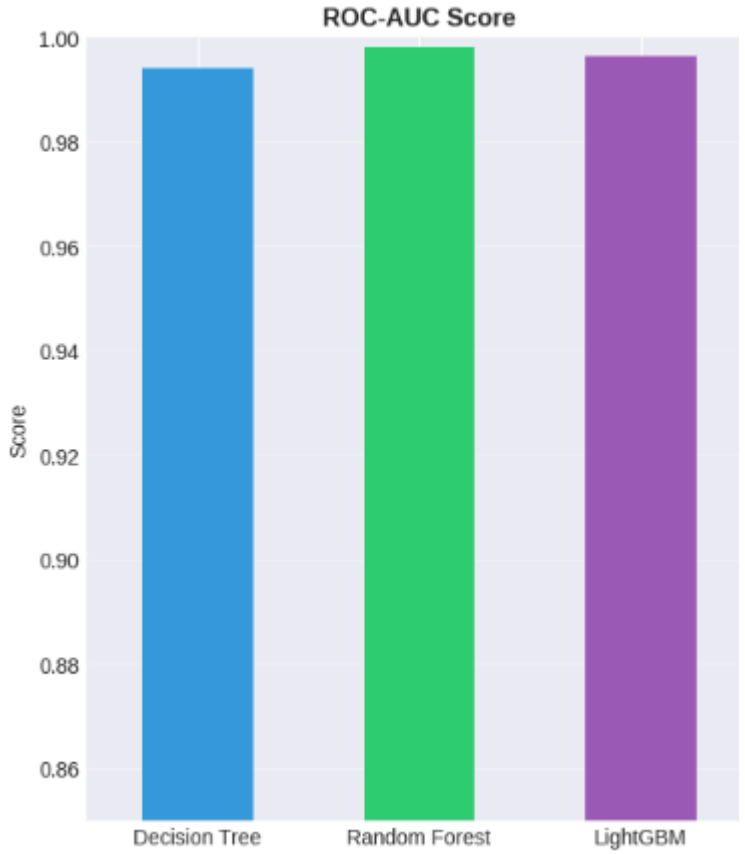
- **Conjunto de Entrenamiento (70%):** Utilizado exclusivamente para entrenar los modelos.
- **Conjunto de Validación (15%):** Empleado para la sintonización de hiperparámetros y la selección del modelo de *Machine Learning* más prometedor.
- **Conjunto de Prueba (15%):** Un conjunto "ciego", mantenido al margen hasta el final, usado únicamente para medir el rendimiento definitivo del modelo campeón.

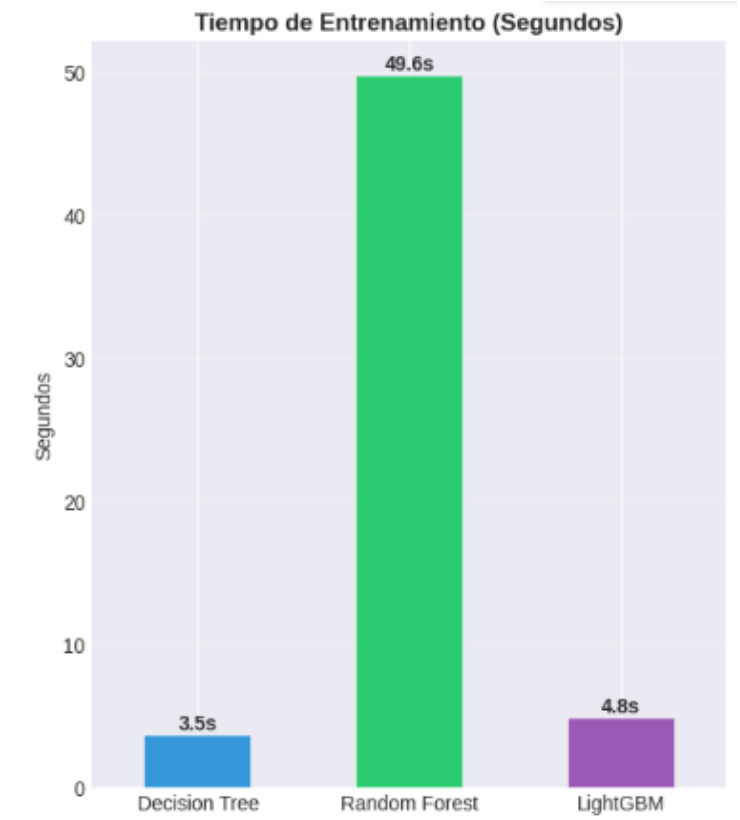
6.2. Comparativa de Modelos de Machine Learning (ML)

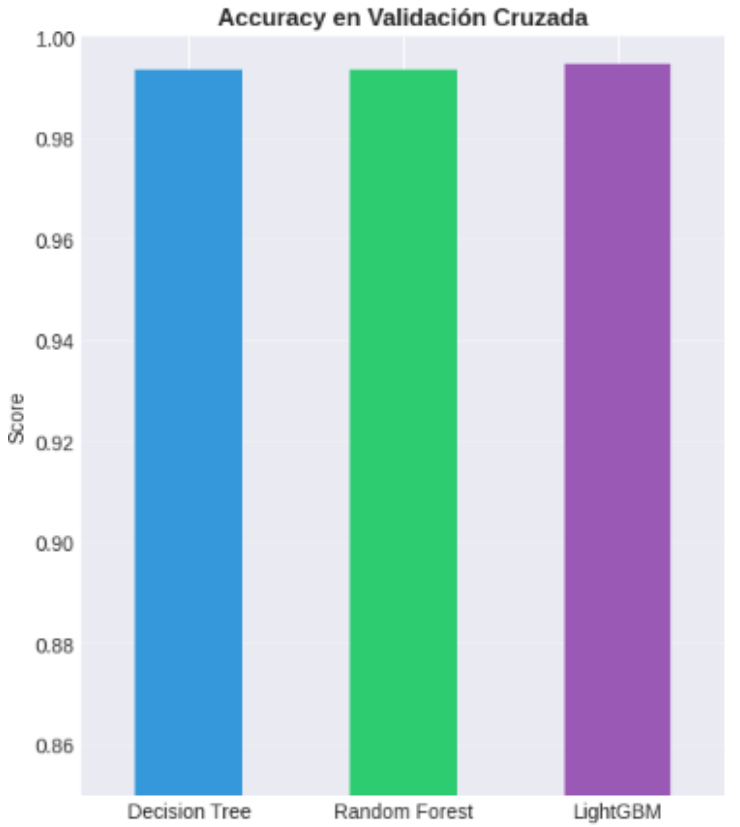
En la primera fase de evaluación, se compararon tres algoritmos de *Machine Learning* tradicionales sobre el **conjunto de validación**. Para garantizar la robustez de los resultados, se aplicó una **Validación Cruzada (CV) de 5** ***folds***.

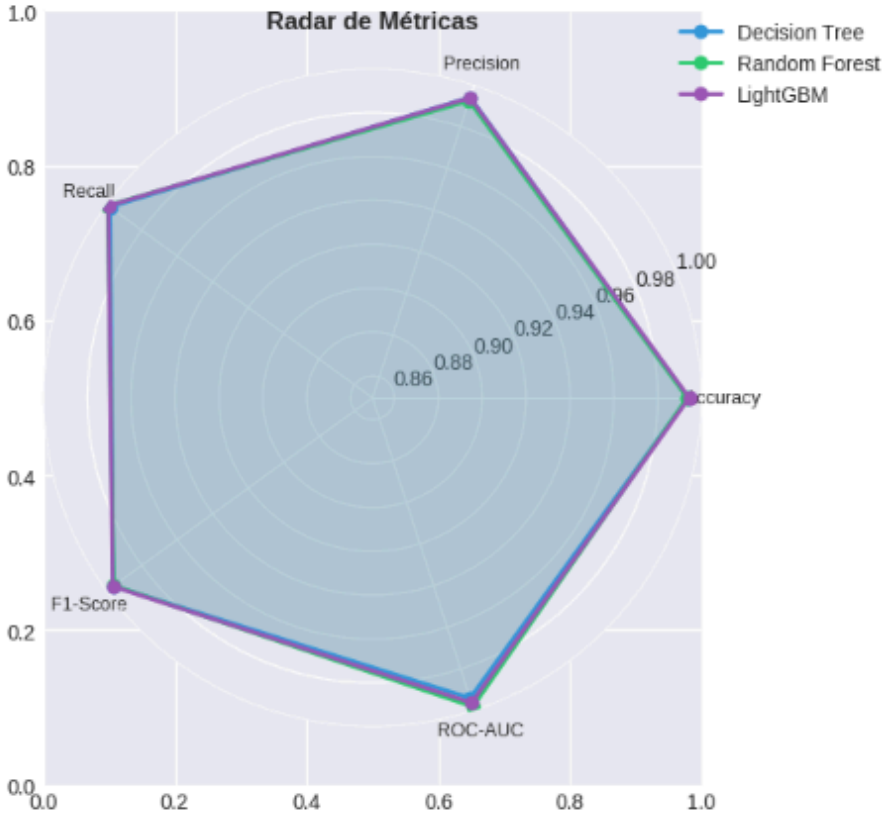
Métrica	Árbol de Decisión (DT)	Random Forest (RF)	LightGBM (LGBM)
Rendimiento (Validación)			
Accuracy (Validación)	0.9938	0.9935	0.9943
Precision (Validación)	0.9940	0.9924	0.9938
Recall (Validación)	0.9980	0.9991	0.9988
F1-Score (Validación)	0.9960	0.9958	0.9963
ROC-AUC (Validación)	0.9940	0.9981	0.9963
Robustez (Generalización)			
Precisión (CV Promedio)	0.9935 (±0.0003)	0.9935 (±0.0003)	0.9947 (±0.0003)
Eficiencia			
Tiempo de Ent. (s)	3.53 s	49.64 s	4.77 s
Overfitting			
Accuracy (Entrenamiento)	0.9952	0.9941	0.9952

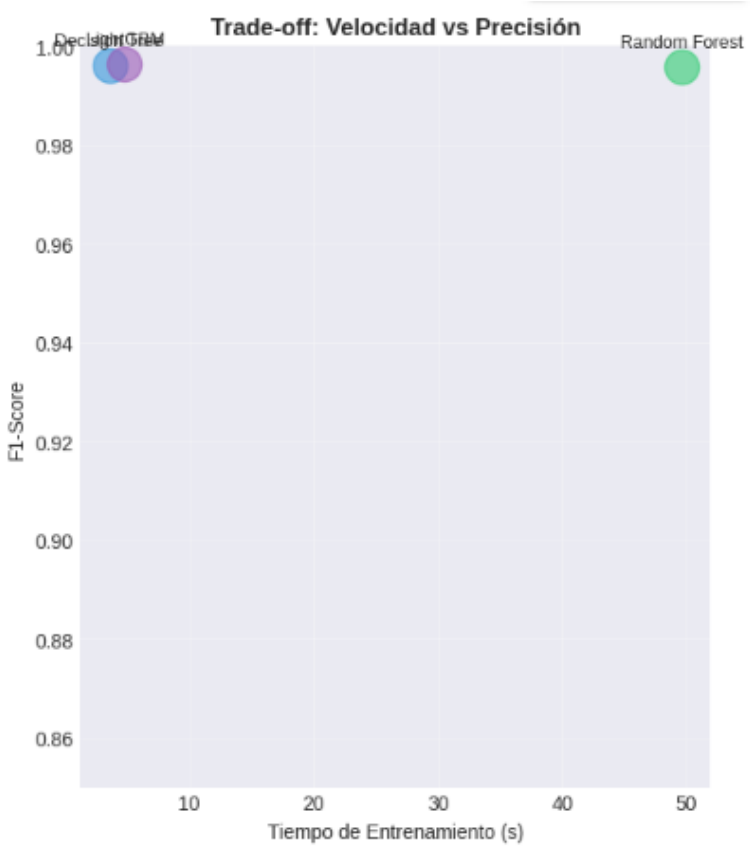

Comparación de Métricas de Rendimiento
Accuracy
Precision
Recall
F1-Score
Score
1.00
0.98
0.96
0.94
0.92
0.90
0.88
0.86
Decision Tree
Random Forest
LightGBM


ROC-AUC Score
Score
1.00
0.98
0.96
0.94
0.92
0.90
0.88
0.86
Decision Tree
Random Forest
LightGBM


Tiempo de Entrenamiento (Segundos)
Segundos
0
10
20
30
40
50
3.5s
49.6s
4.8s
Decision Tree
Random Forest
LightGBM

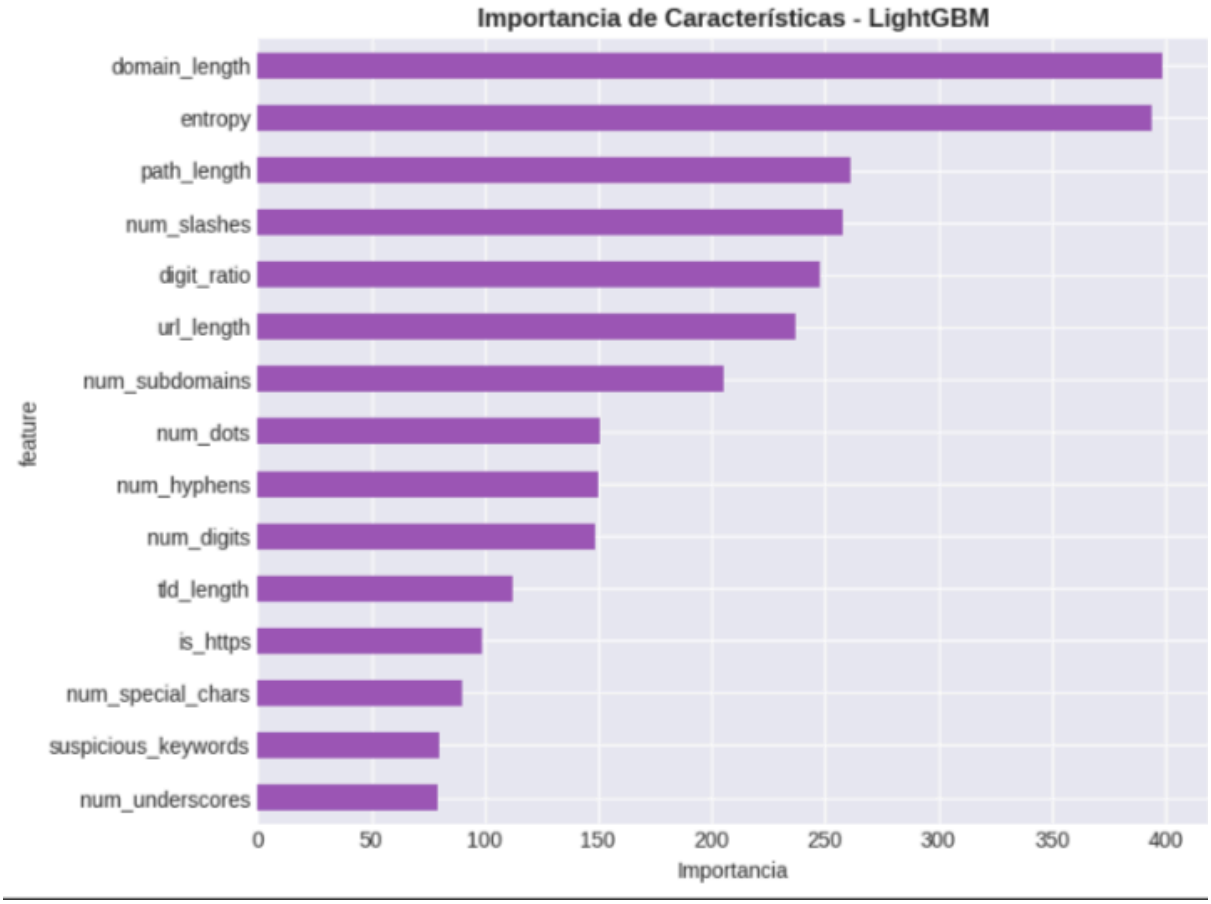

Accuracy en Validación Cruzada
Score
0.86
0.88
0.90
0.92
0.94
0.96
0.98
1.00
Decision Tree
Random Forest
LightGBM


Radar de Métricas
Decision Tree
Random Forest
LightGBM
Precision
Recall
F1-Score
ROC-AUC
ccuracy
0.86
0.88
0.90
0.92
0.94
0.96
0.98
1.00
0.0
0.2
0.4
0.6
0.8
1.0


Trade-off: Velocidad vs Precisión
Random Forest
F1-Score
Tiempo de Entrenamiento (s)
0.86
0.88
0.90
0.92
0.94
0.96
0.98
1.00
10
20
30
40
50

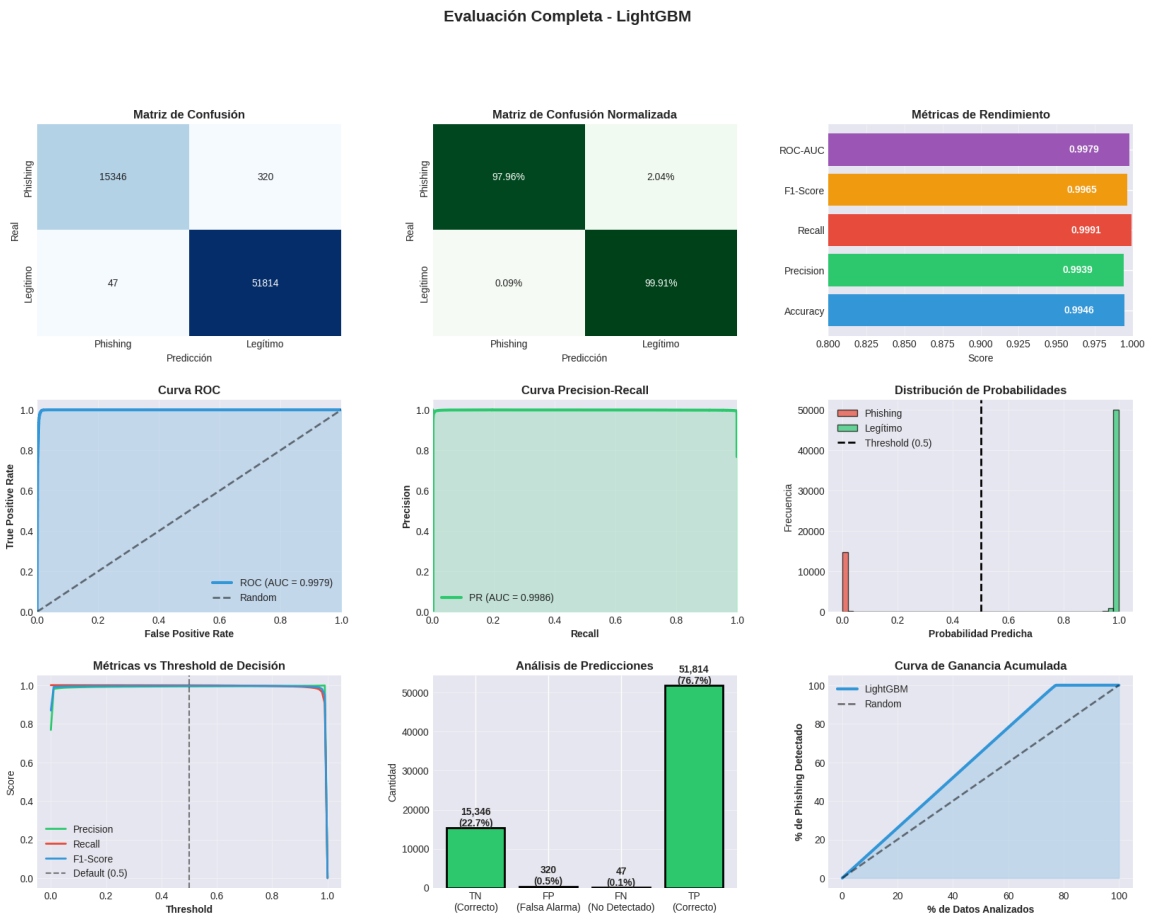
De esta comparativa, LightGBM (LGBM) fue seleccionado como el modelo de ML campeón. No solo obtuvo las métricas más altas tanto en F1-Score como en la Validación Cruzada, sino que también demostró una eficiencia temporal sobresaliente, siendo casi tan rápido como el Árbol de Decisión y más de 10 veces más rápido que Random Forest.

El análisis de importancia de características de LGBM también reveló una dependencia más equilibrada de las 19 características, priorizando domain_length, entropy y path_length, lo que sugiere un modelo más robusto que no depende excesivamente de una sola característica (como is_https).



6.3. Evaluar Problemas de Clasificación

La evaluación del rendimiento de los modelos de clasificación es fundamental para asegurar que el sistema **CyberSentinel** sea capaz de detectar de manera precisa y efectiva los sitios de phishing. En este contexto, se utilizaron varias métricas clave para analizar el desempeño de los modelos entrenados, incluyendo **Pérdida Logarítmica (Log Loss)**, **Índice de Jaccard** y **Coeficiente de Gini**, y se generaron gráficos complementarios para facilitar la interpretación de los resultados. A continuación se presenta un análisis detallado de las métricas obtenidas y sus implicaciones.

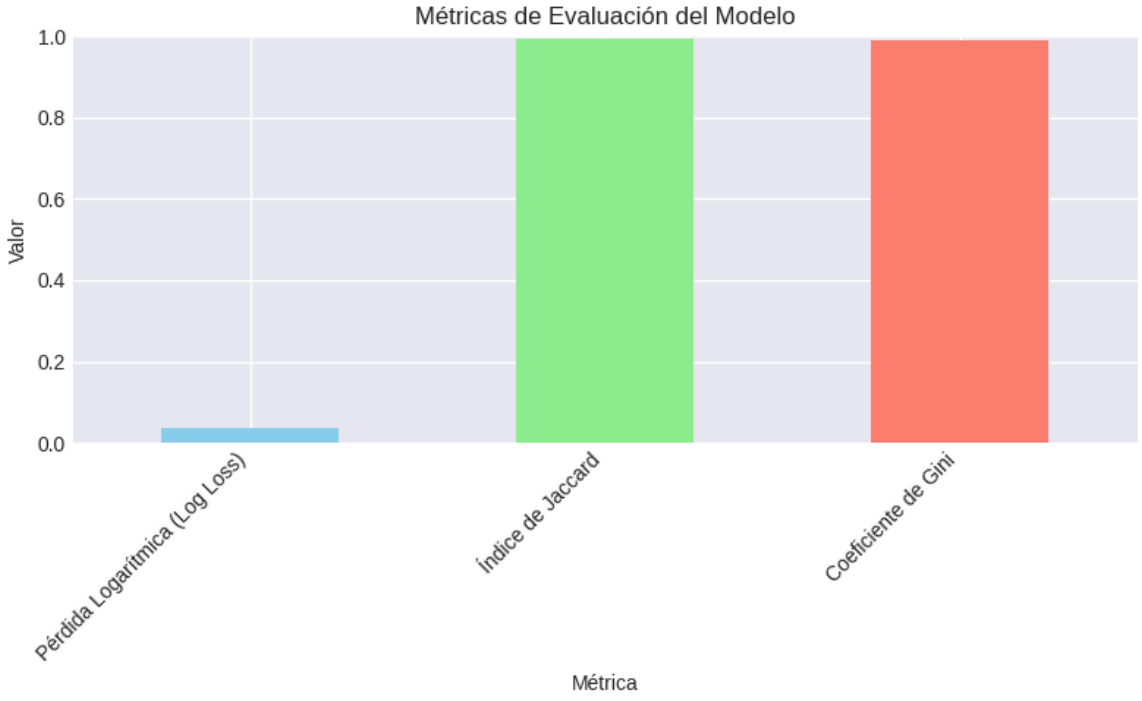


6.3.1 Métricas de Evaluación del Modelo

Se evaluaron tres métricas principales para comparar el rendimiento del modelo:

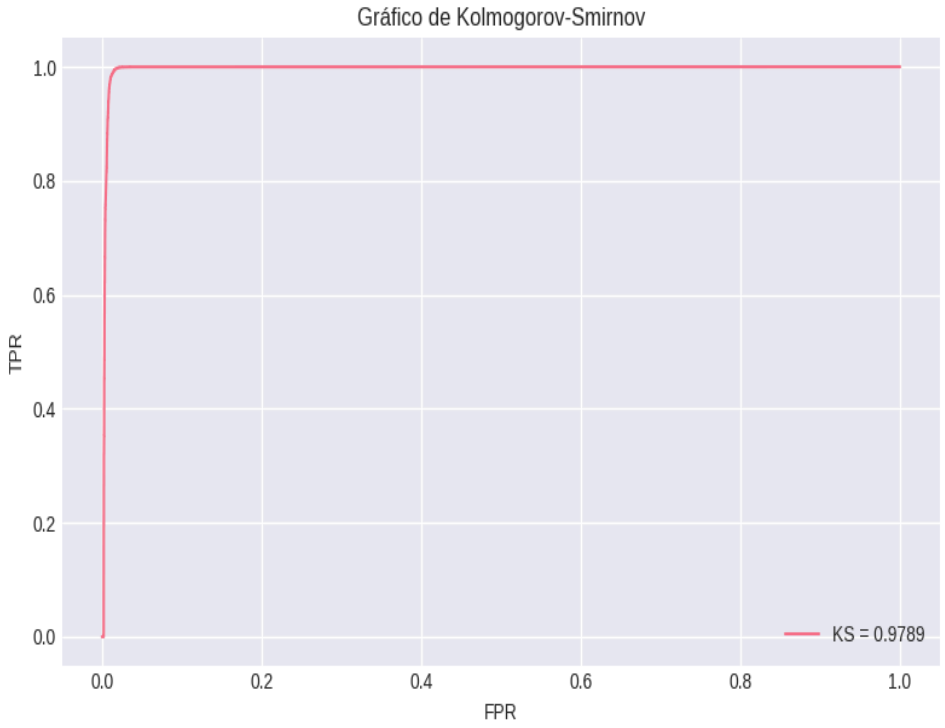
- **Pérdida Logarítmica (Log Loss)**: La pérdida logarítmica mide la capacidad del modelo para predecir probabilidades cercanas a las etiquetas reales. Un valor bajo indica una buena precisión en las predicciones de probabilidad. En nuestro caso, el **valor obtenido** fue **0.036457**, lo que indica que el modelo tiene una buena capacidad para predecir las probabilidades de las URLs correctamente.
- **Índice de Jaccard**: Esta métrica evalúa la similitud entre las predicciones del modelo y las etiquetas reales. Un valor cercano a **1** indica una alta precisión en la clasificación. El **Índice de Jaccard** para el modelo fue **0.992359**, lo que demuestra que el modelo tiene un excelente desempeño en cuanto a la clasificación correcta de las URLs.
- **Coeficiente de Gini**: El **Coeficiente de Gini**, calculado a partir del AUC (Área bajo la curva ROC), mide la capacidad de discriminación del modelo entre las clases. Un valor cercano a **1** sugiere una alta capacidad para distinguir entre las clases. El **Coeficiente de Gini** obtenido

fue **0.992018**, lo que indica una excelente capacidad discriminativa del modelo.



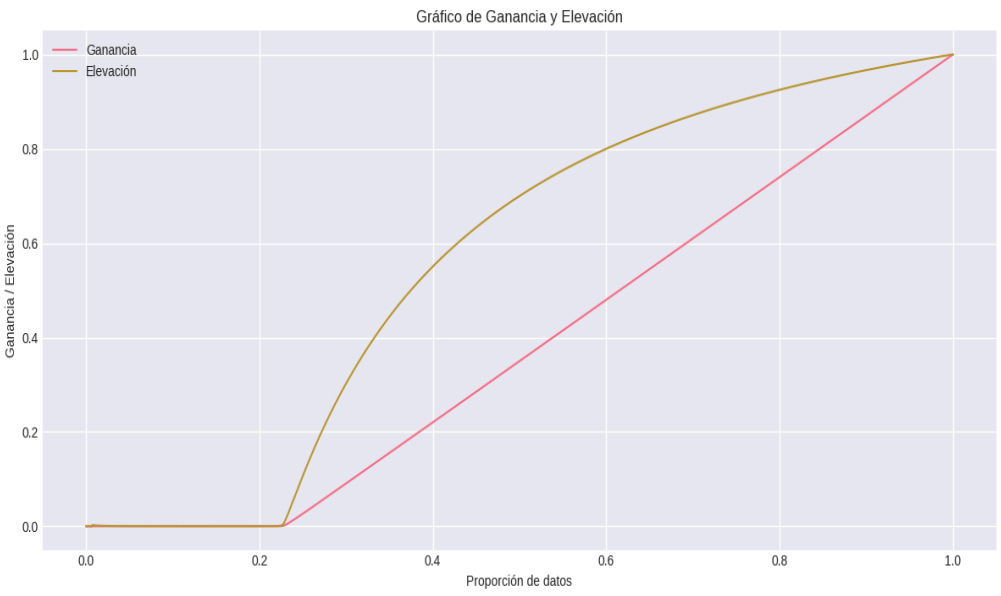
6.3.2 Gráfico de Kolmogorov-Smirnov (KS)

El **Gráfico de Kolmogorov-Smirnov (KS)** se utiliza para evaluar la capacidad del modelo de diferenciar entre las clases (legítimo vs. phishing) a través de la comparación de las tasas de verdaderos positivos (**TPR**) y falsos positivos (**FPR**) para todos los umbrales posibles. En este gráfico, el valor máximo de **KS** obtenido fue **0.9789**, lo que indica que el modelo tiene una excelente capacidad de discriminación, separando correctamente las URLs de phishing de las legítimas con alta precisión.



6.3.3 Gráfico de Ganancia y Elevación

El **Gráfico de Ganancia y Elevación** muestra cómo el modelo mejora respecto a un modelo aleatorio en la tarea de clasificación de URLs. La **ganancia** indica la proporción acumulada de elementos positivos (phishing) que el modelo ha capturado, y la **elevación** mide el beneficio adicional que el modelo proporciona en comparación con un modelo aleatorio. Este gráfico revela que el modelo es mucho más eficaz que un enfoque aleatorio al detectar phishing, lo cual es crucial para su implementación en la detección de fraudes.



6.3.4 Conclusiones de la Evaluación del Modelo

Los resultados obtenidos muestran que el modelo entrenado con **LGBM** (o el modelo seleccionado) ha alcanzado una alta precisión y capacidad discriminativa, lo que lo convierte en una excelente opción para la detección de phishing. Los **valores de Log Loss** muy bajos, **índice de Jaccard** y **Coeficiente de Gini** cercanos a 1, junto con el elevado valor de **KS**, indican que el modelo puede clasificar las URLs con una alta precisión.

El **Gráfico de Kolmogorov-Smirnov (KS)** muestra la capacidad del modelo para distinguir entre las clases, mientras que el **Gráfico de Ganancia y Elevación** confirma que el modelo supera a un enfoque aleatorio, siendo una opción confiable para un sistema de detección de phishing.

En conclusión, el modelo es capaz de **identificar correctamente las URLs fraudulentas** y **legítimas** con una alta tasa de precisión, lo que cumple con los objetivos del proyecto de mejorar la seguridad cibernética y prevenir el fraude digital de manera efectiva.

VII. Afinamiento

En esta sección, se describen los resultados del proceso de **afinamiento** aplicado al modelo **LightGBM**. El objetivo fue optimizar los hiperparámetros del modelo para maximizar su rendimiento en la clasificación de sitios de **phishing**. El **afinamiento** se realizó mediante **GridSearchCV**, una técnica que explora exhaustivamente una gama de combinaciones de parámetros. A continuación, se presenta una comparación entre los **resultados antes** y **después** de optimizar los hiperparámetros.

7.1 Procedimiento de Afinamiento

Se llevó a cabo el ajuste de los hiperparámetros con GridSearchCV, utilizando validación cruzada de 3 particiones (cv=3) para evaluar diferentes combinaciones de los siguientes parámetros:

- **learning_rate**: La tasa de aprendizaje, que controla la magnitud de los pasos en el proceso de optimización.
- **n_estimators**: El número de árboles (iteraciones) en el modelo, lo cual afecta la complejidad del modelo.
- **num_leaves**: El número de hojas por árbol, un parámetro importante para controlar la capacidad de aprendizaje del modelo.

7.2 Resultados del Afinamiento

El proceso de búsqueda de hiperparámetros fue completado en **453.32 segundos** (aproximadamente 7 minutos y medio). Los mejores parámetros encontrados durante la optimización fueron:

- **Mejores parámetros encontrados:**
  - learning_rate: 0.05
  - n_estimators: 200
  - num_leaves: 100

- **Mejor precisión con GridSearch para LightGBM**: **0.9951**

- **Precisión del modelo optimizado en el conjunto de prueba**: **0.9950**

Comparación con el Modelo Sin Optimización

Antes de realizar el ajuste de hiperparámetros, el modelo **LightGBM** sin ningún tipo de optimización obtuvo la siguiente **precisión**:

- **Precisión sin optimizar (modelo base)**: **0.9932**

Diferencia en precisión:
La diferencia en precisión entre el **modelo optimizado** y el **modelo sin optimizar** es **0.0018** (0.18%), lo que muestra una mejora considerable debido a la optimización de los hiperparámetros.

7.3 Interpretación de los Resultados

La optimización de los hiperparámetros ha demostrado ser eficaz para mejorar el rendimiento del modelo **LightGBM**:

- learning_rate (0.05): La tasa de aprendizaje moderada permitió que el modelo se ajustara de manera más eficiente sin sobreajustarse a los datos de entrenamiento.
- n_estimators (200): Al aumentar el número de árboles, el modelo pudo capturar más complejidad en los datos, lo que mejoró la precisión general.
- num_leaves (100): El mayor número de hojas por árbol permitió que el modelo ajustara mejor las relaciones complejas presentes en las URLs.

Referencias

Atuncar E., Chuan A., Pachas G., & Raez H. (2024). *Algoritmos Basados en Inteligencia Artificial para la Detección y Prevención de Ataques de Ingeniería Social: Revisión Sistemática.* LACCEI. https://laccei.org/LACCEI2024-CostaRica/papers/Contribution_1026_final_a.pdf

Crume Jeff. (2025). *Cost of a Data Breach Report 2025*. IBM Security. https://www.ibm.com/reports/data-breach

Diaz A., & Goitia S. (2024). *El delito de phishing en las entidades financieras del Perú*. Repositorio Institucional de la Universidad Autónoma del Perú. https://repositorio.autonoma.edu.pe/bitstream/handle/20.500.13067/3416/Diaz%20Pari%2C%20A.%20C.%20C.%2C%20%26%20Goitia%20Cardenas%2C%20S.%20E.pdf?sequence=1&isAllowed=y

Farias A. & Centeno D. (2024). *Modelos de inteligencia artificial para la prevención de ataques cibernéticos en organizaciones.* Repositorio Institucional de la Universidad Politécnica Salesiana. https://dspace.ups.edu.ec/bitstream/123456789/27884/1/UPS-GT005371.pdf

Hernández Dominguez, A., & Baluja García, W. (2021). Principales mecanismos para el enfrentamiento al phishing en las redes de datos. *Revista Cubana de Ciencias Informáticas*, *15*(4), 413-441. Principales mecanismos para el enfrentamiento al phishing en las redes de datos

Mesa, D. S., & Tabares, R. B. (2015). Técnicas de detección y control de phishing. *Cuaderno activa*, *7*, 75-81. Vista de Técnicas de detección y control de phishing.

Kaur, K., & Jain, A. K. (2025). A Survey on Phishing Attack Taxonomy, Detection Techniques, Datasets, and Security Measures. Journal of Applied Security Research, 1–52. https://doi.org/10.1080/19361610.2025.2508726

Dataset de phishing de Mendeley

Mendeley Data. (2020). *Phishing Website Dataset*. Mendeley. Disponible en: https://data.mendeley.com/datasets/vfszbj9b36/1

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An Introduction to Statistical Learning*. Springer Texts in Statistics.