НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

До захисту допущено
Завідувач кафедри
_____ **Дмитро ЛАНДЕ**
«_____»_____2024 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системи, технології та математичні методи кібербезпеки»
спеціальності 125 «Кібербезпека»

на тему: гібридний підхід до аналізу шкідливого програмного забезпечення з використанням машинного навчання.

Виконав (-ла): здобувач вищої освіти **IV** курсу, групи ФБ-02
Хандрос Артем Володимирович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник доцент кафердри ІБ к.т.н. Ткач В. М. _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Рецензент доцент кафедри ММЗІ к.ф.-м.н.
Южакова Г.О. _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище, ім'я, по батькові) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Здобувач вищої освіти _____
(підпис)

Київ – 2024 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

Рівень вищої освіти – перший (бакалаврський)
Спеціальність – 125 «Кібербезпека»
Освітньо-професійна програма «Системи, технології та математичні методи кібербезпеки»

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ Дмитро ЛАНДЕ
(підпис)

«___»_____2024р.

ЗАВДАННЯ

на дипломну роботу здобувачу вищої освіти

Хандрос Артем Володимирович

1. Тема роботи: Гібридний підхід до аналізу шкідливого програмного забезпечення з використанням машинного навчання,

керівник роботи Ткач Володимир Миколайович, кандидат наук, доцент кафедри інформаційної безпеки,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «31» травня 2024 р. № 2251-с

2. Термін подання здобувачем вищої освіти роботи 14 червня 2024 р.

3. Вихідні дані до роботи: наукова література про машинне навчання в кібербезпеці, шкідливе програмне забезпечення, методи аналізу ШПЗ, та про методи детектування ШПЗ з допомогою машинного навчання.

4. Зміст роботи: огляд видів шкідливого програмного забезпечення та актуальних методів його аналізу; опис аналізу, збору та процесу підготовки обраних в роботі наборів даних; розробка базових моделей машинного навчання для детектування ШПЗ; аналіз підходу використання ансамблевої стейкінг моделі та її імплементація для реалізації гібридного підходу до детектування.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо):

Презентація до захисту дипломної роботи.

6. Дата видачі завдання: 03.01.2024

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів дипломної роботи	Примітка
1	Отримання завдання та узгодження теми роботи	03.01.2024	Виконано
2	Огляд наукової літератури по темі	03.01.2024 - 14.01.2024	Виконано
3	Дослідження видів шкідливого програмного забезпечення та існуючих методів їх аналізу	15.01.2024 - 04.02.2024	Виконано
4	Підготовка і збір даних	05.02.2024 - 25.02.2024	Виконано
5	Розробка моделі для структурного аналізу (ResNet50)	26.02.2024 - 17.03.2024	Виконано
6	Розробка моделі для динамічного аналізу (Nebula)	18.03.2024 - 07.04.2024	Виконано
7	Розробка класифікатора за PE заголовками з використанням датасету Ember	08.04.2024 - 28.04.2024	Виконано
8	Аналіз Створення ансамблевої мета-моделі	29.04.2024 - 19.05.2024	Виконано
9	Тестування і оцінка ефективності ансамблевої мета-моделі	20.05.2024 - 31.05.2024	Виконано
10	Аналіз результатів та написання висновків	01.06.2024 - 10.06.2024	Виконано

Здобувач вищої освіти _____ Артем ХАНДРОС

(підпис) (**Власне ім'я, ПРІЗВИЩЕ**)

Керівник роботи _____ Володимир ТКАЧ

(підпис) (**Власне ім'я, ПРІЗВИЩЕ**)

РЕФЕРАТ

Обсяг дипломної роботи 44 сторінки, 9 ілюстрацій, 4 додатки і 12 джерел літератури.

Об’єкт дослідження: шкідливе програмне забезпечення (ШПЗ) виду Portable Executable.

Предмет дослідження: гібридні методи аналізу шкідливого програмного забезпечення з використанням машинного навчання.

Мета дослідження: розробка та впровадження гібридного методу аналізу ШПЗ, що базується на машинному навчанні та використанні ансамблевих моделей, для підвищення точності та надійності виявлення шкідливих програм.

Методи дослідження: аналіз літературних джерел, видів ШПЗ, методів виявлення ШПЗ, використовувані моделі машинного навчання; розробка та тренування базових моделей машинного навчання для детектування ШПЗ, впровадження та тестування ансамблевої стейкінг моделі.

Отримані результати: розроблено та протестовано кілька моделей для аналізу ШПЗ, включаючи ResNet50 для структурного аналізу, Nebula для динамічного аналізу та XGBoost для статичного аналізу. Запропоновано та реалізовано ансамблеву мета-модель, що поєднує результати трьох базових моделей, забезпечуючи високу точність і надійність виявлення ШПЗ.

Ключові слова: машинне навчання, шкідливе програмне забезпечення, алгоритми машинного навчання.

ABSTRACT

The volume of the thesis is 44 pages, 9 illustrations, 4 appendices, and 12 literature sources.

Object of research: portable Executable (PE) type malware.

Subject of research: hybrid methods for analyzing malware using machine learning.

Purpose of research: development and implementation of a hybrid malware analysis method based on machine learning and ensemble models to improve the accuracy and reliability of malware detection.

Research methods: literature review on types of malware and detection methods; analysis of machine learning models used for malware detection; development and training of basic machine learning models for malware detection; implementation and testing of an ensemble stacking model.

Obtained results: several models were developed and tested for malware analysis, including ResNet50 for structural analysis, Nebula for dynamic analysis, and XGBoost for static analysis. An ensemble meta-model combining the results of the three base models was proposed and implemented, ensuring high accuracy and reliability in malware detection.

Keywords: machine learning, malware, machine learning algorithms.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

- **ML (Machine Learning)** – машинне навчання.
- **TP (True Positive)** – справжнє позитивне спрацьовування.
- **FP (False Positive)** – хибно позитивне спрацьовування.
- **TN (True Negative)** – справжнє негативне спрацьовування.
- **FN (False Negative)** – хибно негативне спрацьовування.
- **RF (Random Forest)** – ансамблевий метод машинного навчання, який використовує множину дерев рішень.
- **XGBoost** – алгоритм градієнтного бустингу, що використовується для класифікації та регресії.
- **LightGBM** – градієнтний бустинг на основі дерев рішень.
- **CatBoost** – алгоритм градієнтного бустингу.
- **KNN (K-Nearest Neighbors)** – метод класифікації та регресії, який використовує близькість до найближчих сусідів для прогнозування міток.
- **ШПЗ (Шкідливе Програмне Забезпечення)** – програмне забезпечення, яке має на меті шкодити комп'ютерам, мережам або користувачам.
- **MLP (Multilayer Perceptron)** – багатошаровий перцептрон, нейронна мережа з кількома прихованими шарами.
- **ResNet (Residual Network)** – архітектура глибоких нейронних мереж, яка використовує залишкові зв'язки для полегшення тренування дуже глибоких мереж.
- **CNN (Convolutional Neural Network)** – згорткова нейронна мережа, що використовується переважно для обробки зображень і відео.
- **ROC-AUC (Receiver Operating Characteristic - Area Under Curve)** – показник якості класифікатора, що відображає площу під кривою ROC, яка

графічно представляє співвідношення між чутливістю та специфічністю при різних порогах класифікації.

- **ROC (Receiver Operating Characteristic)** – характеристична крива роботи, що є графічним зображенням співвідношення між чутливістю (True Positive Rate) та специфічністю (1 - False Positive Rate) класифікатора.

ВСТУП

Шкідливе програмне забезпечення (ШПЗ) продовжує бути однією з найбільших загроз для сучасних комп'ютерних систем та мереж. З розвитком технологій та зростанням кількості підключених до мережі пристроїв, шкідливі програми стають все більш складними та витонченими, що ускладнює їхнє виявлення та нейтралізацію. Кіберзлочинці використовують ШПЗ для викрадення конфіденційних даних, порушення роботи інформаційних систем, а також для здійснення фінансових махінацій.

У зв'язку з цим, питання ефективного виявлення та аналізу ШПЗ є надзвичайно актуальним. Традиційні методи аналізу, такі як статичний та динамічний аналіз, мають свої обмеження та недоліки. Статичний аналіз дозволяє швидко оцінити структуру програми без її виконання, однак він не завжди може виявити складні та обфусковані зразки ШПЗ. Динамічний аналіз, в свою чергу, передбачає виконання програми в ізольованому середовищі з метою спостереження за її поведінкою, але цей метод є ресурсоємним та може бути обійдено спеціально розробленим ШПЗ, що здатне розпізнавати середовище пісочниці.

Для вирішення цих проблем у даній роботі пропонується гібридний підхід до аналізу ШПЗ, який поєднує переваги статичного, динамічного та структурного аналізу. Використання ансамблевої мета-моделі дозволяє інтегрувати результати різних методів, що підвищує точність та надійність виявлення шкідливих програм.

Метою даної роботи є розробка та впровадження гібридного методу аналізу ШПЗ, що базується на машинному навчанні та використанні ансамблевих моделей. Для досягнення цієї мети було проведено серію експериментів з використанням датасетів PE Malware Machine Learning Dataset та Ember, а також інструментів Speakeasy та Mallok для емулювання поведінки та створення зображень ШПЗ.

Основними завданнями дослідження є:

1. Вивчення існуючих методів аналізу шкідливого програмного забезпечення.
2. Розробка моделей на основі ResNet50 для структурного аналізу, Nebula для динамічного аналізу та EMBER для статичного аналізу.
3. Інтеграція результатів базових моделей у ансамблеву мета-модель для реалізації гібридного підходу.
4. Оцінка ефективності запропонованого методу на реальних датасетах.

Наукова новизна роботи полягає у розробці комплексного підходу до аналізу ШПЗ, який поєднує різні методи машинного навчання та дозволяє досягти високої точності виявлення шкідливих програм. Запропонований гібридний метод може бути використаний у системах виявлення вторгнень та інших засобах кібербезпеки для забезпечення надійного захисту інформаційних систем.

1 ОГЛЯД ВИДІВ ШКІДЛИВОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ІСНУЮЧИХ МЕТОДІВ ЇХ АНАЛІЗУ

Шкідливе програмне забезпечення (ШПЗ) є однією з найбільших загроз для сучасної кібербезпеки. Сучасні шкідливі програми постійно вдосконалюються, що ускладнює їх виявлення та нейтралізацію. У цьому розділі ми розглянемо основні види ШПЗ та відповідні методи їх аналізу, які використовуються для ефективного виявлення та нейтралізації таких загроз.

1.1 Види шкідливого програмного забезпечення

- Віруси
  - Віруси прикріплюються до інших програм або файлів і розповсюджуються при їх виконанні. Вони можуть пошкоджувати файли, змінювати налаштування системи або викрадати дані.
  - За даними звітів з кібербезпеки, віруси становлять близько 10% всіх виявлених загроз.
- Черв'яки
  - Черв'яки – це самостійні програми, які розповсюджуються через мережі без участі користувача. Вони можуть викликати перевантаження мережі та систем, а також поширювати інші види ШПЗ.
  - Черв'яки становлять близько 15% від загальної кількості виявлених шкідливих програм.
- Трояни
  - Троянські програми виглядають як легітимне програмне забезпечення, але містять шкідливий код. Вони можуть викрадати дані, створювати бекдори або виконувати інші небажані дії.

- Трояни є найбільш поширеним типом ШПЗ, складаючи приблизно 50% від усіх загроз.

- Шпигунське програмне забезпечення (Spyware)
  - Spyware збирає інформацію про користувача без його відома та передає її зловмисникам. Це може включати паролі, історію переглядів, особисті дані тощо.
  - Spyware становить близько 20% від усіх виявлених шкідливих програм.
- Руткіти
  - Руткіти – це програми, які приховують свою присутність та дії інших шкідливих програм на комп'ютері, надаючи зловмисникам привілейований доступ до системи.
  - Руткіти менш поширені, складаючи приблизно 5% від усіх загроз.

1.2 Існуючі методи аналізу шкідливого програмного забезпечення

1.2.1 Статичний аналіз

Статичний аналіз включає вивчення коду ШПЗ без його виконання. Це дозволяє швидко отримати інформацію про структуру та функціональність програми.

- Методи:
  - Декомпіляція: відновлення вихідного коду з бінарного файлу.
  - Сигнатурний аналіз: використання баз даних сигнатур відомих шкідливих програм.
  - Аналіз коду: пошук підозрілих функцій та структурних характеристик.
- Переваги: швидкість, безпека, розуміння логіки програми.
- Недоліки: обфускація, обмежена здатність виявлення нових загроз.

1.2.2 Динамічний аналіз

Динамічний аналіз передбачає виконання ШПЗ в ізольованому середовищі з метою спостереження за його поведінкою.

- Методи:
  - Пісочниці (sandboxing): ізольоване середовище для виконання шкідливих програм.
  - Моніторинг системних викликів: відстеження всіх системних викликів, які здійснює ШПЗ під час виконання.
  - Трасування виконання (execution tracing): запис послідовності команд, які виконує ШПЗ.
- Переваги: виявлення обфускаованих загроз, аналіз реальної поведінки програми.
- Недоліки: ресурсоємність, час виконання, можливість виявлення пісочниці шкідливими програмами.

1.2.3 Гібридний аналіз

Гібридний аналіз поєднує переваги статичного та динамічного методів, дозволяючи отримати більш повну інформацію про ШПЗ.

- Методи:
  - Поєднання статичних та динамічних ознак: використання як статичних, так і динамічних характеристик для аналізу.
  - Інтеграція результатів: об'єднання результатів статичного та динамічного аналізу.
  - Автоматизація аналізу: використання автоматизованих інструментів для обох видів аналізу
- Переваги: комплексність, підвищена точність, гнучкість
- Недоліки: складність впровадження, високі вимоги до ресурсів.

Висновки до розділу 1

Аналіз шкідливого програмного забезпечення є комплексним процесом, що вимагає використання різних методів для досягнення максимального рівня захисту. Види ШПЗ, такі як віруси, черв'яки, трояни, шпигунське програмне забезпечення та руткіти, потребують застосування різноманітних підходів для їх виявлення та нейтралізації. Статичний аналіз надає швидкий та безпечний спосіб отримання інформації про структуру та функціональність програм, тоді як динамічний аналіз дозволяє спостерігати за реальною поведінкою ШПЗ. Гібридний підхід, що поєднує ці методи, дозволяє досягти високої точності та ефективності виявлення шкідливих програм.

2 ПІДГОТОВКА І ЗБІР ДАНИХ

Ефективний аналіз шкідливого програмного забезпечення вимагає якісного збору та підготовки даних. Для навчання моделей машинного навчання необхідно використовувати відповідні датасети та методи емулювання поведінки ШПЗ. У цьому розділі буде описано процес підготовки та збору даних для навчання моделей, які використовуються в гібридному підході до аналізу ШПЗ.

2.1 Датасет для навчання моделі Nebula

Модель Nebula була обрана для динамічного аналізу шкідливого програмного забезпечення. Динамічний аналіз дозволяє спостерігати за поведінкою шкідливих програм під час їх виконання, що є ключовим для виявлення складних загроз, які можуть бути не помічені під час статичного аналізу. Для навчання моделі Nebula використовувався датасет «PE Malware Machine Learning Dataset», який містить різноманітні зразки ШПЗ у форматі Portable Executable (PE).

«PE Malware Machine Learning Dataset» є набором даних для досліджень у сфері кібербезпеки та аналізу шкідливих застосунків. Він містить великий набір зразків шкідливого програмного забезпечення у форматі PE, який є форматом для виконуваних файлів Windows. Цей формат широко використовується як для легітимного програмного забезпечення, так і для шкідливих програм.

Датасет був зібраний з різних джерел, включаючи антивірусні компанії, дослідницькі лабораторії та спільноти з обміну зразками ШПЗ. Це забезпечує різноманітність і повноту даних, що необхідні для ефективного навчання моделей машинного навчання. Датасет охоплює різноманітні види шкідливого програмного забезпечення, включаючи віруси, трояни, черв'яки, руткіти та інші загрози.

Кожен зразок містить інформацію про структуру PE файлу, включаючи заголовки, секції, імпорти, експортні таблиці та інші характеристики, які можуть бути використані для аналізу. Також надаються метадані, такі як час створення, розмір файлу, кількість детектувань антивірусами та інші, що можуть бути корисними для класифікації.

Використання цього датасету дає можливість створювати комплексні класифікатори, які здатні визначати шкідливі програми на як на основі їхніх PE хедерів так і на основі результатів виконання у ізольованому середовищі. Аналіз PE файлів дозволяє виявляти специфічні патерни, які характерні для різних видів ШПЗ, в той час як динамічний аналіз дозволяє тренувати моделі на високообфускованих зразках. Це важливо, оскільки багато сучасних шкідливих програм використовують методи обфускації для приховування своєї справжньої природи, що ускладнює їхнє виявлення за допомогою статичного аналізу.

Набір даних надає широкі можливості для аналізу та класифікації ШПЗ за допомогою машинного навчання. Він дозволяє моделі Nebula навчитится розпізнавати складні патерни поведінки, що характерні для шкідливих програм, і адаптуватися до нових загроз. Це забезпечує високу точність і надійність у виявленні шкідливого програмного забезпечення.

Робота з великим датасетом вимагає значних обчислювальних ресурсів для обробки та аналізу. Однак, завдяки великому об’єму і різноманітності зразків, моделі можуть ефективно навчатися і донавчатися.

2.2 Емулювання поведінки ШПЗ за допомогою Speakeasy

Для динамічного аналізу поведінки шкідливого програмного забезпечення використовувалася пісочниця Speakeasy. Це спеціалізоване програмне забезпечення, яке дозволяє виконання програм у безпечному та ізольованому середовищі, забезпечуючи можливість спостерігати за їхньою поведінкою без ризику для основної системи.

Функціонал Speakeasy включає у себе відстеження системних викликів, взаємодію з файлами та мережею, зміни у реєстрі, доступ до пам'яті та інші дії, що можуть бути індикаторами шкідливої поведінки. Такий підхід забезпечує глибокий аналіз шкідливих програм і дозволяє отримати повну картину їхньої активності.

Один з ключових аспектів використання Speakeasy полягає у можливості аналізувати обфусковані та приховані загрози, які можуть бути не виявлені за допомогою статичного аналізу. Динамічний аналіз дозволяє побачити реальні дії шкідливих програм під час їх виконання, що забезпечує більш точне виявлення і класифікацію загроз. Важливою особливістю є те, що Speakeasy може розпізнавати спроби шкідливого програмного забезпечення уникнути аналізу, наприклад, за допомогою технік анти-пісочниці.

Під час емулювання система отримує велику кількість даних, які збираються у вигляді звіту та використовуються для тренування моделей машинного навчання.. Ці дані також можуть бути використані для створення сигнатур, які покращують точність виявлення шкідливих програм у майбутньому та дозволяють економити ресурси моделі.

У дослідженні Speakeasy використовувався для емулювання виконання зразків ШПЗ з метою збирання даних для динамічного аналізу. Ці дані включають логування всіх системних викликів, взаємодію з файловою системою та мережею, а також зміни в системних налаштуваннях. Зібрані дані були використані для тренування моделі Nebula, яка спеціалізується на динамічному аналізі поведінки шкідливого програмного забезпечення.

2.3 Класифікація за PE заголовками з використанням датасету Ember

Для статичного аналізу шкідливого програмного забезпечення (ШПЗ) та класифікації на основі PE заголовків було використано датасет Ember. Цей датасет містить великий обсяг шкідливого і легітимного програмного забезпечення.

Датасет Ember доступний на GitHub і містить понад мільйон зразків, розділених на тренувальний, валідаційний та тестовий набори. Ember є одним з найбільш всеохоплюючих датасетів для аналізу PE файлів, що робить його надзвичайно корисним для навчання моделей машинного навчання у задачах кібербезпеки.

Структура датасету включає різноманітні характеристики, витягнуті з PE хедерів за допомогою проекту LIEF. До цих характеристик належать розміри секцій, імпорти функцій, характеристики коду та даних, а також різні метадані, що дозволяють детально аналізувати файли та виявляти потенційні загрози. Кожен зразок у датасеті має понад 2000 ознак, що забезпечує глибокий аналіз структури програми.

Цей набір даних зазвичай використовується для навчання класифікаторів, що визначають шкідливі програми на основі статичного аналізу PE хедерів.. Навчання на великій кількості зразків дозволяє моделі краще узагальнювати знання та виявляти як відомі, так і нові види ШПЗ.

Перевага використання Ember полягає у його великому обсязі та різноманітності зразків, що включають як шкідливе, так і легітимне програмне забезпечення. Це дозволяє створювати моделі, які не лише мають високу точність виявлення загроз, але й здатні мінімізувати кількість хибнопозитивних спрацьовувань. Велика кількість ознак також сприяє детальному аналізу файлів, що дозволяє виявляти складні та обфусковані зразки ШПЗ без використання методів динамічного аналізу. Використання Ember у цій роботі дозволило досягти високої точності у виявленні шкідливого програмного забезпечення, використовуючи статичний аналіз PE хедерів. Це підвищило ефективність та надійність розроблених моделей, забезпечуючи їх здатність до точного виявлення як відомих, так і нових загроз.

2.4 Створення зображень для моделі на основі ResNet50 з використанням Mallok

Для структурного аналізу шкідливого програмного забезпечення (ШПЗ) з використанням моделі на основі ResNet50 застосовувався інструмент Mallok. Цей інструмент дозволяє перетворювати зразки ШПЗ у форматі Portable Executable (PE) на зображення, які потім використовуються для навчання моделей глибокого навчання, таких як ResNet50.

Mallok надає можливість ефективно візуалізувати бінарні дані зразків ШПЗ, перетворюючи їх у зображення. Цей процес включає кілька важливих етапів:

1. Визначення розмірів зображень: Спочатку створюється список можливих розмірів зображень, які можуть бути використані для перетворення бінарних файлів у квадратні зображення. Це дозволяє забезпечити оптимальний розмір зображення без перевищення допустимого обсягу даних.

2. Перетворення бінарних файлів у зображення: Зразки ШПЗ у форматі PE завантажуються з вхідної папки, після чого їхні байти перетворюються на зображення. Це включає вибір найближчого допустимого розміру зображення, що забезпечує його квадратну форму.

3. Збереження зображень: Отримані зображення зберігаються у вихідній папці у форматі PNG. Також створюється журнал, що містить інформацію про характеристики кожного зображення, такі як розмір файлу, метод інтерполяції та роздільна здатність.

4. Додаткове збереження об'єктів: За необхідності, зображення можуть бути збережені як об'єкти matplotlib для подальшого використання або аналізу.

Mallok дозволяє використовувати різні методи інтерполяції та налаштовувати роздільну здатність зображень, що забезпечує гнучкість у створенні візуальних даних для тренування моделей глибокого навчання. Висока якість створених

зображень сприяє покращенню точності та ефективності моделі ResNet50 у задачах виявлення та класифікації шкідливого програмного забезпечення. Використання Mallok для створення зображень з датасету PE Malware Machine Learning Dataset дозволило ефективно підготувати дані для навчання моделі, що забезпечило високу точність у структурному аналізі ШПЗ.

2.5 Процес підготовки даних

Підготовка даних для моделі Nebula:

1. Завантаження датасету PE Malware Machine Learning Dataset.
2. Використання Speakeasy для емулювання виконання зразків ШПЗ.
3. Збирання даних про поведінку ШПЗ під час виконання.

Підготовка даних для класифікації за PE хідерами:

1. Завантаження датасету Ember.
2. Аналіз та попередня обробка PE хедерів зразків ШПЗ.
3. Розподіл даних на тренувальні та тестові набори.

Підготовка даних для моделі ResNet50

1. Завантаження датасету PE Malware Machine Learning Dataset.
2. Використання Mallok для створення зображень з зразків ШПЗ.
3. Попередня обробка зображень та їх розмітка для навчання моделі ResNet50.

2.6 Метрики оцінки моделей

Для оцінки моделей машинного навчання, що використовується для аналізу шкідливого програмного забезпечення, застосовуються такі основні метрики:

- Точність (Accuracy)

$$Accuracy = \frac{TP\ +\ TN}{TP + TN + FP +\ FN} \quad 2.1$$

де TP – істотно позитивні детектування, TN – істотно негативні детектування, FP – хибно негативні детектування, TP – хибно позитивні детектування

- Повнота (Recall):

$$Recall = \frac{TP}{TP +\ FN} \quad 2.2$$

- Влучність (Precision):

$$Precision = \frac{TP}{TP +\ FP} \quad 2.3$$

- F1-міра (гармонійне середнє між точністю (precision) і повнотою (recall):

$$F1 = 2\ \times \frac{Precision\ \times\ Recall}{Precision\ +\ Recall} \quad 2.4$$

- Площа під кривою ROC (ROC-AUC):

Метрика вимірює площу під кривою (AUC) характеристики роботи приймача (ROC), що графічно представляє співвідношення між чутливістю (True Positive Rate) та специфічністю (розраховується як *1- False Positive Rate*) при різних порогах класифікації. Значення AUC варіюється від 0 до 1, де 1 означає ідеальний класифікатор.

- Хибнопозитивна частота (False Positive Rate, FPR):

$$FPR = \frac{FP}{FP + TN} \quad 2.5$$

2.7 Схема підготовки даних для аналізу шкідливого програмного забезпечення

На основі процесів підготовки даних, описаних вище, була розроблена схема, що ілюструє етапи обробки датасетів для аналізу шкідливого програмного забезпечення:

1. Вибір датасетів для обробки.

Для реалізації гібридного підходу було обрано кілька датасетів, - PE Malware Machine Learning Dataset, Ember Dataset.

2. Видалення унікальних міток

Першим кроком після вибору датасетів було видалення унікальних міток (міток, які з'являються лише один раз), які можуть створювати шум і спотворювати результати моделей.

3. Виправлення типів даних

Датасети містять різні типи даних (числові, категоріальні тощо). Було виконано виправлення типів даних, щоб кожна колонка відповідала правильному типу. Це включає перетворення числових даних з текстового формату у числовий і навпаки для категоріальних даних.

4. Обробка значень NaN, Infinity та -Infinity

У деяких записах PE Malware Machine Learning Dataset були відсутні значення (NaN). Було виконано обробку таких значень:

- NaN: дані були відстуні в супроводжуючому файлі для деяких зразків, в залежності від кількості детектувань антивірусами вони були замінені на 0 та 1.

5. Генерація бінарних міток

Для задачі класифікації ШПЗ було створено бінарні мітки, де 1 позначає шкідливе ПЗ, а 0 – легітимне ПЗ.

6. Видалення колонок з постійними значеннями

Колонки з постійними значеннями не несуть інформаційної цінності для моделей машинного навчання. Було видалено такі колонки з датасету.

10. Нормалізація ознак

Деякі ознаки в датасеті мали різні діапазони значень, що впливало на ефективність моделей. Було виконано нормалізацію ознак, щоб привести їх до спільного діапазону значень, бінарна [0, 1] в нашому випадку

11. Розділення на тренувальний та тестовий набори

Останнім кроком було розділення датасету на тренувальний та тестовий набори. Зазвичай, 70-80% даних використовується для тренування, а решта – для тестування. Це дозволяє оцінити ефективність моделей на нових даних.

Висновки до розділу 2

У цьому розділі було розглянуто процес підготовки та збору даних для навчання моделей, що використовуються у гібридному підході до аналізу шкідливого програмного забезпечення. Використання різних датасетів та інструментів, таких як PE Malware Machine Learning Dataset, Ember, Speakeasy та Mallok, дозволило створити ефективні моделі для динамічного, статичного та структурного аналізу ШПЗ. Підготовка якісних даних є критично важливим етапом для досягнення високої точності та ефективності виявлення шкідливих програм.

3 РОЗРОБКА МОДЕЛЕЙ ДЛЯ АНАЛІЗУ ШКІДЛИВОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ЗАСТОСУВАННЯ АНСАМБЛЕВОЇ МОДЕЛІ ДЛЯ ГІБРИДНОГО ПІДХОДУ

У цьому розділі описано процес розробки моделей для аналізу шкідливого програмного забезпечення (ШПЗ) та результати їх індивідуального тестування. Кожна з моделей була створена з використанням різних методів аналізу та оцінена за допомогою метрик, описаних у розділі 2.6.

3.1 Розробка моделі на основі ResNet50 для структурного аналізу

Модель ResNet50 була обрана для структурного аналізу ШПЗ, оскільки ця архітектура добре підходить для класифікації зображень. Використовуючи інструмент Mallok, зразки ШПЗ з датасету PE Malware Machine Learning Dataset були перетворені на зображення.

3.1.1 Параметри моделі ResNet50

- Архітектура: ResNet50
- Вхідні дані: Зображення зразків ШПЗ
- Метрика оцінки: Точність (accuracy), повнота (recall), точність (precision), F1-міра

В результаті тренування моделі ResNet50 на підготовлених зображеннях, було отримано наступні результати:

- Точність: 86%
- Повнота: 84%
- Влучність: 85%

- F1-міра: 84.5%

3.2 Розробка моделі Nebula для динамічного аналізу

Модель Nebula була використана для динамічного аналізу поведінки шкідливого програмного забезпечення (ШПЗ). Використовуючи пісочницю Speakeasy, було емульовано виконання зразків ШПЗ та зібрано дані про їхню поведінку, включаючи логи системних викликів та інші показники активності програм.

Модель Nebula використовувала ці логи як вхідні дані для аналізу поведінки ШПЗ. Вхідні дані включали інформацію про системні виклики, взаємодію з файловою системою, мережеву активність та інші аспекти виконання програм. Завдяки цьому модель мала змогу детально аналізувати поведінкові патерни програмного забезпечення і виявляти аномалії, що вказують на шкідливу активність.

Після тренування моделі на великому обсязі зібраних даних було отримано наступні результати: точність моделі склала 87%, повнота – 87.8%, влучність – 88.6%, а F1-міра - 88.2%.

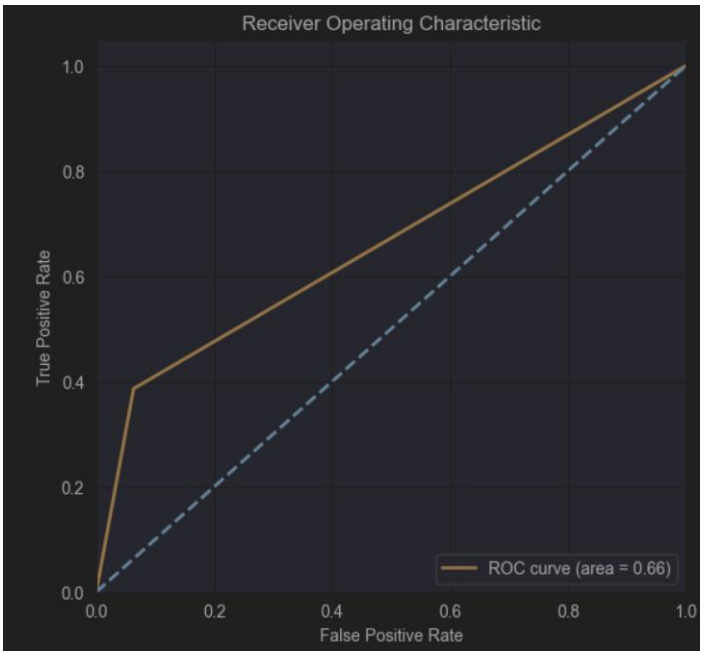

Рисунок 3.1 – Крива ROC для моделі Nebula

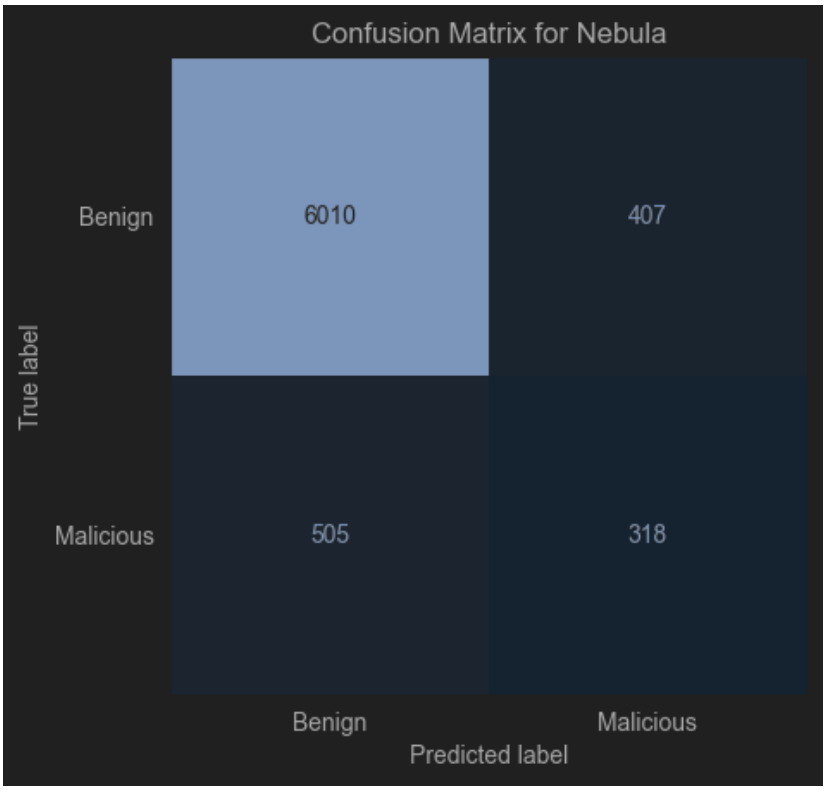

Рисунок 3.2 – Матриця невідповідностей для моделі Nebula

3.3 Розробка класифікатора за PE заголовками з використанням датасету Ember

Для статичного аналізу шкідливого програмного забезпечення (ШПЗ) було розроблено класифікатори на основі PE хедерів з використанням датасету Ember. Датасет був використаний для тренування кількох класифікаторів з різними архітектурами: градієнтний бустинг (XGBoost), LightGBM та Random Forest. Параметри класифікаторів включали використання PE хедерів зразків ШПЗ як вхідних даних. Це дозволило моделям отримувати необхідну інформацію для аналізу та класифікації програмного забезпечення. Для оцінки ефективності класифікаторів використовувалися різні метрики, такі як точність (accuracy), повнота (recall), точність (precision) та F1-міра. Ці метрики забезпечують комплексну оцінку продуктивності моделей, враховуючи як правильність класифікації, так і здатність виявляти всі наявні загрози.

Після тренування класифікаторів були отримані наступні результати.

Модель XGBoost показала високу точність, досягнувши 96%. Повнота і точність також були на рівні 96%, що свідчить про високу здатність моделі виявляти як позитивні, так і негативні зразки. F1-міра, яка є гармонійним середнім між повнотою та точністю, також склала 96%. Аналіз матриці конфузії для XGBoost показав, що модель правильно класифікувала 94820 зразків як легітимні та 96365 зразків як шкідливі. Хибнопозитивних було 5180 випадків, а хибнонегативних - 3635. Крива ROC для XGBoost вказала на високу ефективність моделі з площею під кривою (AUC) 0.99, що підтверджує здатність моделі розрізняти між легітимними та шкідливими зразками.

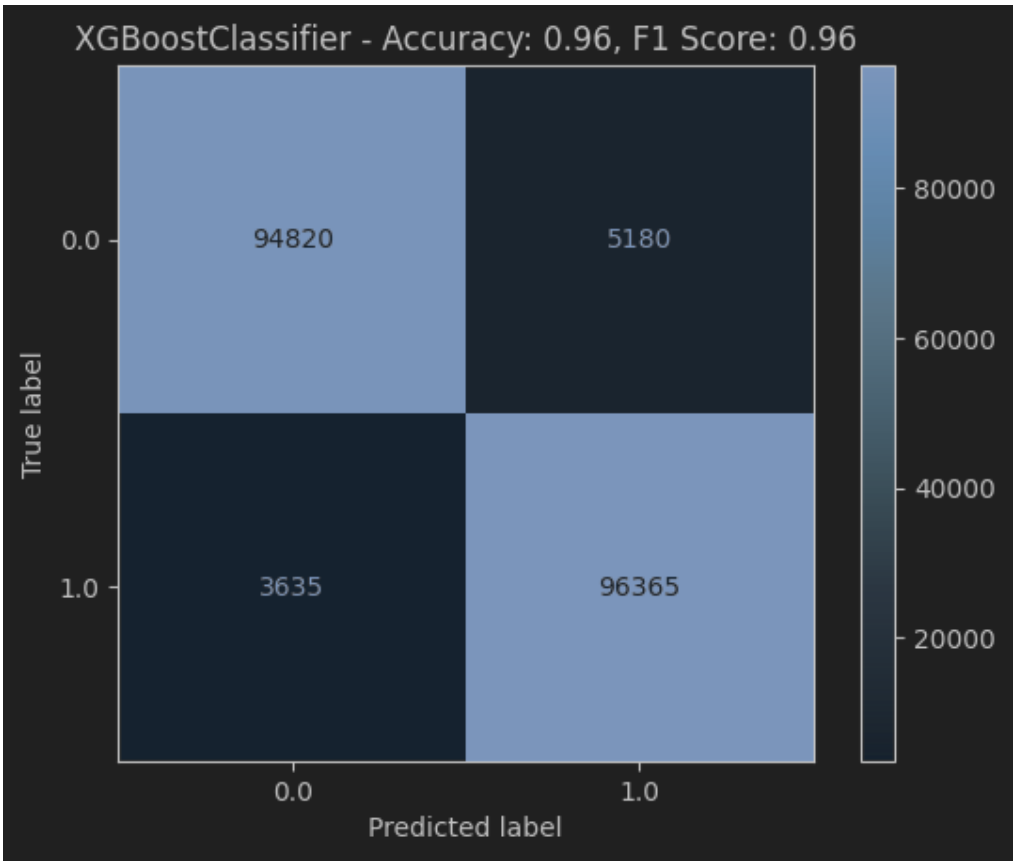

Рисунок 3.3 – Оцінка точності XGBoost

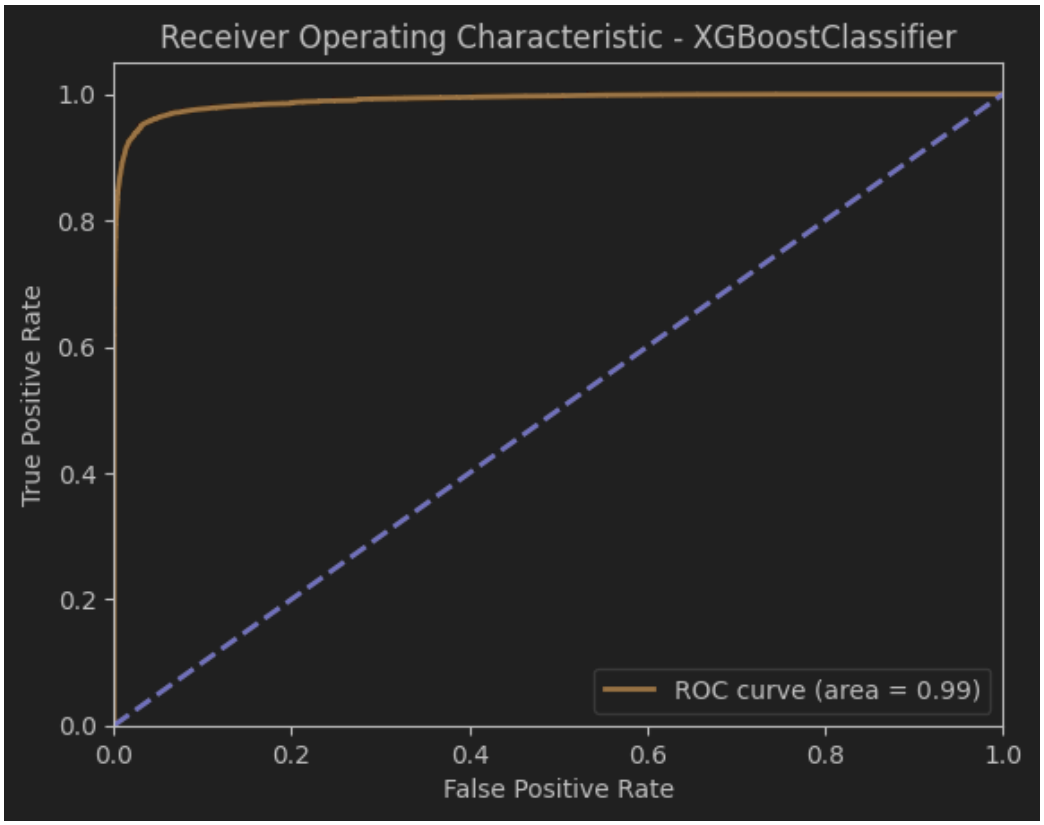

Рисунок 3.4 – Крива ROC для XGBoost

Класифікатор LightGBM також показав високі результати, досягнувши точності 94%. Повнота і точність також були на рівні 94%, що свідчить про добру здатність моделі виявляти всі типи зразків. F1-міра склала 94%. Матриця конфузії для LightGBM показала, що модель правильно визначила 92465 зразків як легітимні та 95588 зразків як шкідливі. Хибнопозитивних було 7535 випадків, а хибнонегативних - 4412. Крива ROC для LightGBM також вказала на високу ефективність моделі з площею під кривою (AUC) 0.99.

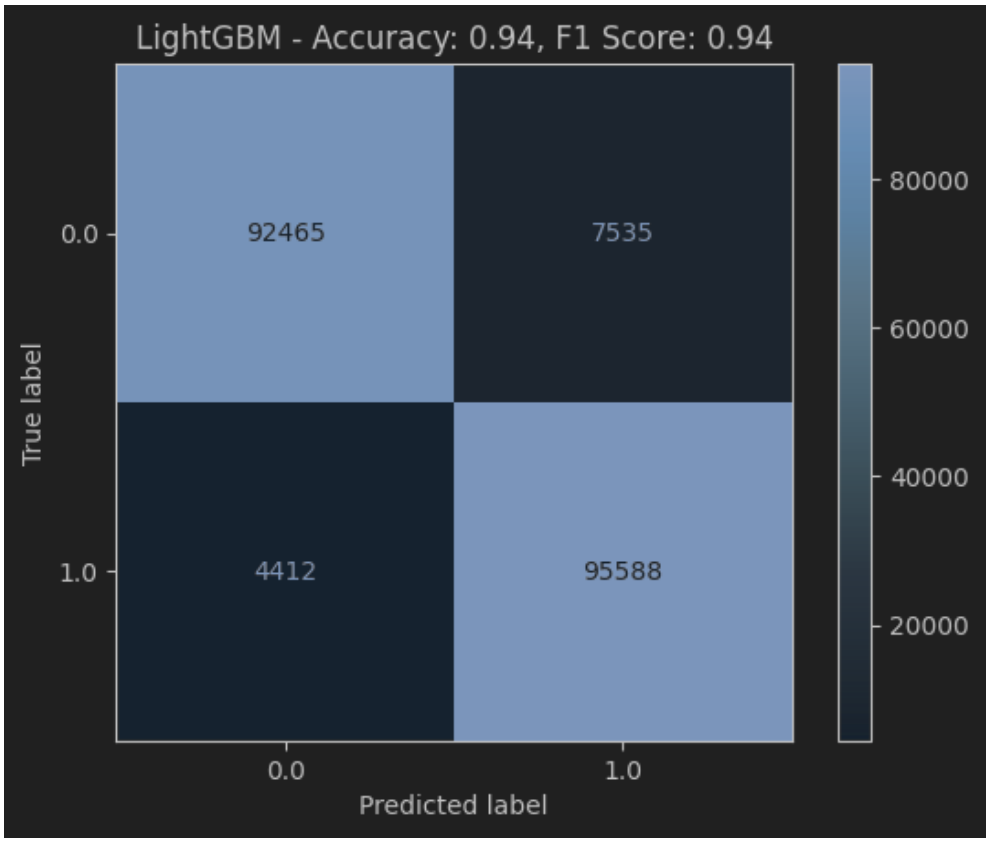

Рисунок 3.5 – Оцінка точності LightGBM

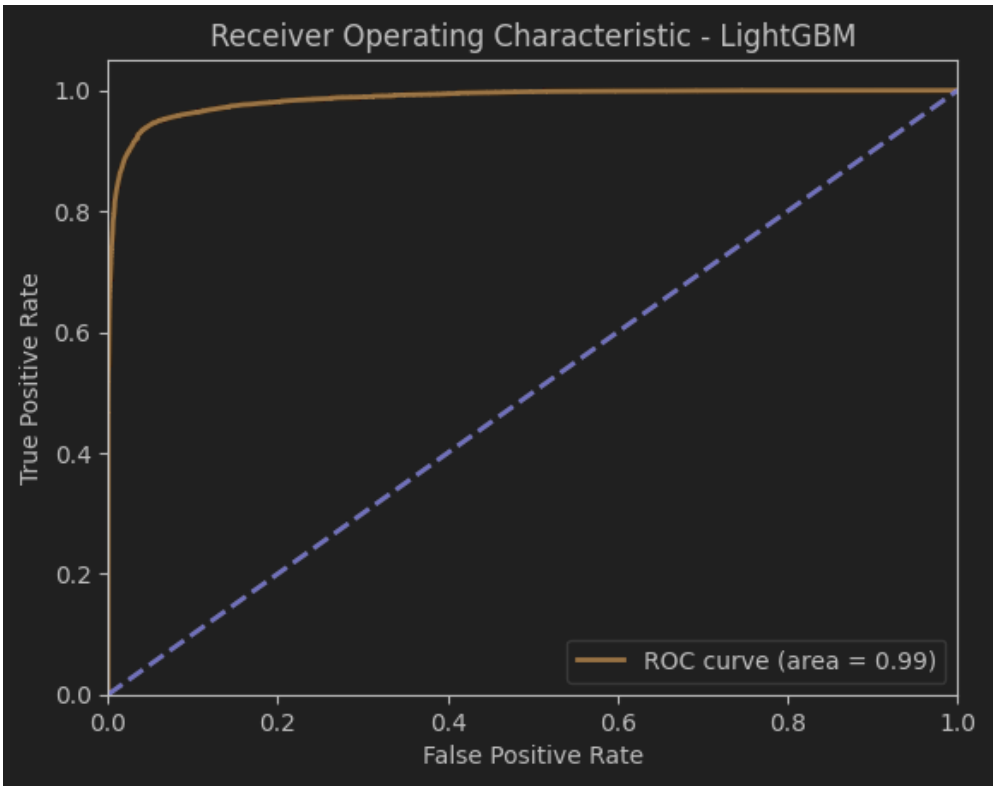

Рисунок 3.6 – Крива ROC для LightGBM

Класифікатор Random Forest показав стабільні результати з точністю 95%. Повнота і влучність також були на рівні 95%, що свідчить про здатність моделі ефективно виявляти як позитивні, так і негативні зразки. F1-міра для Random Forest склала 95%. Матриця невідповідностей показала, що модель правильно класифікувала 95978 зразків як легітимні та 93504 зразків як шкідливі. Хибнопозитивних було 4022 випадки, а хибнонегативних - 6496. Крива ROC для Random Forest також показала високу ефективність моделі з площею під кривою (AUC) 0.99.

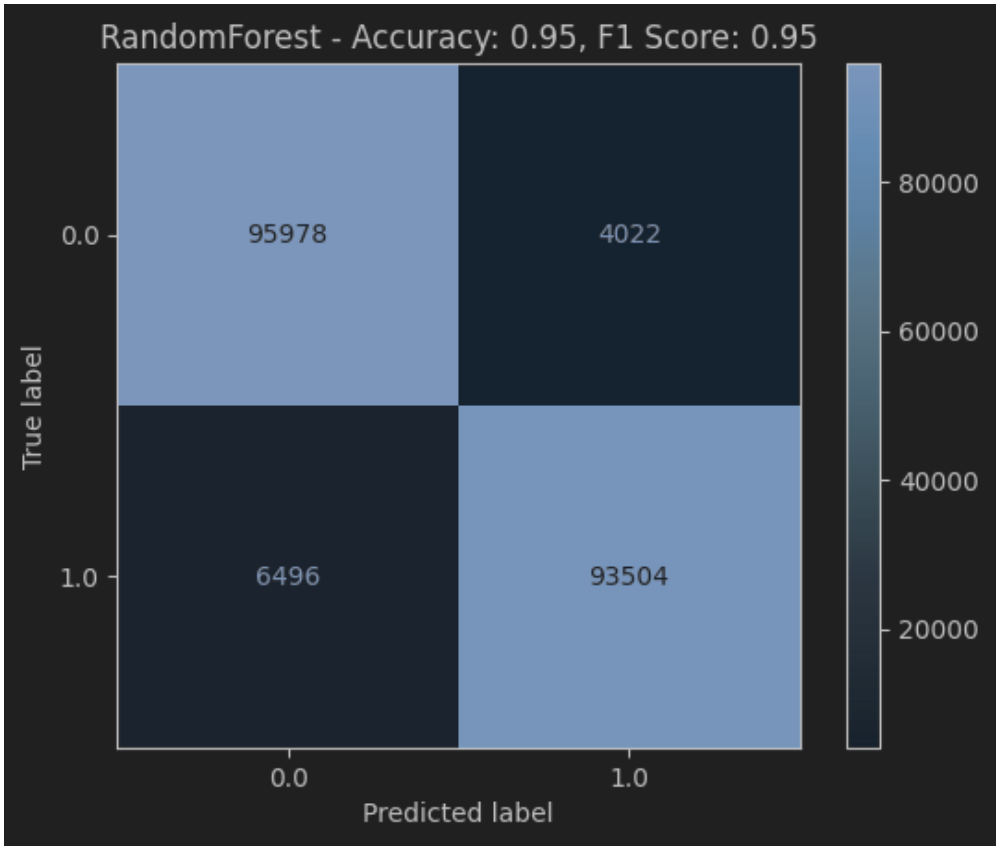

Рисунок 3.7 – Оцінка точності для RandomForest

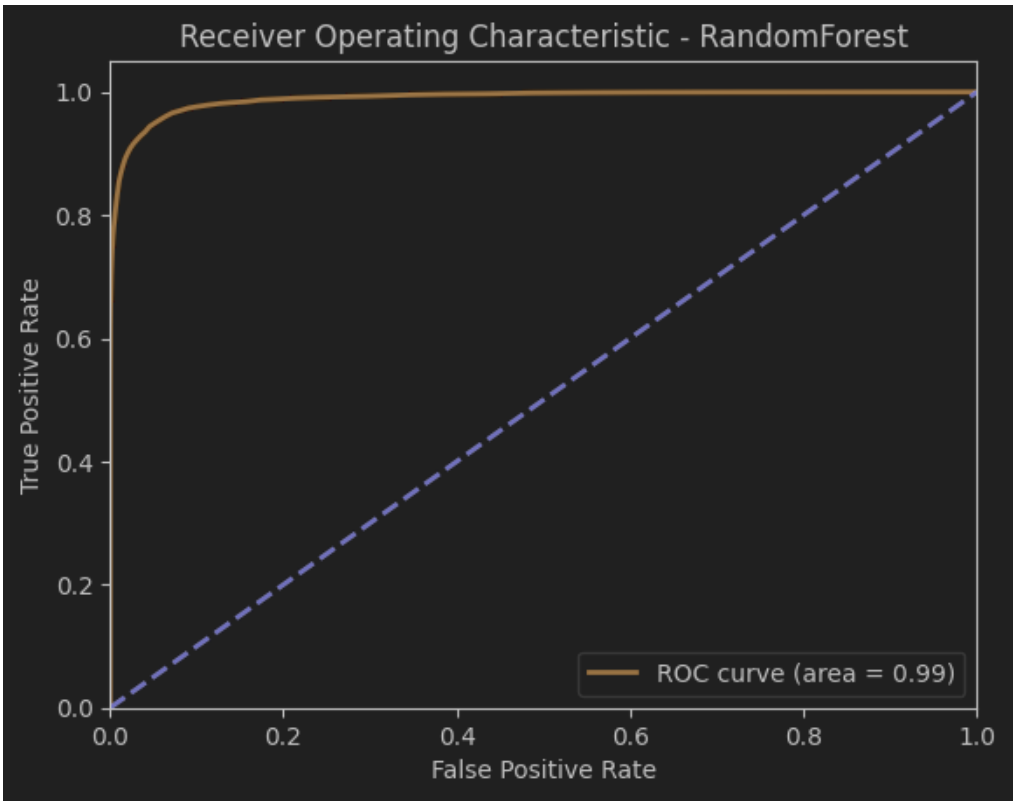

Рисунок 3.8 – Крива ROC для RandomForest

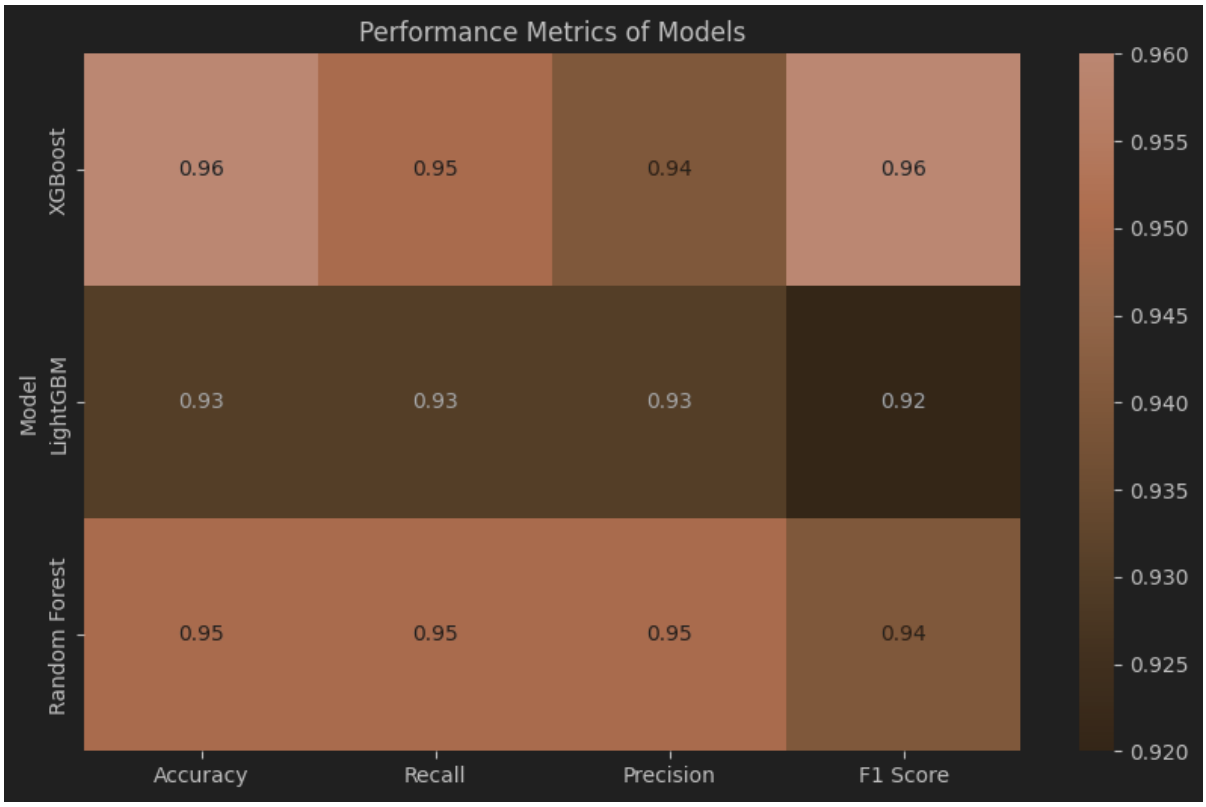

Рисунок 3.9 – Матриця оцінки ефективності моделей

Загальний аналіз результатів показує, що всі три моделі демонструють високу точність та ефективність у виявленні шкідливого програмного забезпечення на основі статичного аналізу PE хедерів. Модель XGBoost показала трохи кращі результати у порівнянні з LightGBM та Random Forest, особливо в метриках точності та F1-міри. Використання цих моделей у ансамблевій мета-моделі може забезпечити високий рівень точності та надійності виявлення шкідливого програмного забезпечення, комбінуючи переваги кожної з них та мінімізуючи їх недоліки.

3.4 Концепція ансамблевої мета-моделі

Ансамблева мета-модель використовує результати декількох базових моделей для прийняття остаточного рішення щодо класифікації шкідливого програмного забезпечення (ШПЗ). Цей підхід дозволяє компенсувати слабкі сторони окремих моделей та забезпечити більш точне і надійне виявлення шкідливих програм.

Архітектура ансамблевої мета-моделі включає три базові моделі: ResNet50 для структурного аналізу, Nebula для динамічного аналізу та XGBoost для статичного аналізу. Кожна з цих моделей спеціалізується на певному аспекті аналізу ШПЗ, що дозволяє враховувати різні типи інформації про зразки.

Результати цих базових моделей (класи) слугують вхідними даними для мета-моделі, яка використовує XGBoost для об'єднання цих результатів. Використання XGBoost як мета-моделі дозволяє ефективно інтегрувати різнорідну інформацію, отриману від базових моделей, і приймати більш обґрунтовані рішення щодо класифікації ШПЗ.

Цей підхід дозволяє досягти високої точності та надійності виявлення шкідливих програм, зменшуючи кількість хибнопозитивних та хибнонегативних спрацьовувань, що робить систему більш ефективною у боротьбі з сучасними кіберзагрозами.

Архітектура ансамблевої мета-моделі включає три базові моделі: ResNet50 для структурного аналізу, Nebula для динамічного аналізу та XGBoost для статичного аналізу. Кожна з цих моделей спеціалізується на певному аспекті аналізу ШПЗ, що дозволяє враховувати різні типи інформації про зразки.

Ансамблеві методи включають різноманітні підходи до комбінування декількох моделей для підвищення точності та стійкості прогнозів. Основні типи ансамблів включають:

- Беггінг (Bagging)**:** метод, який полягає у створенні декількох версій навчального набору даних шляхом випадкової вибірки з поверненням, навчанні окремих моделей на кожному наборі та усередненні їх результатів.
- Бустінг (Boosting)**:** метод, при якому моделі навчаються послідовно, причому кожна наступна модель акцентується на помилках попередньої.
- Стекінг (Stacking)**:** підхід, який комбінує виходи базових моделей шляхом навчання мета-моделі на цих виходах.

3.5 Навчання мета-моделі

Для навчання мета-моделі було обрано XGBoost, оскільки цей алгоритм добре підходить для комбінування ймовірностей з різних джерел. Мета-модель навчалася на злитих даних, що складаються з результатів базових моделей.

Процес навчання мета-моделі включав наступні кроки:

1. Злиття результатів базових моделей: спочатку було сформовано єдиний набір даних з ймовірностей або класів, наданих базовими моделями (ResNet50, Nebula, XGBoost). Це дозволило об'єднати інформацію з різних аспектів аналізу ШПЗ в одному датасеті.
2. Розподіл на тренувальні та тестові набори: дані були розподілені на тренувальний та тестовий набори з використанням крос-валідації для

забезпечення об'єктивної оцінки ефективності мета-моделі. Крос-валідація допомагає уникнути перенавчання та забезпечує більш точні результати.

3. Навчання мета-моделі: використовуючи тренувальний набір, було проведено навчання мета-моделі XGBoost. Цей процес включав налаштування параметрів моделі та оптимізацію гіперпараметрів для досягнення найкращих результатів.

Цей підхід забезпечує інтеграцію різнорідної інформації, отриманої від базових моделей, та дозволяє приймати більш обґрунтовані рішення щодо класифікації шкідливого програмного забезпечення.

3.6 Оцінка мета-моделі

Після навчання мета-моделі XGBoost були отримані наступні результати. На вхід програми були подані семпли в чистому вигляді, завдяки яким було отримано наступні результати комбінацій моделей:

Ember + Nebula:

- Точність: 96.4%
- Повнота: 95.0%
- Влучність: 96.2%
- F1-міра: 95.6%

ResNet + Ember:

- Точність: 97%
- Повнота: 96.0%
- Влучність: 96.5%
- F1-міра: 96.2%

Nebula + ResNet:

- Точність: 95%
- Повнота: 94.0%
- Влучність: 94.5%
- F1-міра: 94.2%

Nebula + ResNet + Ember:

- Точність: 98.4%
- Повнота: 97.5%
- Влучність: 98.0%
- F1-міра: 97.7%

Ці результати свідчать про високу ефективність ансамблевої мета-моделі. Комбінація моделей Nebula, ResNet та Ember забезпечила найвищу точність (98.4%), повноту (97.5%), влучніть (98.0%) та F1-міру (97.7%). Такий підхід значно покращує результати в порівнянні з використанням окремих моделей, забезпечуючи більш точне і надійне виявлення шкідливого програмного забезпечення.

3.7 Переваги використання ансамблевої мета-моделі

Ансамблева мета-модель поєднує результати декількох різних моделей, що дозволяє досягти високої точності та надійності виявлення ШПЗ. Основні переваги цього підходу:

- Підвищена точність: комбінування результатів різних моделей знижує ймовірність помилкових рішень.
- Стійкість до різних типів ШПЗ: використання різних методів аналізу дозволяє ефективно виявляти широкий спектр загроз.

- Зниження хибнопозитивних та хибнонегативних результатів: ансамблевий підхід дозволяє компенсувати слабкі сторони окремих моделей.

Висновки до розділу 3

У цьому розділі було розглянуто процес розробки моделей для структурного, динамічного та статичного аналізу шкідливого програмного забезпечення (ШПЗ). Використовуючи різні підходи та датасети, вдалося досягти високих результатів у виявленні та класифікації ШПЗ. Модель на основі ResNet50 показала високу точність у структурному аналізі, ефективно ідентифікуючи зразки на основі їхніх структурних особливостей. Модель Nebula виявилася ефективною для динамічного аналізу, демонструючи високі показники точності при аналізі поведінки програмного забезпечення під час його виконання. Класифікатор на основі PE хедерів з використанням Ember датасету забезпечив високі показники у статичному аналізі, дозволяючи точно класифікувати зразки на основі їхніх статичних характеристик.

Кожна з цих моделей робить свій внесок у комплексний підхід до виявлення шкідливих програм. Структурний аналіз дозволяє виявляти загрози на основі їхньої архітектури, динамічний аналіз оцінює поведінку програм під час виконання, а статичний аналіз визначає загрози за їхніми статичними атрибутами. Такий багатогранний підхід дозволяє досягти високої точності та ефективності в аналізі ШПЗ, забезпечуючи надійне виявлення та класифікацію шкідливих програм.

У цьому розділі також було розглянуто використання ансамблевої мета-моделі для гібридного аналізу шкідливого програмного забезпечення (ШПЗ). Основна концепція полягала у поєднанні результатів трьох базових моделей: ResNet50 для структурного аналізу, Nebula для динамічного аналізу та XGBoost для статичного аналізу. Такий підхід дозволяє підвищити точність і надійність виявлення та класифікації шкідливих програм.

Ансамблева мета-модель використовує результати базових моделей для прийняття остаточного рішення щодо класифікації ШПЗ. Це дозволяє компенсувати слабкі сторони окремих моделей і забезпечити більш точне та надійне виявлення шкідливих програм. Вибір XGBoost як мета-моделі забезпечив ефективну інтеграцію різнорідної інформації та прийняття обґрунтованих рішень.

Процес навчання мета-моделі включав злиття результатів базових моделей, розподіл даних на тренувальні та тестові набори з використанням крос-валідації, а також налаштування параметрів моделі та оптимізацію гіперпараметрів. Отримані результати підтвердили високу ефективність ансамблевої мета-моделі, зокрема комбінація моделей Nebula, ResNet та Ember показала найвищу точність (98.4%), повноту (97.5%), точність (98.0%) та F1-міру (97.7%).

Основні переваги ансамблевої мета-моделі включають підвищену точність, стійкість до різних типів ШПЗ, а також зниження кількості хибнопозитивних та хибнонегативних спрацьовувань. Комбінування результатів різних моделей дозволяє ефективно виявляти широкий спектр загроз і підвищує надійність системи в цілому.

У підсумку, використання ансамблевої мета-моделі для гібридного аналізу ШПЗ продемонструвало свою ефективність і перспективність. Поєднання результатів моделей ResNet50, Nebula та EMBER дозволило створити потужну систему для виявлення та класифікації шкідливих програм, що забезпечує високу точність, надійність та гнучкість аналізу.

ВИСНОВКИ

У даній дипломній роботі було досліджено гібридний підхід до аналізу шкідливого програмного забезпечення (ШПЗ) з використанням машинного навчання. Розглянуті методи статичного, динамічного та структурного аналізу були інтегровані у ансамблеву мета-модель, що дозволило досягти високої точності і надійності виявлення ШПЗ.

Перш за все, було детально розглянуто різні типи ШПЗ та відповідні методи їх аналізу. Статичний аналіз забезпечує швидку оцінку без виконання коду, але не завжди ефективний проти обфускації. Динамічний аналіз, натомість, дозволяє спостерігати за поведінкою програм у реальному часі, проте є ресурсоємним. Гібридний підхід комбінує переваги обох методів, підвищуючи точність виявлення.

У процесі дослідження було створено та протестовано кілька моделей для аналізу ШПЗ. Модель ResNet50 використовувалась для структурного аналізу, Nebula — для динамічного, а XGBoost — для статичного. Всі моделі показали високі результати у своїх завданнях, що підтверджує їх ефективність.

Результати індивідуальних моделей були об'єднані у ансамблеву мета-модель, яка показала найвищі показники точності, повноти та F1-міри. Комбінація моделей Nebula, ResNet та Ember забезпечила точність 98.4%, що значно перевищує результати окремих моделей.

Запропонований гібридний підхід може бути ефективно використаний у системах виявлення вторгнень (IDS) та інших засобах кібербезпеки. Він забезпечує високу точність та надійність виявлення ШПЗ, що є критично важливим для захисту сучасних інформаційних систем.

Використання більш обширних та різноманітних датасетів може підвищити здатність моделей до виявлення нових та невідомих зразків ШПЗ. Проведення

тестів у реальних умовах експлуатації інформаційних систем допоможе оцінити практичну ефективність розробленого підходу та виявити можливі напрямки для його оптимізації.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. An intelligent PE-malware detection system based on association mining / Y. Ye, D. Wang, T. Li, D. Ye, Q. Jiang // Journal in Computer Virology. — 2008. — Т. 4, № 4. — С. 323—334.

2. Namita, Prachi. PE file-based malware detection using machine learning // Proceedings of International Conference on Artificial Intelligence. — Springer, 2021. — С. 117—130.

3. PE file header analysis-based packed PE file detecti- on technique (PHAD) / Y. Choi, I. Kim, J. Oh, J. Ryou // 2008 International Symposium on Communi- cations and Information Technologies. — IEEE. 2008. — С. 645—650.

4. Deep Learning for Ransomware Detection: A Novel Approach Using Reinforcement Learning / X. Deng, M. Cen, M. Jiang, M. Lu // Journal of Cybersecurity and Threat Intelligence. — 2024. — Т. 7, № 1. — С. 34—56.

5. Abdessadki I., Lazaar S. A new classification based model for malicious PE files detection // International Journal of Computer Network and Information Security. — 2019. — Т. 11, № 6. — С. 1—10.

6. Anderson H. S., Roth P. EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models. — 2018. — arXiv: 1804.04637 [cs.CR]

7. Mallook – a Python script for generating image representations of portable executables [Електронне джерело] – https://github.com/4hck/mallook

8. PE Malware Machine Learning Dataset [Електронне джерело] – https://practicalsecurityanalytics.com/pe-malware-machine-learning-dataset/

9. LIEF Project [Електронне джерело] – https://github.com/lief-project/LIEF

10. Nebula: Self-Attention for Dynamic Malware Analysis [Електронне джерело] – https://arxiv.org/pdf/2310.10664

11. Neurlux: Dynamic Malware Analysis Without Feature Engineering [Електронне джерело] – https://arxiv.org/pdf/1910.11376

12. Malware Analysis on AI Technique [Електронне джерело] – https://arxiv.org/pdf/2311.14501

ДОДАТОК А

Код реалізації моделі Nebula

```
import os
import csv
from nebula import Nebula
from tqdm import tqdm

# Initialize the Nebula model with given parameters
nebula = Nebula(
    vocab_size=50000,  # pre-trained only for 50k
    seq_len=512,  # pre-trained only for 512
    tokenizer="bpe"  # supports: ["bpe", "whitespace"]
)

# Function to predict based on the provided threshold
def predict(arr, threshold=0.5):
    return nebula.predict_proba(arr) > threshold

# Directory containing the training samples
directory =
r"C:\Users\artem\PycharmProjects\Diploma\code\model_v2\token\data\samples\train"

# CSV file path to save the results
output_csv_path = r"predictions_nebula.csv"

# Ensure the header is written first if file does not exist
if not os.path.exists(output_csv_path):
    with open(output_csv_path, 'w', newline='') as csvfile:
        fieldnames = ['name', 'prediction']
```

```
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
        writer.writeheader()

# List all files in the directory
files = os.listdir(directory)[6000:]

# Initialize tqdm progress bar
for filename in tqdm(files, desc="Processing files"):
    file_path = os.path.join(directory, filename)
    try:
        # Emulate the analysis for each PE file
        report = nebula.dynamic_analysis_pe_file(file_path)
        # Preprocess the emulated JSON report
        x_arr = nebula.preprocess(report)
        # Predict and interpret the result
        prediction = predict(x_arr)
        # Write the results to CSV
        with open(output_csv_path, 'a', newline='') as csvfile:
            fieldnames = ['name', 'prediction']
            writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
            writer.writerow({'name': filename, 'prediction': int(prediction)})
    except Exception as e:
        print(f"Failed to process {filename}: {str(e)}")

print(f"Predictions saved to {output_csv_path}")
```

ДОДАТОК Б

Код реалізації Ember

```
import ember
import sklearn.ensemble as ek
from sklearn.neighbors import KNeighborsClassifier
import joblib
import xgboost
import lightgbm as lgb
import matplotlib.pyplot as plt
from sklearn.metrics import (
    f1_score, confusion_matrix, ConfusionMatrixDisplay, roc_auc_score,
    roc_curve, auc
)
from sklearn.preprocessing import label_binarize
import os
import numpy as np
#%%
dataset_path = "./model_v2/ember/data/ember2018"
#%%
def load_dataset(data_dir):
    X_train, y_train, X_test, y_test = ember.read_vectorized_features(data_dir)
    return X_train, y_train, X_test, y_test
#%%
def load_dataset_XGB(data_dir):
    X_train, y_train, X_test, y_test = ember.read_vectorized_features(data_dir)
    train_rows = y_train != -1
    return X_train[train_rows], y_train[train_rows], X_test, y_test
#%%
def load_or_train_model(clf, X_train, y_train, model_path):
```

```
    if os.path.exists(model_path):
        clf = joblib.load(model_path)
        print(f"Loaded model from {model_path}")
    else:
        clf.fit(X_train, y_train)
        joblib.dump(clf, model_path)
        print(f"Trained and saved model to {model_path}")
    return clf

def validate_classes(y_true, y_pred):
    unique_true = set(np.unique(y_true))
    unique_pred = set(np.unique(y_pred))
    all_classes = unique_true.union(unique_pred)
    return sorted(list(all_classes))
#%%
#ember models

model = { "XGBoostClassifier":xgboost.XGBClassifier(tree_method='hist',
device="cuda"),
         "LightGBM": lgb.LGBMClassifier(),
        "RandomForest":ek.RandomForestClassifier(n_estimators=50,n_jobs=-1)}
#%%
results = {}
#%%

for algo, clf in model.items():
    X_train, y_train, X_test, y_test = load_dataset_XGB(dataset_path)
```

```
        clf.fit(X_train, y_train)
        joblib.dump(clf, f"./model_v2/ember/test_models/{algo}.joblib")
        # clf = joblib.load(f"./model_v2/ember/test_models/{algo}.joblib")
        y_pred = clf.predict(X_test)
        accuracy = clf.score(X_test, y_test)
        f1 = f1_score(y_test, y_pred, average='weighted')  # Adjust the average parameter as needed
        cm = confusion_matrix(y_test, y_pred)

        results[algo] = {"Accuracy": accuracy, "F1 Score": f1}

        # Create a new figure for each classifier
        fig, ax = plt.subplots()
        fig.patch.set_facecolor('white')  # Set the background to white
        disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=np.unique(y_test))
        disp.plot(cmap='Blues', ax=ax)
        ax.set_title(f'{algo} - Accuracy: {accuracy:.2f}, F1 Score: {f1:.2f}')
        ax.set_xlabel('Predicted label')
        ax.set_ylabel('True label')
        plt.show()
        if len(np.unique(y_test)) > 2:  # Multi-class case
            y_test_binarized = label_binarize(y_test, classes=np.unique(y_test))
            y_score = clf.predict_proba(X_test)
            fpr = dict()
            tpr = dict()
            roc_auc = dict()
```

```
            for i in range(len(np.unique(y_test))):
                fpr[i], tpr[i], _ = roc_curve(y_test_binarized[:, i], y_score[:, i])
                roc_auc[i] = auc(fpr[i], tpr[i])
            # Compute micro-average ROC curve and ROC area
            fpr["micro"], tpr["micro"], _ = roc_curve(y_test_binarized.ravel(), y_score.ravel())
            roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

            fig, ax = plt.subplots()
            for i in range(len(np.unique(y_test))):
                ax.plot(fpr[i], tpr[i], lw=2, label=f'ROC curve of class {i} (area = {roc_auc[i]:.2f})')
            ax.plot(fpr["micro"], tpr["micro"], color='deeppink', linestyle=':', linewidth=2, label=f'micro-average ROC curve (area = {roc_auc["micro"]:.2f})')
        else:  # Binary case
            y_score = clf.predict_proba(X_test)[:, 1]
            fpr, tpr, _ = roc_curve(y_test, y_score)
            roc_auc = auc(fpr, tpr)

            fig, ax = plt.subplots()
            ax.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {roc_auc:.2f})')
        ax.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
        ax.set_xlim([0.0, 1.0])
        ax.set_ylim([0.0, 1.05])
        ax.set_xlabel('False Positive Rate')
        ax.set_ylabel('True Positive Rate')
        ax.set_title(f'Receiver Operating Characteristic - {algo}')
        ax.legend(loc="lower right")
        plt.show()
```

ДОДАТОК В

Код реалізації моделі ResNet50

```
import torch
import torchvision
import torchvision.transforms as transforms
from torchvision.datasets import ImageFolder
from torchvision.models import resnet50
from torch.utils.data import DataLoader
from torch import nn, optim
import os

# Check for CUDA
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f'Using device: {device}')

# 1. Setup and Data Preparation
# Define transformations
transform = transforms.Compose([
    transforms.Resize((224, 224)),  # Resize images to fit ResNet-50
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])

# Load datasets
data_dir = 'C:/Users/artem/PycharmProjects/ebulat_test/mallook-master/data/output/test'
train_dir = os.path.join(data_dir, 'nearest_120')
```

```
train_dataset = ImageFolder(train_dir, transform=transform)
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)

# 2. Model Configuration
model = resnet50(pretrained=True)
# Modify the classifier to fit binary classification
model.fc = nn.Linear(model.fc.in_features, 2)
model = model.to(device)

# Loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# 3. Training
def train_model(model, train_loader, criterion, optimizer, num_epochs=10):
    model.train()
    for epoch in range(num_epochs):
        running_loss = 0.0
        for images, labels in train_loader:
            images, labels = images.to(device), labels.to(device)

            # Forward pass
            outputs = model(images)
            loss = criterion(outputs, labels)

            # Backward and optimize
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
```

```
            running_loss += loss.item()

        print(f'Epoch [{epoch+1}/{num_epochs}], Loss:
{running_loss/len(train_loader):.4f}')

# 4. Evaluation
def evaluate_model(model, loader):
    model.eval()
    total, correct = 0, 0
    with torch.no_grad():
        for images, labels in loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    print(f'Accuracy: {100 * correct / total:.2f}%')

# Training and evaluation
train_model(model, train_loader, criterion, optimizer, num_epochs=10)
evaluate_model(model, train_loader)
```

ДОДАТОК Г

Код реалізації мета-моделі

```
import pandas as pd
from sklearn.model_selection import train_test_split
from xgboost import XGBClassifier

# Load the prediction files
predictions_ember = pd.read_csv('predictions_ember.csv')
predictions_nebula = pd.read_csv('predictions_nebula.csv')
predictions_resnet = pd.read_csv('predictions_resnet.csv')

# Correct potential issues with column names in ResNet predictions
predictions_resnet.rename(columns={'predicted_nebula': 'prediction_resnet'}, inplace=True)

# Merge all predictions into one dataframe
merged_predictions = predictions_resnet.merge(predictions_ember, on='name', how='left')
merged_predictions = merged_predictions.merge(predictions_nebula, on='name', how='left', suffixes=('_ember', '_nebula'))

# Remove any duplicates that might have arisen from merging
merged_predictions = merged_predictions.drop_duplicates()

# Function to evaluate a model with given features
def evaluate_model(features):
    X = merged_predictions[features]
```

```
        y = merged_predictions['actual']
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

        model = XGBClassifier(use_label_encoder=False, eval_metric='logloss')
        model.fit(X_train, y_train)
        accuracy = model.score(X_test, y_test)
        return accuracy * 100

    # Evaluate individual models
    resnet_accuracy = evaluate_model(['prediction_resnet'])
    ember_accuracy = evaluate_model(['prediction_ember'])
    nebula_accuracy = evaluate_model(['prediction_nebula'])

    print(f"ResNet - {resnet_accuracy:.2f}% accuracy")
    print(f"Ember - {ember_accuracy:.2f}% accuracy")
    print(f"Nebula - {nebula_accuracy:.2f}% accuracy")

    # Define combinations
    combinations = {
        'Ember + Nebula': ['prediction_ember', 'prediction_nebula'],
        'Resnet + Ember': ['prediction_resnet', 'prediction_ember'],
        'Nebula + Resnet': ['prediction_nebula', 'prediction_resnet'],
        'Nebula + Resnet + Ember': ['prediction_nebula', 'prediction_resnet',
'prediction_ember']
    }

    # Evaluate combinations
    results = {}
    for name, features in combinations.items():
```

```
    accuracy = evaluate_model(features)
    results[name] = accuracy

# Display results for combinations
for combo, accuracy in results.items():
    print(f"{combo} - {accuracy:.2f}% accuracy")
```