Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти перший (бакалаврський)

Програмні засоби розпізнавання шкідливого програмного забезпечення з використанням машинного навчання
(тема)

Виконав:
здобувач 4 року навчання,
групи КІУКІ-21-2
Валерія Свєтлична
(власне ім'я, прізвище)

Спеціальність
123 «Комп'ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма
Комп'ютерна інженерія
(повна назва освітньої програми)

Керівник: ас. Олександр РОМАНЮК
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ЕОМ _____ Андрій КОВАЛЕНКО
(підпис) (власне ім'я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти перший (бакалаврський)

Спеціальність 123 «Комп’ютерна інженерія»
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Комп’ютерна інженерія
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
“____” _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Свєтличній Валерії Андріївні
(прізвище, ім’я, по батькові)

1. Тема роботи Програмні засоби розпізнавання шкідливого програмного забезпечення з використанням машинного навчання

затверджена наказом по університету від “ 26 ” травня 2025 р. № 424 Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 17 червня 2025 р.

3. Вхідні дані до роботи
машинне навчання
шкідливе програмне забезпечення
аналіз програм у пісочниці
виявлення комп’ютерних загроз
Google Colab

4. Перелік питань, що потрібно опрацювати у роботі
1 Аналіз предметної області
2 Методи машинного навчання. робота з даними
3 Практична частина
4 Аналіз результатів

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій 15 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Аналіз проблеми та огляд існуючих рішень	26.05.2025–30.05.2025	
2	Вибір технології розробки та інструменталь-	31.05.2025–03.06.2025	
	них засобів		
3	Розробка програмних засобів	04.06.2025–08.06.2025	
4	Моделювання та реалізація	09.06.2025–11.06.2025	
5	Аналіз результатів	12.06.2025–13.06.2025	
6	Оформлення ПЗ	14.06.2025–16.06.2025	

Дата видачі завдання " 26 " травня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ ас. Олександр РОМАНЮК
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 63 с., 25 рис.,15 табл., 2 дод., 10 джерел.

ШТУЧНИЙ ІНТЕЛЕКТ, МАШИННЕ НАВЧАННЯ, ШКІДЛИВЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, КІБЕРБЕЗПЕКА, КЛАСИФІКАЦІЯ, АНАЛІЗ ОЗНАК, МОДЕЛІ ВИЯВЛЕННЯ, НАВЧАЛЬНІ ВИБІРКИ, ОБРОБКА ДАНИХ, АВТОМАТИЗОВАНЕ ВИЯВЛЕННЯ ЗАГРОЗ, ВИПАДКОВИЙ ЛІС, НЕЙРОННІ МЕРЕЖІ, ПІДТРИМКА ПРИЙНЯТТЯ РІШЕНЬ, ПОВЕДІНКОВИЙ АНАЛІЗ, АНТИВІРУСНІ ТЕХНОЛОГІЇ.

Метою кваліфікаційної роботи є розробка програмних засобів виявлення шкідливого програмного забезпечення з використанням методів машинного навчання.

У ході виконання кваліфікаційної роботи Обрано та застосовано методи вилучення та представлення ознак, а також оцінено алгоритми машинного навчання. Як метод представлення ознак обрано комбіновану матрицю, що визначає частоту успішних та невдалих викликів API разом із кодами повернення.

Ця репрезентація обрана через відображення фактичної поведінки файлу та об'єднання інформації про різноманітні системні зміни, включаючи модифікації реєстру та файлів, на відміну від інших методів. Порівняльний аналіз існуючих підходів до виявлення шкідливого програмного забезпечення виявив обмеженість традиційних методів на основі сигнатур у протидії сучасним загрозам. Шкідливе програмне забезпечення нового покоління, що використовує методи обфускації, поліморфізму та метаморфізму, потребує принципово нових підходів до виявлення та класифікації.

ABSTRACT

Bachelor’s thesis: 63 pages, 25 figures, 15 tables, 2 appendices, 10 sources.

ARTIFICIAL INTELLIGENCE, MACHINE LEARNING, MALWARE, CYBERSECURITY, CLASSIFICATION, FEATURE ANALYSIS, DETECTION MODELS, TRAINING DATASETS, DATA PROCESSING, AUTOMATED THREAT DETECTION, RANDOM FOREST, NEURAL NETWORKS, DECISION SUPPORT, BEHAVIORAL ANALYSIS, ANTIVIRUS TECHNOLOGIES.

The major goal of this thesis is the development of software tools for malware detection using machine learning methods.

In order to feature extraction and representation methods were selected and applied, and machine learning algorithms were evaluated. A combined matrix was chosen as the method for feature representation, which captures the frequency of successful and unsuccessful API calls along with return codes.

This representation was selected due to its ability to reflect the actual behavior of the file and to combine information about various system-level changes, including registry and file modifications, in contrast to alternative approaches. A comparative analysis of existing malware detection methods revealed the limitations of traditional signature-based techniques in countering modern threats. Next-generation malware that employs obfuscation, polymorphism, and metamorphism requires fundamentally new approaches to detection and classification.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ДА – динамічний аналіз

ПЗ – програмне забезпечення

СА – статичний аналіз

API – Application Programming Interface

IoT – Internet of Things

JSON – JavaScript Object Notation

LCS – Longest Common Subsequence

ML – Machine Learning

MSA – Multiple Sequence Alignment

NP – Non-deterministic Polynomial time

ОС – операційна система

PCAP – Packet Capture

SAT – Boolean Satisfiability Problem

URL – Uniform Resource Locator

ВСТУП

Сучасний етап розвитку інформаційних технологій характеризується експоненціальним зростанням використання мобільних пристроїв для отримання та обробки інформації. Персональні комп'ютери, смартфони та інші цифрові девайси стали невід'ємними компонентами повсякденного життя сучасної людини. Первісна мета створення цих технологічних рішень полягала в автоматизації складних завдань та спрощенні процесів, які важко виконати виключно за допомогою людських ресурсів.

Останні десятиліття засвідчили трансформацію інформаційних технологій у критично важливу інфраструктуру функціонування суспільства. Пандемія COVID-19 продемонструвала абсолютну залежність сучасного суспільства від цифрових технологій, особливо в контексті підтримки соціального дистанціювання та забезпечення можливості віддаленої роботи. Це призвело до колосального збільшення обсягів даних, що циркулюють у глобальних мережах.

Статистичні дані, оприлюднені корпорацією Microsoft, демонструють приріст використання функціоналу дзвінків та відеоконференцій програмного продукту Teams на 775% в Італії протягом місяця локдауну. Використання технології Windows Virtual Desktop зросло втричі. Американський постачальник медійних послуг Netflix повідомив про залучення понад 16 мільйонів нових користувачів. Глобальний потік інтернет-трафіку збільшився на 57%, а відсоток завантажень різних типів даних на хмарні платформи зріс на 80%.

Зазначені тенденції актуалізують питання захисту інформації, що завантажується у хмарні сховища. Проблема кібербезпеки, завжди актуальна для інформаційного суспільства, набула критичного значення в умовах цифрової трансформації.

Метою кваліфікаційної роботи є створення комп'ютерної системи

розпізнавання шкідливого програмного забезпечення з використанням методів машинного навчання та розробка концепції класифікації шкідливих програм на основі віртуального середовища Cuckoo Sandbox. Дане середовище використовуватиметься для аналізу поведінкових характеристик зразків шкідливих програм, які слугуватимуть вхідними даними для алгоритмів машинного навчання. Основне завдання полягає у визначенні оптимального методу представлення ознак та найточнішого алгоритму для диференціації сімейств шкідливих програм з мінімальним рівнем помилок.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Вплив комп'ютерних атак на розвиток індустрії

У 2025 році практично кожен член суспільства активно використовує інтернет-технології у повсякденному житті. Соціальні мережі, інтернет-банкінг, електронні платежі та онлайн-покупки становлять лише частину щоденних цифрових активностей. Інтеграція розумних пристроїв, здатних самостійно вирішувати та виконувати різноманітні завдання без безпосередньої участі людини, призвела до експоненціального зростання використання технологій у всіх сферах життєдіяльності.

Хоча технологічний прогрес сприяв підвищенню рівня життя та продуктивності завдяки автоматизації численних процесів, він одночасно збільшив вразливість користувачів до атак кіберзлочинців. Кожен новий пристрій або додаток створює потенційну можливість для викрадення даних або завдання шкоди користувачам та їхнім девайсам.

Кібербезпека визначається як сукупність технологій, процесів та практик, призначених для захисту мереж, пристроїв, програм та даних від атак, пошкоджень або несанкціонованого доступу. Альтернативною назвою цієї галузі є безпека інформаційних технологій.

Проблематика кібербезпеки однаково хвилює як приватних користувачів, так і великі корпорації. Незважаючи на значні інвестиції більшості компаній у системи захисту та розробку нових алгоритмів безпеки, їхня вразливість до кіберагак не зменшується порівняно із звичайними користувачами мережі. Більше того, корпорації часто стають пріоритетними мішенями через зберігання великих обсягів конфіденційних даних.

Дослідження 2018-2023 року засвідчили, що близько 68% підприємств у світі зазнали атак із використанням фішингових листів або інших форм соціальної інженерії. У 2024 році кожні 14 секунд деяка організація ставала

жертвою програми-шантажиста. Середня вартість збитків від витоку інформації для типової компанії становила 3,86 мільйона доларів.

Початок пандемії кардинально змінив статистичні показники кіберзлочинності. Кількість кібератак зросла на 600%. Порівняно з попереднім роком відсоток викрадених даних кредитних карт збільшився на 212%, даних облікових записів на 129%, а поширення шкідливого програмного забезпечення на 102%.

Характерними прикладами кіберінцидентів періоду пандемії стало викрадення 500 тисяч паролів сервісу Zoom та їх розміщення для продажу в мережі даркнет, а також атака на американську медичну компанію Magellan Health, внаслідок якої було вкрадено дані 365 тисяч пацієнтів.

Кіберзлочинці демонструють високу адаптивність до актуальних ситуацій. У період пандемії інформація у фішингових листах з високою імовірністю стосувалася COVID-19, зокрема засобів індивідуального захисту, продуктів харчування або вакцинації.

Аналіз представлених даних дозволяє зробити висновок про критично важливу роль кібербезпеки у сучасному світі. Без постійного дослідження, розробки та впровадження нових методів та алгоритмів захисту переважна більшість користувачів мережі стали б жертвами кіберзлочинців.

1.2 Існуючі типи шкідливого програмного забезпечення

Шкідливе програмне забезпечення [1] являє собою програмні продукти, які навмисно виконують зловмисні дії на пристроях жертв, включаючи комп'ютери, смартфони, мережі та інші цифрові системи. Такі програми можуть перешкоджати нормальному функціонуванню пристроїв, збирати конфіденційну інформацію або отримувати несанкціонований доступ до приватних комп'ютерних систем. Шкідливе програмне забезпечення може бути представлене у вигляді коду, скрипту, активного контенту та іншого програмного забезпечення.

Класифікація шкідливого програмного забезпечення включає віруси, рекламне програмне забезпечення, хробаки, троянські програми, руткіти, клавіатурні логери, дозвонювачі, шпигунські програми, здирницькі програми, шкідливі плагіни та інші типи зловмисного програмного забезпечення.

Комп'ютерний вірус представляє собою програму, що володіє здатністю до прихованого самопоширення. Одночасно зі створенням власних копій віруси можуть завдавати значної шкоди системам, включаючи знищення, пошкодження або викрадення даних, а також повне унеможливлення подальшого функціонування операційної системи. Поширення вірусу в системі відбувається після певної взаємодії з користувачем, наприклад, виконання інфікованої програми.

Рекламне програмне забезпечення [2] являє собою програми, що відображають небажану рекламу на комп'ютері користувача. Таке програмне забезпечення може змінювати домашню сторінку браузера, додавати шпигунські модулі для викрадення конфіденційної інформації та перевантажувати пристрій рекламним контентом. Додатково воно може призводити до зниження швидкості інтернет-з'єднання та блокування інших програм.

Комп'ютерний хробак являє собою зловмисну програму, яка, подібно до вірусу, має здатність до самопоширення, проте відтворюється не в межах одного пристрою, а активно поширюється через мережу. Джерелами поширення є фішингові листи, ресурси для обміну файлами та інтернет-посилання. Хробаки можуть спричиняти системні помилки, незвичайну поведінку комп'ютера, зниження продуктивності або повне блокування системи.

Троянські програми представляють собою шкідливі програми, нездатні до самопоширення, тому потребують активації внаслідок взаємодії з користувачем. Після активації такі програми дозволяють кіберзлочинцям здійснювати шпигунську діяльність, викрадати конфіденційні дані та

отримувати backdoor-доступ до системи. Цей доступ надає зловмисникам віддалений контроль над інфікованим комп'ютером, дозволяючи виконувати будь-які операції, включаючи надсилання, отримання, запуск та видалення файлів, відображення даних та перезавантаження системи. Backdoor часто використовуються для об'єднання комп'ютерів-жертв у ботнет-мережі для кримінальних цілей.

Руткіт являє собою програму, основною метою якої є приховування слідів власного існування, що ускладнює виявлення інфікування. Ядерний руткіт призначений для модифікації функціональності операційної системи шляхом додавання власного коду та структур даних до ядра ОС, що дозволяє встановлювати шкідливі драйвери, модулі та служби. Залежно від середовища поширення та типу ці програми відрізняються за механізмами дії.

Програма-шантажист [3] являє собою шкідливе програмне забезпечення, що обмежує або блокує доступ користувачів до системи, блокуючи екран або файли до сплати викупу. Деякі види таких програм шифрують певні типи файлів на заражених системах та вимагають оплату через специфічні методи онлайн-платежів для отримання ключа дешифрування.

Тип	Сімейство	Тип	Сімейство
Backdoor	*Backdoor.Agent*	Trojan	*Trojan-Banker*
	Backdoor.Androm		*Trojan-DDoS*
	Backdoor.Bifrose		*Trojan-Downloader*
	Backdoor.BlackHole		*Trojan-Dropper*
	Backdoor.Chyopic		*Trojan-FakeAV*
	Backdoor.DarkKomet		*Trojan-GameThief*
	Backdoor.Delf		*Trojan-IM*
	Backdoor.Gbot		*Trojan-Ransom*
	Backdoor.Hupigon		*Trojan-SMS*
	...		...

Рисунок 1.1 – Сімейства типів шкідливого ПЗ Trojans та Backdoor

Кожен тип шкідливого програмного забезпечення розроблений для специфічного впливу на систему жертви. Враховуючи способи, методи та

цілі, їх можна класифікувати у певні групи-сімейства, як наприклад сімейства типів Trojans та Backdoor (рисунок 1.1).

Сучасна класифікація шкідливого програмного забезпечення стає дедалі складнішим завданням, оскільки багато екземплярів можуть поєднувати характеристики декількох типів. На початковому етапі розвитку такі програми створювались для простих цілей, що спрощувало їх виявлення та класифікацію як традиційного шкідливого програмного забезпечення.

Сучасне шкідливе програмне забезпечення [4] може функціонувати в режимі ядра, приховувати свою активність, чинити більш руйнівний вплив на системи та бути значно складнішим для виявлення порівняно з традиційними видами. Таке програмне забезпечення класифікується як шкідливе програмне забезпечення нового покоління, здатне легко обходити захисні системи, що працюють у режимі ядра, включаючи брандмауери та антивірусне програмне забезпечення.

Параметр для порівняння	Традиційне	Нове покоління
Рівень реалізації	простий	складний
Поширення	всі копії однакові	всі копії різні
Джерело поширення	файли .exe розширення	файли з різними розширеннями
Знаходження у системі	тимчасове	постійне
Взаємодія з іншими процесами	декілька процесів (або один)	багато процесів
Приховування	ні	так
Ціль атаки	звичайні комп'ютери	різні девайси(комп'ютери,смартфони, сервери, ...)

Рисунок 1.2 – Порівняння традиційного та нового покоління шкідливого ПЗ

Традиційне шкідливе програмне забезпечення зазвичай складається з одного процесу та не використовує складних методів маскування. Шкідливі програми нового покоління використовують множинні процеси одночасно, складні техніки приховування та інтеграції у систему на постійній основі. Додатково вони мають більш руйнівний характер, спрямовані на досягнення комплексних цілей та використовують декілька типів шкідливого

програмного забезпечення одночасно (рисунок 1.2).

1.3 Проблема задачі виявлення вірусу

Фредерік Б. Коен, американський інформатик, який винайшов перший вірус та дав йому визначення, описав комп'ютерний вірус як програму, здатну інфікувати інші програми, включаючи у їхній код свою еволюційну копію. Завдяки властивості зараження вірус може поширюватися комп'ютерною системою або мережею, використовуючи права користувача, при цьому кожна інфікована програма може функціонувати як вірус, забезпечуючи експоненціальне зростання інфекції.

Згідно з теорією Коена, задача виявлення вірусу є принципово нерозв'язною через внутрішню суперечність процесу виявлення. Якщо розглядати задачу виявлення вірусу як задачу прийняття рішень D, то D повинен визначити, чи є програма P вірусом. Це неможливо, оскільки якщо P дійсно є вірусом та буде ідентифікований D як вірус, то він не зможе вносити зміни в інші програми та не буде поводитися як вірус. Якщо P є вірусом, а D визначає його як невірус, то P продовжуватиме поширювати копії та інфікувати систему, створюючи логічну суперечність [5].

Неможливість створення програми для безпомилкового виявлення вірусів обумовлена поліморфною природою вірусів, здатних змінювати код без модифікації алгоритму, та існуванням у різноманітних формах.

Хоча у загальних випадках зазначені проблеми нерозв'язні, можливо отримати прийнятні результати при припущенні обмежень на час або обсяг пам'яті шкідливого програмного забезпечення. Спінеліс довів, що використання SAT-задачі та накладання обмежень на задачу Коена дозволяє звести її до NP-повної, роблячи її розв'язною.

NP-повна природа задачі означає відсутність оптимального рішення для найгіршого випадку, проте прийнятні рішення можуть існувати для середнього випадку або отримуватися з використанням апроксимаційних чи

евристичних алгоритмів.

З практичної перспективи проблема полягає в тому, що зловмисне програмне забезпечення нового покоління використовує поширені методи обфускації коду, що дозволяє легко обходити захисні системи, такі як антивіруси або брандмауери. Зокрема, використовується шифрування для зміни коду шкідливого програмного забезпечення, поліморфні методи з різними ключами шифрування та дешифрування, метаморфні методи з динамічним прихованням коду, коли програма змінює код при кожному виконанні [6].

Таке програмне забезпечення складно виявити через унікальність кожної копії та неможливість виділення певного шаблону. Для обходу захисного програмного забезпечення широко використовуються методи архівування. Додатково воно може поєднувати характеристики декількох класів одночасно, роблячи правильну класифікацію з використанням одного підходу практично неможливою.

1.4 Аналіз існуючих підходів задачі виявлення шкідливого програмного забезпечення

1.4.1 Постановка задачі виявлення шкідливого програмного забезпечення

Ефективний та своєчасний захист компаній та користувачів мережі потребує надійних методів виявлення зловмисного програмного забезпечення. Традиційно широко використовувався підхід на основі сигнатур, проте в сучасних умовах він неефективний для виявлення нових та невідомих шкідливих програм нового покоління.

Еволюція наукових підходів призвела до появи динамічного та евристичного аналізу з використанням методів машинного навчання та глибинного аналізу даних. Сучасний тренд міграції даних та інфраструктур у

хмарні платформи створює нові можливості, але одночасно розширює простір для атак зловмисників. Аналогічна ситуація спостерігається з мобільними пристроями, розумними девайсами та інтернетом речей, що обумовило розвиток специфічних підходів для кожної категорії.

Cloud-based підхід [7] використовує ресурси хмарних платформ для виявлення зловмисного програмного забезпечення. Відсутність доказів переваги одного підходу над іншими пояснюється специфічними перевагами, недоліками та різною ефективністю у конкретних ситуаціях. Наприклад, для відомих типів шкідливого програмного забезпечення евристичний підхід демонструє кращі результати, тоді як для невідомих більш ефективним є підхід з використанням глибинного навчання.

Хоча деякі методи намагаються виявляти нові зловмисні програми до завдання шкоди жертві, жоден з них не досягає високої точності в цьому завданні. Досягнення такої мети є складним завданням, що потребує численних досліджень та створення нових методів і підходів [8].

Виявлення шкідливого програмного забезпечення представляє собою процес дослідження змісту програми з метою визначення її зловмисного або доброякісного характеру. Процес включає три етапи: аналіз роботи програми, виділення ознак та класифікацію.

Аналіз роботи програми спрямований на з'ясування механізмів функціонування програми, впливу на інші процеси та систему загалом, ідентифікацію пошкоджених або вкрадених даних. Існують два основні підходи: статичний та динамічний аналіз коду. Статичний аналіз досліджує хеш-функції, рядки, бібліотеки, імпортовані функції для отримання базового уявлення про поведінку програми до її запуску. Динамічний аналіз передбачає виконання досліджуваної програми у безпечному середовищі-пісочниці, що дозволяє спостерігати за поведінкою шкідливого програмного забезпечення без ризику інфікування системи.

Виділення ознак здійснюється за допомогою методів глибинного аналізу даних - процесу виділення значущої інформації з великих наборів

даних або баз даних. На цьому етапі відбувається обробка зібраних даних для генерації нових змінних, що несуть корисну або раніше невідому інформацію для подальшого аналізу.

Класифікація являє собою процес визначення цільового класу, що може включати простий поділ на зловмисні та доброякісні програми або детальну ідентифікацію типу чи сімейства шкідливого програмного забезпечення залежно від поставленої задачі. На цьому етапі обирається алгоритм для використання [9].

Широкого поширення набули алгоритми машинного навчання [10], що забезпечують методи вірогідної оцінки класу шкідливого програмного забезпечення без явного програмування. Популярними методами стали баєсова мережа, наївний баєсів класифікатор, алгоритм C4.5 для генерації дерев рішень, випадковий ліс, метод k-найближчих сусідів, логістична регресія, метод опорних векторів. Поряд з методами машинного навчання може використовуватися глибинне навчання або комбінація декількох методів одночасно.

1.4.2 Підхід на основі сигнатур

Сигнатура являє собою унікальну ознаку шкідливого програмного забезпечення, аналогічну відбитку пальця для людини. Зазвичай це набір унікальних даних або біт коду, що можуть містити ключову фразу чи команду для ідентифікації.

Підхід на основі сигнатур широко використовується в комерційних антивірусах, забезпечуючи швидке та ефективне виявлення відомого шкідливого програмного забезпечення, проте неефективний для невідомих загроз. Екземпляр шкідливого програмного забезпечення може легко уникнути виявлення за допомогою методів обфускації.

Згенеровані сигнатури зберігаються в базі даних, а визначення класу програми відбувається шляхом порівняння з існуючими підписами.

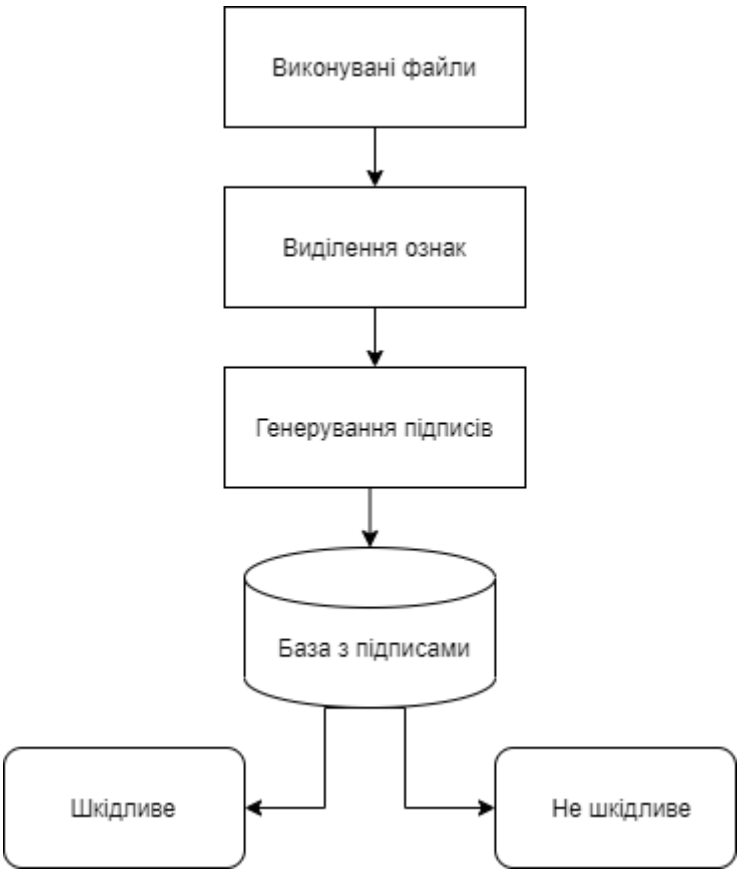


Рисунок 1.3 – Діаграма виконання підходу на основі сигнатур

На рисунку 1.3 представлена діаграма виконання підходу на основі сигнатур.

1.4.3 Підхід на основі поведінкових сигнатур

Цей підхід спостерігає за поведінкою програми під час виконання з використанням спеціальних інструментів моніторингу. Це дозволяє ігнорувати обфускацію коду, оскільки поведінка залишається незмінною або схожою, забезпечуючи виявлення нових видів шкідливого програмного забезпечення.

Недоліком підходу є те, що деякі зловмисні програми не функціонують у захищеному середовищі, що може призвести до неправильної класифікації як доброякісних.

Моніторинг виконання включає системні виклики, зміни файлової

системи, порівняння знімків реєстру, мережеві взаємодії, роботу процесів, бібліотек та драйверів.

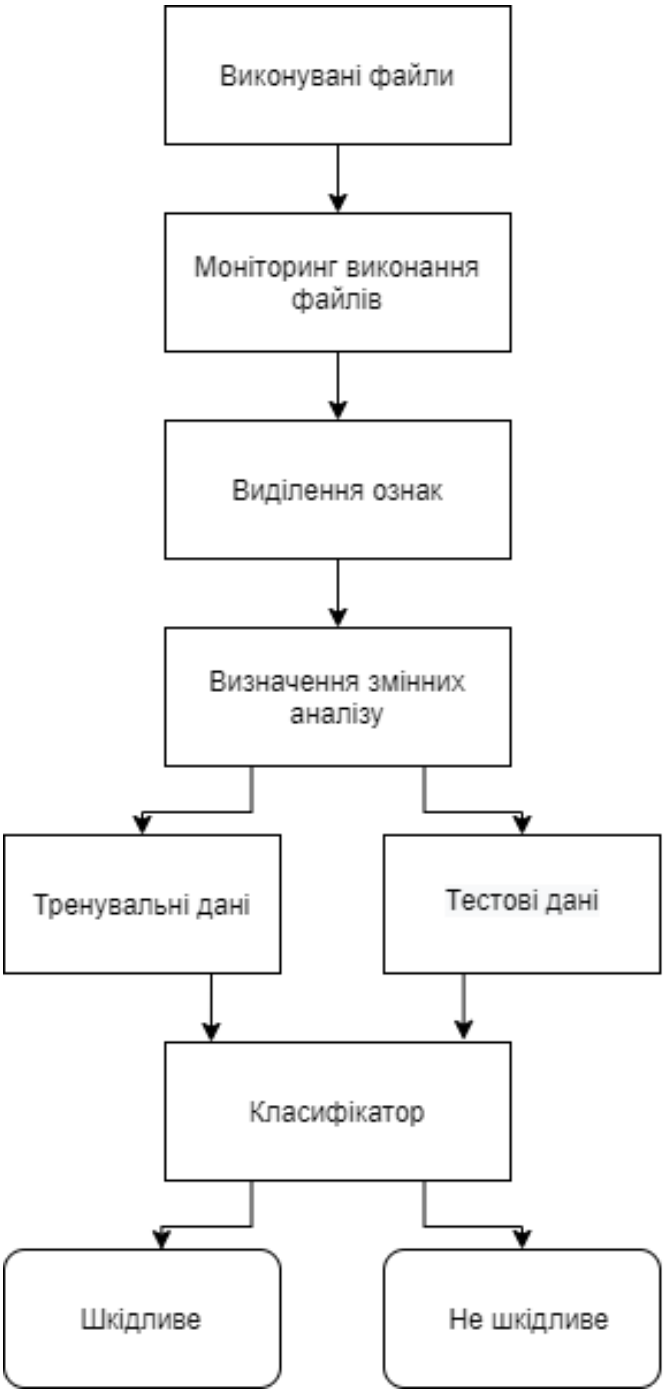


Рисунок 1.4 – Діаграма виконання підходу на основі поведінкових сигнатур

Моніторинг охоплює як активні дії, так і їх відсутність при запуску вірусу, що генерує додаткові змінні для аналізу (рисунок 1.4).

1.4.4 Підхід з використанням глибинного навчання

Глибинне навчання являє собою галузь машинного навчання, що використовує штучні нейронні мережі для навчання на прикладах. Цей підхід широко застосовується для обробки зображень, мовлення та розробки безпілотних транспортних засобів, проте має обмежене застосування в аналізі шкідливого програмного забезпечення, незважаючи на достатню ефективність.

Рисунок 1.5 – Діаграма виконання підходу з використанням глибинного навчання

Недоліком методу є нестійкість до evasion-атак або атак ухиляння, що полягають у додаванні шуму до тестового зразка. У контексті шкідливого програмного забезпечення це може проявлятися у додаванні зайвих API або системних викликів (рисунок 1.5).

1.4.5 Підхід з використанням хмарних обчислень

Хмарні обчислення набули популярності завдяки численним перевагам, включаючи легку доступність, абстрагування від інфраструктури,

різноманітні способи зберігання даних та зменшення витрат, що обумовило їх використання для виявлення шкідливого програмного забезпечення.

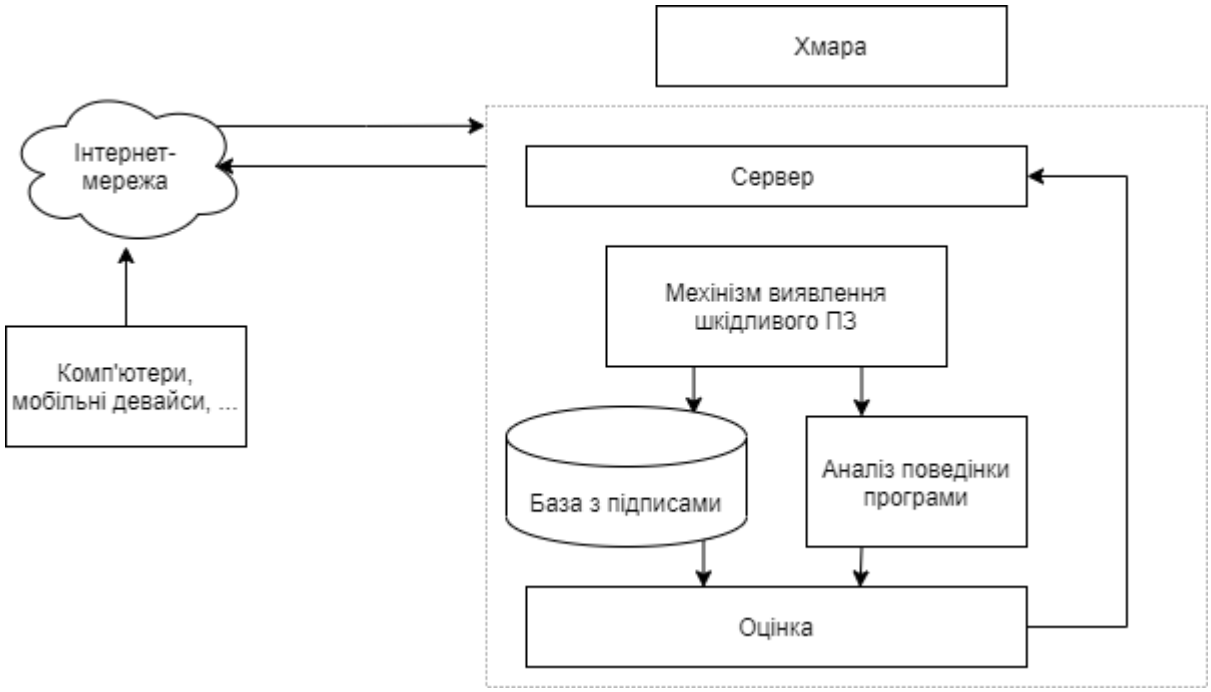


Рисунок 1.6 – Діаграма виконання підходу з використанням хмарних обчислень

Концепція(рисунок 1.6) полягає у функціонуванні хмарного сервера, куди користувач може завантажити файл та отримати звіт про його потенційну інфікованість. Це знижує навантаження на операційну систему користувача та дозволяє використовувати значно більші бази даних для навчання моделей або порівняння, а також забезпечує накопичення бази файлів для майбутніх досліджень.

Незважаючи на переваги, підхід має недоліки: необхідність завантаження файлів з конфіденційною інформацією у хмару, додатковий час передачі даних через мережу, особливо критичний для мобільних та IoT пристроїв, та відсутність моніторингу в реальному часі.

.

2 МЕТОДИ МАШИННОГО НАВЧАННЯ. РОБОТА З ДАНИМИ

2.1 Попередній аналіз даних та обробка

2.1.1 Схема генерації навчальних даних

Враховуючи домінуючу позицію Microsoft Windows на ринку настільних операційних систем з часткою 75,56 відсотків, для дослідження виправдано використовувати виконувані файли саме цієї операційної системи.

Набір даних для дослідження було отримано шляхом застосування динамічного підходу до аналізу. На окремій віртуальній машині було запущено виконувані файли, отримані з публічно доступних ресурсів Malicia-project та VirusTotal, що зберігають екземпляри шкідливого програмного забезпечення.

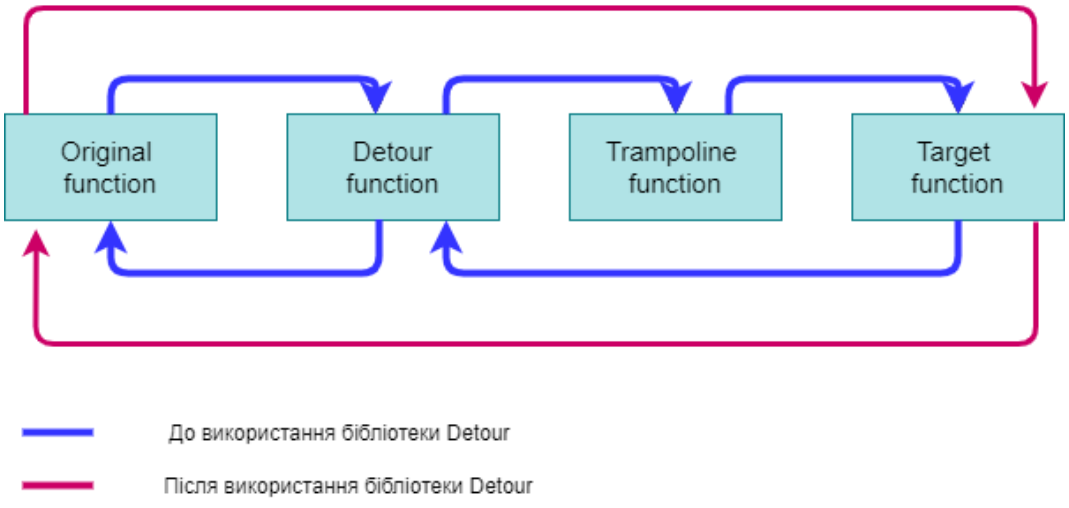


Рисунок 2.1 – Схема перехоплення API-викликів з використання бібліотеки Detours

Для відстеження послідовності API-викликів під час виконання використовується бібліотека Detours, розроблена Microsoft для перехоплення,

моніторингу та контролю бінарних функцій в операційній системі Windows. Призначення бібліотеки полягає у зміні стандартної поведінки операційної системи в процесі перехоплення викликів (рисунок 2.1).

На відміну від звичайного виконання програми, де вихідна функція викликає цільову функцію, з'являються дві проміжні функції: Detour та Trampoline. Detour містить початкові команди вихідної функції та оригінальний код користувача, тоді як Trampoline включає видалені команди цільової функції та безумовний перехід до її завершення. Після виконання цільової функції викликається Detour функція з подальшим виконанням коду користувача, що дозволяє відстежувати та зберігати послідовність API-викликів.

У цій роботі не виконується самостійний запуск шкідливого програмного забезпечення через відсутність додаткової обчислювальної машини та недостатній досвід роботи з таким програмним забезпеченням, що могло призвести до інфікування операційної системи та втрати конфіденційних даних. Для дослідження використовуються заздалегідь згенеровані дані, доступні у відкритому доступі.

2.1.2 Аналіз якості даних

Дані представлені 23147 записами, з яких лише 300 є доброякісними, що свідчить про незбалансованість набору даних. Класифікатори, побудовані на незбалансованих даних, під час практичного використання схильні з більшою імовірністю відносити нові спостереження до класу, представленого більшою кількістю навчальних прикладів.

Експериментальна модель логістичної регресії, побудована на даних з відношенням цільової змінної 11:1, показала точність 92% на тестових даних, проте прогнозовані значення містили лише один клас.

Семплінг представляє собою зміну відношення класів цільової змінної у вибірці для отримання збалансованої навчальної множини. Основні методи

включають oversampling (дублювання міноритарного класу) та undersampling (видалення прикладів мажоритарного класу). У роботі використано oversampling для отримання даних з відношенням 1:1 за допомогою модуля resample бібліотеки Scikit-learn.

2.1.3 Попередня обробка даних

Навчальні дані представляють послідовність API-викликів. Аналіз показав необхідність попередньої обробки через схожість назв із подібною функціональністю. Об'єднання схожих викликів дозволило зменшити кількість унікальних API-викликів з 1165 до 705, що становить зниження майже на 40%.

Аналогічна ситуація спостерігалася з назвами категорій шкідливого програмного забезпечення, де групування цільових категорій зменшило їх кількість з 4874 до 264.

```
asr_ldm.exe,AdjustTokenPrivileges,LookupPrivilegeValueW,
OpenProcessToken,DeviceIoControl,SetupDiEnumDeviceInterfaces,
SetupDiGetDeviceInterfaceDetailW,SetupDiGetClassDevsW,
SetupDiDestroyDeviceInfoList,AsrAddSifEntryW,LoadStringW,
MessageBoxW,CoUninitialize
```

Рисунок 2.2 – Приклад вхідних даних виконуваного файлу asr_ldm.exe

Для побудови ефективної моделі машинного навчання недостатньо лише вибрати правильний тип моделі та параметри. Більшість методів машинного навчання базуються на математичних алгоритмах, що потребують числового формату інформації, а деякі додатково потребують нормалізації.

Обробка включала кодування категорійних змінних шляхом

номінального кодування API-викликів на числа від 1 до 709 та розділення вибірки на тренувальну та тестову у співвідношенні 80:20 з використанням стандартного методу scikit-learn із забезпеченням рівномірного розподілення екземплярів різних категорій (рисунок 2.3).

2.2 Аналіз математичних основ розв'язання задачі

Враховуючи послідовний характер даних, важливим є порядок викликів. Для порівняння послідовностей використовується задача пошуку найдовшої спільної підпослідовності, широко застосована в порівнянні файлів, виявленні плагіату, оптимізації запитів до баз даних та біоінформатиці.

s_1	**a**	**b**	**c**	a	d	c	c
s_2	d	**a**	**a**	d	**b**	**c**	d
s_3	d	c	**a**	**b**	c	**a**	

Рисунок 2.3 – Приклад застосування задачі LCS для трьох послідовностей S_1, S_2, S_3 з алфавітом Σ = {a, b, c, d}. Оптимальне рішення виділено жирним шрифтом

Математична модель (рисунок 2.3) задачі складається з пари (S, Σ), де S - множина з n послідовностей алфавіту Σ, а метою є знаходження послідовності, що є підпослідовністю для всіх елементів множини S. Задача вирішується методами динамічного програмування за поліноміальний час.

При застосуванні до шкідливого програмного забезпечення необхідно враховувати використання зловмисниками методів обфускації, що може призвести до додавання зайвих викликів до послідовностей.

Для задач великої розмірності часто застосовується попереднє вирішення задачі множинного вирівнювання послідовностей, класифікованої

як NP-повна з експоненціальним зростанням простору пошуку зі збільшенням кількості послідовностей.

84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104
243	243	349	117	655	93	93	456	647	647	433	433	190	174	601	634	258	630	503	630	623
243	243	349	117	655	93	93	456	647	647	433	433	190	174	590	590	601	634	258	630	503
243	243	349	117	488	655	93	93	456	647	647	433	433	190	174	601	634	258	630	503	630
243	243	349	117	655	93	93	456	647	647	433	433	190	174	601	634	258	630	503	630	623
243	243	349	117	655	93	93	456	647	647	433	433	190	174	601	634	258	630	503	630	623

Рисунок 2.4 – Множина з п’яти послідовностей API-викликів до вирівнювання

84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104
243	243	349	117	-	655	93	93	456	647	647	433	433	190	174	-	-	601	634	258	630
243	243	349	117	-	655	93	93	456	647	647	433	433	190	174	590	590	601	634	258	630
243	243	349	117	488	655	93	93	456	647	647	433	433	190	174	-	-	601	634	258	630
243	243	349	117	-	655	93	93	456	647	647	433	433	190	174	-	-	601	634	258	630
243	243	349	117	-	655	93	93	456	647	647	433	433	190	174	-	-	601	634	258	630

Рисунок 2.5 – Множина з п’яти послідовностей API-викликів після вирівнювання

Спроба застосування генетичного алгоритму для множинного вирівнювання показала часткові результати для невеликих множин, проте при збільшенні розміру результат погіршувався (рисунки 2.4 та 2.5).

85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104	105	106	107
243	349	117	655	93	93	456	647	647	433	433	190	174	-	-	601	634	258	630	503	630	623	0
243	349	117	655	93	93	456	647	647	433	433	190	174	590	590	601	634	258	630	503	630	623	0
243	349	117	488	655	93	93	456	647	647	433	433	190	174	-	-	601	634	258	630	503	630	623
243	349	117	655	93	93	456	647	647	433	433	190	174	-	-	601	634	258	630	503	630	623	0
243	349	117	655	93	93	456	647	647	433	433	190	174	-	-	601	634	258	630	503	630	623	0
243	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	242
243	349	117	655	93	93	456	647	647	433	433	190	174	-	-	601	634	258	630	503	630	623	0
243	349	117	655	93	93	456	647	647	433	433	190	174	-	-	601	634	258	630	503	630	623	0
243	349	117	655	93	93	456	647	647	433	433	190	174	590	590	601	634	258	630	503	630	623	0
243	349	117	655	93	93	456	647	647	433	433	190	174	590	590	601	634	258	630	503	630	623	0

Рисунок 2.6 – Результат вирівнювання множини з 11 API-викликів

Складність алгоритму є експоненціальною, що потребує значних обчислювальних ресурсів та пам'яті, роблячи його використання неможливим при обмеженнях на ресурси(рисунок 2.6).

2.3 Методи машинного навчання

2.3.1 Формулювання задачі у термінах машинного навчання

Машинне навчання, за визначенням А. Семюеля, являє собою тип штучного інтелекту, що дозволяє програмним додаткам покращувати результати прогнозування без спеціального програмування. Алгоритми використовують історичні дані для прогнозування значень нових зразків, виявляючи та формалізуючи принципи, закладені в даних.

У контексті дослідження алгоритм може прогнозувати категорію шкідливого програмного забезпечення на основі знань про послідовність API-викликів. Математично формалізований набір принципів називається моделлю машинного навчання.

У задачах виявлення шкідливого програмного забезпечення використовуються три підходи: навчання без вчителя для виявлення структури даних без правильних відповідей, навчання з вчителем при наявності даних та правильних відповідей, та методи глибинного навчання.

Навчання з вчителем включає два етапи: вибір та навчання моделі на тренувальних даних та використання навченої моделі для прогнозування нових даних. Математично завдання полягає у знаходженні відображення множини X (дані для навчання) на Y (цільову категорію).

Для дослідження використовуються методи навчання з вчителем: випадковий ліс, дерево рішень, логістична регресія та метод k-найближчих сусідів.

2.3.2 Дерево рішень

Дерево рішень (рисунок 2.7) представляє деревовидну ієрархічну структуру вузлів та гілок, побудовану рекурсивним поділом вхідних даних за певним критерієм. Кожен вузол має один батьківський, а критерій представляє умову типу "якщо ознака > 100".

Процес побудови полягає у послідовному розбитті навчальної множини на підмножини до оголошення всіх кінцевих вузлів листям природним

шляхом або після досягнення умови зупинки.

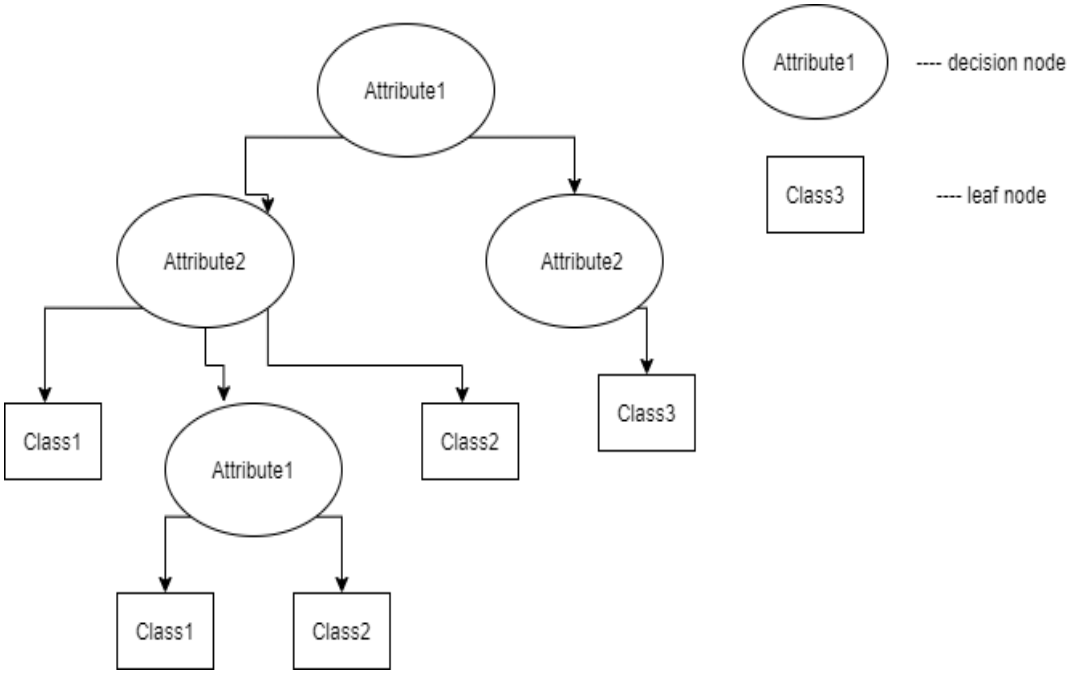


Рисунок 2.7 – Проста модель дерева рішень

Алгоритми побудови відносяться до жадібних, діючи за принципом, що локальні оптимальні рішення призводять до глобального оптимуму. Для ефективної побудови критично важливий вибір правильного атрибуту для розбиття з використанням інформаційного посилення або індексу Gini.

2.3.3 Випадковий ліс

Випадковий ліс являє собою ансамбль декількох дерев рішень, де сукупний результат важливіший за окремі складові. Ансамбль захищає від перенавчання, зменшує кореляцію між деревами та покращує точність моделі, зазвичай забезпечуючи кращі результати ніж окреме дерево рішень.

Недоліком методу є великий розмір моделі, що збільшує час побудови, обчислювальну складність та знижує швидкість прийняття рішень.

2.3.4 Метод k-найближчих сусідів

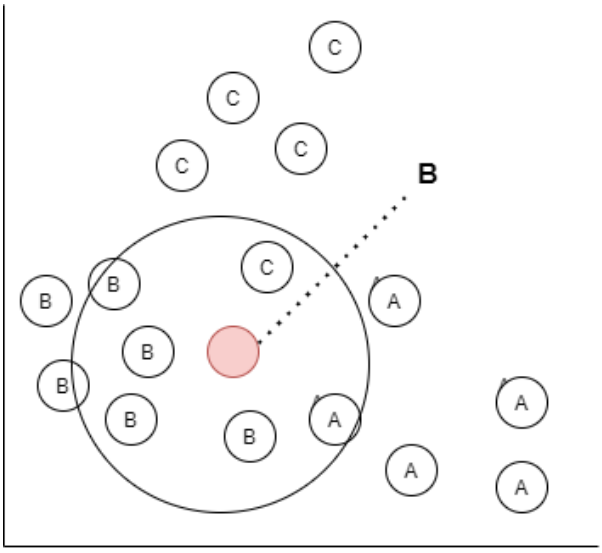


Рисунок 2.8 – Метод k-найближчих сусідів для k = 6

Концепція методу (рисунок 2.8) полягає у присвоєнні невідомому елементу класу, найбільш поширеного серед k найближчих сусідів. Близькість визначається функцією відстані, часто евклідовою, проте вибір залежить від специфіки даних та задачі.

При значних відмінностях у значеннях атрибутів проводиться нормалізація даних для зведення до діапазону 0-1, найчастіше мінімаксна нормалізація, хоча деякі атрибути можуть потребувати переваги над іншими залежно від задачі.

2.3.5 Логістична регресія

Логістична регресія використовується для прогнозування імовірності настання події, часто в бінарній класифікації. У контексті багатокласової класифікації застосовується підхід One-Vs-The-Rest з побудовою n моделей для n категорій та вибором категорії з найвищою імовірністю як прогнозу.

3 ПРАКТИЧНА ЧАСТИНА

Мета роботи полягає у визначенні оптимальних методів представлення та вилучення ознак, найточнішого алгоритму для виявлення сімейств шкідливих програм з мінімальним рівнем помилок та порівнянні з поточною точністю системи оцінок.

3.1 Дані

Для реалізації проекту було зібрано 2140 файлів, переважно через геші з звітів про зараження або зворотну розробку, отримані з сервера VirusTotal. Набір включає 1156 шкідливих файлів дев'яти сімейств (Dridex, Locky, TeslaCrypt, Vawtrak, Zeus, DarkComet, CyberGate, Xtreme, CTB-Locker) та 984 доброякісних файли різних форматів. Використовується виключно шкідливе програмне забезпечення останніх двох років для забезпечення актуальності результатів.

3.2 Cuckoo Sandbox

Дослідження базується на Cuckoo Sandbox - інструменті аналізу шкідливого програмного забезпечення з відкритим кодом, що забезпечує детальний опис поведінки файлів або URL за секунди. Підтримуються різноманітні формати файлів, включаючи виконувані файли Windows, DLL, PDF, документи Microsoft Office, URL, скрипти та інші.

Cuckoo має модульну архітектуру для використання як окремо, так і в інтегрованих рішеннях. Основні компоненти включають хост-машину для управління та гостьові машини для аналізу. Процес передбачає виділення віртуального середовища для нового файлу, його виконання та фіксацію всіх операцій у системі.

3.2.1 Система оцінки

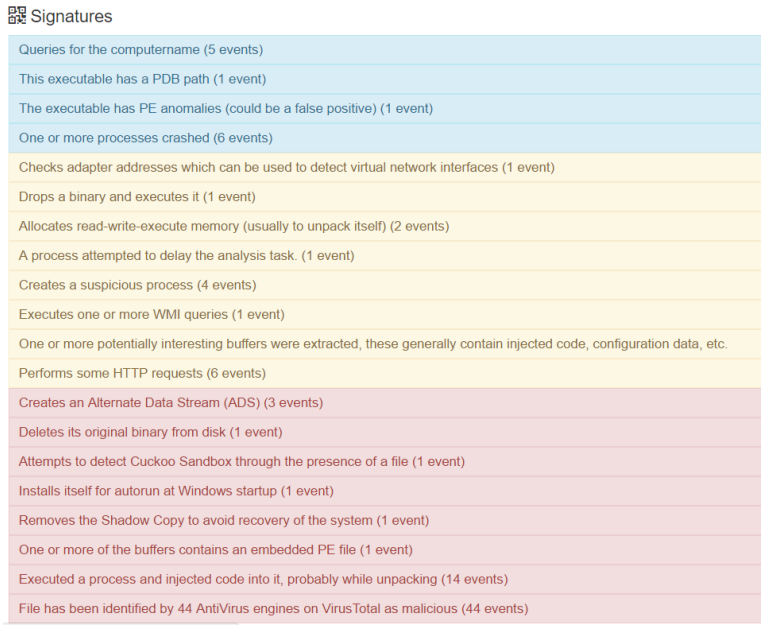


Рисунок 3.1 – Сигнатури Cuckoo

Оцінка Cuckoo Sandbox (рисунок 3.1) індикує рівень зловмисності аналізованого файлу через підрахунок шкідливих дій. Використовуються сигнатури трьох рівнів важкості: низького, середнього та високого. Під час аналізу всі дії зберігаються для подальшої обробки модулями, включаючи модуль сигнатур, що аналізує дані та знаходить відповідні шаблони.

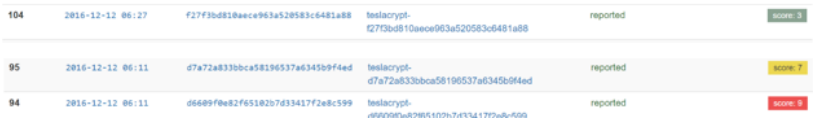


Рисунок 3.2 – Приклад різних оцінок файлів

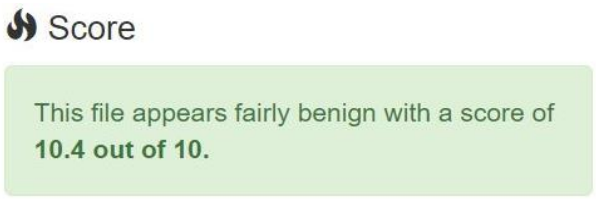


Рисунок 3.3 – Помилки у системі оцінки

При збігу сигнатури лічильник збільшується на відповідну кількість балів, а фінальна оцінка обчислюється діленням на 5.0. Графічний інтерфейс використовує кольорові індикатори: зелений для оцінок 4 і нижче, жовтий для 4-7, червоний для 7-10, проте ця функція перебуває в альфа-тестуванні та має помилки (рисунки 3.3 та 3.4).

3.2.2 Звіти та ознаки

Для застосування алгоритмів машинного навчання необхідно визначити дані для вилучення та методи їх представлення. Деякі роботи використовують властивості рядків або форматів файлів, проте специфічні для форматування функції не є оптимальним рішенням через варіативність форматів аналізованих файлів.

Інші дослідження покладаються на n-грами - перекриваючі підрядки, зібрані ковзним вікном фіксованого розміру. Основна складність байтових n-грамів полягає у надмірно великому наборі, непридатному для прямого застосування методів класифікації. Додатково такий підхід обмежує виявлення поліморфного шкідливого програмного забезпечення.

Через зазначені причини в дослідженні використовується фактична поведінка файлів, що включає файли, ключі реєстру, м'ютекси, процеси, IP-адреси та DNS-запити, виклики API.

Файли містять інформацію про відкриті та створені файли, корисну для прогнозування сімейства, проте в деяких випадках, як програми-шифрувальники, ця інформація може бути недостатньою через переважання зашифрованих файлів над налаштуваннями.

Ключі реєстру зберігають системні налаштування та можуть бути джерелом інформації про системні зміни, спричинені шкідливим програмним забезпеченням.

М'ютекси як унікальні ідентифікатори можуть бути хорошими ідентифікаторами зразків, проте для визначення сімейств у великому

масштабі їх недостатньо через малу кількість створюваних м'ютексів порівняно з розміром набору даних.

Процеси рідко використовуються для ідентифікації сімейства через схожість імен процесів з геш-значеннями зразків.

IP-адреси та DNS-запити можуть точно ідентифікувати командні сервери, проте зловмисники часто змінюють доменні імена або IP-адреси, що знижує надійність такого підходу.

Виклики API представляють широкий опис поведінки зразків, включаючи всі вищезгадані властивості, забезпечуючи численні різноманітні значення та можливість цифрового представлення. Набір ознак визначається кількістю унікальних викликів API та кодів повернення.

3.3 Репрезентація ознак

Враховуючи великий розмір набору ознак, що включає успішні та невдалі виклики API з кодами повернення, необхідно знайти ясний та компактний спосіб представлення. Обраним методом є матриця частот.

3.3.1 Бінарне представлення

$$API_{bin} = \begin{array}{c} \\ S_1 \\ S_2 \\ \vdots \\ S_n \end{array} \begin{array}{c} \begin{array}{cccc} API_1 & API_2 & \dots & API_n \end{array} \\ \begin{bmatrix} 1 & 1 & \cdots & 0 \\ 1 & 0 & \cdots & 1 \\ \vdots & \ddots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{bmatrix} \end{array} \quad (3.1)$$

Найпростіший спосіб представлення через створення матриці з рядками як зразками та стовпцями як викликами API (рисунок 3.1), де 0 означає невдалий виклик, а 1 - успішний. Недоліком є ігнорування кодів повернення та частоти викликів, що знижує точність.

3.3.2 Частотне представлення

Подібна до бінарної структура, проте замість маркування як невдалий/успішний відображає частоту кожного API-виклику. Цей підхід забезпечує більше деталей та кращу точність порівняно з бінарним представленням.

3.3.3 Комбіноване представлення

$$Combination = \begin{array}{c} \\ S_1 \\ S_2 \\ \vdots \\ S_n \end{array} \begin{array}{c} \begin{array}{cccccccccc} Pass_1 & \dots & Pass_n & Fail_1 & \dots & Fail_n & RetC_1 & \dots & RetC_n \end{array} \\ \begin{bmatrix} 23 & \cdots & 3 & 224 & \cdots & 123 & 23 & \cdots & 27 \\ 52 & \cdots & 21 & 224 & \cdots & 57 & 224 & \cdots & 1 \\ \vdots & \ddots & \ddots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 52 & \cdots & 22 & 210 & \cdots & 46 & 72 & \cdots & 111 \end{bmatrix} \end{array} \tag{3.3}$$

Для максимального використання корисних даних оптимальним є об'єднання ознак попередніх методів. Результуюча матриця (рисунок 3.3) визначає частоту невдалих API, успішних API та кодів повернення, забезпечуючи справедливу продуктивність за рахунок значного збільшення кількості ознак.

Метою вибору ознак є видалення неважливих функцій із надмірно великого набору. Після вилучення та комбінованого представлення отримано 70518 ознак, що є неприйнятно великою кількістю для ефективної обробки та точного прогнозування.

Методи вибору ознак включають методи фільтрації (статистична оцінка), обгорткові методи (тестування комбінацій з моделлю) та вбудовані методи (оцінка під час створення моделі).

3.5 Реалізація

Процес включає конфігурацію пісочниці, витяг ознак засобами Python 2.7, вибір ознак та застосування методів машинного навчання засобами R, оцінку результатів.

3.5.1 Конфігурація пісочниці

Для отримання коректних звітів про поведінку важливо правильно налаштувати Cuckoo Sandbox з широким спектром послуг у віртуальних машинах. Використовується гіпервізор Virtualbox з автоматичним створенням машин засобами VMCloack.

Всі віртуальні машини мають стандартизовані характеристики та встановлене програмне забезпечення, включаючи Windows 7 Professional 64-біт без оновлень, Adobe PDF reader, Flash Player, пакети Visual Studio, Java JRE та .NET Framework.

3.5.2 Витяг ознак

Обраний метод виділення ознак представляє комбіновану матрицю успішних та невдалих викликів API з кодами повернення, що витягуються зі звітів пісочниці. Звіти зберігаються локально після обробки та використовуються як вхід до скрипта вилучення ознак для створення CSV-файлу з матрицею.

3.5.3 Вибір функцій

Для видалення надлишкових ознак використовується мова R та пакет Boruta - метод обгортки навколо алгоритму випадкового лісу. Алгоритм створює тіньові копії всіх функцій, навчає класифікатор та призначає ваги на

основі середнього зменшення точності.

Через обмеження пам'яті набір даних розподіляється на підмножини для окремого вибору ознак з подальшим об'єднанням релевантних функцій. Після процедури отримано 306 ознак із покращенням точності KNN приблизно на 1% та скороченням часу прогнозування до трьох секунд.

3.5.4 Застосування методів машинного навчання

Після вилучення та вибору ознак застосовуються методи машинного навчання з використанням відповідних пакетів R: class для K-найближчих сусідів, kernlab для методу опорних векторів, Rweka для дерева рішень, e1071 для наївного баєсового класифікатора, randomForest для випадкових лісів.

4 АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

Точність виявлення вимірюється як відсоток правильно визначених випадків від загальної кількості випадків.

4.1 K-Найближчих сусідів

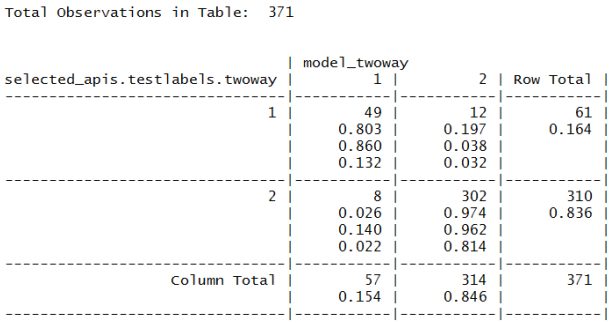
Тестовий набір (таблиця 4.1) складався з 371 зразка. Найкраща загальна точність 87% досягнута при k=1, що може свідчити про чіткі границі між класами при застосуванні алгоритму KNN. Для бінарної класифікації точність становила 94,6% при k=1.

Таблиця 4.1 – Результати к-найближчих сусідів

Клас	Сім'я	Кількість правильно класифікованих зразків	Кількість неправильно класифікованих зразків	Точність	Середня оцінка Cuckoo
1	Нешкідливі файли	49	12	80.3%	1.04
2	Dridex	31	6	83.8%	5.26
3	Locky	22	5	81.5%	6.41
4	TeslaCrypt	43	1	97.7%	6.27
5	Vawtrak	15	3	83.3%	2.66
6	Zeus	30	10	75%	6.46
7	DarkComet	47	2	95.9%	5.15
8	CyberGate	38	0	100%	6.57

Алгоритм забезпечив хорошу точність з рівномірним розподілом класів у багатокласовій класифікації та точні результати навіть при нерівномірному

розподілі в бінарній класифікації.



```
Total Observations in Table:  371

                               | model_twoway
selected_apis.testlabels.twoway |         1 |         2 | Row Total |
--------------------------------|-----------|-----------|-----------|
                              1 |        49 |        12 |        61 |
                                |     0.803 |     0.197 |     0.164 |
                                |     0.860 |     0.038 |           |
                                |     0.132 |     0.032 |           |
--------------------------------|-----------|-----------|-----------|
                              2 |         8 |       302 |       310 |
                                |     0.026 |     0.974 |     0.836 |
                                |     0.140 |     0.962 |           |
                                |     0.022 |     0.814 |           |
--------------------------------|-----------|-----------|-----------|
                   Column Total |        57 |       314 |       371 |
                                |     0.154 |     0.846 |           |
--------------------------------|-----------|-----------|-----------|
```

Рисунок 4.1 – Результат методу knn, бінарна класифікація

На рисунку 4.1 представлено результат методу knn, бінарна класифікація.

Далі результати представлені в таблицях 4.2 та 4.3.

Таблиця 4.2 – Результати к найближчих сусідей

Клас	Кількість правильно класифікованих зразків	Кількість неправильно класифікованих зразків	Точність
Нешкідливі файли	49	12	80.3%
Шкідливі файли	302	8	97.4%

Таблиця 4.3 – Результати методу к-найближчих сусідів

Істинно-позитивні спрацювання	Істинно-негативні спрацювання	Хибно позитивні спрацювання	Хибно негативні спрацювання
302	49	12	8

4.2 Метод опорних векторів

В таблиці 4.4 наведено результати методу опорних векторів.

Таблиця 4.4 – Результати методу опорних векторів

Клас	Сім'я	Кількість правильно класифікованих зразків	Кількість неправильно класифікованих зразків	Точність	Середня оцінка Cuckoo
1	Нешкідливі файли	56	5	91.8%	1.04
2	Dridex	32	5	86.5%	5.26
3	Locky	21	6	77.8%	6.41
4	TeslaCrypt	37	7	84%	6.27
5	Vawtrak	10	8	55.6%	2.66
6	Zeus	31	9	77.5%	6.46
7	DarkComet	48	1	98%	5.15
8	CyberGate	37	1	97.4%	6.57
9	Xtreme	31	3	91.2%	5.15
10	CTB-Locker	22	0	100%	4.76

Загальна точність становила 87,6% для багатокласової та 94,6% для бінарної класифікації.

```
   Cell Contents
|-------------------------|
|                       N |
| Chi-square contribution |
|           N / Row Total |
|           N / Col Total |
|         N / Table Total |
|-------------------------|

Total Observations in Table:  371

```

selected_apis.testlabels.twoway	predictions 1	2	Row Total
1	41	20	61
	174.102	21.631	
	0.672	0.328	0.164
	1.000	0.061	
	0.111	0.054	
2	0	310	310
	34.259	4.256	
	0.000	1.000	0.836
	0.000	0.939	
	0.000	0.836	
Column Total	41	330	371
	0.111	0.889	

Рисунок 4.2 – Результат методу опорних векторів, бінарна класифікація

Результати (рисунок 4.2) майже ідентичні методу K-найближчих сусідів, проте алгоритм не допустив помилкових негативних результатів у бінарній класифікації, що означає відсутність випадків визначення шкідливих програм як доброякісних.

4.3 Дерева рішень

Точність становила 93,3% для багатокласової та 94,6% для бінарної класифікації. Перевагою методу є можливість відстеження рішень, що призвели до прогнозу. Результат для багатокласової класифікації значно кращий за попередні методи.

4.4 Наївний баєсів класифікатор

Точність становила 72,23% для багатокласової та лише 55% для бінарної класифікації. Такі результати неприйнятні для реального застосування через ризик епідемії шкідливих програм. Погана точність

імовірно обумовлена високим зв'язком між ознаками, що суперечить основному припущенню методу про незалежність ознак.

Таблиця 4.7 – Результат методу дерева рішень

Клас	Сім’я	Кількість правильно класифікованих зразків	Кількість неправильно класифікованих зразків	Точність	Середня оцінка Cuckoo
1	Нешкідливі файли	54	7	88.5%	1.04
2	Dridex	37	0	100%	5.26
3	Locky	24	3	88.9%	6.41
4	TeslaCrypt	44	0	100%	6.27
5	Vawtrak	16	2	88.9%	2.66
6	Zeus	33	7	82.5%	6.46
7	DarkComet	47	2	95.9%	5.15
8	CyberGate	38	0	100%	6.57
9	Xtreme	32	2	94.1%	5.15
10	CTB-Locker	21	1	95.5%	4.76

	predictions		
selected_apis.testlabels.twoway	1	2	Row Total
1	46	15	61
	168.727	26.891	
	0.754	0.246	0.164
	0.902	0.047	
	0.124	0.040	
2	5	305	310
	33.201	5.291	
	0.016	0.984	0.836
	0.098	0.953	
	0.013	0.822	
Column Total	51	320	371
	0.137	0.863	

Рисунок 4.3 – Результат дерева рішень, бінарна класифікація

Таблиця 4.8 – Результат методу дерева рішень

Клас	Кількість правильно класифікованих зразків	Кількість неправильно класифікованих зразків	Точність
Нешкідливі файли	46	15	75.4%
Шкідливі файли	305	5	98.4%

Таблиця 4.9 – Результат методу дерева рішень

Істинно-позитивні спрацювання	Істинно-негативні спрацювання	Хибно позитивні спрацювання	Хибно негативні спрацювання
305	46	15	5

Загальна точність дерева рішень була хороша: 93,3% для багатокласової класифікації та 94,6% для бінарної класифікації. Для багатокласової класифікації цей результат є набагато кращим, ніж той, який отриманий за допомогою методу K-найближчих сусідів та методу опорних векторів. Для бінарної класифікації результат однаковий.

4.4 Наївний Байєсів класифікатор

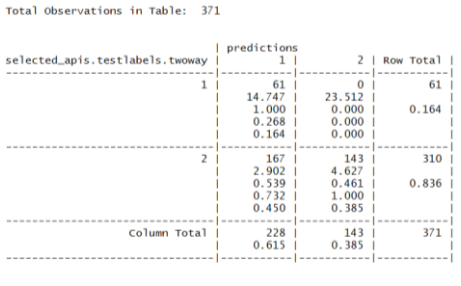
Четвертим алгоритмом, який було протестовано, був наївний байєсів класифікатор. Точність становила 72,23% для багатокласової класифікації і 55% для бінарної класифікації.

Детальні результати, які вказують на точність кожної з родин зловмисного програмного забезпечення, можна знайти в таблиці 4.10.

Таблиця 4.10 – Результат наївного Байєсова класифікатора

Клас	Сім'я	Кількість правильно класифікованих зразків	Кількість неправильно класифікованих зразків	Точність	Середня оцінка Cuckoo
1	Нешкідливі файли	34	27	55.8%	1.04
2	Dridex	1	36	2.7%	5.26
3	Locky	25	2	92.6%	6.41
4	TeslaCrypt	33	11	75%	6.27
5	Vawtrak	8	10	44.4%	2.66
6	Zeus	28	12	70%	6.46
7	DarkComet	49	0	100%	5.15
8	CyberGate	37	1	97.4%	6.57
9	Xtreme	31	3	91.2%	5.15
10	CTB-Locker	22	0	100%	4.76

Для бінарної класифікації алгоритм виконується із низькою точністю. Кількість правильно ідентифікованих доброякісних випадків склала 61, правильно ідентифікованих екземплярів зловмисних програм - 143, неправильно визначених доброякісних випадків - 0, неправильно визначених випадків – 167 (рисунок 4.4, таблиці 4.11-4.12)



Total Observations in Table: 371

selected_apis.testlabels.twoway	predictions 1	2	Row Total
1	61	0	61
	14.747	23.512	
	1.000	0.000	0.164
	0.268	0.000	
	0.164	0.000	
2	167	143	310
	2.902	4.627	
	0.539	0.461	0.836
	0.732	1.000	
	0.450	0.385	
Column Total	228	143	371
	0.615	0.385	

Рисунок 4.4 – Результат методу, бінарна класифікація

Таблиця 4.12 – Результат методу, бінарна класифікація

Істинно-позитивні спрацювання	Істинно-негативні спрацювання	Хибно позитивні спрацювання	Хибно негативні спрацювання
143	61	0	167

Загалом наївний баєсівський класифікатор показав погані результати. Точність багатокласової класифікації становила 72,23%, а бінарної - лише 55%. Цей результат неприйнятний для застосуваннф у реальному світі. У реальному середовищі такий результат може призвести до величезної епідемії шкідливих програм протягом короткого періоду часу.

Швидше за все, така погана точність є результатом високого зв'язку між ознаками. Як ми знаємо, основним недоліком наївного байєсівського класифікатору є те, що кожна ознака розглядається незалежно, хоча в більшості випадків це не може бути правдою. У нашому випадку, швидше за все, певні виклики API залежать один від одного, тобто один виклик не може бути викликаний без іншого. Це найбільш вірогідна причина поганого результату.

4.5 Випадковий ліс

Випадковий ліс – це ансамблевий алгоритм машинного навчання, який використовується для задач класифікації та регресії. Він складається з великої кількості незалежних дерев рішень (decision trees), які працюють разом для підвищення точності прогнозів.

Основні характеристики випадкового лісу:

ансамблевий підхід: об'єднує прогнози багатьох дерев для досягнення більш надійного та стабільного результату;

- випадковість: кожне дерево навчається на випадковій підмножині даних (метод бутстрепу) та використовує випадкову підмножину ознак при

побудові розгалужень;

- уникнення перенавчання: завдяки використанню багатьох дерев і випадковості, модель має меншу ймовірність перенавчання в порівнянні з окремим деревом рішень;

- інтерпретованість: хоча інтерпретувати випадковий ліс складніше, ніж окреме дерево, він може надати важливу інформацію про значущість ознак.

В таблиці 4.13 наведено результати.

Таблиця 4.13 – Результат методу випадковий ліс

Клас	Сім'я	Кількість правильно класифікованих зразків	Кількість неправильно класифікованих зразків	Точність	Середня оцінка Cuckoo
1	Нешкідливі файли	58	3	95%	1.04
2	Dridex	35	2	94.6%	5.26
3	Locky	25	2	92.6%	6.41
4	TeslaCrypt	44	0	100%	6.27
5	Vawtrak	15	3	83.3%	2.66
6	Zeus	35	5	87.5%	6.46
7	DarkComet	49	0	100%	5.15
8	CyberGate	38	0	100%	6.57
9	Xtreme	34	0	100%	5.15
10	CTB-Locker	22	0	100%	4.76

```
Total Observations in Table:  371

                                 | predictions
selected_apis.testlabels.twoway  |         1 |         2 | Row Total |
---------------------------------|-----------|-----------|-----------|
                               1 |        52 |         9 |        61 |
                                 |   204.055 |    35.516 |           |
                                 |     0.852 |     0.148 |     0.164 |
                                 |     0.945 |     0.028 |           |
                                 |     0.140 |     0.024 |           |
---------------------------------|-----------|-----------|-----------|
                               2 |         3 |       307 |       310 |
                                 |    40.153 |     6.989 |           |
                                 |     0.010 |     0.990 |     0.836 |
                                 |     0.055 |     0.972 |           |
                                 |     0.008 |     0.827 |           |
---------------------------------|-----------|-----------|-----------|
                    Column Total |        55 |       316 |       371 |
                                 |     0.148 |     0.852 |           |
---------------------------------|-----------|-----------|-----------|
```

Рисунок 4.5 – Результат методу випадковий ліс, бінарна класифікація

На рисунку 4.5 та в таблиця 4.14, 4.15 результат методу випадковий ліс, бінарна класифікація

Таблиця 4.14 – Результат методу випадковий ліс

Клас	Кількість правильно класифікованих зразків	Кількість неправильно класифікованих зразків	Точність
Нешкідливі файли	52	9	85.2%
Шкідливі файли	307	3	99%

Таблиця 4.15 – Результат методу випадковий ліс

Істинно-позитивні спрацювання	Істинно-негативні спрацювання	Хибно позитивні спрацювання	Хибно негативні спрацювання
307	52	9	3

Алгоритм забезпечив найвищу точність: 95,69% для багатокласової та 96,8% для бінарної класифікації, демонструючи найкращу продуктивність серед усіх тестованих методів.

4.6 Результати з Google Colab

В додатку Б наведено код реалізації в середовищі Google Colab.

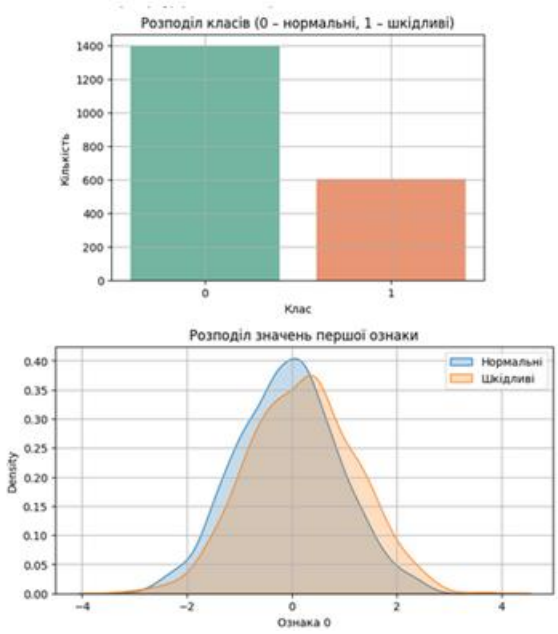


Рисунок 4.6 – Результати роботи

На рисунку 4.6 представлено розподіл класів у наборі даних, де видно, що нормальні зразки значно переважають над шкідливими. На другому рисунку зображено щільність розподілу значень першої ознаки для обох класів, що демонструє подібну форму розподілу, але з деяким зсувом між нормальними та шкідливими зразками. Це свідчить про потенційні відмінності в ознаковому просторі, які можуть бути використані для класифікації.

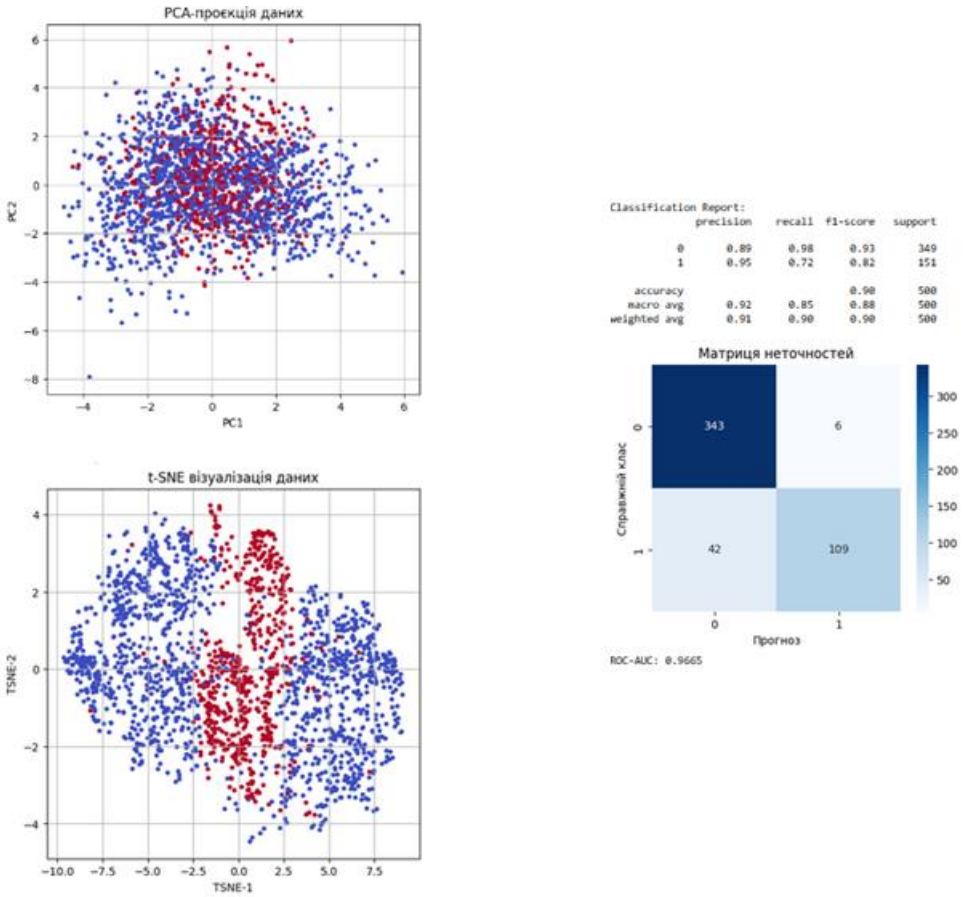


Рисунок 4.7 – Результати роботи

На рисунку 4.7 представлено результати аналізу та класифікації даних. Візуалізації за допомогою PCA і t-SNE демонструють розподіл двох класів у зниженому вимірі, де t-SNE краще розділяє кластери. Класифікаційний звіт свідчить про високу точність моделі, зокрема значення точності 0.92 і ROC-AUC 0.9665. Матриця неточностей відображає незначну кількість помилкових передбачень, що вказує на ефективність застосованого класифікатора.

ВИСНОВКИ

Обрано та застосовано методи вилучення та представлення ознак, а також оцінено алгоритми машинного навчання. Як метод представлення ознак обрано комбіновану матрицю, що визначає частоту успішних та невдалих викликів API разом із кодами повернення.

Ця репрезентація обрана через відображення фактичної поведінки файлу та об'єднання інформації про різноманітні системні зміни, включаючи модифікації реєстру та файлів, на відміну від інших методів.

Порівняльний аналіз існуючих підходів до виявлення шкідливого програмного забезпечення виявив обмеженість традиційних методів на основі сигнатур у протидії сучасним загрозам. Шкідливе програмне забезпечення нового покоління, що використовує методи обфускації, поліморфізму та метаморфізму, потребує принципово нових підходів до виявлення та класифікації.

Експериментальне дослідження п'яти алгоритмів машинного навчання на наборі даних із 2140 файлів продемонструвало значні відмінності в ефективності різних методів. Алгоритм випадкового лісу показав найвищу точність, досягнувши 95,69% для багатокласової класифікації та 96,8% для бінарної класифікації, що перевищує результати інших досліджуваних методів.

Метод дерев рішень забезпечив другий за ефективністю результат з точністю 93,3% для багатокласової класифікації, демонструючи додаткову перевагу у вигляді можливості відстеження логіки прийняття рішень. Методи k-найближчих сусідів та опорних векторів показали близькі результати з точністю близько 87% для багатокласової класифікації.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. R. Jing та Y. Zhang, «A View of Support Vector Machines Algorithm on Classification Problems,» в International Conference on Multimedia Communications, 2010.

2. P. H. Swain та H. Hauska, «The Decision Tree Classifier: Design and Potential,» в IEEE Transactions on Geoscience Electronics, 1977.

3. M. Aquino, «Fake BACS Remittance Emails Delivers Dridex Malware,» 2014. [Електронний ресурс]. Available: https://blog.cyren.com/articles/fake-bacs-remittance-emails-delivers-dridex-malware.html.

4. Kaspersky Lab, "Kaspersky Security Bulletin 2015. Overall statistics for 2015," 2016. [Електронний ресурс]. Available: https://securelist.com /analysis/kaspersky-security-bulletin/73038/kaspersky-security-bulletin-2015-overall-statistics-for-2015/.

5. J. Horton та J. Seberry, «Computer Viruses. An Introduction,» University of Wollongong, Wollongong, 1997.

6. C. Smith, A. Matrawy, S. Chow та B. Abdelaziz, «Computer Worms: Architectures, Evasion Strategies, and Detection Mechanisms,» Journal of Information Assurance and Security, № 4, pp. 69-83, 2009.

7. M. Moffie, W. Cheng, D. Kaeli та Q. Zhao, «Hunting Trojan Horses,» в Proceedings of the 1st Workshop on Architectural and System Support for Improving Software Dependability, San Jose, USA, 2006.

8. E. Chien, «Techniques of Adware and Spyware,» 2005.

9. A. Chuvakin, "An Overview of Unix Rootkits," 2003.

10. W. Lopez, H. Guerra, E. Pena, E. Barrera та J. Sayol, «Keyloggers,» 2013.