專題報告

多模態監督式機器學習判斷釣魚網站

資工四 羅宣祐
資工四 黃亭皓

一.摘要

隨著詐騙案件日益猖獗，釣魚網站的數量與產生速度不斷上升，傳統的黑名單與規則比對技術難以有效應對。本專題提出一種**多模態監督式機器學習方法**，透過整合 URL、HTML、JavaScript 與 DNS 等不同層面的特徵，並以模型融合（Fusion）方式進行判斷，以提升檢測的準確率與穩定性。實驗結果顯示，單一模態雖能在部分情境下有效檢測釣魚網站，但因適應性不足，容易受特定特徵限制而導致誤判或漏判；相較之下，多模態融合能有效互補不同特徵間的不足，降低誤判率並提升整體識別效能。本研究驗證了**特徵與模型融合**在釣魚網站偵測上的優勢，提供一個更可靠且具實用性的檢測方案，以協助降低使用者遭受網路詐騙的風險

二.研究動機

近年台灣詐騙案件急遽上升，2024 年 10 月單月受理案件高達 18,094 件，財損逾 119 億元，其中釣魚網站為主要手法之一。傳統黑名單與規則比對方式在面對自動化工具與 AI 技術生成的大量新型釣魚網站時，往往無法即時更新，導致偵測成效有限。加上釣魚網站常利用縮網址、符號混淆或 JavaScript 隱藏代碼等方式規避檢測，使得傳統方法更加無力。

為解決此問題，本研究嘗試透過**多模態監督式機器學習**，整合 URL、HTML、JavaScript 與 DNS 多層面特徵，並採用模型融合（Fusion）方式，以提升偵測效能與穩定性。本研究聚焦以下三個問題：

1. **如何提升準確性並降低誤判率**

   避免模型過度依賴單一特徵，造成合法網站被誤判或漏判。

2. **如何因應釣魚網站的高度變化性**

   在面對快速更換網域、結構或仿冒網頁時，仍能準確提取特徵。

3. **如何整合多模型輸出並確保一致性**

探討不同融合策略對最終判斷的影響，以維持檢測的可靠性與實用性。

三.文獻探討

根據 Rasha Zieni 等人 [1] 在其釣魚網站檢測調查研究中，目前主要的檢測釣魚網站的方法主要分為三個面向:List-Based Detection、Similarity-Based Detection 以及目前被廣泛發展研究的Machine Learning-Based Detection

3.1 List-Based Detection

List-Based Detection 係指透過傳統的Blacklist/Whitelist 防禦惡意網站，黑名單（記錄已知的釣魚網站）和白名單（記錄可信任的網站）。
大多數瀏覽器(如Google)都內建這類清單來保護用戶。如[1]所述，
清單偵測的方式的優點在於簡單且快速，對已知網站有效，可以準確封鎖已識別的釣魚網站，但缺點在於需要時間辨識未知的網站，對於未知或新型的釣魚網站的防禦能力有限，現今的工具包很多可以快速產生新型的釣魚網站，且 List-Based Detection 會需要頻繁更新list 中的資料，無法有效率的應對快速且大量生成的釣魚網站。攻擊方(Adversary)也發展了十分成熟的隱匿技術如 URL 變形、誤植域名(typosquatting)及域名劫持(Combosquatting)等方式去規避檢測，能延緩該域名被檢測且被列入名單的時間。

3-2 Similarity-Based Detection

Similarity-Based Detection 透過比對網站的文本、版面結構、圖片與網址，與其合法網站的內容進行比對，以判斷是否為釣魚網站。鑒於目前許多釣魚網站的目的為透過社交工程去模仿模仿合法網站的外觀來欺騙使用者，以降低使用者的戒心，所以此方可以應對目前新型釣魚攻擊及zero-day 攻擊網站。相似度通常以文本內容(Textual Content)或視覺內容(Visual Content)進行比較，文本內容比較標籤HTML 結構、CSS 樣式與 DOM 樹結構來評估相似度，例如透過關鍵字權重去評估相似度(Term Frequency-Inverse Document Frequency)，由Yue Zhang 等人於2007 年提出的CANTINA 方法[2]。而視覺內容相似度透過圖片特徵、頁面佈局、顏色匹配來分析相似度。這類檢測方法的優點在於能應對新型釣魚網站有效，特別對於模仿合法網頁的類型能有效識別，缺點在於運算成本較高，特別針對圖片對比會需要需要較高的計算資源。且需要大量存儲需求，需要維護大量合法網站的數據庫，以進行比對。

3-3 Machine Learning-Based Detection

目前基於機器學習(Machine Learning-Based)的方法已成為目前最廣泛研究的釣魚網站檢測技術，因為其它具備學習能力，靈活且可有自適應性，可以偵測zero-day 攻擊，並能夠應對新型的釣魚網站設計，目前機器學習方法將釣魚網站檢測視為二元分類(Binary Classification)，其主要目標是區分網頁是合法(Legitimate)還是釣魚(Phishing)結果。

Machine Learning-Based是利用特徵工程(Feature Engineering)提取網頁的屬性，並透過分類演算法來判斷網站是否為釣魚網站，如何選取適當特徵對於提升模型準確率、降低計算成本、避免過擬合等攸關模型效能等是重要的因素，根據[1]中將主要特徵分為三類:URL-based、HTML-based 和Network-based。URL 是釣魚網站最常見的特徵媒介，特別是上文提到為了規避傳統偵測而使用URL 隱蔽技術，進而成為釣魚網站的重大特徵，此外，也可以透過URL 熵值，用來衡量 URL 的隨機性，隨機性越高可疑性也越強，透過工具包產生的可能性也越高。而HTML-based 則是取用了Similarity-Based Detection 相關的特徵，如HTML 及DOM 結構特徵，還包括其他與網站原始碼相關的資訊，如前端事件驅動語言JavaScript 或CSS 皆可做為特徵選取的元素。Jian Mao 等人[3]於2019 年透過研究了網頁 CSS 規則的相似性，並提出了基於靜態特徵提取(Static Feature Extraction)的方法量化網頁的視覺佈局，該方法透過CSS 屬性轉換為特徵向量，並可以透過比較其相似性來識別釣魚網站而除了網頁的內容，研究者還會分析網頁的網絡行為，此為Network-based 的特徵，透過分析DNS 及相關數據，也可以提出相關可能的特徵，例如觀察Domain Age 太短、IP 歷史變更頻率高或是如果特定 IP 地址同時託管大量釣魚網站，則可能存在風險

除了特徵工程外，機器學習分類器(Machine Learning Classifiers)對模型訓練成效也會有影響，目前對於這類的研究以監督式學習演算法為主，主要為傳統機器學習模型和深度學習。 傳統機器學習演算法適合中小型數據集，且解釋性較強，如隨機森林（Random Forest, RF）和支援向量機（Support Vector Machine, SVM）為兩種十分常見和被比較的傳統機器學習演算法，而隨著數據規模增大，深度學習也被應用於這類研究中，Peng Yang 等人的研究 [4]提出了一種基於多維特徵的深度學習方法，結合 URL、HTML 和 DNS 特徵，並透過深度學習來識別釣魚網路，在系統中建立多種模型，如下所述:

LSTM：用於分析 URL 和 HTML 代碼的模式。

CNN（卷積神經網絡）：用於提取網頁的視覺特徵。

多層感知機（MLP）：用於學習網站的結構化數據(如 DNS record)
研究使用了多個公開數據集，如Alexa Top 1M，該模型達到了 98.6% 的準確率，針對Zero-day 釣魚攻擊的檢測上表現良好，但是缺點在於LSTM 模型計算開銷較大，不適用於即時檢測系統。且需要大量標註數據來訓練，以確保模型的泛化能力。

Lei Zhang 等人於2021 研究提出了MultiPhish[5]，融合多模態特徵，並使用神經網絡進行釣魚網站檢測的方法，該系統依賴於三項特徵:網站域名(Domain)、網站圖標(Favicon)及 URL，透過Encoder 去提取個別的特徵，domain encoder 首先將域名字符嵌入向量，隨後CNN 卷積神經網絡提取不同粒度(層級)的特徵，得到相應的特徵映射，再來透過最大池化(Max Pooling)(降維技術，主要用於減少特徵維度及提高效率和泛化能力)提取最重要的數據，Favicon encoder 則使用預訓練的VGG-19 模型提取圖片特徵，隨後使用全連接層（FC） 進行特徵映射，使域名特徵與圖標特徵具有相同維度。結果顯示，三者融合的效果與移除特定特徵或單一特徵相比，具有最高的準確率，高達97.79%，可知多重特徵對於判斷釣魚網站有更穩定的表現，單一特徵可能無法充分捕捉釣魚網站的多樣性，而多重特徵融合能夠提供更穩健的識別能力。

除了深度學習的應用，根據相關文獻，我們可以針對不同的潛在特徵識別出最適合的分類器候選模型，以確保在各個特徵模態中選擇最具代表性的機器學習演算法，Norzaidah Binti Md Noh 等人於2021 提出對於釣魚網站檢測對比[6]，針對隨機森林(Random Forest)和支持向量機(SVM)兩種演算法對URL 特徵的表現做對比，使用的資料集來自馬來西亞砂勞越大學（UNIMAS）計算機科學與信息技術學院（FSIT）提供的公開資料集。該資料集包含30,000 個網站，其中包括15,000 個釣魚網站和15,000 個合法網站，資料集在數據預處理階段經過處理，通過對HTML 代碼進行標記化處理，因為需要處理大量數據。標記器是通過一種名為字節對編碼(Byte Pair Encoding，BPE)的算法實現的。在字節對編碼標記器創建後，它將用於標記化HTML 文件並提取特徵以供機器學習使用隨機森林（Random Forest）和支持向量機（SVM）算法進行分類。資料集被劃分為90％的訓練數據和10％的測試數據。研究結果顯示，隨機森林(Random Forest)算法在準確度、精度、召回率和F1 分數等性能指標方面比支持向量機（SVM）表現更佳，隨機森林對釣魚類別的正確預測數量最高，且對合法類別的錯誤預測數量最低，僅為2。而支持向量機（SVM）模型對釣魚類別的錯誤預測數量較高，該值為1494，且隨機森林的處理時間遠低於SVM。
Sadia Parvin Ripa 等人於2021 年的研究[7]提出對於不同的機器學習演

算法針對不同網頁特徵進行實驗，測試演算法包含XGBoost、隨機森林等， 針對釣魚網址(Phishing URL)偵測使用Kaggle 數據集訓練，共4799筆數據，實驗結果如下:

準確率
XGBoost 94.44%
隨機森林（Random Forest） 94.16%
決策樹（Decision Tree） 91.80%
K-近鄰演算法 90.48%
邏輯回歸（ Logistic Regression）89.37%

根據[6],[7]可知對於URL 檢測，適合使用XGBoost 或是隨機森林。而針對Address Bar based Features 、Abnormal Based Features、HTML 、JavaScript based Features 和Domain based Features 等四大類特徵，使用UCI 機器學習數據集，實驗結果如下:

準確率
隨機森林 96.8%
支持向量機（SVM） 96.6%
決策樹（Decision Tree） 96.47%
K-近鄰演算法 94.09%
邏輯回歸（Logistic Regression） 92.28%

目前多數的多模態研究皆聚焦於深度學習的部分，具有自動提取特徵、普遍擁有良好效果，但是訓練成本較高，計算開銷較大，因此我們希望研究傳統機器學習演算法配合多模態多模型，施用最適合的分類器處理每個特徵模態融合，最後融合多個模型的預測結果，形成最終分類結果

四.實驗步驟

在研究中，我們透過多個來源蒐集釣魚網站與合法網站的資料，url的部分使用Hannousse, Abdelhakim[8] 提供的釣魚資料集，篩選出url及外部查詢特徵如web rank等。HTML、Javascript及DNS相關之釣魚網站由PhishTank，合法之網站由Tranco取得（Tranco 提供網站熱門度排行，故一般認為高排名域名為合法網站）

- URL和外部查詢特徵

從[8]資料集篩選出符合url和相關外部查詢特徵
資料總數:11430
Legal:5715(50%) Phish:5715(50%)

特徵:

url基礎結構與字元統計

length_url	整個 URL 字元數
length_hostname	主機名字元數
ip	URL 是否直接用 IP
nb_subdomains	子網域數量
prefix_suffix	網域是否含 - 分隔品牌
port	是否指定非標準埠
punycode	是否含 Punycode（xn--）
path_extension	路徑副檔名種類/標誌
https_token	協定名是否為 https
http_in_path	路徑中出現 http/https 字樣
nb_dots, nb_hyphens, nb_at, nb_qm, nb_and, nb_or, nb_eq, nb_underscore, nb_tilde, nb_percent, nb_slash, nb_star, nb_colon, nb_comma, nb_semicolumn, nb_dollar, nb_space	對應符號在 URL 出現次數（nb_or由於值一律為0 不具備判斷價值於訓練時捨棄）
nb_www	www 出現次數
nb_com	.com 出現次數
nb_dslash	// 出現次數
ratio_digits_url, ratio_digits_host	數字比例（高比例常見自動化或隨機）
length_words_raw	以非文字分隔後的詞數
char_repeat	字元重複度量

shortest_words_raw longest_words_raw avg_words_raw	URL 整體詞長度統計

外部查詢與TLD特徵

statistical_report	來自第三方惡意回報統計
web_traffic	網站流量指標
google_index	是否被 Google 收錄
page_rank	網頁排名
tld_in_path, tld_in_subdomain	TLD 名字是否出現在路徑/子域
abnormal_subdomain	子域是否異常
suspecious_tld	是否為高風險 TLD（黑名單表）
nb_redirection	轉址次數
nb_external_redirection	是否導向外網域
phish_hints	是否含關鍵字
domain_in_brand	domain 是否出現在已知品牌詞庫
brand_in_subdomain	子域包含品牌詞
brand_in_path	路徑包含品牌詞
random_domain	域名是否近似隨機（Shannon Entropy）
shortening_service	是否短網址服務

- html 特徵

  由 kaggle 網站上所提供的 html 檔篩選特徵，且提供的檔案已有分類為正常網站和釣魚網站，只需提取特徵並標記就可

  資料總數:10308

  Legal:6138(59.5%) Phish:4170(40.5%)

  特徵:

num_links	頁面中 <a> 超連結的數量
num_forms	頁面中 <form> 表單的數量
num_inputs	頁面中 <input> 欄位的數量
num_scripts	頁面中 <script> 標籤的數量
num_iframes	頁面中 <iframe> 內嵌框架的數量

num_meta	頁面 <head> 中 <meta> 標籤的數量（如描述、關鍵字、編碼）
num_divs	頁面 容器標籤的數量
num_images	頁面 <img> 圖片的數量。
num_inline_styles	HTML 內直接寫在標籤的 style="..." 次數
num_css_files	頁面中引用的 CSS 檔案總數（<link rel="stylesheet" ...> + <style>）
num_external_css	外部載入的 CSS 檔案數量
has_login	頁面是否含有登入相關功能
has_password_field	是否存在 <input type="password">
has_https_links	頁面中是否有 HTTPS 的連結
text_length	頁面中可見文字的總長度
num_exclamations	文字中!的出現次數
num_uppercase_words	全大寫單字的數量
has_sensitive_keywords	是否包含敏感詞彙
external_script_ratio	外部 JavaScript 檔案數量 ÷ 總 JavaScript 數量。
external_links_ratio	外部連結數量 ÷ 總連結數量
avg_link_length	所有連結（href）平均長度

- Javascript 特徵

特徵參考[8]提供的資料集少部分特徵及"Detection of Malicious JavaScript Code in Web Pages"[9]作者所參考的文獻特徵以及作者自行提出之特徵

釣魚網站資料來源於 Phishtank
合法網站資料來源於 tranco 排名前 70000 筆

由於 tranco 僅提供域名，需要預先處理 resolve 為 url

使用爬蟲取得網頁 javascript 資料
針對每一個 URL，我們先解析 HTML，擷取內嵌‘<script>‘ 的文字內容以及所有外部 js 檔案（對相對路徑使用‘urljoin‘ 轉為絕對 URL，再以 HTTP 請求取得）

考量資料量龐大（數萬筆）且網路 I/O 為瓶頸，透過進行多線程 thread_pool 抓取與處理

過程中由於部分網頁防禦爬蟲或網頁連線逾時
或者合法域名並非常見型態 導致無法取得資料
故實際取得的資料會少於原始資料數
。

資料總數: 79,021 筆
Legal: 41,475 (52.5%) Phish: 37,546 (47.5%)

特徵

eval_count	使用 eval() 動態執行程式碼次數
function_count	自定義函數的數量
document_write	動態注入內容次數
setTimeout	setTimeout 呼叫次數，常用於延遲惡意行為執行
length	整體 JavaScript 長度
entropy	字元熵用來衡量混淆程度
iframe_count	iframe 元素出現次數
window_location	window.location 的使用次數，可能做跳轉
createElement_count	createElement() 呼叫次數，動態建立 DOM
appendChild_count	appendChild() 呼叫次數
dispatchEvent_count	模擬事件觸發次數
onmouseover_count	滑鼠移動事件次數
fromCharCode_count	字串動態組合次數，可能有混淆目的
charCodeAt_count	charCodeAt() 呼叫次數
escape_count	escape() 呼叫次數
unescape_count	unescape() 呼叫次數
digit_count	數字字元數
hex_count	十六進位數出現次數
backslash_count	反斜線數量(編碼與逃脫字元)
pipe_count	管線符號數量(混淆或特殊語法)
percent_count	百分比出現次數

curly_brace_count	大括號出現次數
space_count	空白字元出現次數
parseInt_count	字串轉整數使用次數
classid_count	classid 屬性出現次數 ActiveX 常見於舊版 IE 漏洞攻擊
ActiveXObject_count	ActiveXObject() 呼叫次數 ActiveX 常見於舊版 IE 漏洞攻擊
concat_count	拼接字串使用次數
indexOf_count	indexOf() 呼叫次數
substring_count	substring() 呼叫次數
replace_count	replace() 呼叫次數
addEventListener_count	addEventListener() 呼叫次數
attachEvent_count	attachEvent() 呼叫次數
getElementById_count	getElementById() 呼叫次數
search_count	search() 呼叫次數
split_count	split() 呼叫次數
onerror_count	onerror 錯誤事件的次數
onload_count	onload 載入事件的次數
onbeforeunload_count	onbeforeunload 卸載事件次數（阻止關閉頁面）
setAttribute_count	setAttribute() 呼叫次數
charAt_count	charAt() 呼叫次數
consoleLog_count	console.log() 呼叫次數
js_file_count	引入 js 檔案次數
php_file_count	引入 php 檔案次數
random_count	Math.random() 呼叫次數
decode_count	decodeURI/URIComponent 函數使用次數
toString_count	toString() 呼叫次數
encoded_char_count	編碼字元的數量
long_string_count	超過 200 字元的長字串數量
max_word_length	最長詞彙的長度
min_word_length	最短詞彙的長度
entropy_longest_word	最長詞彙的字元熵
popup_window	是否出現 prompt() 彈窗
right_clic	是否偵測右鍵
share_of_digits	數字字元佔比
share_of_hex	0x 類十六進位字元佔比
share_of_backslash	反斜線字元佔比

share_of_pipe	‘|‘ 符號佔比
share_of_percent	‘%‘ 符號佔比
share_of_curly_braces	‘{}‘ 括號佔比
share_of_spaces	空白字元佔比
unsafe_anchor_percent	超連結元未加 rel=noopener 的比例

- DNS 特徵

  釣魚網站資料來源於 Phishtank

  合法網站資料來源於 tranco

  由於進行特徵提取時部分網頁可能會沒有回應，因此需要過濾並篩選出有回應的網站來篩選特徵

  資料總數:6887

  Legal:3500(50.8%) Phish:3387(49.2%)

domain	網域名稱本身
domain_length	網域名稱的長度
has_suspicious_keywords	網域名稱中是否包含可疑關鍵字
A_record_count	A 記錄（IPv4 位址）
ttl_avg	A 記錄的平均 TTL
CNAME_count	CNAME 記錄數量
MX_count	MX_count
NS_count	NS 記錄數量
TXT_count	TXT 記錄數量
has_AAAA	是否有 AAAA 記錄（IPv6 位址）
whois_days_to_expire	網域剩餘的到期天數
whois_success	是否能成功查詢 Whois 資料

- 模型訓練

在完成各模態（URL、HTML、JavaScript、DNS）特徵的擷取與清理後，將資料集依照特徵類別分別儲存為 CSV 檔，再導入 COLAB Python 環境進行訓練與測試。整體流程如下：

1 匯入資料

- 使用 pandas 將特徵資料讀入程式。
- 檢查並處理缺失值，針對部分特徵若為空值則填補為 0 或刪除異常樣本。
- 標籤（label）為二元分類，0 代表合法網站 (Legitimate)，1 代表釣魚網站 (Phishing)。

2. 資料切分

將完整資料集依照 75% 訓練集、25% 測試集進行隨機劃分，並保持類別分布比例一致（stratified split），確保模型在正負樣本數量上平衡。

3. 資料標準化

- 針對連續型特徵（如字元長度、字元比例、熵值）進行標準化處理，使其均值為 0、變異數為 1，以避免不同量級的特徵對模型產生偏差。
- 部分模型（如樹模型）不依賴標準化，但為確保公平比較，在所有模型中統一採用標準化版本的資料。

4. 分類器選擇

本研究使用多種監督式學習分類器進行比較：

- Logistic Regression：線性分類模型。
- K-Nearest Neighbors (KNN)：基於距離度量
- Support Vector Machine (SVM)：利用最大化間隔的超平面進行分類
- Random Forest (RF)：集成式樹模型，能有效處理高維與非線性特徵
- Gradient Boosting: 建立弱分類器並提升權重
- XGBoost (XGB)：梯度提升樹模型，對於釣魚網站偵測任務表現出色，

5. 模型訓練與測試

- 實驗過程中使用網格搜尋 GridSearchCV 進行超參數搜尋最佳參數組合。
- 使用訓練集進行模型訓練，測試集進行評估。
- 評估指標包括：Accuracy（準確率）、Precision（精確率）、Recall（召回率）、F1-score，並搭配 Confusion Matrix 分析模型的誤判情況。

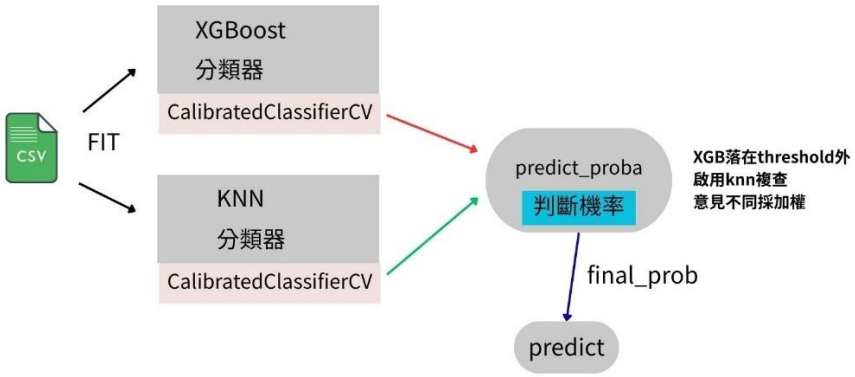
- Javascript cascade分層架構

在 JavaScript 特徵的子模型中，我們後續採用了 XGBoost 與 KNN 的級聯（Cascade）設計

- XGBoost 在整體效能上最為突出，能有效捕捉非線性關係並提供高準確率與穩定的召回率，但在可能對於邊界樣本上的判斷偏樂觀，容易出現誤報。
- KNN 雖然整體表現不如 XGBoost，但具有較高的 recall，能依據鄰近樣本的分布對難以判斷的案例做補強

JS_CASCADE 流程設計如下：

1. 由於需要針對模型輸出之機率進行判別，但是原生 sklearn 套件之機率存在較嚴重之誤差，所以先使用 calibratedClassifierCV 進行機率之優化(isotonic regression)
2. 第一層：XGBoost 先進行預測，若其輸出機率超過 threshold 區間（>0.7 或<0.3）則視為對預測具備信心，直接輸出結果。
3. 不確定區（0.3–0.7 之間）：再交由 KNN 進行複查。
    - 若 KNN 與 XGBoost 意見一致，直接採用結果
    - 若兩者意見相左，則進行加權融合，再以閾值判斷。
4. 僅對少數難例進行二次判斷，延遲成本低。
5. 框架使用 sklearn 規定之格式，以便與其他功能相容



- 多模型融合fusion

fusion測試資料由Phishtank和Tranco提取
總合法樣本:1508筆 總釣魚樣本:2269筆
由於網站可能針對爬蟲建立防禦或是釣魚網站被關閉導致資料無法透過取得

- url: 合法樣本:1508筆(39.93%) 釣魚樣本:2269筆(60.07%)
- html: 合法樣本:1508筆(40.26%) 釣魚樣本: 2238筆(59.74%)
- javascript: 合法樣本: 1415筆(45.48%) 釣魚樣本: 1696筆(54.52%)
- dns: 合法樣本: 1508筆(74.51%) 釣魚樣本: 516 筆 (25.49%)

根據URL及外部查詢、HTML、DNS針對訓練資料的測試結果選擇準確率最高的分類器，javascript使用級聯模型

(1)權重初始設定與模態模型

設定了四個模態的順序與初始權重分配：

- 模態順序：JavaScript、HTML、URL、DNS。
- 初始權重：JS 權重最高 (0.6)，URL 次之 (0.3)，HTML 與 DNS 權重較低 (各 0.05)。

同時，將四個模態的「已訓練模型」與「對應的測試資料」組合起來，建立一個模態對應表，方便後續統一處理。

(2) 預測機率計算與缺失值處理

在測試階段，我們需要取得每個模態模型對於樣本的預測機率，但資料中可能存在缺失或無法解析的特徵，為了解決未知缺值的情況進行以下處理：

- NaN 標記：對於每一筆樣本，若該模態特徵中存在缺失，則記錄一個 nan_mask，以便後續融合時排除這些模態。
- 特殊值填補：將缺失特徵填入一個特殊常數 (-9999)，使得模型仍能接收輸入並輸出預測機率。

最終，我們為每個模態建立了一個儲存每個模態模型計算出來的預測機率 probas，以及一個 nan_masks 記錄哪些網站樣本有缺失值，確保後續的加權融合能忽略特徵資料不可靠的模態。

(3) 定義加權 F1 函數

為了在不同模態之間找到最佳的權重組合，定義以 F1-score 為核心的目標函數 weighted_f1，排除了資料缺失的模態，並後續將其作為全域搜尋的依據。

- 輸入一組候選權重向量，對應到四個模態（JS、HTML、URL、DNS），形成 w_dict。
- 對每一筆樣本，先篩選出有效模態，即該樣本在該模態中沒有缺失值。
- 若該樣本的所有模態皆無效，則直接給予中性機率 0.5。
- 排除無效模態後，對於有效模態，將其權重重新正規化（確保加總為 1），再計算加權平均的惡意機率。
- 若最終機率大於等於 0.5，則預測為「惡意」；否則為「合法」。
- 計算這組權重下整體測試集的 F1-score，並取其負值作為回傳值，方便後續最佳化演算法進行「最小化」搜尋。

(4) 全域搜尋最佳權重

後續使用差分演化演算法（Differential Evolution）進行全域搜尋

- 搜尋範圍設定
  - 每個模態的權重下限為 0，上限為 1
  - 確保權重向量的長度與模態數一致

- 差分演化搜尋
  - 全域搜尋會在定義範圍內隨機生成多組候選權重，並透過突變與交叉不斷更新，尋找能讓目標函數（F1-score）最小化的解。
  - 因為 F1 函數回傳的是負的 F1-score，因此最小化過程即對應到最大化 F1-score。

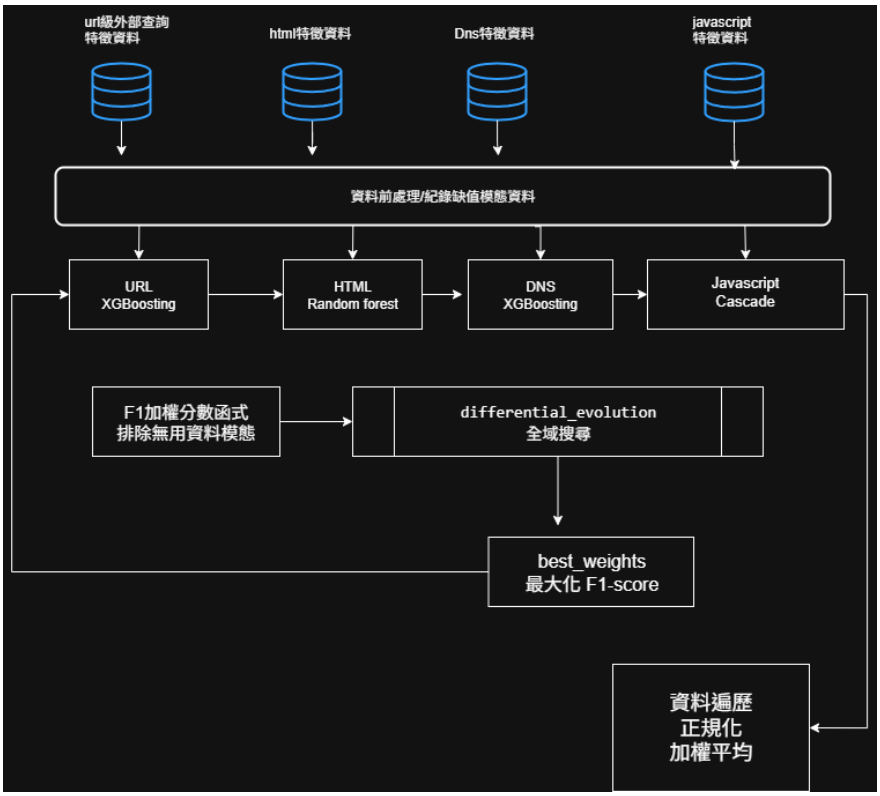
- 正規化處理: 將權重向量進行正規化，使得所有權重加總為 1，

- 加權平均預測
  - 最終得到一組全域最佳權重分配。
  - 使用最佳權重對各模態機率進行加權平均。
  - 若某筆樣本缺少部分模態輸出，則僅使用有效模態的權重並正規化。
  - 對所有樣本計算加權機率，並進行閾值搜尋（0.01 至 0.99）以找出最佳分類門檻

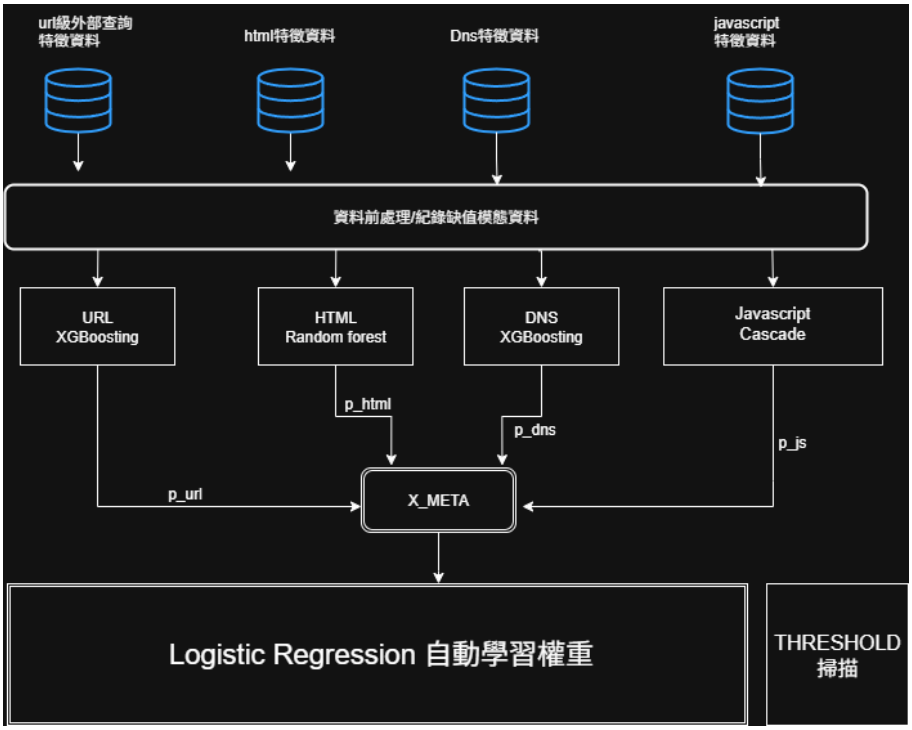
(5) 使用 Logistic Regression 進行 Meta-learning

除了透過差分演化演算法搜尋權重外，也使用另一方法 使用次層學習（Meta-learning）的方式，讓模型自動學習各模態之間的最佳組合。

1. 建立次層特徵矩陣
    - 將四個模態輸出的機率結果視為新的特徵，組合成一個特徵矩陣 X_meta，其維度為「樣本數 × 模態數」。
2. 缺失值處理
    - 若某模態無法輸出機率（NaN），則以中性值 0.5 進行填補，表示該模態對這筆樣本沒有偏向，避免影響學習
3. 次層模型訓練
    - 使用 Logistic Regression 作為次層模型，輸入為 X_meta，輸出為真實標籤 y_test
    - Logistic Regression 能自動為不同模態學習適合的權重，並且在必要時考慮模態之間的交互效果。
4. 最終機率輸出
    - 訓練完成後，次層模型會對每個樣本輸出最終的釣魚機率，作為融合後的分類依據。



全域搜尋最佳權重架構圖



Meta-learning 架構圖

五.實驗結果

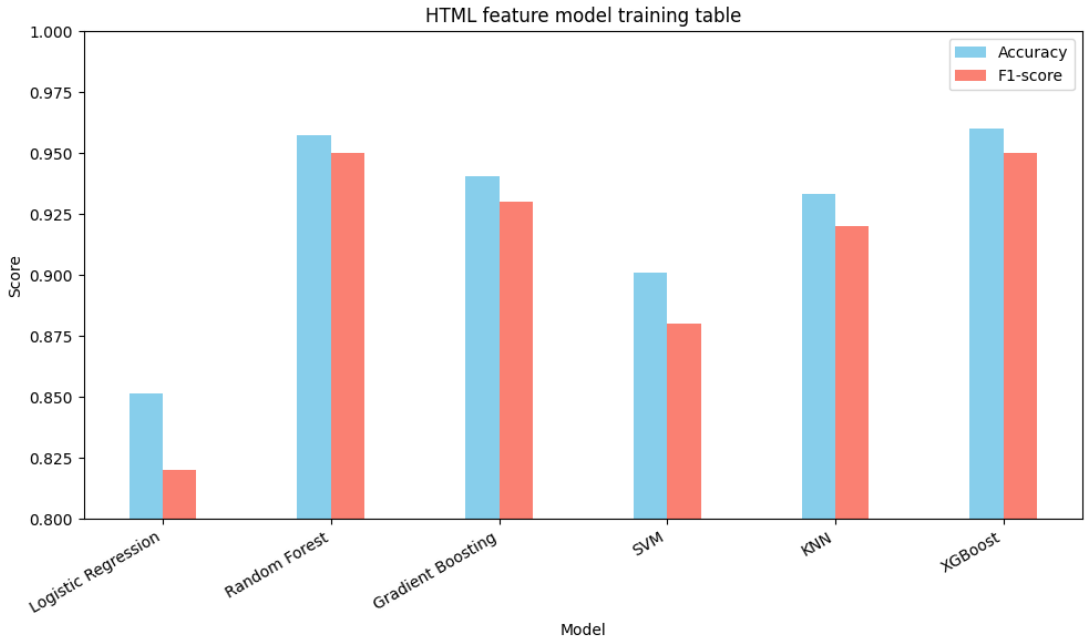
以下為根據訓練資料各特徵類型針對各分類器之訓練結果
url 表現最佳為 XGBoost
最佳參數: {'colsample_bytree': 1, 'learning_rate': 0.3, 'max_depth': 10, 'n_estimators': 100, 'subsample': 1}
交叉驗證準確率: 0.963076187353118
測試集準確率: 0.9632610216934919

Html 表現最佳為 Random Forest

最佳參數 {'max_depth': 20, 'n_estimators': 100}

交叉驗證準確率 0.9550089861624125

測試集準確率: 0.9573229873908826

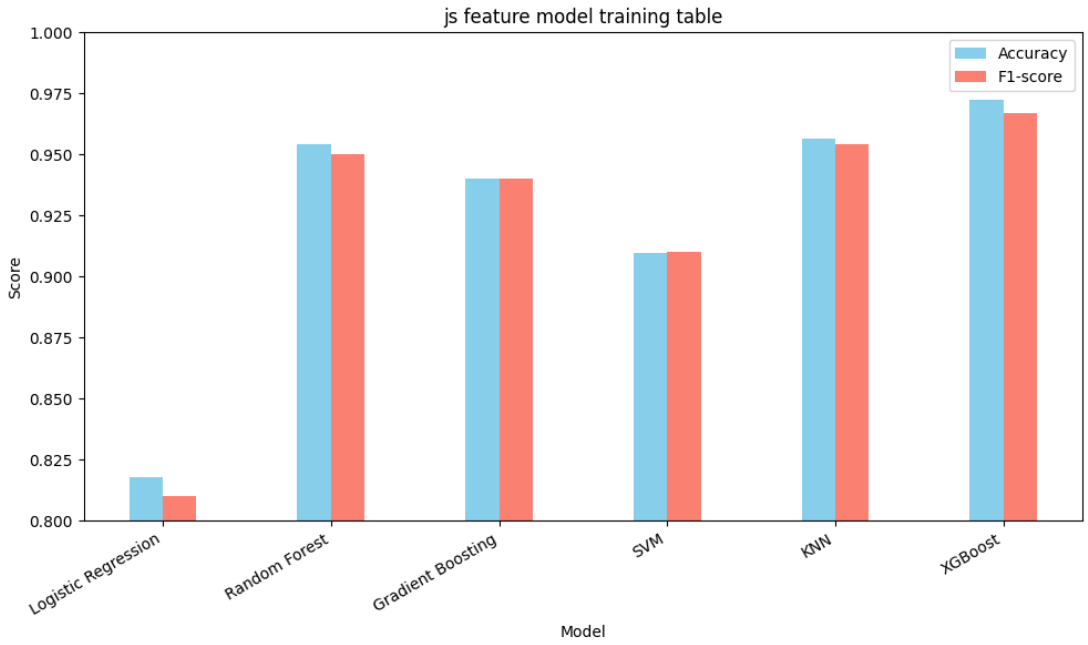


Javascript 表現最佳為 XGBoost

最佳參數: {'colsample_bytree': 1, 'learning_rate': 0.2, 'max_depth': 8, 'n_estimators': 150, 'subsample': 0.8}

交叉驗證準確率: 0.9701510166202649

測試集準確率: 0.9724640615509212



Dns 表現最佳為 XGBoost

最佳參數 {'colsample_bytree': 1, 'learning_rate': 0.1, 'max_depth': 6, 'n_estimators': 200, 'subsample': 1}

交叉驗證準確率 0.8935000000000001

測試集準確率: 0.897

JS_cascade 分層架構結果

```
Custom Cascade Classifier Evaluation
測試集準確率: 0.9734257946952825
混淆矩陣:
 [[10249   120]
 [  405  8982]]
分類報告:
              precision    recall  f1-score   support

           0       0.96      0.99      0.98     10369
           1       0.99      0.96      0.97      9387

    accuracy                           0.97     19756
   macro avg       0.97      0.97      0.97     19756
weighted avg       0.97      0.97      0.97     19756
```

XGB 與 KNN

```
測試集準確率: 0.9722109738813525
混淆矩陣:
 [[10220   149]
 [  400  8987]]
分類報告:
              precision    recall  f1-score   support

           0       0.96      0.99      0.97     10369
           1       0.98      0.96      0.97      9387

    accuracy                           0.97     19756
   macro avg       0.97      0.97      0.97     19756
weighted avg       0.97      0.97      0.97     19756
```

```
測試集準確率: 0.9650739015995141
混淆矩陣:
 [[10170   199]
 [  491  8896]]
分類報告:
              precision    recall  f1-score   support

           0       0.95      0.98      0.97     10369
           1       0.98      0.95      0.96      9387

    accuracy                           0.97     19756
   macro avg       0.97      0.96      0.96     19756
weighted avg       0.97      0.97      0.97     19756
```

結果顯示：

- 單用 XGBoost 的測試集準確率為 97.22%，誤報（FP）149、漏報（FN）400。
- 單用 KNN 的準確率為 96.51%。
- 採用 JS_CASCADE 後，準確率提升至 97.34%，誤報數降至 120（-19%），漏報僅小幅增加至 405。
- 在幾乎不影響召回的情況下，降低了誤報率

多模型融合fusion

使用新的提取的測試資料進行測試

- 子模型表現

- URL Model (Excluding NaN Samples):
  - Accuracy: 0.7466243050039714

- JS Model (Excluding NaN Samples):
  - Accuracy: 0.8344583735133397

- HTML Model (Excluding NaN Samples):
  - Accuracy: 0.39829151094500803

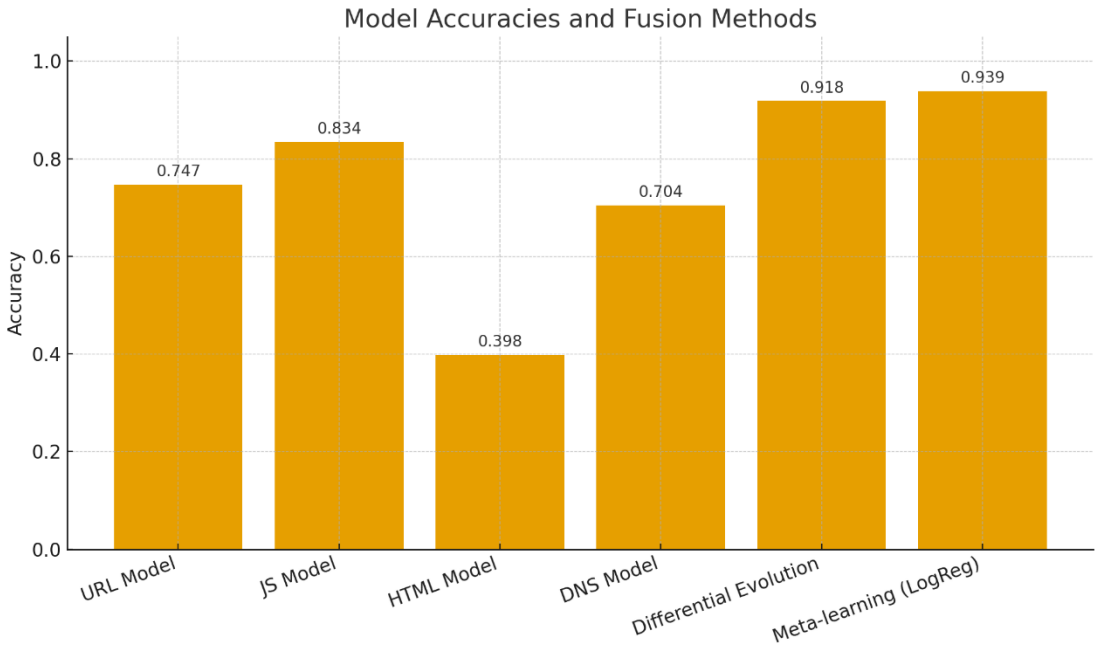
- DNS Model (Excluding NaN Samples):
  - Accuracy: 0.7040513833992095

- differential_evolution 全域搜尋
  - 最佳權重:
    - 'js': 0.5252164716997149
    - 'html': 0.000947031494330571
    - 'url': 0.3968887944650153
    - 'dns': 0.07694770234093935
  - 最佳閾值: 0.200
  - 對應最佳 F1-score: 0.9347
  - 準確率: 0.918453799311623

  混淆矩陣:
  [[1264 244]
  [64 2205]]

- Meta-learning (Logistic Regression)
- 最佳閾值: 0.600
- 對應最佳 F1-score: 0.9479
- 準確率: 0.9385755890918719

混淆矩陣:
[[1433 75]
[157 2112]]



六.結論

本研究針對釣魚網站的偵測問題，提出一套整合多層特徵的多模態監督式機器學習架構，結合 URL、HTML、JavaScript 與 DNS 四種模態的特徵，並透過多模型融合（Fusion）提升整體辨識效能。各模態均採用最適化之監督式分類器進行訓練與測試，並在融合層中分別採用差分演化（Differential Evolution）與次層學習（Meta-learning）策略進行最終判斷

- **JS_CASCADE 架構效益**

在 JavaScript 模態中，單一 XGBoost 模型雖能達到 97.22% 的準確率，但誤報數達 149。透過結合 KNN 的級聯架構，針對 XGBoost 判斷不確定區間（0.3–0.7 機率）進行二次複查，最終準確率提升至 **97.34%**，誤報數降至 **120**（降低約 19%），在召回率僅微幅變動下有效降低誤判

● fusion 單模態模型效能

各模態的最佳模型分別為：

- **URL 模型**：XGBoost，準確率 0.7466
- **JavaScript 模型**：XGBoost + KNN 級聯架構（JS_CASCADE），準確率 0.8345
- **HTML 模型**：Random Forest，準確率 0.3983
- **DNS 模型**：XGBoost，準確率 0.7041

由結果可知，不同模態對於釣魚網站辨識的貢獻度差異明顯，其中 JavaScript 模態在特徵豐富度與可辨識性方面表現最佳，HTML 模態則因資料特性限制而準確率偏低。

各子模型在訓練階段表現良好，但在融合測試資料中準確率皆有一定程度下降。此現象主要源於：

- 資料時效性差異：訓練資料與新提取的 Fusion 測試資料之間相隔數月，
- 釣魚網站快速變化：網頁結構與特徵隨時間持續演化，導致舊模型無法完全捕捉新型特徵，
- 監督式學習的適應性限制：雖能在固定特徵空間內學習，但對於資料分布轉移反應有限，需定期進行模型重訓與特徵更新

在多模態融合模型中，HTML 提供的相對增益有限，而 其他特徵提供更直接、增益更大的訊號，模型因此分配給 HTML 的權重較低。此外，HTML 特徵設計可能較單一，無法捕捉隱藏釣魚行為。

● 多模態融合結果

為整合不同模態的預測結果，本研究分別採用：

- **差分演化演算法**（Differential Evolution）

  進行權重全域搜尋，最終取得最佳權重組合
  準確率 **0.9185**，F1-score **0.9347**。

- Meta-learning (Logistic Regression)

  準確率 0.9386，F1-score 0.9479，整體表現最佳。

結果顯示，即使基底模型的表現一般，但是透過 fusion 的結果卻有出色的表現，多模態融合的偵測效果明顯優於單一模態，顯示不同層面的特徵能有效互補，提升模型對釣魚網站的判斷能力。特別是 Meta-learning 能自動調整模態權重與交互關係，取得最高準確率與穩定性。

未來展望與優化

- 引入深度學習或圖神經網路進行特徵融合
- 加入視覺相似度特徵如 favicon 辨識，更容易辨識 ai 工具包的產物
- 優化爬蟲工具，加進資料分析效率
- 建立自動化流程，應對資料漂移，確保長期檢測效能
- 擴增特徵，javascript 深入對 AST 語法樹進行分析

參考文獻

[1] R. Zieni, L. Massari and M. C. Calzarossa, "Phishing or Not Phishing? A Survey on the Detection of Phishing Websites," in *IEEE Access*, vol. 11, pp. 18499-18519, 2023, doi:10.1109/ACCESS.2023.3247135.

[2] Y. Zhang, J. I. Hong and L. F. Cranor, "Cantina: A content-based approach to detecting phishing Web sites", Proc. 16th Int. Conf. World Wide Web, pp. 639-648, May 2007.

[3] J. Mao, W. Tian, P. Li, T. Wei and Z. Liang, "Phishing-Alarm: Robust and Efficient Phishing Detection via Page Component Similarity," in *IEEE Access*, vol. 5, pp. 17020- 17030, 2017, doi: 10.1109/ACCESS.2017.2743528.

[4] P. Yang, G. Zhao and P. Zeng, "Phishing Website Detection Based on Multidimensional
Features Driven by Deep Learning," in *IEEE Access*, vol. 7, pp. 15196-15209, 2019, doi:10.1109/ACCESS.2019.2892066.

[5] L. Zhang, P. Zhang, L. Liu and J. Tan, "Multiphish: Multi-Modal Features Fusion Networks for Phishing Detection," ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Toronto, ON, Canada, 2021, pp.
3520-3524, doi: 10.1109/ICASSP39728.2021.9415016.

[6] N. Binti Md Noh and M. N. Bin M. Basri, "Phishing Website Detection Using Random Forest and Support Vector Machine: A Comparison," *2021 2nd International Conference on Artificial Intelligence and Data Sciences (AiDAS)*, IPOH, Malaysia, 2021, pp. 1-5, doi:10.1109/AiDAS53897.2021.9574282.

[7] *S. P. Ripa, F. Islam and M. Arifuzzaman, "The Emergence Threat of Phishing Attack andThe Detection Techniques Using Machine Learning Models,"* 2021 International Conference on Automation, Control and Mechatronics for Industry 4.0 (ACMI), *Rajshahi,Bangladesh, 2021, pp. 1-6, doi: 10.1109/ACMI53878.2021.9528204.*

[8]
Hannousse, Abdelhakim; Yahiouche, Salima (2021), “Web page phishing detection”, Mendeley Data, V3, doi: 10.17632/c2gw7fy2j4.3

[9]
D. R. Patil and J. B. Patil, "Detection of Malicious JavaScript Code in Web Pages," *Indian Journal of Science and Technology*, vol. 10, no. 19, pp. 1–7, May 2017, doi:10.17485/ijst/2017/v10i19/112331.