Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»
Інститут електроенергетики
(інститут)
факультет інформаційних технологій
(факультет)
Кафедра інформаційних технологій та комп'ютерної інженерії
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня бакалавра
(бакалавра, спеціаліста, магістра)

студента Ващука Дмитра Олександровича
(ПІБ)
академічної групи 126-20-1
(шифр)

спеціальності 126 Інформаційні системи та технології
(код і назва спеціальності)

за освітньо-професійною програмою

«Інформаційні системи та технології»
(офіційна назва)

на тему Програмний інструмент виявлення потенційно небезпечних URL-адрес за допомогою бібліотеки scikit-learn
(назва за наказом ректора)

Керівники	Прізвище, ініціали	Оцінка за шкалою		Підпис
		рейтинговою	інституційною	
кваліфікаційної роботи	доц. Каштан В.Ю.			
розділів:				

Рецензент				

Нормоконтролер	проф. Коротенко Г.М.			

Дніпро
2024

ЗАТВЕРДЖЕНО:
завідувач кафедри
_____інформаційних технологій _____
_____та комп'ютерної інженерії_____
(повна назва)
_____ _Гнатушенко В.В._
(підпис) (прізвище, ініціали)

«_____»_____2024 року

ЗАВДАННЯ
на кваліфікаційну роботу
ступеня ____**бакалавра**_____
(бакалавра, спеціаліста, магістра)

студенту_____Ващуку Д.О._____ **академічної групи** _____126-20-1____
(прізвище та ініціали) (шифр)
спеціальності _____126 «Інформаційні системи та технології»_____
за освітньою-професійною програмою _____
(за наявності)
_____«Інформаційні системи та технології»_____
на тему ___Програмний інструмент виявлення потенційно небезпечних URL-адрес за_____
допомогою бібліотеки scikit-learn_____

затверджену наказом ректора НТУ «Дніпровська політехніка» від 23.05.2024 469-с

Розділ	**Зміст**	**Термін виконання**
Розділ 1	Аналіз стану області рішення задач	05.02.2024 – 11.03.2024
Розділ 2	Розробка методу виявлення потенційно небезпечних url-адрес за допомогою бібліотеки scikit-learn	12.03.2024 – 15.04.2024
Розділ 3	Реалізація інструменту виявлення потенційно небезпечних url-адрес за допомогою бібліотеки scikit-learn	16.04.2024 – 31.05.2024

Завдання видано_____ _____В.Ю. Каштан_____
(підпис керівника) (прізвище, ініціали)

Дата видачі __05.02.2024 р._____

Дата подання до екзаменаційної комісії____17.06.2024 р.___

Прийнято до виконання_____ _____Ващук Д.О._____
(підпис студента) (прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 71 стор., 18 рис., 1 табл., 2 додатки, 13 джерел.

Об'єктом кваліфікаційної роботи є програмний інструмент виявлення потенційно небезпечних URL-адрес.

Предметом кваліфікаційної роботи є алгоритм випадковий ліс, який використовуються для аналізу та класифікації URL-адрес на основі отриманої інформації з них, без необхідності завантаження вмісту веб-сторінок.

Мета кваліфікаційної роботи полягає в розробці програмного інструменту для виявлення потенційно шкідливих URL-адрес з використанням керованого навчання. Для цього використовується алгоритм випадковий ліс, який навчається за допомогою бібліотеки scikit-learn.

Для реалізації мети потрібно вирішити **задачі:**

1. Розробити алгоритм для виділення лексичних ознак з URL-адрес.
2. Побудувати модель для класифікації URL-адрес за лексичними ознаками, використовуючи методи машинного навчання.
3. Визначити та врахувати особливості популярності сайтів для покращення класифікації.
4. Розробити функції на основі хостів для додаткової класифікації фішингових URL-адрес.
5. Використати навчальний набір даних з позначеними URL-адресами для навчання моделі.
6. Розробити графічний інтерфейс для керування програмним інсттрументом.

Ключові слова: scikit-learn, небезпечна адреса, випадковий ліс, класифікація URL-адрес.

ABSTRACT

Explanatory note: 71 pages, 18 figures, 1 table, 2 appendices,13 sources.

The object of the qualification work is a software tool for detecting potentially dangerous URLs.

The subject of the qualification work is a random forest algorithm used to analyze and classify URLs based on the information without the need to download the content of web pages.

The aim of the qualification work is to develop a software tool for detecting potentially malicious URLs using supervised learning. For this purpose, a random forest algorithm is used and trained using the sci-kit-learn library.

To realize the goal, the following tasks need to be solved:

1. Develop an algorithm for extracting lexical features from URLs.

2. Build a model for classifying URLs by lexical features using machine learning methods.

3. Identify and implement the features of website popularity to improve classification.

4. Develop host-based features for additional classification of phishing URLs.

5. Use a training dataset with labeled URLs to train the model.

6. Develop a graphical interface to control the software tool.

Keywords: sci-kit-learn, dangerous address, random forest, URL classification.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД – база даних;

ORM – об'єктно-реляційним відображенням (;

IDEF – Information Modeling – методологія моделювання інформаційних потоків;

UML – Universal Modeling language – це універсальна мова моделювання.

.

ВСТУП

Онлайн-послуги та веб-сайти стали необхідною складовою сучасного життя в різних сферах, включаючи бізнес, освіту, банківську справу та особисте життя. Зі зростанням популярності і використання онлайн-ресурсів зростає і кількість шкідливих веб-сайтів. Шкідливий веб-сайт може містити небажаний вміст з метою збору конфіденційних даних або встановлення шкідливого програмного забезпечення на комп'ютер користувача. Часто це відбувається без уведення користувача, зокрема під час драйвового завантаження, коли шкідливе програмне забезпечення автоматично встановлюється без його дозволу.

Захист від таких атак складний, оскільки іноді навіть обережне користування Інтернетом не є достатнім. Зловмисники можуть використовувати вразливості у веб-додатках для впровадження шкідливого коду без відома власника. Згідно з дослідженням Webroot Threat Report за 2019 рік, 40% шкідливих URL-адрес було виявлено на добропорядних доменах [1]. Це означає, що навіть легітимні веб-сайти можуть стати потенційною загрозою для користувачів.

У зв'язку з цим виникає необхідність розробки методів та інструментів, які допоможуть відрізнити шкідливі URL-адреси від безпечних. Одним зі широко використовуваних методів захисту є створення чорних списків, але цей підхід має свої обмеження. Іншим методом є застосування методів машинного навчання для виявлення шаблонів у шкідливих URL-адресах. У цій роботі ми зосередимося саме на цьому підході.

Мета цієї роботи полягає в розробці програмного інструменту для виявлення потенційно шкідливих URL-адрес з використанням керованого навчання. Для цього використовується алгоритм випадковий ліс, який навчається за допомогою бібліотеки scikit-learn.

1 АНАЛІЗ СТАНУ ОБЛАСТІ ВИЯВЛЕННЯ ШКІДЛИВИХ URL-АДРЕС

1.1 Характеристика небезпечних URL-адрес

Небезпечні URL-адреси, або URL (абревіатура від Uniform Resource Locator), є глобальними адресами документів та інших ресурсів у всесвітній павутині. У сутності, це рядок символів, що вказує на існування певного Інтернет-ресурсу. Зазвичай URL складається з трьох-п'яти компонентів (рис.1.1) [2]:

– схема, що визначає протокол, який використовується хостом;

– хост, що вказує IP-адресу або шлях до зареєстрованого доменного імені;

– шлях, що ідентифікує конкретний ресурс на хості; рядок запиту номера порту;

– параметри та їх значення, які можуть супроводжувати URL;

– фрагмент, що ідентифікує вторинний ресурс, зазвичай частина сторінки.

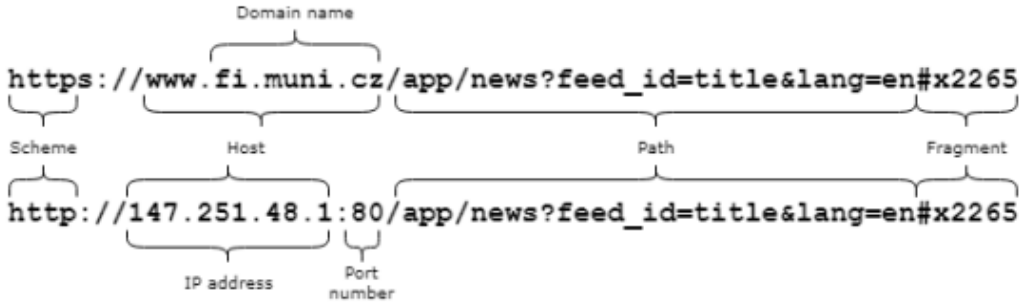

Рисунок 1.1 – Приклад URL-адреси

Термін "небезпечні URL-адреси" використовується для посилань на веб-сайти, що містять різноманітний небажаний вміст. Ці адреси становлять загрозу для користувачів, які нічого не підозрюючи відвідують їх, оскільки містять шкідливий вміст, що може бути використаний для різних видів

кібератак. Вони можуть бути випадково згенерованими або скомпрометованими легальними веб-сайтами.

Розрізняють різні типи шкідливих URL-адрес залежно від небажаного вмісту, який вони містять, і типу кібератак, на які вони спрямовані. Найпоширенішими є спам, фішинг та автоматичне завантаження. Кожен тип може мати свої відмінні риси, і для їх виявлення може бути успішною інша методологія.

1.1.1 Спам

Веб-спам - це веб-сайти, які намагаються обдурити пошукові системи, щоб піднятися в рейтингу вище і збільшити таким чином свій трафік. URL-адреси, що ведуть на такі веб-сайти, можна назвати спам-адресами. Незважаючи на вищий рейтинг, користувачі не знайдуть там легального контенту, який вони шукали. Загалом, існує два найпоширеніші види веб-спаму: контент-спам і посилальний спам.

Контентний спам включає всі методи, які змінюють вміст сторінки, щоб отримати вищий рейтинг. Це включає в себе додавання популярних слів до фактичного вмісту, зміну частин, які мають більше значення для ранжування, таких як заголовок сторінки або якірний текст та ін.

Посилальний спам використовує переваги алгоритмів ранжування, заснованих на посиланнях. Ці алгоритми дають веб-сайтам вищий рейтинг, чим більше на них посилаються інші високорейтингові веб-сайти [3].

1.1.2 Фішинг

Фішингові веб-сайти намагаються викрасти у користувачів конфіденційну інформацію, таку як номери карток, банківських рахунків або паролі, обманом змушуючи їх повірити, що вони знаходяться на легальному

веб-сайті. URL-адреси та вміст фішингових сайтів схожі на оригінальні, тому жертві важко розпізнати різницю. Отримана конфіденційна інформація здебільшого використовується для крадіжки особистих даних або грошей жертви [3,4].

1.1.3 Випадкове завантаження

Випадкове завантаження - це ненавмисне завантаження шкідливого коду (шкідливого програмного забезпечення) на пристрій користувача після відвідування скомпрометованих веб-сайтів. Жертві не потрібно нічого натискати, щоб ініціювати завантаження. Простий доступ до зараженого веб-сайту може завантажити та встановити шкідливе програмне забезпечення непомітно у фоновому режимі без відома користувача. Встановлене шкідливе програмне забезпечення може бути використане для отримання контролю над скомпрометованим комп'ютером, викрадення паролів або іншої конфіденційної інформації, вимагання викупу або інших небажаних дій.

1.2 Особливості URL-адрес

Використання всього рядка URL-адреси як єдиної інформації для виявлення URL-адрес може виявитися недостатнім. Тому обов'язковим першим кроком є вибір і вилучення корисних ознак, які достатньою мірою описують URL-адресу. Робота з ознаками, а не з повним рядком URL, дозволить створити більш ефективні та успішні алгоритми виявлення. Для кожного типу шкідливих URL-адрес різний набір ознак може дати кращі результати. У роботах, присвячених виявленню шкідливих URL-адрес, запропоновано кілька типів ознак, які можна використовувати. Ми можемо класифікувати ці ознаки на чорні списки, лексичні, на основі хостів, на

основі вмісту та на основі рангу [3]. Огляд властивостей цих ознак можна знайти в Таблиці 1.1.

Таблиця 1.1 – Огляд функцій, що використовуються для виявлення шкідливих URL-адрес

Тип	Категорія	Властивості				
		Зовнішня залежність	Час збору	Час обробки	Розмір	Ризик
Чорний список	Чорний список.	так	середній	низький	низький	низький
Лексичні	Традиційний	немає	низький	низький	високий	низький
	Розширений	немає	низький	середній	низький	низький
Хост	WHOIS	так	високий	низький	високий	низький
	Доменне ім'я	так	високий	низький	низький	низький
	Географічний	так	високий	низький	середній	низький
	Швидкість з'єднання	немає	висока	низька	низька	низька
Зміст	HTML	немає	низький	високий	низький	високий
	Javascript	немає	низький	середній	низький	високий
	Інше	немає	високий	високий	низький	високий
Rank (Рейтинг)	Рейтинг	так	середній	низький	низький	низький

1.2.1 Чорний список

Чорний список - це, по суті, список URL-адрес, які раніше були визначені як шкідливі. Він постійно оновлюється з часом за допомогою поєднання автоматичних механізмів і людей. Використання чорних списків для виявлення шкідливих URL-адрес вважається дуже швидким і простим у застосуванні методом з дуже низьким показником хибно позитивних результатів. На жаль, цей метод не здатний виявляти новостворені URL-адреси, а отже, страждає від високого рівня хибно негативних результатів. Однак, замість того, щоб використовувати чорний список як самостійний інструмент ідентифікації, він може бути зручною функцією для більш просунутих методів. Інструменти виявлення можуть використовувати власні

створені чорні списки або, зважаючи на великі обсяги даних, один з багатьох публічних API чорних списків, таких як Google Safe Browsing або PhishTank.

1.2.2 Лексичні ознаки

Ідея лексичних ознак базується на припущенні, що шкідливі веб-сайти мають різні візуальні характеристики, тому мають відмінні шаблони тексту URL-адреси. У багатьох випадках користувач може виявити підозрілі URL-адреси лише за їх зовнішнім виглядом. Це особливо стосується фішингових URL-адрес, де URL-адреса часто виглядає так само, як і оригінальна сторінка, яку вона намагається імітувати. Наприклад, розглянемо URL-адреси: https://www.paypal.com/ та https://paypal.co.uk.b8wt.icu/n/. Друга URL-адреса з першого погляду має надто багато доменних імен і виглядає підозріло.

Під лексичними ознаками ми розуміємо всі текстові характеристики самої URL-адреси, не враховуючи вмісту сторінки, на яку вона вказує. Лексичні ознаки привабливі для використання, оскільки їх легко обробити, вони потребують невеликого обсягу пам'яті і можуть бути отримані без необхідності звертатися до інших сервісів. Зазвичай методи виявлення розділяють лексичні ознаки імені хоста та шляху окремо, оскільки вони впливають на класифікацію по-різному.

Згідно з дослідженнями Sahoo, Liu та Hoi, лексичні ознаки можна розділити на традиційні та розширені.

Традиційні лексичні ознаки включають усі загальноприйняті характеристики тексту URL-адреси. Більшість з них - це комбінація ознак, які були вперше описані McGrath та Gupta та Kolari та іншими. Ці ознаки включають довжину імені хоста, довжину всієї URL-адреси, а також частоту символів у URL-адресі. Також враховується наявність або порядок токенів в

імені хоста (розділені символом '.') і в шляху (розділені символами '.', '/', '?', '=', '-', ' ') [5].

Розширені лексичні ознаки є складнішими за традиційні та вимагають більше обчислень. Однак вони ефективніші у виявленні різних методів обфускації. Типові приклади методів обфускації включають заміну хоста на IP-адресу, включення цільового домену в шлях, використання великих імен хостів та додавання додаткових доменів, а також використання невідомого або неправильно написаного домену.

Le та інші [6] запропонували розширені ознаки, такі як наявність IP-адреси, кількість спеціальних символів або статистичні властивості токенів, такі як їхня кількість, середня або максимальна довжина. Особливістю, яку використовували Daeef та інші [7] або Verma та інші [8], є використання n-грам, де замість наявності окремих слів ми визначаємо наявність n-символьних під рядків токенів. Таким чином, ми можемо виявити під слова або неправильно написані слова. Верма і Дайер використовували евклідову відстань як одну з характеристик, щоб виявити відмінності між фішинговими URL-адресами і стандартною англійською мовою. Складність Колмогорова (або так звана ентропія Колмогорова, алгоритмічна ентропія), яку використовують автори Рао та інші, може розглядатися як більш просунута ознака. Цей метод, у разі виявлення шкідливих URL-адрес, вирішує, чи є URL-адреса безпечною або шкідливою, порівнюючи, чи має вона більше схожих шаблонів символів із заданою базою даних безпечних або шкідливих URL-адрес.

1.2.3 Ознаки за хостовою приналежністю

Елементи на основі хосту описують властивості, які ідентифікуються за частиною URL-адреси, що містить ім'я хоста[2]. Вони дозволяють приблизно

визначити місцезнаходження, власника, дату реєстрації, стиль управління та інші властивості зловмисних хостів. Ці ознаки підвищують загальну точність виявлення, але вони завжди дорогі, оскільки не виводяться з самої URL-адреси, а збираються зі сторонніх сервісів. Типовими характеристиками хостів, що використовуються, є Властивості WHOIS - дата реєстрації/оновлення/закінчення терміну дії, реєстратори та реєстранти доменного імені. За даними [9], шкідлива URL-адреса часто має дату реєстрації або оновлення в недалекому минулому. Крім того, якщо група шкідливих URL-адрес зареєстрована однією особою, це слід розглядати як зловмисну особливість. Властивості доменного імені - час життя (TTL), наявність запису покажчика (PTR), або якщо PTR-запис дозволяє розпізнати одну з IP-адрес хоста. Крім того, сюди можна віднести властивості, які можна розглядати як лексичні, наприклад, наявність слів "сервер" і "клієнт" або IP-адреси в імені хоста. Географічні властивості - фізична географічна інформація - континент/країна/місто, до якого належить IP-адреса. Швидкість з'єднання - швидкість висхідного з'єднання. Поважні веб-сайти, як правило, мають вищу швидкість з'єднання.

1.2.4 Ознаки на основі вмісту

Функції на основі вмісту - це ті, які отримуються після відкриття та завантаження веб-сайту. Вони вимагають великих обсягів даних, але надають найточнішу інформацію про вміст веб-сайту. Однак для фішингових веб-сайтів вони можуть бути неефективними, оскільки фішинговий веб-сайт повинен мати подібний вміст до оригінального веб-сайту, за який він себе видає. Ми можемо розділити функції на основі вмісту на три під категорії: HTML, JavaScript та інші функції на основі вмісту. HTML-функції - здебільшого містять статистичні характеристики HTML-документа, такі як довжина сторінки, середня довжина слів, кількість слів, кількість окремих

слів або відсоток пробілів. Крім того, також використовуються статистичні характеристики певних елементів HTML, такі як підрахунок кількості тегів HTML, рядків, гіперпосилань або iframe. Характеристики JavaScript - статистичні властивості коду JavaScript, що використовується на веб-сайтах. Чой та інші підрахували кількість семи підозрілих власних функцій JavaScript: escape(), eval(), link(), unescape(), exec(), link() та search(). Інші функції на основі вмісту - зазвичай більш специфічні та складні функції, такі як візуальні функції, які зосереджені на пошуку схожості з іншими веб-сайтами.

1.2.5 Ознаки засновані на ранжируванні

Рангові ознаки отримують від сервісів, які певним чином упорядковують або ранжирують домени або окремі сторінки. Це робиться на основі припущення, що шкідливі веб-сайти повинні мати нижчий рейтинг, ніж безпечні.

Для цього зазвичай використовується Alexa rank4, який обчислює популярність веб-сайту за допомогою комбінації оціночного трафіку та активності відвідувачів за останні три місяці. На жаль, ця система ранжування базується лише на домені, що може призвести до того, що скорочувачі посилань або веб-хостинги отримають високі бали. Інший варіант - використання даних про популярність сайту в пошукових системах. Ризик цієї функції полягає в тому, що вмістом веб-сайту можуть маніпулювати, щоб отримати вищий рейтинг у пошукових системах. Вирішенням цієї проблеми може бути використання комбінації декількох пошукових систем, які використовують різні алгоритми ранжування. Наприклад, Choi та ін. використовували для виявлення популярності посилань дані з п'яти різних пошукових систем: Altavista, AllTheWeb, Google, Yahoo! і Ask.

1.3 Машинне навчання для виявлення фішингових атак

Штучний інтелект - це нова інноваційна наука, яка розглядає і створює гіпотези, стратегії, процедури і додатки, що відтворюють, розвивають і розширюють людські знання. Машинне навчання є частиною штучного інтелекту і часто перетинається з обчислювальними вимірюваннями [10], які також зосереджені на прогнозуванні з використанням ПК. Машинне навчання має тісний зв'язок з науковим вдосконаленням, яке підказує методи, гіпотези та області застосування. Іноді ML поєднується з інтелектуальним аналізом даних, але підгалузь інтелектуального аналізу даних зосереджується більше на дослідженні підготовчої інформації і називається неконтрольованим навчанням (unsupervised learning). ML також може бути неконтрольованим і використовуватися для навчання та створення профілів шаблонів для різних об'єктів, а потім використовуватися для пошуку важливих аномалій.

Кібербезпека - це набір інновацій та процедур, призначених для захисту комп'ютерів, мереж, проектів та інформації від атак та несанкціонованого доступу, модифікації або знищення [11]. Структура безпеки системи складається зі структури забезпечення безпеки системи та структури захисту комп'ютерів. Кожна з цих систем включає брандмауери, антивірусні програми та системи виявлення вторгнень (IDS). IDS допомагають знаходити, визначати та розрізняти несанкціоновану поведінку системи , наприклад, використання, реплікацію, зміну та знищення.

Існують три важливих типи мережевого аналізу для систем виявлення вторгнень: на основі зловживань, також відомий як аномалії, на основі сигнатур та гібридний. Стратегії виявлення на основі зловживань полягають у тому, щоб відрізнити реалізовані атаки, використовуючи ознаки цих атак. Методи на основі аномалій вивчають типову систему та її поведінку і розрізняють аномалії як відхилення від звичайної поведінки. Гібридне

виявлення поєднує в собі виявлення аномалій та зловживань, щоб збільшити швидкість виявлення прийнятних вторгнень і зменшити швидкість помилкових спрацьовувань невідомих атак.

Застосування методів машинного навчання (ML) в кібербезпеці зростає як ніколи раніше, як показано на рис. 1.2. Починаючи з категоризації IP-трафіку, відокремлення шкідливого трафіку для виявлення вторгнень, машинне навчання є однією з найкращих відповідей, які можуть вплинути на атаки "нульового дня". Нові дослідження проводяться шляхом використання вимірюваних характеристик трафіку та методів ML [12].

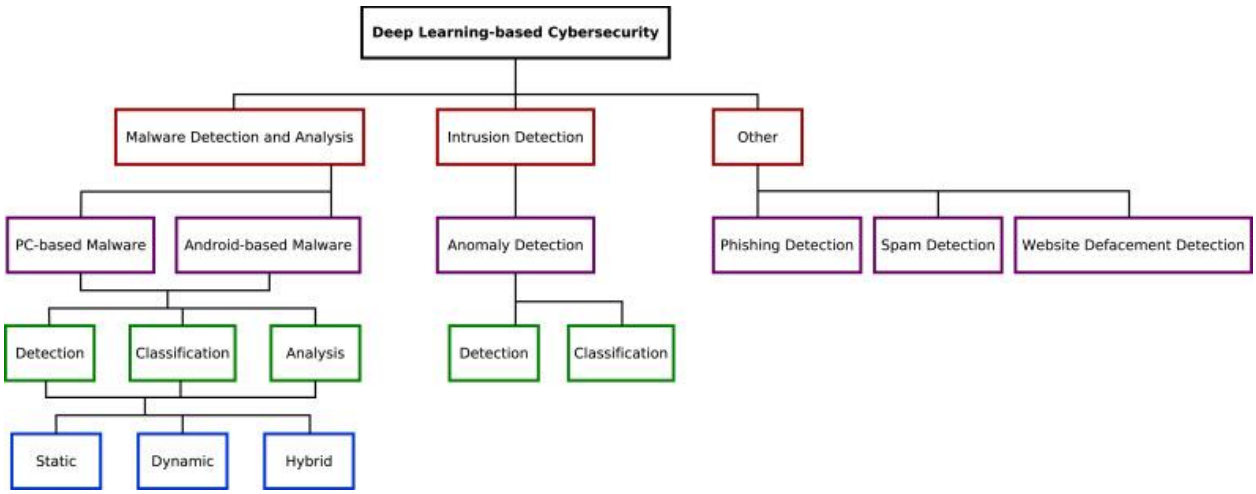

Рисунок 1.2 – Застосування машинного навчання в кібербезпеці [12]

Слово "фішинг" було введено в 1987 році [9]. Фішинг - це онлайн-злодійство, яке викрадає приватні дані та ідентифікаційні дані людини. Це різновид вимагання, коли зловмисник отримує повний доступ до приватних даних іншої особи [10]. Веб-сайт-шахрай, схожий на справжній, легко створюється вправним дизайнером, і тому розпізнавання веб-сайту як шахрайського може бути виснажливим. Ось чому ми потрапляємо в такі

пастки. Ці фішингові сайти закликають користувачів надати дані свого облікового запису, стверджуючи, що вони є справжніми, наприклад, за допомогою HTTPS. Це переконує користувача покластися на цей фейковий сайт. Вони запевняють у безпеці та конфіденційності, хоча й отримують ідентифікаційні дані користувача. Люди здійснюють більшість грошових обмінів в Інтернеті. Оплата рахунків або переказ грошей, майже все відбувається через сайти або додатки. Тому виявлення таких фальшивих сайтів має велике значення. За даними Anti-Phishing Working Group, загальна кількість характерних фішингових сайтів, зафіксованих до вересня 2018 року, становила 647 592. Як тільки зловмисник отримує доступ до паролів, будь-яка шкідлива мета спрощується.

У зв'язку зі збільшенням кількості фішингових атак, запропоновано багато результатів, які генерують вирішення цієї проблеми. Для побудови фреймворку, який гарантує захист від фішингових атак, існує декілька способів. Існують різні інші методи виявлення фішингових атак, такі як чорний список, нечіткі правила, білий список, кантина, машинне навчання, евристичні підходи та підходи на основі зображень. Існує ще кілька досліджень, в яких йдеться про різноманітні методи і техніки для виявлення різних типів фішингових атак. Фішингові сайти виглядають як справжні, і деякі люди мають проблеми з розпізнаванням таких сайтів. У деяких браузерах вбудовано лише кілька антифішингових технологій.

1.4 Обмеження сучасних рішень на основі машинного навчання

Атаки соціальної інженерії взагалі, а фішингові атаки зокрема, не є успішними через вразливість систем, а через те, що люди не можуть відрізнити легітимних суб'єктів від фейкових. Відповідно, в літературі вивчається широкий спектр методів протидії таким атакам різного рівня

складності. Машинне навчання спрямоване на автоматизацію процесів навчання на основі існуючих прикладів та досвіду без явного програмування [13]. Алгоритми машинного навчання показали багатообіцяючі результати . Ця методика вимагає наявності попередніх реальних даних, які були класифіковані або марковані для проведення навчання. Однак, використання методів машинного навчання для виявлення фішингових веб-сайтів у наявних підходах призводить до наступних обмежень.

Контрольоване глибоке навчання здається перспективним підходом для виявлення фішингу [13]. Машинне навчання потребує великого обсягу навчальних даних, а їх збір порушує конфіденційність кінцевих користувачів. Для отримання характеристик підозрілих веб-сайтів доводиться використовувати сторонні сервіси, такі як пошукові системи, що може розкривати історію переглядів користувачів та порушувати їхню конфіденційність.

Наступним обмеженням є відсутність реальних даних про атаки або непридатність даних. У системах кібербезпеки загрози є рідкісними подіями, тому набори даних зміщені в бік нормальних подій. Такі набори містять значно більше нормальних випадків, ніж нетипових, що ускладнює навчання на основі такого набору даних. Інтерес викликають набори даних меншості, де рідкісні випадки належать до фішингових атак. Крім того, дослідники майже не діляться своїми наборами даних з проблем кібербезпеки з міркувань конфіденційності та приватності. Лише 10% дослідників поділилися своїми наборами даних у подібних проблемах безпеки мереж, що унеможливлює створення набору даних, що ґрунтується на реальності. Зважаючи на малий обсяг існуючих наборів даних про фішинг, класифікатор, що навчається, може не збігатися, а його ефективність буде непослідовною. Коротше кажучи, навчальна модель може бути недосконалою через відсутність адекватних даних.

Перепредставлені цілі є ще одним обмеженням. Деякі зразки в наборах даних перепредставлені, а інші - недостатньо представлені. Ці упереджені набори даних можуть призвести до того, що модель навчання буде расистською, сексистською або несправедливою по відношенню до груп меншості. Наприклад, медичний алгоритм машинного навчання, який був навчений розпізнавати рак шкіри за фотографіями, не був протестований на темношкірих людях через недостатню кількість зразків цієї групи, приблизно 5% зображень темношкірих людей були включені в набір даних. Наше спостереження полягає в тому, що якщо зразки фішингу у навчальних наборах даних зміщені в бік найбільш популярних веб-сайтів, то рівень виявлення фішингових атак для цих сайтів буде вищим, ніж для менш відомих. Хоча загальні результати алгоритму виявлення фішингу можуть бути високими, результати виявлення значно відрізняються для різних цільових веб-сайтів в залежності від їхньої популярності. У такому випадку алгоритм буде спрямований на популярні сайти, тобто він ефективніше виявлятиме атаки на найбільш популярних сайтах порівняно з менш відомими.

Фішингові атаки продемонстрували неабияку стійкість до безлічі захисних заходів, і зловмисники продовжують створювати витончені фішингові веб-сайти, які точно імітують легальні веб-сайти. Одне з найважливіших припущень у використанні підходів машинного навчання полягає в тому, що процес збору навчальних даних не залежить від дій зловмисників. Однак у контексті протидії, наприклад, фішингу, це далеко від реальності, оскільки зловмисники або генерують зашумлені вибірки даних, або створюють нові зразки атак, маніпулюючи особливостями існуючих фішингових інстанцій. Більше того, маніпулювання ознаками може призвести до небезпечного сценарію, коли зловмисник може обійти згенерований класифікатор без особливих зусиль. Ретельно створена вибірка фішингових даних, яка видається класифікатору машинного навчання як

легітимна, називається зразком супротивника. Безпосередній вплив ворожих зразків полягає в тому, що вони знижують точність класифікатора машинного навчання. Ключовою проблемою для зловмисника буде вибір ознак, якими потрібно маніпулювати, і пов'язана з цим вартість такої маніпуляції. В ідеалі зловмисник хотів би обійти класифікатор з найменшими витратами на маніпуляції ознаками вибірки даних.

1.5 Постановка задачі на розробку

Існує багато методів захисту від фішингу, які допомагають нам захиститися від фішингових сайтів. Браузери, такі як Mozilla Firefox, Safari та Google Chrome, використовують сервіс Google Safe Browsing (GSB), що блокує фішингові сайти. Крім того, існують інші подібні інструменти, такі як McFee Site Advisor, Quick Heal, Avast і Netcraft, які широко використовуються. GSB аналізує URL-адресу за допомогою підходу чорного списку. Однак, основним недоліком GSB є те, що він не завжди виявляє фішингові сайти через відсутність оновлення чорного списку. У разі з Netcraft, веб-сайт, який вчиняє фішинг, може бути зафіксований як фішинговий, але не буде автоматично заблокований. Netcraft блокує сайти лише в тому випадку, якщо на 100% впевнений, що вони є фішинговими. Попередження видається тільки тоді, коли користувач натискає правою кнопкою миші на іконці для перевірки рейтингу ризику. Ризик виникає, коли користувач ігнорує перевірку рейтингу або приймає рішення використовувати сайт після інформації про ризик. Деякі програмні продукти, такі як QuickHeal та Avast, забезпечують захист від атак в Інтернеті. Проте, функціонування антивірусу Avast було перевірено після його встановлення, і він не завжди успішно виявляв фішингові URL-адреси, які успішно визначали Netcraft та GSB. Це підтверджує необхідність використання просунутих антифішингових інструментів. Важливо зауважити, що ці

інструменти потрібно встановлювати самостійно, тому обізнаність про них та про фішинг є дуже важливою. Крім того, необхідно розуміти, що навіть з використанням таких інструментів, необхідно самостійно перевіряти достовірність сайтів, оскільки іноді вони можуть призвести до неправильної класифікації.

Ця проблема виникла після ретельного спостереження та вивчення методів класифікації фішингових веб-сайтів з використанням методів машинного навчання. Тому, мета роботи полягає в розробці програмного інструменту для виявлення потенційно шкідливих URL-адрес з використанням керованого навчання. Для цього використовується алгоритм випадковий ліс, який навчається за допомогою бібліотеки scikit-learn.

Завдання кваліфікаційної роботи полягають у наступному:

1. Розробити алгоритм для виділення лексичних ознак з URL-адрес.
2. Побудувати модель для класифікації URL-адрес за лексичними ознаками, використовуючи методи машинного навчання.
3. Визначити та врахувати особливості популярності сайтів для покращення класифікації.
4. Розробити функції на основі хостів для додаткової класифікації фішингових URL-адрес.
5. Використати навчальний набір даних з позначеними URL-адресами для навчання моделі.
6. Розробити графічний інтерфейс для керування програмним інструментом.

1.6 Висновки

Аналіз лексичних ознак URL-адрес виявив відмінності між фішинговими та легітимними сайтами. Довжина URL, кількість рівнів та токенів у домені та шляху є важливими характеристиками для класифікації. Виявлено, що фішингові та шкідливі сайти можуть використовувати довші URL та більшу кількість токенів, включаючи назви популярних брендів, для маскування своєї справжньої природи.

Популярність сайту може служити індикатором його безпеки, зазвичай шкідливі сайти менш популярні. Оцінка популярності сайту може бути корисною для розробки алгоритмів виявлення фішингових та шкідливих сайтів, допомагаючи виділити підозрілі домени для подальшого аналізу.

Шкідливі сайти частіше реєструються в менш авторитетних хостинг-центрах або регіонах. Це може бути важливим індикатором для класифікації сайтів, допомагаючи ідентифікувати потенційно небезпечні домени. Аналіз функцій на основі хостингу може допомогти покращити точність та ефективність системи виявлення шкідливих URL-адрес.

2 РОЗРОБКА МЕТОДУ ВИЯВЛЕННЯ ПОТЕНЦІЙНО НЕБЕЗПЕЧНИХ URL-АДРЕС ЗА ДОПОМОГОЮ БІБЛІОТЕКИ SCIKIT-LEARN

2.1 Огляд методів на основі машинного навчання

Підходи на основі машинного навчання використовують список URL-адрес, визначених як набір ознак, і на їх основі тренують модель прогнозування для класифікації URL-адреси як безпечної або шкідливої. Це дає їм можливість виявляти нові потенційно небезпечні URL-адреси. Цей підхід складається з двох етапів: перший - вибір і відповідне представлення ознак, а другий - використання цього представлення для навчання механізму прогнозування. Машинне навчання можна класифікувати як контрольоване, неконтрольоване і напівконтрольоване, залежно від того, чи відомо алгоритму, чи є URL-адреси шкідливими/безпечними. Залежно від того, як модель приймає навчальні дані і навчається на них, можна виділити дві основні групи: Пакетне навчання (навчання на обмеженій групі даних) і навчання в режимі реального часу (приймання і подальше навчання на даних, що надходять потоками).

Підготовка даних - це етап, який використовується для перетворення необроблених даних у чистий набір даних. Першим кроком є вибір та вилучення корисних ознак, які достатньою мірою представляють URL-адреси у вигляді векторів ознак. Відбір ознак зазвичай здійснюється за допомогою евристичних методів з використанням попередніх знань про шкідливі URL-адреси. На наступному кроці ці вектори ознак потрібно перетворити у форму, яка сприймається алгоритмом як вхідні дані. Зазвичай, якщо вони не перетворюють дані самостійно, вони приймають лише числові вектори, оскільки навчання ґрунтується на математичних рівняннях і розрахунках. Перед подачею даних у модель необхідна певна санітарна обробка та нормалізація даних, оскільки якість і форма даних можуть безпосередньо впливати на здатність моделі до навчання.

Числове представлення тексту є ключовою частиною обробки ознак, яка передбачає перетворення нечислових (текстових) значень у числові. Нам необхідно перетворити ознаки, які містять одне текстове значення (наприклад, країна перебування, головний домен) або вектор текстових значень (наприклад, токени у шляху). У контексті URL-адрес, всі ці значення є дискретними та категоріальними даними. На початку кожного методу кодування створюється словник усіх можливих значень (слів) для певних ознак. А потім, використовуючи цей словник, ми можемо виконати:

Bag-of-Words (Мішок слів) є поширеною технікою представлення текстових даних у вигляді векторів, що використовується в алгоритмах машинного навчання. Після побудови словника вимірюється проста наявність слів. Кожна ознака тексту перетворюється на вектор з 0 та 1 в розмірі словника, де 1 означає наявність, а 0 - відсутність. Якщо значення ознаки може містити одне й те саме слово кілька разів, як у випадку токенів у шляху, можна зберегти їхню частоту(рис.2.1).

```
url_one = {'country':'Russia', 'path_tokens': {'be', 'successful'}}
url_two = {'country':'Slovakia', 'path_tokens': {'think','happy', 'be', 'happy'}}

vocabulary_country = {'Russia', 'Slovakia'}
vocabulary_path_tokens = {'be', 'successful', 'think', 'happy'}

url_one_bow = {{1,0},{1,1,0,0}}
url_two_bow = {{0,1},{1,0,1,1}}
url_two_bow_freq = {{0,1},{1,0,1,2}}
```

Рисунок 2.1 – Фрагмент реалізації методу Bag-of-Words

Цей метод називається "мішок" слів, оскільки будь-яка інформація про порядок або структуру слів відкидається, що призводить до втрати порядку слів або їх схожості. Крім того, він дуже обтяжливий за розміром, оскільки виводить дані у вигляді розрідженого вектора (більшість значень дорівнює 0).

Однократне кодування. Цей метод схожий на метод Bag-of-Words, але на відміну від нього, він зберігає порядок слів. Ми перетворюємо кожне слово у вектор 0 та 1. Це означає, що ознака, яка містить вектор слів, буде перетворена у вектор векторів. Таким чином, кожне слово буде закодовано у вектор, де тільки одне значення дорівнює 1, а решта - 0.

```
url_one_one_hot = {{1,0},{{1,0,0,0},{0,1,0,0}}}
url_two_one_hot = {{0,1},{{0,0,1,0},{0,0,0,1},{1,0,0,0},{0,0,0,1}}}
```

Рисунок 2.2 – Фрагмент реалізації методу однократне кодування

Кодування в унікальні числа, кожне унікальне слово кодується як унікальне число, зберігаючи порядок слів. Хоча порядок слів не втрачається, але, подібно до попереднього методу, відсутній зв'язок між схожістю будь-яких двох слів. На виході отримуємо щільний вектор, що робить цей метод більш ефективним.

```
url_one_unique_num = {1,{1,2}}}
url_two_unique_num = {2,{3,4,1,3}}
```

Рисунок 2.3 – Фрагмент реалізації методу кодування в унікальні числа

Вкладення слів представляє собою спосіб представлення слів, в якому схожі слова мають схоже кодування. Результатом є щільний вектор значень з плаваючою комою. Значення не обчислюються вручну, а визначаються як вагові результати моделі машинного навчання. Таке представлення дозволяє зафіксувати зв'язок між словами та підсловами (наприклад, "банк" - "банківська справа") або тісний зв'язок з контекстом (наприклад, "платити" -

"банк"). Найвідомішим методом представлення вкладених слів є Word2vec, розроблений Міколовим та іншими. Word2vec використовує комбінацію двох різних методів - Continuous Bag-of-Words (CBOW) та Skip-gram model. CBOW передбачає слово за контекстом, а Skip-grama передбачає контекст за словом. Побудова словника у цьому методі набагато складніша, оскільки враховує відстані між кожним словом.

Стохастичний градієнтний спуск є спрощеною версією алгоритму градієнтного спуску. Обидва алгоритми ітеративно оновлюють ваговий вектор wt для мінімізації рівня помилок з метою отримання моделі, яка надає найточніші прогнози. Градієнтний спуск використовує всю пакетну вибірку на кожній ітерації для оцінки оновлення. Тому цей метод є обчислювально дорогим і не підходить для великих обсягів даних з багатьма характеристиками. Великий масив даних може призвести до дуже тривалого часу обчислення однієї ітерації. Градієнтний спуск оновлює вектор ваг за допомогою такої формули:

$$w_{t+1}=w_t-\eta\nabla Q(w_t), \tag{2.1}$$

де η - швидкість навчання, а Q - деяка наперед визначена функція помилки. Стохастичний градієнтний спуск обчислює градієнт на одній випадково вибраній вибірці xt, а не на всьому наборі даних. Градієнт можна оцінити так:

$$\nabla Q(w_t)\approx\nabla Q(w_t) \tag{2.2}$$

Через випадковий вибір вибірки, стохастичний градієнтний спуск не завжди сходиться до мінімально можливої похибки, але при великій кількості вибірок та ітерацій вона буде надзвичайно близькою. Крім того, перевагою SDG є те, що він набагато швидший та ефективніший, ніж алгоритм градієнтного спуску.

2.2 Дані

Вибірки даних були надані компанією ESET, яка спеціалізується на інформаційній безпеці та надає антивірусні та захисні продукти для Інтернету. URL-адреси були зібрані протягом 20 днів, з 11 по 20 листопада 2018 року. Дані включають два набори URL-адрес: менший набір "Змінені" та більший "Вибірка". Кожен з цих наборів містить три різні підмножини: CLN, MAL, FMAL, які також розбиті за днями, коли вони були виявлені. CLN - чисті/безпечні URL-адреси; FMAL - URL-адреси, виявлені як шкідливі/небезпечні; MAL - заблоковані/шкідливі URL-адреси. Підмножина "Змінені" містить URL-адреси, які змінили свій статус з "безпечний" на "шкідливий" протягом досліджуваних днів. У підмножині "Вибірка" URL-адреси також можуть змінювати свій статус, але переважна більшість з них мала лише один статус.

Використання цього набору даних має як певні переваги, так і недоліки. Найбільшою перевагою є те, що безпечні URL-адреси візуально дуже схожі на шкідливі. Це велика проблема для багатьох досліджень, оскільки безпечні зразки можуть бути занадто ідеальними порівняно з шкідливими. Наприклад, можна згадати використання URL-адрес з генератора випадкових сторінок від Yahoo або топ-1000 сторінок за рейтингом Alexa. Однак недолік полягає в обмеженій кількості ознак, які ми можемо використовувати, оскільки ознаки з зовнішніх ресурсів можуть змінювати своє значення з часом.

2.3 Реалізація методу «випадковий ліс» для виявлення небезпечних URL-адрес за допомогою бібліотеки scikit-learn

Метод "випадковий ліс" полягає в побудові багатьох дерев рішень під час навчання і використанні їх для прогнозування. Кожне дерево випадкового лісу навчається на випадковій підвибірці навчальних даних, і приймає рішення незалежно від інших дерев. При класифікації кожне дерево

"голосує" за своїм класом, а клас з найбільшою кількістю "голосів" стає загальним прогнозом моделі, як показано на рисунку 2.4.

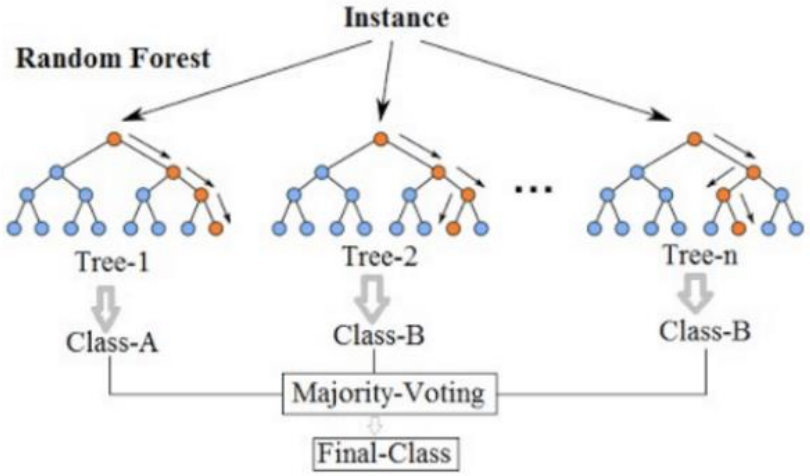

Рисунок 2.4 – Метод "випадковий ліс"

У випадковому лісі кожне дерево будується на випадковому підмножині тренувальних даних, а також на випадковій підмножині ознак. Це дозволяє збільшити різноманіття дерев у лісі, що зменшує ризик перенавчання і робить модель більш універсальною.

Випадковий ліс побудований на основі багатьох дерев рішень, що використовуються в якості "комітету" (ensemble) дерев. Прогноз моделі випадкового лісу визначається наступним чином:

1. Кожне дерево навчається на випадковому підмножині тренувальних даних, які вибираються з поверненням. Це означає, що кожен екземпляр може бути вибраний більше одного разу для навчання дерева.
2. Під час побудови кожного дерева також випадковим чином вибирається підмножина ознак.
3. Після навчання кожне дерево вирішує, до якого класу належить кожен вхідний екземпляр даних.
4. Результат ансамблю обчислюється шляхом вибору класу, який найчастіше з'являється серед прогнозів всіх дерев.

Математично, прогноз класу для ансамблю випадкового лісу може бути виражений наступним чином:

$$y\hat{}=mode(f_1(x),f_2(x),...,f_N(x)) \quad (2.3)$$

де $y\hat{}$ - прогнозований клас для вхідного екземпляра x, N - кількість дерев у випадковому лісі, а $)fi(x)$ - прогноз, зроблений i-м деревом для екземпляра x. Функція $mode$ вибирає клас, який найчастіше з'являється серед прогнозів всіх дерев.

Було створено файл trainer.py, фрагмент якого наведено на рис.2.5.

```
import pandas
import pandas as pd
from sklearn import preprocessing
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
import numpy
from sklearn import svm
from sklearn.model_selection import cross_validate as cv
import matplotlib.pylab as plt
import warnings
from sklearn.metrics import confusion_matrix

warnings.filterwarnings("ignore", category=DeprecationWarning, module="pandas", lineno=570)


def return_nonstring_col(data_cols):
    cols_to_keep = []
    train_cols = []
    for col in data_cols:
        if col != 'URL' and col != 'host' and col != 'path':
            cols_to_keep.append(col)
            if col != 'malicious' and col != 'result':
                train_cols.append(col)
    return [cols_to_keep, train_cols]


def svm_classifier(train, query, train_cols):
    clf = svm.SVC()

    train[train_cols] = preprocessing.scale(train[train_cols])
    query[train_cols] = preprocessing.scale(query[train_cols])

    print (clf.fit(train[train_cols], train['malicious']))
    scores = cv.cross_val_score(clf, train[train_cols], train['malicious'], cv=30)
    print('Estimated score SVM: %0.5f (+/- %0.5f)' % (scores.mean(), scores.std() / 2))

    query['result'] = clf.predict(query[train_cols])

    print (query[['URL', 'result']])


# Called from gui
def forest_classifier_gui(train, query, train_cols):
    rf = RandomForestClassifier(n_estimators=150)

    print (rf.fit(train[train_cols], train['malicious']))

    query['result'] = rf.predict(query[train_cols])

    print (query[['URL', 'result']].head(2))
```

Рисунок 2.5 – Фрагмент trainer.py

Спочатку імпортуємо необхідні бібліотеки, такі як pandas для роботи з даними у вигляді датафреймів, scikit-learn для навчання моделей машинного навчання, numpy для обробки числових даних та інші.

Функція return_nonstring_col визначає, які колонки в вашому наборі даних не є рядковими, і вибирає тільки числові ознаки для подальшого використання. Це важливо для того, щоб правильно підготувати дані для навчання моделі.

Навчання моделі за допомогою SVM: Функція svm_classifier навчає модель машини опорних векторів (SVM) на вхідних даних і використовує її для класифікації небезпечних URL-адрес. Важливою частиною цього процесу є стандартизація ознак за допомогою методу preprocessing.scale, що допомагає моделі працювати краще.

Функція forest_classifier використовує випадковий ліс для навчання моделі класифікації. Потрібно вказати кількість дерев у лісі за допомогою параметра n_estimators. Після навчання моделі вона оцінюється за допомогою крос-валідації та виводяться результати.

Крім того, код містить деякі інші дії, такі як імпорт даних, обробка відмови від попереджень, виведення результатів та інше.

2.4 Висновки

У даному розділі проведено огляд методів на основі машинного навчання, спрямованих на виявлення небезпечних URL-адрес. Розглянуті різні підходи до аналізу та класифікації URL-адрес, зокрема, підходи на основі векторів ознак та моделі "випадкового лісу".

Надані дані були отримані від компанії ESET, яка спеціалізується на інформаційній безпеці та надає антивірусні та захисні продукти для Інтернету. Дані містять два набори URL-адрес: менший набір "Змінені" та

більший "Вибірка", які включають різні підмножини (CLN, MAL, FMAL), розбиті за днями виявлення.

Реалізація методу "випадковий ліс" для виявлення небезпечних URL-адрес була виконана за допомогою бібліотеки scikit-learn. У використаному коді використовуються класифікатори Random Forest Classifier та SVM Classifier для навчання моделі та прогнозування класів URL-адрес. Відзначимо, що код реалізації також включає попередження та обробку винятків для забезпечення безпеки та надійності виконання.

Загальна концепція методу "випадковий ліс" полягає в ансамблюванні декількох дерев рішень, які працюють як ансамбль. Кожне дерево надає прогноз класу, і кінцеве рішення моделі визначається більшістю голосів дерев. Цей підхід дозволяє покращити точність та стійкість моделі до перевантаження та забезпечує надійні результати класифікації навіть у випадку великої кількості вхідних даних.

Отже, реалізація методу "випадковий ліс" за допомогою бібліотеки scikit-learn є ефективним інструментом для виявлення небезпечних URL-адрес, що відображається у високій точності та надійності класифікації.

3 РЕАЛІЗАЦІЯ ІНСТРУМЕНТУ ВИЯВЛЕННЯ ПОТЕНЦІЙНО НЕБЕЗПЕЧНИХ URL-АДРЕС ЗА ДОПОМОГОЮ БІБЛІОТЕКИ SCIKIT-LEARN

3.1 Підготовка даних

Правильна підготовка даних є однією з найважливіших частин процесу машинного навчання і призводить до кращих результатів. Метою цієї дипломної роботи є навчання та оцінка результатів різних моделей машинного навчання. Для цього необхідно підготувати простий у використанні інструмент для обробки необроблених вхідних даних, який можна запускати багато разів з різними вхідними даними та конфігурацією. Для цього я розробив інструмент Feature Generator - розширюваний, конфігурований додаток, написаний на мові Python для обробки вхідних необроблених даних URL-адрес. На виході ми отримуємо підготовлений набір ознак, збережений у необхідному форматі.

Весь потік даних складається з чотирьох етапів: попередня обробка, генерація ознак, зберігання даних та пост обробка. Кожен етап має конфігуровану кількість компонентів, які можна додавати або видаляти залежно від бажаного результату. Компоненти динамічно завантажуються на основі конфігурації програми. Залежно від конкретної реалізації компонента, може підтримуватися багатопотоковість для прискорення обробки. Структурна схема представлена на рисунку 3.1.

Попередня обробка - це перший етап, на якому отримується і зберігається основна інформація про вхідні дані. Сюди входить вся інформація, необхідна для подальшої обробки ознак, наприклад, словник або значення для нормалізації даних.

Генерація ознак - це основний етап обробки даних, на якому відбувається виділення ознак з вхідних даних і їх збереження для подальшого використання. Результатом є список вилучених ознак із відповідними значеннями.

Зберігання даних - це етап, на якому проводиться перетворення та зберігання ознак у відповідному форматі, необхідному для конкретного алгоритму машинного навчання.

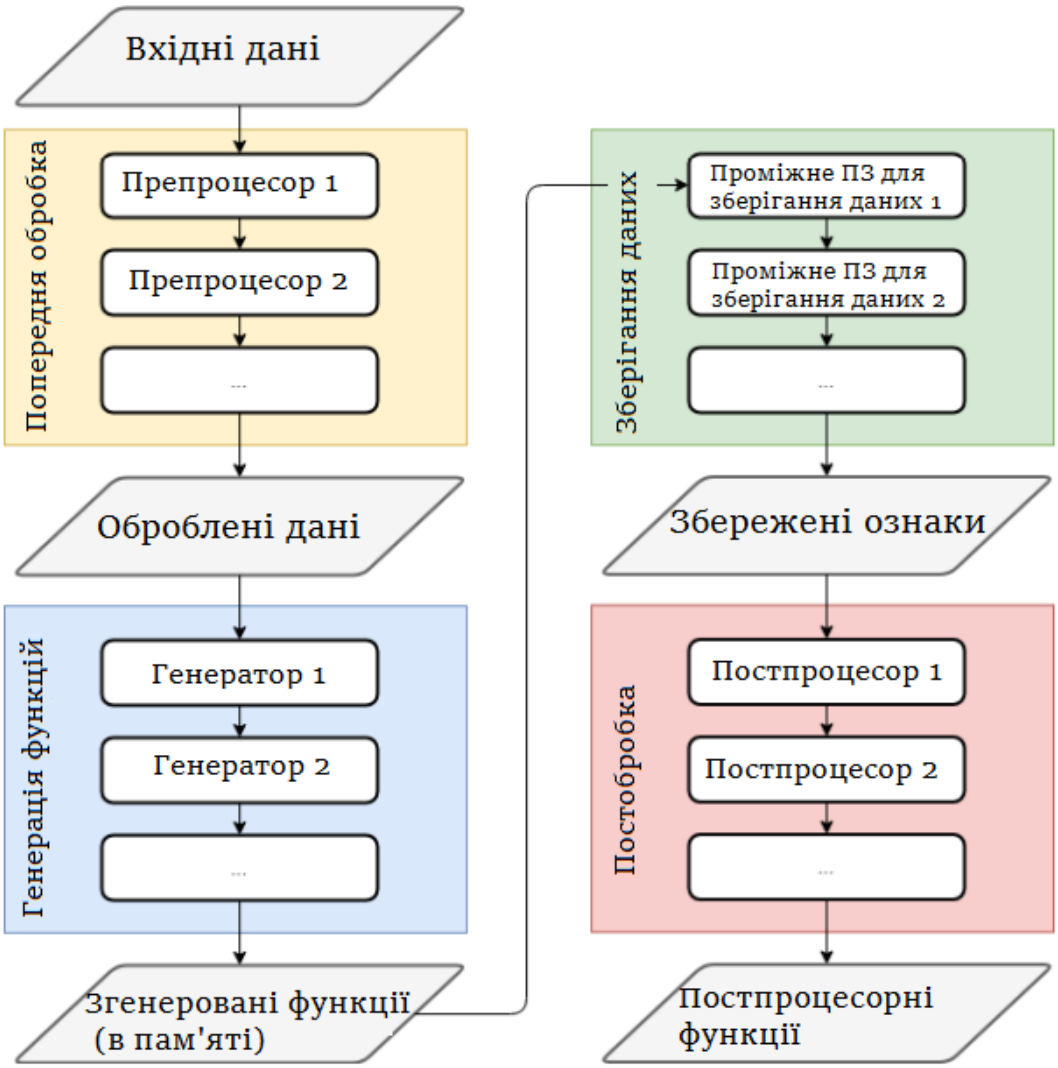

Рисунок 3.1 – Структурна схема обробки даних

Останнім етапом є постобробка даних. Основна мета постпроцесора полягає у фінальній підготовці вихідних даних для подальшого використання в оціночних експериментах з алгоритмами машинного навчання.

Конфігурація є важливою частиною програми і включає конфігураційний файл config.py, що містить інформацію про компоненти,

їхній простір імен та специфічні налаштування. Ядро програми гарантує, що кожен компонент має певні налаштування, які передаються як аргумент у конструкторі, і використовуються лише ті, які увімкнені у потоці даних.

3.2 Програмна реалізація та налаштування генератора функцій

Існує багато способів налаштування та використання генератора функцій. Було розроблено Feature_extraction.py призначений для витягування ознак з URL-адрес, що можуть бути корисні для подальшого аналізу та класифікації цих адрес за допомогою алгоритмів машинного навчання. Лістінг наведено в додатку А. Цей файл є ключовим компонентом у попередній обробці та аналізі URL-адрес для використання їх в алгоритмах машинного навчання. Він дозволяє витягувати різноманітні ознаки з URL-адрес, які потім можна використовувати для класифікації адрес за їхньою безпекою або іншими характеристиками. Розглянемо логіку виконання програмного файлу.

Функція Tokenise(url) використовується для розбиття URL-адреси на окремі токени (слова), щоб подальше обчислення різних характеристик, таких як середня довжина токенів, кількість токенів та найдовший токен.

Ind_ele_with_attribute(dom, ele, attribute) це функція для пошуку елементу з вказаним атрибутом у DOM-структурі XML-документа.

Функція sitepopularity(host) визначає популярність сайту за допомогою Alexa Rank API. Вона отримує рейтинги сайту від Alexa та повертає їх у вигляді рангу хоста та рангу країни.

Security_sensitive(tokens_words) перевіряє наявність безпечних ключових слів в URL-адресі. Вона шукає певні ключові слова, які можуть вказувати на можливість загрози безпеці.

Функція exe_in_url(url) перевіряє наявність розширення .exe в URL-адресі, оскільки файлові розширення .exe можуть вказувати на вміст, який може бути потенційно небезпечним.

Check_IPaddress(tokens_words) для перевірки наявності IP-адреси в URL-адресі. Вона перевіряє кожен токен URL на предмет того, чи є він числом, якщо так, то перевіряється, чи є кількість цих чисел більше 4, що може вказувати на IP-адресу.

Функція getASN(host) отримує номер автономної системи (ASN) для вказаного хоста. ASN ідентифікує сукупність IP-адрес, які управляються однією або кількома організаціями, тому ця інформація може бути корисною для визначення походження та власника хоста.

Функція safebrowsing(url) використовує Google Safe Browsing API для перевірки безпеки URL-адреси. Вона перевіряє URL на предмет наявності в базі даних Google Safe Browsing, яка містить список потенційно небезпечних або шкідливих URL-адрес.

Основна функція feature_extract(url_input) для аналізу ознак з URL-адреси. Вона викликає раніше описані функції для обчислення різних ознак та повертає словник, що містить значення цих ознак для заданої URL-адреси.

Препроцесори словників збирають слова з усіх URL-адрес у оброблюваних файлах. Результатом є словник унікальних слів із певною кількістю входжень у всі файли. Залежно від конфігурації, ми можемо виключати рідкісні слова. Кожному слову присвоюється унікальний числовий ідентифікатор, який використовується як числовий ідентифікатор слова. Існують дві версії цього компонента: Dictionary Preprocessor та Dictionary Split Preprocessor.

Препроцесор словника створює лише один словник, що містить слова з усіх URL. Цей препроцесор використовується в основному для алгоритму TensorFlow, який використовує кодування слів для керування порядком слів у URL. Препроцесор Dictionary Split Preprocessor створює два словники: один із хост-частини URL, а інший - з шляху. Dictionary Split Preprocessor

використовується для алгоритму SVMlight, який приймає текст, закодований у модифікованій версії one-hot-кодування. Тому, якщо ми хочемо розрізнити позиції слів у хості або шляху, нам потрібно створити два окремі словники і вектори з одним гарячим кодуванням.

Обидва препроцесори підтримують багатопоточність, де кожен потік обробляє рівно один файл URL. За замовчуванням словники зберігаються у двійковому серіалізованому форматі у файлі GeoIPASNum.dat.

Файл mai.py містить набір функцій для обробки списку URL-адрес та генерації їхніх функціональних ознак. Фрагмент навдено на рис.3.2 та додатку А.

```
import csv
import Feature_extraction as urlfeature
import trainer as tr


def resultwriter(rowdict, output_dest):
    with open(output_dest, 'w', newline='') as f:
        fieldnames = rowdict[0].keys()  # Вибираємо ключі першого словника у списку
        writer = csv.DictWriter(f, fieldnames=fieldnames)

        writer.writeheader()
        for row in rowdict:
            # Конвертуємо значення в байтовий об'єкт, якщо це необхідно
            for key, value in row.items():
                if isinstance(value, str):
                    row[key] = value.encode()
            writer.writerow(row)


def process_URL_list(file_dest, output_dest):
    feature = []
    with open(file_dest) as file:
        for line in file:
            url = line.split(',')[0].strip()
            malicious_bool = line.split(',')[1].strip()
            if url != '':
                print
                'working on: ' + url
                ret_dict = urlfeature.feature_extract(url)
                ret_dict['malicious'] = malicious_bool
                feature.append([url, ret_dict]);
    resultwriter(feature, output_dest)


def process_test_list(file_dest, output_dest):
    feature = []
    with open(file_dest) as file:
        for line in file:
            url = line.strip()
            if url != '':
                print
                'working on: ' + url
                ret_dict = urlfeature.feature_extract(url)
                feature.append([url, ret_dict]);
    resultwriter(feature, output_dest)


# change
def process_test_url(url, output_dest):
    feature = []
    url = url.strip()
    if url != '':
        print('working on: ' + url)  # showoff
        ret_dict = urlfeature.feature_extract(url)
        feature.append({'URL': url, 'Features': ret_dict})
```

Рисунок 3.2 – Фрагмент файлу для обробки списку URL-адрес та генерації їхніх функціональних ознак

Функція resultwriter(rowdict, output_dest) призначена для запису результатів обробки URL-адрес у вихідний CSV-файл. Вона приймає словник rowdict, який містить результати обробки URL-адрес, і шлях output_dest, куди потрібно зберегти CSV-файл.

Process_URL_list(file_dest, output_dest) обробляє список URL-адрес з файлу file_dest. Вона викликає функцію urlfeature.feature_extract(url), яка витягує функціональні ознаки з кожного URL-адреси. Результати зберігаються у вихідному CSV-файлі output_dest.

Process_test_list(file_dest, output_dest) аналогічно до попередньої функції, ця функція обробляє список тестових URL-адрес з файлу file_dest. Вона також викликає urlfeature.feature_extract(url) для отримання функціональних ознак кожного URL-адреси, які зберігаються у вихідному CSV-файлі output_dest.

Функція process_test_url(url, output_dest) призначена для обробки одного тестового URL-адресу url. Вона викликає urlfeature.feature_extract(url) для отримання функціональних ознак цього URL-адреси, які потім зберігаються у вихідному CSV-файлі output_dest.

Головна функція main(), яка виконує основні операції програми. Вона викликає process_test_list() для обробки тестового списку URL-адрес, а потім викликає функцію тренування tr.train() двічі: раз з тренувальними функціональними ознаками, що зберігаються у файлі url_features.csv. на рис.3.3 наведо загальну структуру, розробленого програмного інструменту.

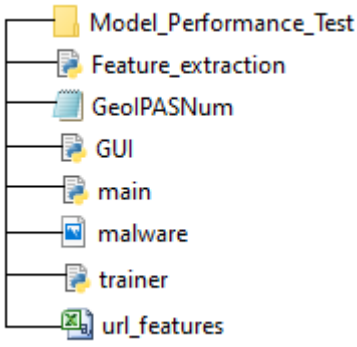

Рисунок 3.3 – Структура файлів

3.3 Розробка графічного інтерфейсу

Графічний інтерфейс реалізований за допомогою бібліотеки Tkinter у файлі main.py. Основна функціональність і логіка роботи інтерфейсу полягає в тому, щоб дозволити користувачеві вводити URL-адресу, а потім аналізувати цю адресу з використанням попередньо розроблених функцій.

Основні компоненти графічного інтерфейсу (рис.3.4):

1. Використовується поле введення (Entry), куди користувач може вводити URL-адресу для аналізу.
2. Використовується текстове поле (Text), де виводяться результати аналізу URL-адреси.
3. Кнопка "Опрацювати", яка викликає функцію submitCallBack, що обробляє введений URL-адресу і виводить результати в текстове поле.
4. Кнопка "Довідка", яка викликає функцію about, що відображає вікно з інформацією про розробника.

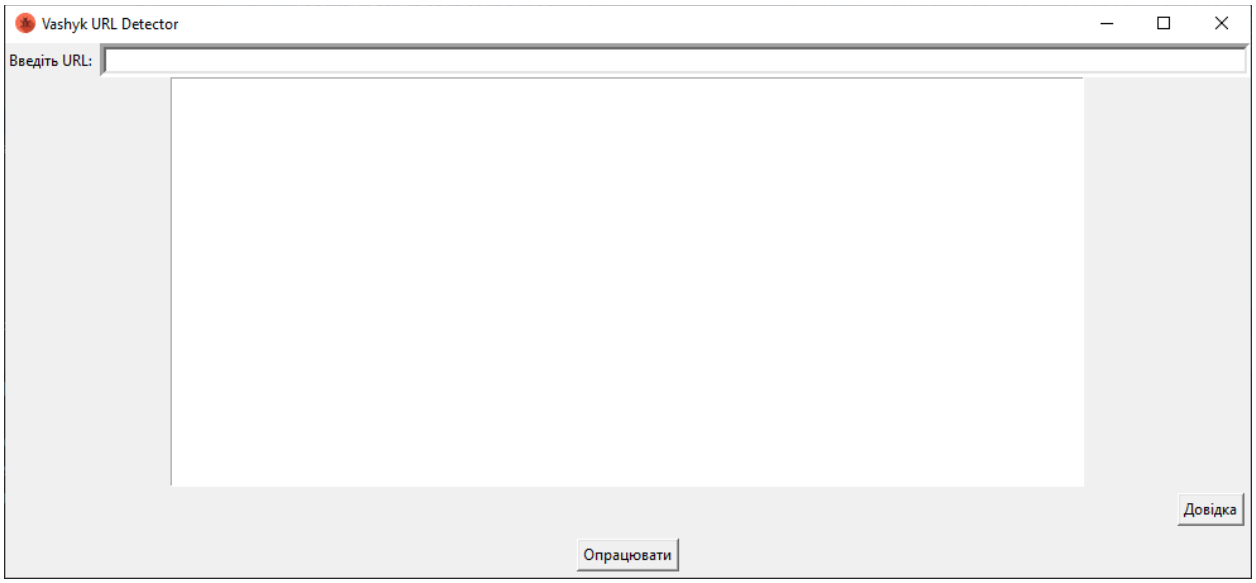

Рисунок 3.4 – Графічний інтерфейс

Основна логіка роботи: при натисканні кнопки "Опрацювати", отримується введений користувачем URL-адреса. Викликається функція feature_extract з файлу Feature_extraction, яка аналізує URL-адресу і повертає

характеристики. Результати аналізу виводяться в текстове поле, де користувач може побачити, чи є URL-адреса безпечною або небезпечною, а також інші характеристики, такі як довжина URL, наявність підтверджуючих IP-адрес та інше.

Архітектура інструменту показана на рис. 3.5. URL-адреси, які потрібно класифікувати як легітимні або фішингові, подаються на вхід відповідного класифікатора. Потім класифікатор, який навчається класифікувати URL-адреси як фішингові або легальні з навчального набору даних, використовує розпізнаний ним шаблон для класифікації нових вхідних даних. З URL-адреси витягуються такі характеристики, як IP-адреса, довжина URL-адреси, домен, наявність іконки тощо, і формується список їхніх значень. Цей список подається на класифікатор випадкових лісів. Потім оцінюється продуктивність цих моделей і генерується оцінка точності. Навчений класифікатор, використовуючи згенерований список, визначає, чи є URL-адреса легітимною, чи фішинговою. Список містить значення 1, 0 і -1, якщо ознаки існують, не застосовуються і якщо ознаки не існують відповідно. У цьому проекті розглядається 30 ознак.

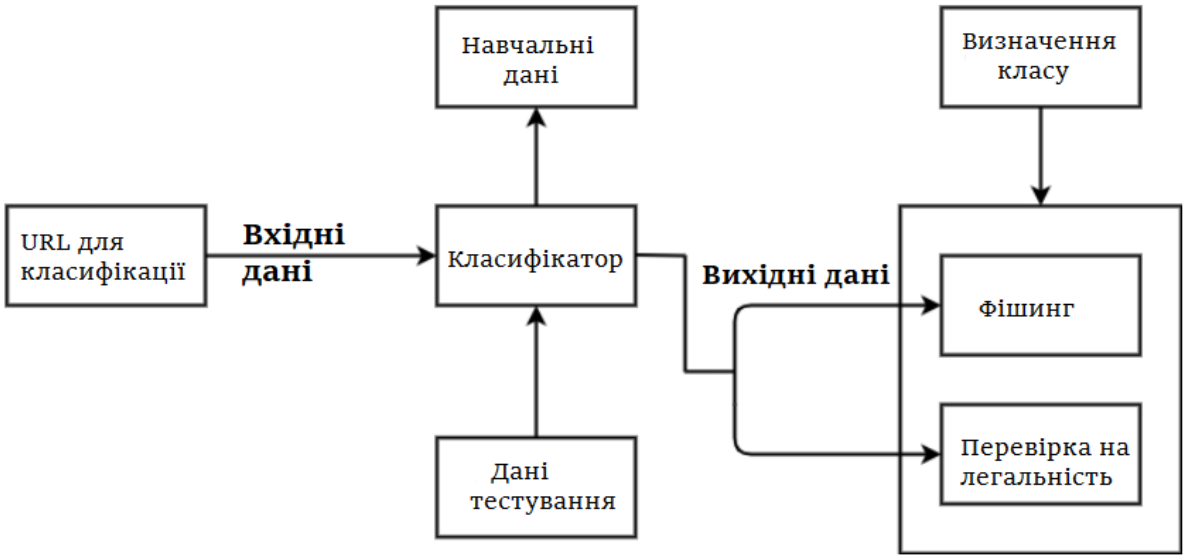

Рисунок 3.5 – Архітектура інструменту виявлення потенційно небезпечних URL-адрес

Діаграми потоків даних (DFD) є важливим інструментом для моделювання та аналізу процесів у системах. Вони дозволяють графічно зображувати потік даних від введення до виведення і визначають взаємозв'язки між різними компонентами системи. DFD широко використовуються в інженерії програмного забезпечення, бізнес-аналітиці та управлінні проектами для розуміння та візуалізації складних процесів. У цьому вступному слові ми розглянемо основні поняття DFD і їх значення в контексті розробки програмного забезпечення та бізнес-аналізу.

DFD використовуються для графічного зображення потоку даних у системі [43]. Вони пояснюють процеси, які відбуваються в системі, від введення даних до створення звіту, і показують всі можливі шляхи від одного об'єкта до іншого в системі. Деталі діаграми потоків даних можуть бути представлені на трьох різних рівнях, які нумеруються 0, 1 і 2. Існує багато типів нотацій для малювання діаграм потоків даних, серед яких найпопулярнішими є YourdonCoad та метод Гейна-Сарсона. DFD, зображені в цій главі, використовують нотації Gane-Sarson DFD.

DFD рівня 0 називають контекстною діаграмою. Це простий огляд всієї системи, що моделюється. На рис. 3.6 показано DFD рівня 0 системи. Вона відображає систему як високорівневий процес з його взаємозв'язками із зовнішніми сутностями. Ця діаграма повинна бути легко зрозумілою широкому колу осіб - від зацікавлених сторін до розробників і аналітиків даних.

Діаграма діяльності є одним з важливих інструментів в аналізі та моделюванні систем. Вона дозволяє візуалізувати послідовність операцій або дій, які відбуваються в системі, та виявити різні можливі шляхи та альтернативні варіанти виконання. Діаграми діяльності зазвичай використовуються для моделювання бізнес-процесів, системних функцій або алгоритмів. Вони дозволяють аналізувати та вдосконалювати ефективність системи, визначати проблемні аспекти та вдосконалювати її функціональність. Діаграма діяльності допомагає розкрити взаємозв'язки між

різними елементами системи та визначити потік управління, допомагаючи у вирішенні складних завдань та прийнятті рішень.

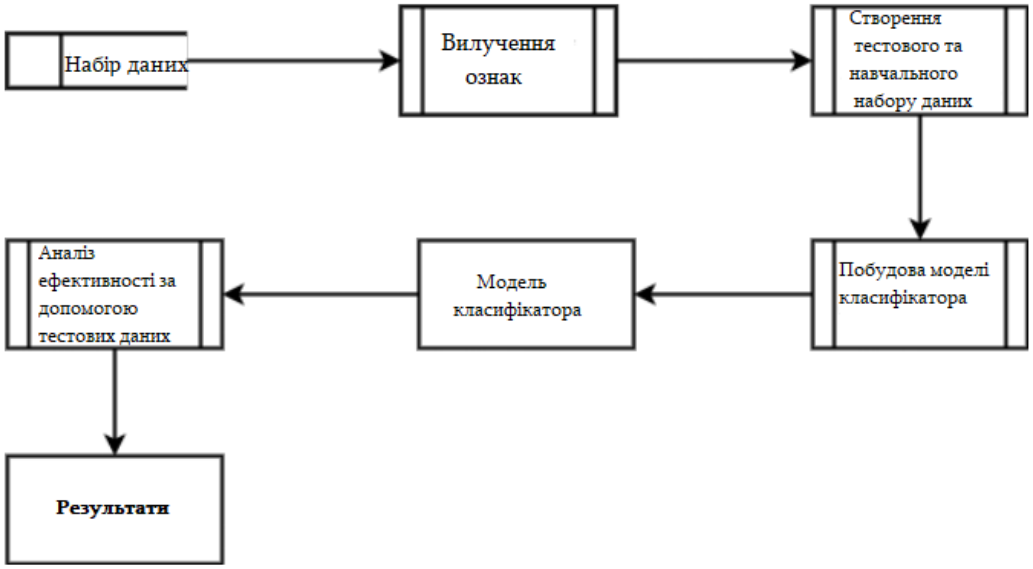

Рисунок 3.6 – Діаграма DFD рівня 0

На рис. 3.7 наведено діаграма діяльності програмного інструменту. Вона зображує потік управління від початкової точки до кінцевої точки, демонструючи різні шляхи, які існують під час виконання діяльності.

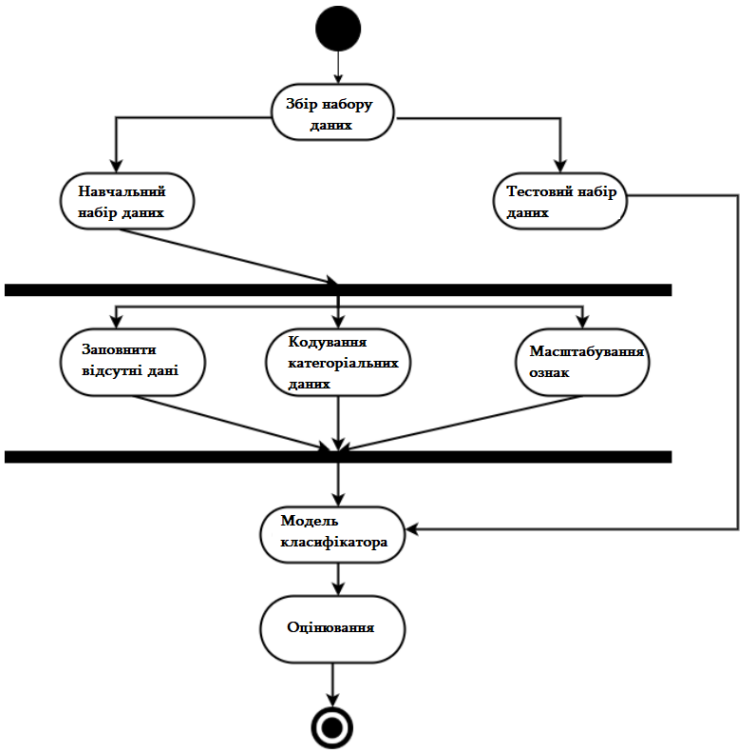

Рисунок 3.7 – Діаграма діяльності програмного інструменту

3.4 Вимоги до апаратного забезпечення

Враховуючи вимоги до апаратного та програмного забезпечення, звернімо увагу на необхідні характеристики для встановлення системи.

Вимоги до апаратного забезпечення:

- процесор: Intel Pentium Dual Core і вище;
- об'єм жорсткого диска: Мінімум 512 МБ;
- оперативна пам'ять: Мінімум 4 ГБ.

Вимоги до програмного забезпечення:

- мова програмування: Python;
- операційна система: Windows 8.1 або вище;
- IDE: Anaconda, iPython версії 3.x.

3.5 Перевірка функціоналу програмного інструменту

Для користування інтерфейсом Vashyk URL Detector потрібно ввести URL-адресу, яку бажаєте перевірити, у відповідне поле введення. Починати з «http://». Натисніть кнопку "Опрацювати", щоб розпочати аналіз URL-адреси.

Після натискання кнопки "Опрацювати" буде проведений аналіз та перевірка URL-адреси та виведено статус «безпечна» (рис.3.8) чи «небезпечна» адреса (рис.3.9).

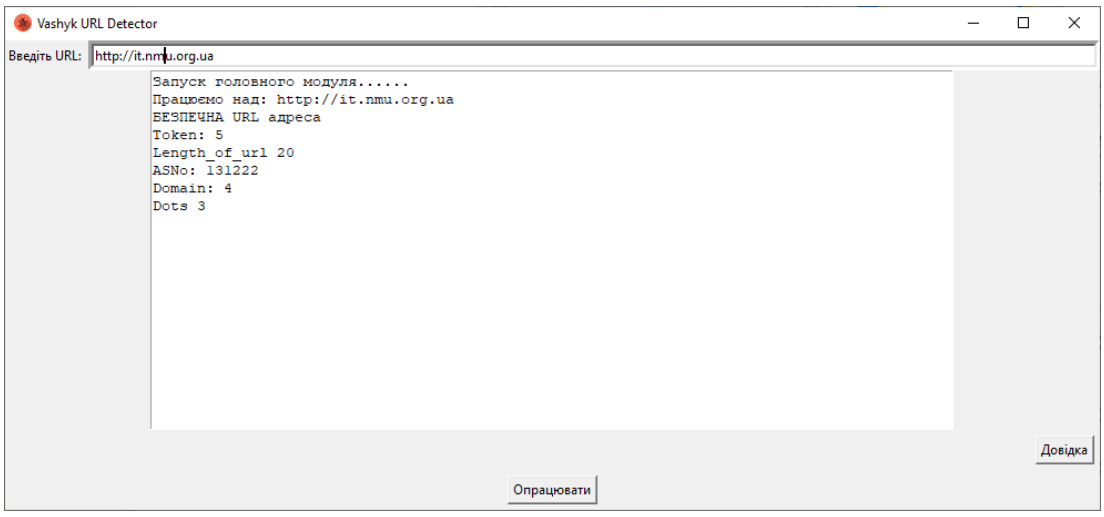

Рисунок 3.9 – Перевірка адреси кафедри

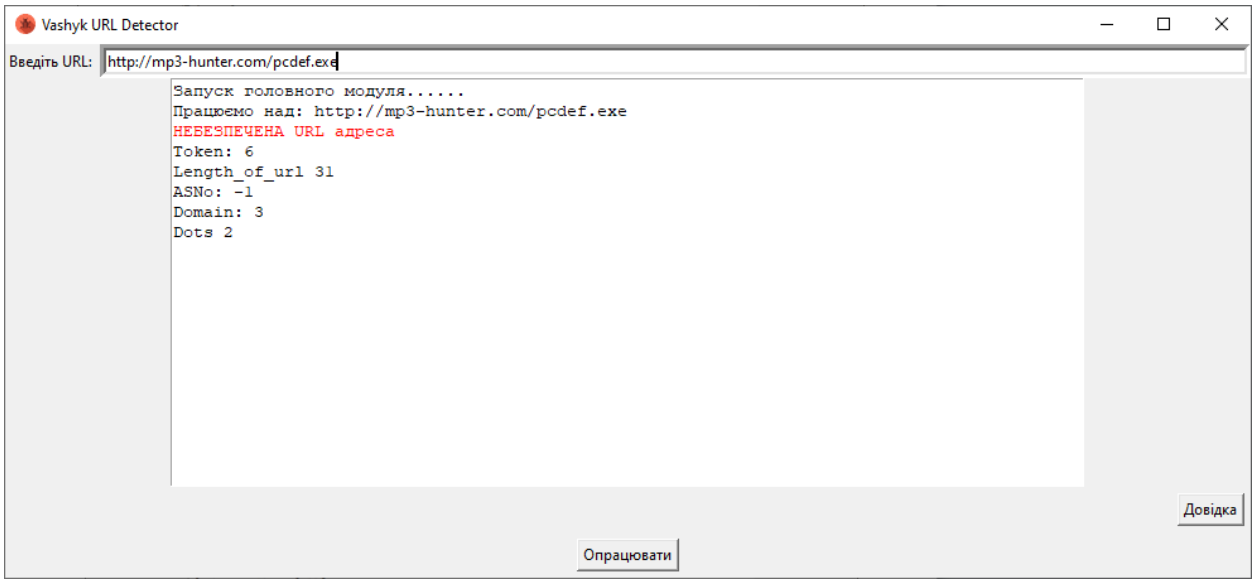

Рисунок 3.10 – Зафіксовано «небезпечну» URL-адресу

Якщо URL-адреса містить домен ".ru", буде виведено повідомлення про потенційну небезпеку. Результати аналізу будуть відображені у полі тексту (рис.3.11 та додаток Б).

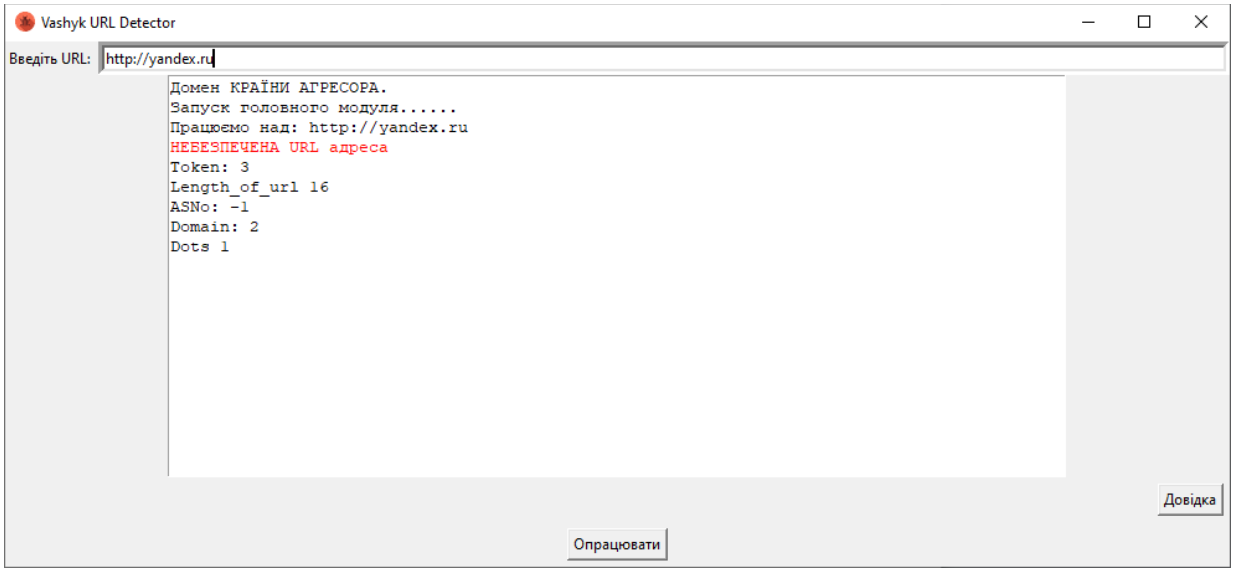

Рисунок 3.11 – Зафіксовано «небезпечну» URL-адресу країни агресора

У результаті аналізу будуть відображені такі характеристики URL-адреси як токени, довжина URL-адреси, номер автономної системи, домен та кількість крапок. Наприклад для рис.3.10: Token: 6 – це кількість токенів у URL-адресі. Токени - це частини URL-адреси, розділені спеціальними

символами. "Length_of_url 31" – це довжина URL-адреси у символах. У цьому випадку довжина URL-адреси складає 31 символ.

"ASNo: -1" – це номер автономної системи (ASNo) для URL-адреси. У цьому випадку ASNo дорівнює -1, що може вказувати на відсутність цієї інформації або неможливість отримати її.

"Domain: 3" – це кількість токенів у доменному імені. У цьому випадку доменне ім'я складається з трьох токенів.

"Dots 2" – це кількість крапок у URL-адресі. У цьому випадку URL-адреса містить дві крапки.

Для отримання додаткової довідки щодо програми натисніть кнопку "Довідка" (рис.3.12). Відображає інформацію про розробника інструменту.

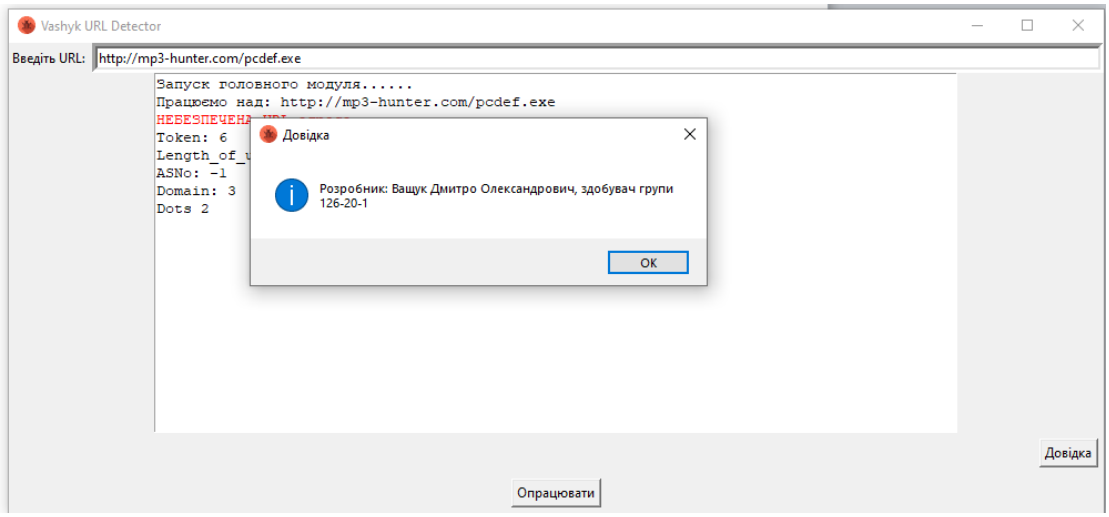

Рисунок 3.12 – Довідкова інформацію

3.6 Висновки

У третьому розділі описано етап підготовки первинних даних для подальшого аналізу та використання у нашій програмі. Цей процес включав в себе збір даних, їхню обробку та конвертацію у формат, придатний для подальшого використання. Важливою частиною цього процесу була групування та сортування даних з урахуванням їхнього потенційного впливу на роботу програми.

Наступним кроком було налаштування та реалізація генератора функцій, необхідних для виконання аналізу даних. Ми ретельно перевірили роботу генератора та впевнилися у його відповідності поставленим завданням. Після успішного завершення цього етапу ми використовували генератор для створення необхідних функцій та алгоритмів для обробки даних.

Описано розробку графічного інтерфейсу програми. Ретельно працювали над кожним компонентом інтерфейсу, забезпечуючи його зручність та ефективність для користувача. Після завершення розробки графічного інтерфейсу, провели його перевірку, щоб переконатися у коректності роботи та відповідності вимогам.

.

ВИСНОВКИ

Задача виявлення шкідливих URL-адрес, без завантаження вмісту сторінки, є актуальною та важливою для забезпечення кібербезпеки. У рамках цієї кваліфікаційної роботи було досліджено та описано різноманітні ознаки, які можуть бути використані для ефективного представлення URL-адреси. Однак, проблема правильного представлення цих ознак та їх нормалізації виникла через неправильне представлення даних, що ускладнювало досягнення бажаних результатів.

Для вирішення цієї проблеми було запропоновано та реалізовано програмний інструмент для виявлення потенційно небезпечних URL-адрес за допомогою бібліотеки scikit-learn. Робота включала в себе аналіз, обробку та підготовку даних, налаштування та реалізацію генератора функцій, а також розробку графічного інтерфейсу для зручного використання програмного забезпечення. В ході роботи було проведено вивчення та використання різноманітних алгоритмів машинного навчання, зокрема класифікаторів scikit-learn, для побудови моделі виявлення потенційно небезпечних URL-адрес.

Отриманий програмний інструмент може бути корисним для виявлення та блокування потенційно небезпечних URL-адрес у реальному часі, що допомагає забезпечити безпеку користувачів в інтернеті. Результати роботи вказують на ефективність та точність розробленого інструменту у виявленні небезпечних URL-адрес, що може сприяти підвищенню рівня кібербезпеки. Додаткові можливості та функціональність програмного забезпечення можуть бути розширені та вдосконалені у майбутньому для ще більш ефективного виявлення потенційно небезпечних URL-адрес та підвищення його корисності для користувачів.

ПЕРЕЛІК ПОСИЛАНЬ

1. INC., Webroot. 2019 Webroot Threat Report. ¶[Електронний ресурс] – Режим доступу до ресурсу: https : / / www - cdn . webroot . com / 9315 / 5113 / 6179 / 2019 _Webroot_Threat_Report_US_Online.pdf.

2. S. Moore, Gartner Forecasts Worldwide Information Security Spending to Exceed $124 Billion in 2019, Gartner, accessed July 13, 2020.

3. CHOI, Hyunsang; ZHU, Bin B.; LEE, Heejo. Detecting Malicious Web Links and Identifying Their Attack Types. In: Proceedings of the 2Nd USENIX Conference on Web Application Development. 2011.

4. GOOGLE. Google Safe Browsing - Blacklist service provided by Google. ¶[Електронний ресурс] – https : / / safebrowsing . google.com/.

5. SAHOO, Doyen; LIU, Chenghao; HOI, Steven C. H. Malicious URL Detection using Machine Learning: A Survey. 2017.

6. LE, Anh; MARKOPOULOU, Athina; FALOUTSOS, Michalis. PhishDef: URL Names Say It All. 2010.

7. YAHYA, Ammar; AHMAD, R.Badlishah; MOHD YACOB, Yasmin; MOHD WARIP, Mohd Nazri Bin. Lightweight phishing URLs detection using N-gram features. 2016, vol. 8, pp. 1563–1570.

8. VERMA, Rakesh; DAS, Avisha. What's in a URL: Fast Feature Extraction and Malicious URL Detection. In: 2017, pp. 55–63.

9. ALTHOBAITI, Kholoud; RUMMANI, Ghaidaa; VANIEA, Kami. A Review of Human-and Computer-Facing URL Phishing Features. In: 2019.

10. J. Jang-Jaccard and S. Nepal, "A survey of emerging threats in cybersecurity," Journal of Computer and System Sciences, vol. 80, no. 5, pp. 973–993, 2014.

11. D. Milkovich, 15 Alarming Cyber Security Facts and Stats, Cybint, accessed July 13, 2020.

12. Mahdavifar, Samaneh & Ghorbani, Ali. (2019). Application of Deep Learning to Cybersecurity: A Survey. Neurocomputing. 347. 10.1016/j.neucom.2019.02.056.

13. S. Patil and S. Dhage. A methodical overview on phishing detection along with an organized way to construct an anti-phishing framework. In 2019 5th International Conference on Advanced Computing Communication Systems (ICACCS), pages 588–593, 2019.

Додаток А. Фагмент лістингу програми

Фрагмент лістингу **Feature_extraction.py**

```
from urllib.parse import urlparse

import re

import urllib.request

from xml.dom import minidom

import csv

import pygeoip as pygeoip



opener = urllib.request.build_opener()

opener.addheaders = [('User-agent', 'Mozilla/5.0')]

print(opener)


nf = -1



def Tokenise(url):

    if url == '':

        return [0, 0, 0]

    token_word = re.split('\W+', url)

    no_ele = sum_len = largest = 0

    for ele in token_word:

        l = len(ele)

        sum_len += l

        if l > 0:
```

```
                no_ele += 1
            if largest < l:
                largest = l
        try:
            return [float(sum_len) / no_ele, no_ele, largest]
        except:
            return [0, no_ele, largest]



    def find_ele_with_attribute(dom, ele, attribute):
        for subelement in dom.getElementsByTagName(ele):
            if subelement.hasAttribute(attribute):
                return subelement.attributes[attribute].value
        return nf



    def sitepopularity(host):
        xmlpath = 'http://data.alexa.com/data?cli=10&dat=snbamz&url='
+ host
        try:
            xml = urllib.request.urlopen(xmlpath)
            dom = minidom.parse(xml)
            rank_host = find_ele_with_attribute(dom, 'REACH', 'RANK')
            rank_country = find_ele_with_attribute(dom, 'COUNTRY',
'RANK')
            return [rank_host, rank_country]
        except:
            return [nf, nf]
```

```
    def Security_sensitive(tokens_words):
        sec_sen_words = ['confirm', 'account', 'banking', 'secure',
'ebayisapi', 'webscr', 'login', 'signin']
        cnt = 0
        for ele in sec_sen_words:
            if ele in tokens_words:
                cnt += 1
        return cnt



    def exe_in_url(url):
        if url.find('.exe') != -1:
            return 1
        return 0



    def Check_IPaddress(tokens_words):
        cnt = 0
        for ele in tokens_words:
            if ele.isnumeric():
                cnt += 1
            else:
                if cnt >= 4:
                    return 1
                else:
```

```
                cnt = 0
        if cnt >= 4:
            return 1
        return 0



    def getASN(host):
        try:
            g = pygeoip.GeoIP('GeoIPASNum.dat')
            asn = int(g.org_by_name(host).split()[0][2:])
            return asn
        except:
            return nf



    def safebrowsing(url):
        api_key = "ABQIAAAA8C6Tfr7tocAe04vXo5uYqRTEYoRzLFR0-
nQ3fRl5qJUqcubbrw"
        name = "URL_check"
        ver = "1.0"


        req = {}
        req["client"] = name
        req["apikey"] = api_key
        req["appver"] = ver
        req["pver"] = "3.0"
        req["url"] = url
```

```
        try:
            params = urllib.parse.urlencode(req)
            req_url = "https://sb-
ssl.google.com/safebrowsing/api/lookup?" + params
            res = urllib.request.urlopen(req_url)
            if res.code == 204:
                print("safe")
                return 0
            elif res.code == 200:
                print("The queried URL is either phishing, malware or
both, see the response body for the specific type.")
                return 1
            elif res.code == 204:
                print("The requested URL is legitimate, no response
body returned.")
            elif res.code == 400:
                print("Bad Request The HTTP request was not correctly
formed.")
            elif res.code == 401:
                print("Not Authorized The apikey is not authorized")
            else:
                print("Service Unavailable The server cannot handle
the request. Besides the normal server failures, it could also
indicate that the client has been throttled by sending too many
requests")
        except:
            return -1
```

Фрагмент лістингу **main. py**

```
import csv
import Feature_extraction as urlfeature
import trainer as tr



def resultwriter(rowdict, output_dest):
    with open(output_dest, 'w', newline='') as f:
        fieldnames = rowdict[0].keys()  # Вибираємо ключі першого словника у списку
        writer = csv.DictWriter(f, fieldnames=fieldnames)

        writer.writeheader()
        for row in rowdict:
            # Конвертуємо значення в байтовий об'єкт, якщо це необхідно
            for key, value in row.items():
                if isinstance(value, str):
                    row[key] = value.encode()
            writer.writerow(row)



def process_URL_list(file_dest, output_dest):
    feature = []
    with open(file_dest) as file:
        for line in file:
            url = line.split(',')[0].strip()
            malicious_bool = line.split(',')[1].strip()
```

```
            if url != '':
                print
                'working on: ' + url
                ret_dict = urlfeature.feature_extract(url)
                ret_dict['malicious'] = malicious_bool
                feature.append([url, ret_dict]);
    resultwriter(feature, output_dest)



def process_test_list(file_dest, output_dest):
    feature = []
    with open(file_dest) as file:
        for line in file:
            url = line.strip()
            if url != '':
                print
                'working on: ' + url
                ret_dict = urlfeature.feature_extract(url)
                feature.append([url, ret_dict]);
    resultwriter(feature, output_dest)
```

Фрагмент лістингу **trainer. py**

```
from sklearn import svm
from sklearn.model_selection import cross_validate as cv
import matplotlib.pylab as plt
import warnings
from sklearn.metrics import confusion_matrix
```

```
warnings.filterwarnings("ignore", category=DeprecationWarning,
module="pandas", lineno=570)



def return_nonstring_col(data_cols):
    cols_to_keep = []
    train_cols = []
    for col in data_cols:
        if col != 'URL' and col != 'host' and col != 'path':
            cols_to_keep.append(col)
            if col != 'malicious' and col != 'result':
                train_cols.append(col)
    return [cols_to_keep, train_cols]



def svm_classifier(train, query, train_cols):
    clf = svm.SVC()


    train[train_cols] = preprocessing.scale(train[train_cols])
    query[train_cols] = preprocessing.scale(query[train_cols])


    print (clf.fit(train[train_cols], train['malicious']))
    scores = cv.cross_val_score(clf, train[train_cols],
train['malicious'], cv=30)
    print('Estimated score SVM: %0.5f (+/- %0.5f)' % (scores.mean(),
scores.std() / 2))
```

```
    query['result'] = clf.predict(query[train_cols])


    print(query[['URL', 'result']])



# Called from gui
def forest_classifier_gui(train, query, train_cols):
    rf = RandomForestClassifier(n_estimators=150)


    print(rf.fit(train[train_cols], train['malicious']))


    query['result'] = rf.predict(query[train_cols])


    print(query[['URL', 'result']].head(2))
    return query['result']



def forest_classifier(train, query, train_cols):
    rf = RandomForestClassifier(n_estimators=150)


    print(rf.fit(train[train_cols], train['malicious']))
    scores = cv.cross_val_score(rf, train[train_cols],
train['malicious'], cv=30)
    print('Estimated score RandomForestClassifier: %0.5f (+/- %0.5f)'
% (scores.mean(), scores.std() / 2))


    query['result'] = rf.predict(query[train_cols])
    print(query[['URL', 'result']])
```

```
def train(db, test_db):
    query_csv = pandas.read_csv(test_db)
    cols_to_keep, train_cols = return_nonstring_col(query_csv.columns)
    # query=query_csv[cols_to_keep]

    train_csv = pandas.read_csv(db)
    cols_to_keep, train_cols = return_nonstring_col(train_csv.columns)
    train = train_csv[cols_to_keep]

    svm_classifier(train_csv, query_csv, train_cols)

    forest_classifier(train_csv, query_csv, train_cols)


def gui_caller(db, test_db):
    query_csv = pandas.read_csv(test_db)
    cols_to_keep, train_cols = return_nonstring_col(query_csv.columns)

    train_csv = pandas.read_csv(db)
    cols_to_keep, train_cols = return_nonstring_col(train_csv.columns)
    train = train_csv[cols_to_keep]

    result = forest_classifier_gui(train_csv, query_csv, train_cols)
    return result
```

Фрагмент лістингу **GUI. py**

```
from tkinter import *
import tkinter.messagebox as tkMessageBox
import trainer as tr
import webbrowser
import main
import Feature_extraction
import csv


def submitCallBack():
    url = E1.get()  # Отримуємо URL з поля введення
    result_text.delete(1.0, END)  # Очищаємо вміст поля Text
    if '.ru' in url:
        result_text.insert(END, "Домен КРАЇНИ АГРЕСОРА.\n")
        # Додайте код для відображення повідомлення на формі або в іншому місці за потреби


    # Викликаємо функцію feature_extract для обробки URL
    result = Feature_extraction.feature_extract(url)


    # Додаємо результати у поле Text
    result_text.insert(END, "Запуск головного модуля......\n")
    result_text.insert(END, "Працюємо над: " + url + "\n")
    # Перевіряємо значення rank_host та rank_country
    if result['ASNno'] == -1 and result['rank_country'] == -1:
        result_text.tag_configure("danger", foreground="red")  # Налаштовуємо новий тег для червоного коліру
```

```
        result_text.insert(END, "НЕБЕЗПЕЧЕНА URL адреса\n", "danger")
# Використовуємо тег для цього рядка тексту
    else:
        result_text.insert(END, "БЕЗПЕЧНА URL адреса\n")


    result_text.insert(END, "Token: " + str(result.get('token_count',
'N/A')) + "\n")
    result_text.insert(END, "Length_of_url " +
str(result.get('Length_of_url', 'N/A')) + "\n")
    result_text.insert(END, "ASNo: " + str(result.get('ASNno', 'N/A'))
+ "\n")
    result_text.insert(END, "Domain: " +
str(result.get('domain_token_count', 'N/A')) + "\n")
    result_text.insert(END, "Dots " + str(result.get('No_of_dots',
'N/A')) + "\n")
    # Відображаємо результати, якщо URL співпадає з введеним
користувачем
     # Додаємо кількість токенів у домені
                # Додайте інші ключі та значення за потреби
    # result_text.insert(END, "Інші ключі та значення: " +
str(return_ans['інший_ключ']) + "\n")


def about():
    tkMessageBox.showinfo("Довідка", "Розробник: Ващук Дмитро
Олександрович, здобувач групи 126-20-1")


root = Tk()
root.title("Vashchuk URL Detector")
root.iconbitmap(r'malware.ico')
```

```
frame = Frame(root)

frame.pack()

bottomframe = Frame(root)

bottomframe.pack(side=BOTTOM)


L1 = Label(frame, text="Введіть URL: ")

L1.pack(side=LEFT)
```

Фрагмент лістингу **test. py**

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score, confusion_matrix


# File Paths

INPUT_PATH = "../inputs/breast-cancer-wisconsin.data"

OUTPUT_PATH = "../inputs/breast-cancer-wisconsin.csv"


# Headers

HEADERS = ["CodeNumber", "ClumpThickness", "UniformityCellSize",
"UniformityCellShape", "MarginalAdhesion",

           "SingleEpithelialCellSize", "BareNuclei", "BlandChromatin",
"NormalNucleoli", "Mitoses", "CancerType"]



def read_data(path):

    """

    Read the data into pandas dataframe
```

```
    :param path:
    :return:
    """
    data = pd.read_csv(path)
    return data


def get_headers(dataset):
    """
    dataset headers
    :param dataset:
    :return:
    """
    return dataset.columns.values


def add_headers(dataset, headers):
    """
    Add the headers to the dataset
    :param dataset:
    :param headers:
    :return:
    """
    dataset.columns = headers
    return dataset
```

```
def data_file_to_csv():
    """
    :return:
    """

    # Headers
    headers = ["CodeNumber", "ClumpThickness", "UniformityCellSize",
"UniformityCellShape", "MarginalAdhesion",
               "SingleEpithelialCellSize", "BareNuclei",
"BlandChromatin", "NormalNucleoli", "Mitoses",
               "CancerType"]
    # Load the dataset into Pandas data frame
    dataset = read_data(INPUT_PATH)
    # Add the headers to the loaded dataset
    dataset = add_headers(dataset, headers)
    # Save the loaded dataset into csv format
    dataset.to_csv(OUTPUT_PATH, index=False)
    print("File saved ...!")


def split_dataset(dataset, train_percentage, feature_headers,
target_header):
    """
    Split the dataset with train_percentage
    :param dataset:
    :param train_percentage:
    :param feature_headers:
    :param target_header:
```

```
    :return: train_x, test_x, train_y, test_y
    """

    # Split dataset into train and test dataset
    train_x, test_x, train_y, test_y =
train_test_split(dataset[feature_headers], dataset[target_header],

train_size=train_percentage)
    return train_x, test_x, train_y, test_y



def handel_missing_values(dataset, missing_values_header,
missing_label):
    """
    Filter missing values from the dataset
    :param dataset:
    :param missing_values_header:
    :param missing_label:
    :return:
    """

    return dataset[dataset[missing_values_header] != missing_label]



def random_forest_classifier(features, target):
    """
    To train the random forest classifier with features and target
data
```

```
    :param features:
    :param target:
    :return: trained random forest classifier
    """
    clf = RandomForestClassifier()
    clf.fit(features, target)
    return clf


def dataset_statistics(dataset):
    """
    Basic statistics of the dataset
    :param dataset: Pandas dataframe
    :return: None, print the basic statistics of the dataset
    """
    print(dataset.describe())


def main():
    """
    Main function
    :return:
    """
    # Load the csv file into pandas dataframe
    dataset = pd.read_csv(OUTPUT_PATH)
    # Get basic statistics of the loaded dataset
    dataset_statistics(dataset)
```

```
    # Filter missing values
    dataset = handel_missing_values(dataset, HEADERS[6], '?')
    train_x, test_x, train_y, test_y = split_dataset(dataset, 0.7,
HEADERS[1:-1], HEADERS[-1])

    # Train and Test dataset size details
    print("Train_x Shape :: ", train_x.shape)
    print("Train_y Shape :: ", train_y.shape)
    print("Test_x Shape :: ", test_x.shape)
    print("Test_y Shape :: ", test_y.shape)

    # Create random forest classifier instance
    trained_model = random_forest_classifier(train_x, train_y)
    print("Trained model :: ", trained_model)
    predictions = trained_model.predict(test_x)

    for i in range(0, 5):
        print("Actual outcome :: {} and Predicted outcome ::
{}".format(list(test_y)[i], predictions[i]))

    print("Train Accuracy :: ", accuracy_score(train_y,
trained_model.predict(train_x)))
    print("Test Accuracy  :: ", accuracy_score(test_y, predictions))
    print(" Confusion matrix ", confusion_matrix(test_y, predictions))

if __name__ == "__main__":
    main()
```

Додаток Б. Аналіз функціональності застосунку

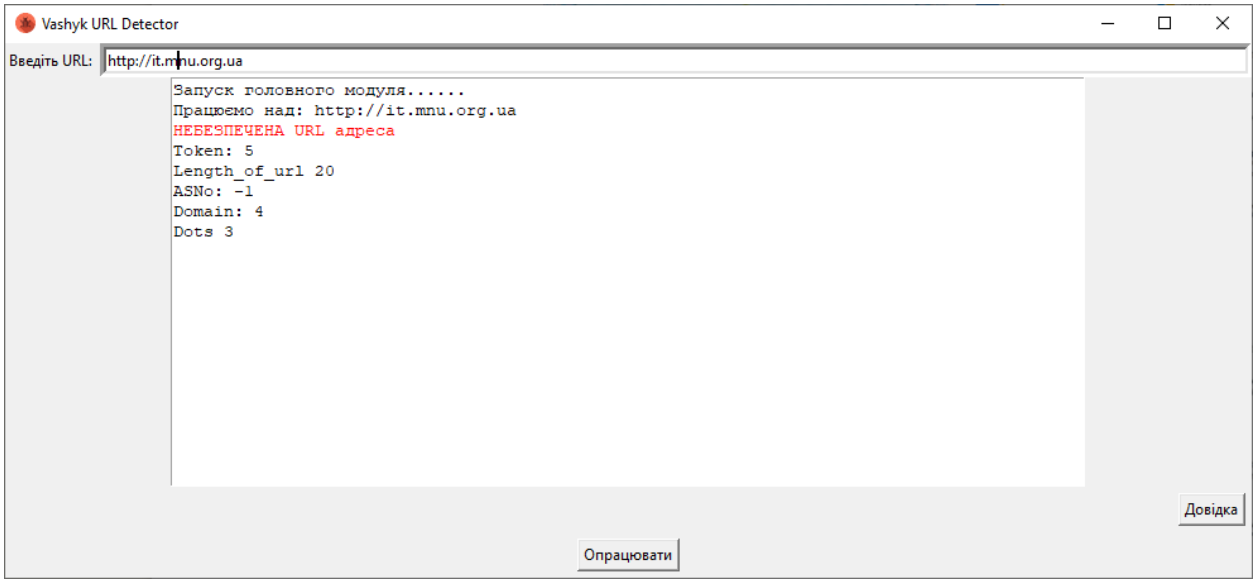

Рисунок Б.1 – Поля пункту «Додати сповіщення»

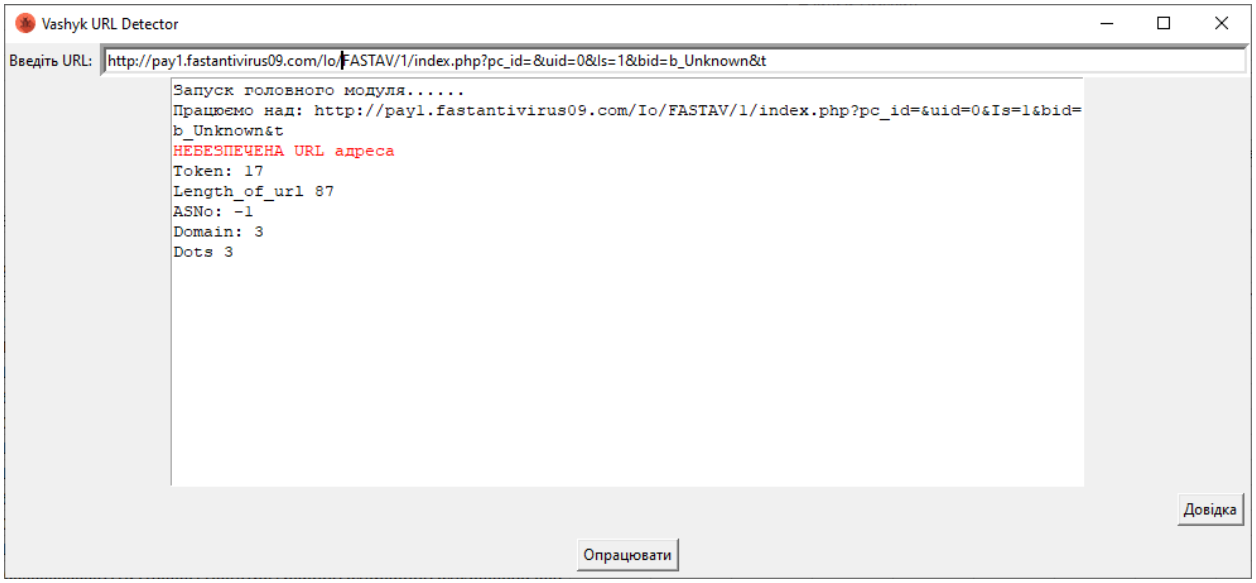

Рисунок Б.2 – Перевірка заповнених полів пункту «Додати сповіщення»

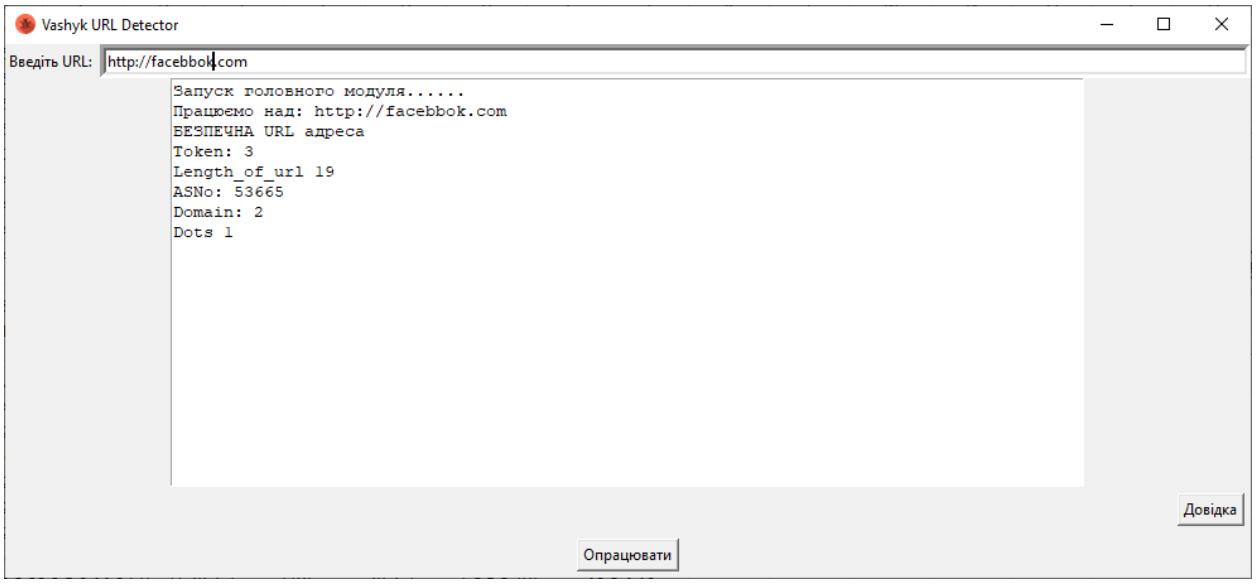

Рисунок Б.3 – Перевірка заповнених полів пункту «Додати сповіщення»

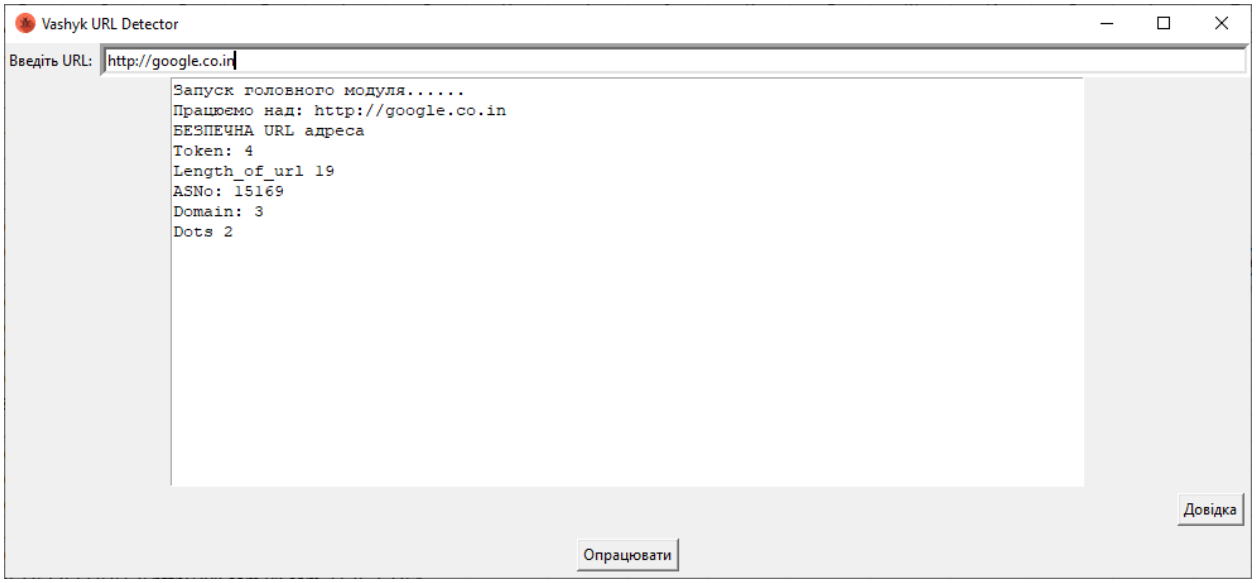

Рисунок Б.4 – Перевірка заповнених полів пункту «Додати сповіщення»