КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

автоматизації і інформаційних технологій
(факультет)

інформаційних технологій проєктування та прикладної математики
(кафедра)

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА ЗДОБУТТЯ ОСВІТНЬОГО РІВНЯ «БАКАЛАВР»

на тему: «Метод виявлення шкідливих програм за допомогою машинного навчання**»**

КУШНІР ВОЛОДИМИР ЮРІЙОВИЧ
(прізвище, ім’я та по батькові студента повністю)

Київ 2026 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

автоматизації і інформаційних технологій
(факультет)

інформаційних технологій проєктування та прикладної математики
(кафедра)

ЗАТВЕРДЖУЮ
Завідувач кафедри ІТППМ
д.т.н., професор Бородавка Є.В.

_____
„___” _____2026 року

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА ЗДОБУТТЯ ОСВІТНЬОГО РІВНЯ «БАКАЛАВР»

на тему: «Метод виявлення шкідливих програм за допомогою машинного навчання»

Виконав: студент 4-го курсу, групи ІСТ-22

Спеціальності: 126 «Інформаційні системи та технології»
(шифр і назва спеціальності)

Кушнір В.Ю.
(прізвище та ініціали)

Керівник доктор філософії Бугров А.А.
(прізвище та ініціали)

Рецензент доктор філософії Рябчун Ю.В.
(прізвище та ініціали)

Київ 2026 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

Факультет: автоматизації і інформаційних технологій

Кафедра: інформаційних технологій проєктування та ПМ

Освітній рівень: «бакалавр» за ОПП

Спеціальність: 126 «Інформаційні системи та технології»

Освітня програма: «Інформаційні системи та технології»

ЗАТВЕРДЖУЮ
Завідувач кафедри ІТППМ
д.т.н., професор Бородавка Є.В.
_____
“____”_____ 2026р.

З А В Д А Н Н Я
ДО ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА ЗДОБУТТЯ ОСВІТНЬОГО РІВНЯ «БАКАЛАВР»

Кушнір Володимир Юрійович

1. Тема роботи: «Метод виявлення шкідливих програм за допомогою машинного навчання»,
затверджена наказом ректора КНУБА № 444/52-15/13.6/26 від “08” квітня 2026 року
2. Керівник роботи: Бугров Анатолій Анатолійович, доктор філософії, старший викладач кафедри ІТППМ
3. Строк подання студентом роботи до захисту: червень 2026 року

4. Зміст пояснювальної записки за розділами:
   Р.1. Методи машинного навчання та досліджувані файли
   Р.2. Алгоритми машинного навчання
   Р.3. Аналіз та порівняння знаходження вірусів за допомогою методів машинного навчання
5. Інформаційні слайди:
   С.1. Титульний слайд.
   С.2. Актуальність роботи.
   С.3. Мета, завдання.
   С.4. Алгоритми машинного навчання.
   С.5. Статистика по завантажених файлах.
   С.6. Розподіл файлів по голосам

С.7. Приклад крос-валідації
С.8. Важливість ознак
С.9. Порівняння алгоритмів
С.10. Залежність якості алгоритму від розміру даних
С.11. Реалізаація методу
С.12. Висновок

6. Календарний план виконання кваліфікаційної роботи

Види робіт та їх зміст	Дата виконання
1. Завдання на виконання роботи. 2. 1-ий розділ роботи	15.03.26
1-ий та 2-ий розділи роботи	12.04.26
Всі розділи роботи	17.05.26
Ілюстрації роботи програмного продукту та (або) технічних розробок	24.05.26
Довідка про проходження перевірки на академічний плагіат	30.05.26
Готова робота, матеріали презентації, доповідь, відповіді до питання	01.06.26 – 05.06.26
Рецензія на роботу	10.06.26
Захист кваліфікаційної роботи	17.06.26 – 24.06.26

7. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта, представника комісії	Дата	Підпис
Прийом програмного продукту	Доктор філософії, доцент Рябчун Ю.В.		

8. Дата видачі завдання: 12 січня 2026 року

Керівник _____ Бугров А.А.
(підпис) (прізвище та ініціали)

Бакалавр _____ Кушнір В.Ю.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

«Метод виявлення шкідливих програм на базі машинного навчання».

Кваліфікаційна робота бакалавра за спеціальністю: 126 «Інформаційні системи та технології», освітня програма: «Інформаційні системи та технології». – Київський національний університет будівництва та архітектури. – Київ, 2026.

Виявлено найбільш важливі ознаки для визначення шкідливості файлу в умовах обмежених ресурсів і часу, а також порівняно сучасні алгоритми машинного навчання.

Теоретична та практична реалізація методу виявлення шкідливих програм на базі машинного навчання.

Ключові слова : Алгоритми машинного навчання, градієнтний бустінг, випадковий ліс, portable executable file, LIGHTGBM.

SUMMARY

"Method of detecting malware on the basis of machine learning".

Bachelor's qualification work in the specialty: 126 "Information systems and technologies", educational program: "Information systems and technologies". - Kyiv National University of Construction and Architecture. - Kyiv, 2026.

The most important features for determining the harmfulness of the file in conditions of limited resources and time, as well as relatively modern machine learning algorithms.

Theoretical and practical implementation of the method of detecting malware on the basis of machine learning.

Keywords: Machine learning algorithms, gradient boosting, random forest, portable executable file, LIGHTGBM.

ЗМІСТ

ВСТУП

У наш час одна з найголовніших цінностей на планеті вважається інформація. Комп'ютерні програми і додатки розробляються з великою швидкістю. З розвитком комп'ютерних технологій та всесвітньої мережі, все більше людей починають хвилюватися про їхню безпеку та безпеку своїх даних. А з розвитком технологій також розвивається і шкідливе програмне забезпечення і це змушує людей розробляти нові більш сучасні механізми захисту. За даними McAfee Lab в першому кварталі 2019 року кількість нападів шкідливого програмного забезпечення зросло на 18 % відносно попереднього кварталу, зафіксовано понад 65 мільйонів шкідливого програмного забезпечення, а загальна кількість майже достигла помітки в 1 мільярд [1]. Одні з найпоширеніших нових шкідливих ПЗ в першому кварталі 2019 року були Dharma, GandCrab та Ryuk. Дослідження такої кількості даних вимагає від антивірусних компаній великих зусиль. Віруси стає все більш складно виявити. Сучасне шкідливе програмне забезпечення здатне як впроваджуватися і заражати чисті файли системи та інші програми, так і самостійно поширюватися і міняти своє тіло в процесі життєдіяльності. Використовуються різні способи приховування шкідливого коду. Застосовуються інструменти шифрування коду, секцій, даних. Шкідливе програмне забезпечення може не тільки викрасти або завдати шкоди даним користувача, вимагати гроші, уповільнити роботу комп'ютера, але і перетворити комп'ютер користувача в шпигуна, використовувати апаратне забезпечення для своїх потреб, заволодіти його управлінням.

Мета дипломної роботи – створення методу знаходження шкідливого програмного забезпечення на базі машиного забезпечення.

Для досягнення мети потрібно вирішити наступні задачі:

– дослідити існуючі підходи;

– виявити найбільш важливі ознаки для визначення шкідливості файлу в умовах обмежених ресурсів і часу;

– створення методу на основі машинного навчання для статичного аналізу PE файлів для операційної системи Windows з подальшою інтеграцією в програмний продукт.

Об’єкт дослідження – статичний аналіз PE файлів для операційної системи Windows.

Предмет дослідження – алгоритми машинного навчання.

Методи досліджень. Проведені дослідження в даній роботі базуються на сучасних алгоритмах машинного навчання з використанням статичного аналізу файлів.

Новизна – реалізовано новий метод на базі машинного навчання з використанням найбільш значущих ознак файлу, що дозволяє виявляти шкідливе програмне забезпечення до того як антивірусні компанії дослідять його

Практична цінність роботи полягає в тому, що запропонований метод машинного навчання можна вбудувати в антивірусне програмне забезпечення, і це допоможе вберегти інформацію та інформаційну систему від загроз, які ще не дослідили антивірусні компанії. За допомогою методу машинного навчання, встановленого в антивірусне програмне забезпечення, в інформаційній системі з'являється можливість відразу аналізувати сигнатуру файлу і виявити чи він шкідливий.

РОЗДІЛ 1

МЕТОДИ МАШИННОГО НАВЧАННЯ ТА ДОСЛІДЖУВАНІ ФАЙЛИ

1.1. Існуючі рішення

У наш час одна з найголовніших цінностей на планеті вважається інформація. Комп'ютерні програми і додатки розробляються з великою швидкістю. З розвитком комп'ютерних технологій та всесвітньої мережі, все більше людей починають хвилюватися про їхню безпеку та безпеку своїх даних. А з розвитком технологій також розвивається і шкідливе програмне забезпечення і це змушує людей розробляти нові більш сучасні механізми захисту. За даними McAfee Lab в першому кварталі 2019 року кількість нападів шкідливого програмного забезпечення зросло на 18 % відносно попереднього кварталу, зафіксовано понад 65 мільйонів шкідливого програмного забезпечення, а загальна кількість майже достигла помітки в 1 мільярд [1]. Одні з найпоширеніших нових шкідливих ПЗ в першому кварталі 2019 року були Dharma, GandCrab та Ryuk. Дослідження такої кількості даних вимагає від антивірусних компаній великих зусиль. Віруси стає все більш складно виявити. Сучасне шкідливе програмне забезпечення здатне як впроваджуватися і заражати чисті файли системи та інші програми, так і самостійно поширюватися і міняти своє тіло в процесі життєдіяльності. Використовуються різні способи приховування шкідливого коду. Застосовуються інструменти шифрування коду, секцій, даних. Шкідливе програмне забезпечення може не тільки вкрасти або пошкодити дані користувача, вимагати гроші, уповільнити роботу комп'ютера, але і перетворити комп'ютер користувача в шпигуна, використовувати апаратне забезпечення для своїх потреб, заволодіти його управлінням.

Традиційний підхід до виявлення шкідливих програм заснований на зіставленні сигнатур досліджуваних файлів [2]. Процедура полягає в наступному:

1) шкідливе ПЗ або новий вірус починає поширюватися;

2) експертами антивірусних компаній отримуються зразки для дослідження поведінки вірусу;

3) вірусу присвоюється експертами унікальна сигнатура, що представляє собою послідовність інструкцій;

4) сигнатура додається в базу даних сигнатур шкідливих програм;

5) клієнтам повідомляють про оновлення бази сигнатур;

6) клієнти оновлюють бази сигнатур, таким чином вони стають захищеними від даного виду шкідливого ПЗ.

Потрібно відзначити, що за допомогою сигнатури гарантовано визначити тип вірусу. Це допомагає в базу сигнатур вносити способи лікування заражених файлів та вірусів. Але в даного підходу є декілька ключових недоліків:

1) клієнт захищений тільки від відомих вірусів. Для того щоб отримати сигнатуру нового вірусу, потрібно дослідити його. Як правило, сигнатури створюються так, щоб покривати якомога більшу кількість вірусів чи сімейство вірусів. Проте при зміні тіла вірусу, сигнатура перестає виявлятися;

2) бази даних сигнатур постійно зростає. Зі збільшенням кількості вірусів, їх типів, а також здатність вірусів змінюватися збільшується і швидкість наповнення бази;

3) при появі вірусу і до оновлення бази сигнатур клієнт вразливий для нової шкідливої програми. Тільки визначивши досліджуваний файл як вірус можна отримати його сигнатуру і додати в базу.

Більш того, розробники шкідливого програмного забезпечення навчилися обходити пошук сигнатур за допомогою обфускації тіла вірусу. Поліморфні і метаморфічні віруси змінюють свій зовнішній вигляд в процесі життєдіяльності. Все це спонукало антивірусні компанії розробляти альтернативні способи захисту. А саме, можна виділити два великих напрямки досліджень [3]:

1) статичний аналіз (аналіз структури бінарного файлу, логічних структур, його атрибутів, даних і потоку виконання);

2) динамічний аналіз (відстеження дій програми при її виконанні, побудова її профілю).

Кожен з цих методів має свої переваги і недоліки. Зазвичай, для кращого пошуку шкідливого програмного забезпечення вони використовуються разом. Кожнен з цих методів може помилково визначити чистий файл як вірус. Від такого помилкого визначення передбачуване лікування файлу може його зіпсувати, що спричинить до втрати інформації.Більшість статичних аналізаторів грунтуються на добуванні структур PE (Portable Executable) файлу, як в деревовидному вигляді, так і у вигляді сукупності полів. Використовуються як ознаки, одержувані з машинного коду, виклики функцій, так і ознаки, одержувані з розгляду файлу в вигляді послідовності байт, визначення статистик, ентропій, підрахунок *n*-грам.

Виявляти аномалії в структурах бінарних файлів також можливо за допомогою PE парсеру – PortEX, більш детально про нього можна прочитати в [4]. Для створення цього парсеру було проведено дослідження стратегій зараження файлів, виділені найбільш ймовірні атрибути чистих файлів. Також проводилася оцінка залежності ентропії секцій PE файлу і ймовірності того, що файл шкідливий. Для отримання ймовірності того, що файл шкідливий застосовуються евристики. Було визначено:

1) бустери (шаблони, які вказують на поведінку або зовнішній вигляд шкідливого ПЗ). Збільшують імовірність того, що файл буде ідентифікований як шкідливий;

2) стопери (шаблони, які вказують на поведінку або зовнішній вигляд, який є нетиповим для шкідливого ПЗ). Зменшують імовірність того, що файл буде визначений як шкідливий.

В парсері обчислюється сумарний бустер і сумарний стоппер на основі евристик і статистики по зібраних файлах і на основі отриманого значення видається ймовірність шкідливості файлу. Ентропія секцій PE файлу допомогла визначати шкідливість файлу.

Межі та визначення бустерів і стопперів являють собою евристики, засновані на статистиці з багатьох файлів, які можна поліпшити методами машинного навчання.

В [3] аналіз будується на добуванні 197 атрибутів з бінарного файлу. Атрибути представляють собою або характер присутності (DLLs referred, APIs referred), або значення полів структур файлу. Виділення значущих атрибут відбувається на основі статистичних тестів (t-тест, x^2-тест). Використовуючи відібрані 19-20 ознак навчають моделі бустінга, випадкового лісу. Відсутній більш глибокий аналіз PE-формату, розгляд ділянок файлу в вигляді послідовності байт. Також немає інформації про швидкість роботи алгоритму.

Досить повна картина вибору важливих атрибут PE файлу наведена в [5]. Наведена табл. 1.1 використовуваних ознак в попередніх роботах на дану тематику. Ознаки – ознаки, витягнуті з PE файлу. Або з заголовка, або з тіла. У колонці «Структура» показані додаткові ознаки, що збираються з файлу. Додатково введено такі скорочення: API: Application Programming Interface, BYT: Byte code, FC: Function Call, STC: Structural features, OP: Operation code.

Завдання стояло в класифікації шкідливих програм на типи (malware family). Стаття підготовлена на основі конкурсу kaggle, де учасникам Microsoft надала 21741 файлів без заголовка PE файлу (PE header) для класифікації по 9 типам. У дослідженні основний упор був зроблений на виборі кращого набору атрибут. Вибір проводився як на основі полів структур PE файлу, так і розглядаючи файл у вигляді послідовності байт.

Більш конкретно, уявімо класифікацію ознак, що розглядаються в статтях:

1) hex dump-based features (Ознаки, витягнуті з послідовності байт файлу):

Таблиця 1.1

Методи статичного аналізу з типами видобутих ознак

Рік	**Стаття**	**Тип**	**Ознаки заголовку**	**Ознаки тіла**	**Структура**
2008	Ye et al. [6]	Детекція	API	—	Itemset
2009	PE-Miner [7]	детекція	STC	STC	—
2009	Tabish et al. [8]	детекція	BYT	BYT	N-gram
2009	Griffin et al. [9]	детекція	BYT	BYT	Sequence
2009	Hu et al. [10]	детекція	—	FC	Graph
2010	Sami et al. [11]	детекція	API	—	Itemset
2011	Natara j et al. [12]	класифікація	BYT	BYT	—
2012		класифікація	STC	BYT	N-gram
2013	Jacob et al. [13]	детекція	—	OP	Sequence
2014	Santos et al. [14]	детекція	BYT	BYT	N-gram
2015	Nissim et al. [15]	детекція	DLL	—	Tree
	DLLMiner [16]				

– n-грам. Послідовність N байт. 1-gram – частота байтів і була обрана в статті;

– метадані. Наприклад, розмір файлу;

– ентропія. Підрахунок ентропії всього файлу або методом ковзаючого вікна;

– подання файлу як зображення [12];

– довжини рядків. Виймання довжин можливих рядків ASCII символів з файлу.

2) features extracted from disassembled files (Витяг ознак з структур PE файлу):

– метадані. Наприклад, кількість рядків у файлі;

– частота спеціальних символів. Частота символів: −, +, ∗,], [, ?, @;

– коди операцій. Підрахунок статистики по асемблерним інструкціям;

– регістри. Підрахунок статистики по використовуваних регістрам;

– application Programming Interface. Перевірка присутності/відсутності певного набору API викликів;

– секції. Статистика по секціях;

– встановлення байт. Статистика по db, dw і dd інструкціям.

Така класифікація дозволяє в повній мірі побачити дослідження в даній області по вилученню ознак і поточні результати. Зауважимо, що частина цих ознак недоступна в нашому дослідженні. Підрахунок статистики по довгій послідовності байт представляє собою дорогу операцію. А саме, підрахунок статистики по всьому файлу або подання файла як зображення будемо уникати.

Також були оригінальні ідеї з інших статей, а саме, подання файлу в вигляді зображення [12]. У статті наведено графік важливості ознак для визначення класу шкідливого ПЗ. В якості алгоритму використаний бустінг на вирішальних деревах (XGBoost) [17], а також, для отримання кращого результату в конкурсі беггінг (bagging) [18]. У реальних же умовах у нас доступна інформація з PE header. Також, цікаво дослідити значимість атрибут в задачі визначення шкідливого ПЗ.

Для повноти картини наведемо короткий опис оригінального дослідження [12], в якому PE файл представлявся у вигляді чорно-білого зображення. Завдання стояло у визначенні типу (malware family) шкідливого ПЗ. Послідовність байт представляли у вигляді матриці (0: чорний, 1: білий). Ширина зображення фіксувалася в залежності від розміру файлу (32-1000kB). На основі зображень були визначені загальні патерни, текстури для 25 типів шкідливих ПО. Для аналізу текстури застосовувався фільтр Габора.

Ознаки, отримані із зображень використовували в класифікаторі – метод k-найближчих сусідів з метрикою Евкліда. Хоч дослідження і представляє інтерес, але в силу наших вимог, а саме, суворого обмеження часу роботи методу, ми змушені відмовитися від цього виду ознак.

Згадаємо про існування бібліотек, націлених на боротьбу з деобфускацією коду. Один з таких інструментів FLOSS. FLOSS комбінує в собі кілька підходів статичного аналізу для пошуку обфусцірованних ASCII рядків в аналізуємому файлі. А саме, аналізує потік управління для розбору файлу на функції, базові блоки. Використовує евристики для пошуку декодуючих функцій. Емулює поведінку функцій декодування для вилучення зчитуваних ASCII рядків. Такий інструментарій покликаний допомогти статичному аналізу з зашифрованими файлами і отримати більше інформації про них. Використовуючи FLOSS, наприклад, можна отримати список імпортованих API і бібліотек. У даній роботі ми не будемо використовувати цей інструментарій. Завдання полягає в створенні швидкого і якісного алгоритму машинного навчання. Для більш глибокого аналізу варто досліджувати відмовостійкість подібних бібліотек і порівняти існуючі інструменти деобфускації. У даній роботі це питання не висвітлюється.

1.2. PE формат

Для аналізу досліджуваного файлу класифікатор отримує інформацію від PE-парсера. Знання про PE (Portable Executable) формат необхідно для оцінки корисності витягується інформації і розуміння завантаження файлу в пам'ять для виділення найбільш значущих атрибут.

Далі наводиться короткий опис PE формату, його структури, як для 32 бітних, так і 64 бітних систем.

Повна специфікація доступна на сайті Microsoft [19], проте, в документації є двозначності. Деякі моменти покриті статтею Кріса Касперски [20]. Потрібно розуміти, що специфікація, наведена на сайті Microsoft, має, скоріше, оглядовий характер. Як насправді надходять завантажувачі можна спостерігати лише експериментально.

В особливо хитрих випадках один і той же файл може бути запущений одним завантажувачем, але, запустивши іншим, може привести всю систему до

перезавантаження. Варто пам'ятати, що завантажувач при запуску бінарного файлу може його змінювати.

«Portable Executable (PE, переносимий виконуваний) – формат виконуваних файлів, об'єктного коду та динамічних бібліотек, який використовується в 32- і 64-розрядних версіях операційної системи Microsoft Windows. Формат PE являє собою структуру даних, що містить всю інформацію, необхідну PE-завантажувачу для відображення файлу в пам'ять» [21]

Всі PE файли можна розділити на файли з розширенням EXE і DLL. DLL файли призначені для експортування функцій або даних для використання іншими програмами. Тобто, вони зазвичай можуть бути запущені в контексті інших програм. Такі файли мають розширення .dll, .sys, .ocx, .cpl, .fon, .drv [22]. EXE файли запускаються в своїх власних процесах. Вони зазвичай мають розширення .exe і не експортують символи. Незважаючи на такий умовний поділ, формат не забороняє створювати гібриди. Наприклад, EXE файли може експортувати символи.

Структура PE файлу представлена на рис. 1.1. Атрибути цієї структури, а також статистика по послідовностям байт цієї структури використовується для отримання ознак.

Будь-який PE файл починається з MS-DOS Stub'a. Залишений для сумісності і являє собою заглушку. Перші два байта якої повинні бути рівними "MZ". Атрибут e_lfanew, являє собою зсув PE File заголовка відносно початку файлу і вказує на сигнатуру "PEx0x0". Не обов'язково, щоб e_lfanew вказувало на область відразу після MS-DOS Stub, що також відображено на рис. 2 – невикористана область. Секції розташовуються відразу після PE File заголовка.

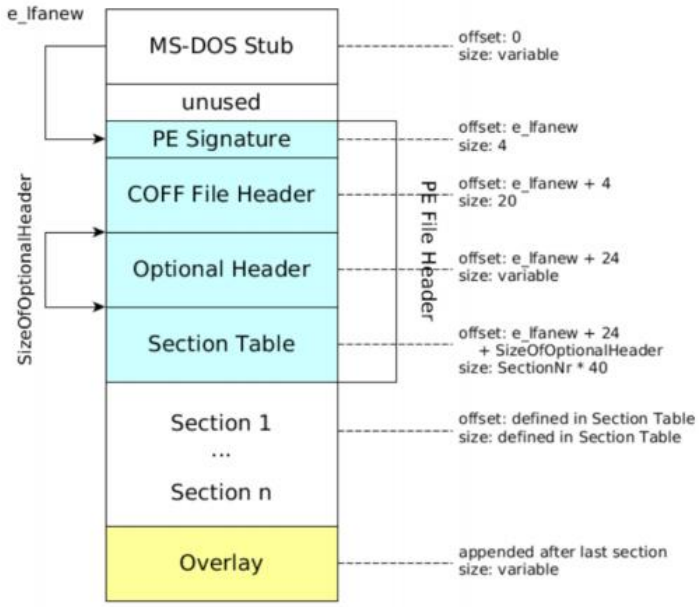

Рис. 1.1. Структура PE файлу

Після PE сигнатури "PEx0x0" в PE File Header розташовується COFF File Header. Має фіксований розмір. Містить загальну інформацію про файл, таку як: тип цільової машини (x64, ARM, MIPS і так далі), кількість секцій – NumberOfSections, дата створення файлу, розмір опціонального заголовка – SizeOfOptionalHeader. Атрибути всього файлу – Characteristics (саме тут містяться прапори – чи є файл EXE або DLL). Практично всі атрибути цієї структури варто розглядати при детектуванні вірусів, зараження файлу. Виключимо з розгляду дату створення файлу – він не повинен впливати на ймовірність зараження, але вибірка може виявитися зміщеною за цим параметром.

Далі йде опціональний заголовок (Optional Header). Хоч ця структура і має таку назву, її присутність обов'язкова для завантаження файлу. Структура містить уточнюючу інформацію про файл. Таку, наприклад, як 32х або 64х бітовий адресний простір. Сумарний розмір секцій коду, ініціалізованих і неініціалізованих даних (відповідно до [20] ці поля ніким не перевіряються також як і відносні базові адреси кодової секції та секції даних). У деяких антивірусів є евристики на значення адреси точки входу – AddressOfEntryPoint. Передбачається, що точка входу розташовується в першій секції файлу

(як правило .text секція). Базова адреса завантаження програми (ImageBase) повинна бути кратна 64кб. Також, для аналізу корисні вирівнювання – FileAlignment, SectionAlignment, а вірніше поля PE файлу, від яких вимагається вирівнювання. Також, в опціонально заголовку міститься каталог даних (DataDirectory), кількість елементів якої – NumberOfRvaAndSizes.

В каталозі даних (DataDirectory) кожен елемент – структура, що складається з покажчика та розміру. Кожна із записів має певну роль. Так, IMAGE_DIRECTORY_ENTRY_EXPORT – покажчик на таблицю експортованих функцій і даних (зустрічається в основному в DLL). Покажчик IMAGE_DIRECTORY_ENTRY_SECURITY представляє особливий інтерес. Він вказує на Certificate Table. Таблицю, що знаходиться на диску. При IMAGE_DIRECTORY_ENTRY_SECURITY! = 0 практично виключена ймовірність того, що файл є шкідливим. Також, якщо IMAGE_DIRECTORY_ENTRY_COM_DESCRIPTOR! = 0, то файл являє собою .NET додаток, що складається з байт-коду, що також знижує ймовірність шкідливості файлу.

Після опціонального заголовка йде таблиця секцій (Section Table), яка містить NumberOfSections секцій. Як правило, секції мають наступний порядок. Спочатку описана секція з кодом, після – кілька секцій ініціалізованих даних, а після – секція неініціалізованих даних. Кожна секція має ім'я (не має ніякого значення при завантаженні файлу), адреса початку секції в пам'яті і в файлі (вирівняні), віртуальну і фізичну довжину секції (VirtualSize і SizeOfRawData). Віртуальні адреси секцій повинні йти підряд, не накладаючись і не утворюючи пропусків. Поле Characteristics являє собою права доступу до секції і особливості її завантаження. Варто розглянути як секції, так і їх атрибути. Для детекції вірусів і аналізу також корисні поля таблиці експорту, імпорту.

Таблиця експорту потрібна для зв'язку експортованих функцій з їх адресами. Містить покажчики на 3 таблиці: таблиця імен, ординалів, адрес.

Таблиця імпорту співвідносить виклики функції динамічних бібліотек з їх адресами. Існує 3 режими імпорту: стандартний, зв'язуючий (bound import), відкладений (delay import). Для аналізу PE файлу варто аналізувати назви бібліотек і функцій (API). Тут виникає питання про спосіб подання цієї інформації. Вектор ознак повинен бути фіксований, що накладає певні обмеження. З одного боку ми повинні вибирати часто зустрічі API, з іншого – ті, які статистично значущі при відділенні шкідливого ПО від чистих файлів.

Виклавши загальну картину структур PE файлів, ми можемо приступити до формування атрибутів і їх відбору. Стають помітні такі властивості одержуваних ознак. Ознаки суворо діляться по ролі і значенню структури з якої були вилучені:

1) чисельна ознака (кількість секцій, символів, розмір опціонального заголовка, адреса точки входу);

2) прапор присутності (секції, API таблиці імпорту);

3) значення функції від послідовності байт (ентропія секцій).

Тут не розглядаються функції від порівняно довгих послідовностей байт, так, наприклад, ентропію всього файлу, розподіл певних символів в файлі або подання файлу в вигляді зображення. Метод повинен швидко працювати навіть на слабких машинах, тому завдання полягає в відборі найцінніших з точки зору визначення шкідливого ПО ознак.

Тут стають видні і головні проблеми статичного аналізу. Одна з основних – зашифровані, упаковані секції. Віруси шифрують свій код виконання так, що він більше виявляється не помітний з точки зору статичного аналізатора. Для вирішення цієї проблеми або додають дешифратор, або ґрунтуються на статистиці по цій секції. У першому випадку завдання постає окремим дослідженням, яке не буде розглянуте тут. Більшість дешифраторів використовують евристики і перебирають відомі методи шифрування, а є ті, які дешифрують при виконанні бінарного файлу. При підрахунку ентропій ми опираємося на припущенні, що хороше шифрування вирівнює частоту байт секцій, тим самим збільшуючи її ентропію.

1.3. Евристичні методи

Евристичний аналізатор – це програма, яка аналізує код об'єкта, що перевіряється і за непрямими ознаками визначає, чи є об'єкт шкідливим [23]. Причому на відміну від сигнатурного методу евристичний може детектувати як відомі, так і невідомі віруси.

Робота аналізатора, зазвичай, починається з пошуку в коді підозрілих ознак (команд), характерних для вірусів. Цей метод називається статичним аналізом. Наприклад, багато шкідливого програмного забезпечення шукає виконувані програми, відкриває знайдені файли і змінює їх. Евристичний аналізатор переглядає код програми та, зустрівши підозрілу команду, збільшує «лічильник підозрілості» для цього додатка. Якщо після перегляду всього коду значення лічильника перевищує задане граничне значення, то об'єкт визнається підозрілим. Розуміється це може бути застосовано не тільки до програм зберігаються на комп'ютері [24], але і до інших файлів, які передаються по мережі, що зберігаються на поштових серверах, і т. д. Наприклад, як потенційно небезпечні маркуються вкладення з відсутніми розширеннями файлів. Команди, без дозволу відкриваючі адресну книгу Outlook Express, створюючі ключі реєстру або активуючі мережеві порти, теж ідентифікуються як потенційні шкідники.

Евристичні методи виявляються надзвичайно успішними в разі дуже коротких програмних частин в завантажувальному секторі: якщо, наприклад, програма робить запис в сектор 1, доріжку 0, сторону 0, то це призводить до зміни розділу накопичувача. Але, крім допоміжної програми FDISK ця команда більше ніде не використовується, і тому в разі її несподіваної появи мова (з великою ймовірністю) йде про завантажувальний вірус.

Евристичні алгоритми здатні виявити нові або мутовані віруси, для яких поки немає вірусних визначень. Оскільки цей метод пошуку базується на емпіричних припущеннях, повністю виключити помилкові спрацьовування (false positive) не можна.

Перевагою цього методу є простота реалізації і висока швидкість роботи, проте рівень виявлення нових шкідливих кодів залишається досить низьким, а ймовірність помилкових спрацьовувань високою [23].

1.4. Інформаційний аналіз

Метод полягає у виділенні особливостей великих наборів даних, тобто необхідно визначити найбільш компактний опис нормальної поведінки на основі перебору всіх шаблонів, що відображають нормальну поведінку. На основі певних особливостей необхідно зробити узагальнення для визначення нормальних шаблонів, пропущених в тренувальних даних.

Для реалізації даного методу використовувалася система RIPPER (Repeated Incremental Pruningto Produce Error Reduction) – система, побудована на підставі навчальних правил і використовувана для вирішення задач класифікації. Тренування полягає у визначенні зразків, що складаються з множини атрибутів, що описують об'єкт, що класифікується, і цільового класу, до якого належить об'єкт. На підставі багатьох прикладів RIPPER виділяє правила в такій формі:

ClassA: - attrib1 = x, attrib5 = y,

ClassB: - attrib2 = z,

ClassC: - true (за замовчуванням).

Для задач виявлення порушника даний метод можна застосовувати тільки в тому випадку, якщо відомо повна множина прикладів анормальних класів, на підставі яких тренується система. Лі адаптував систему RIPPER для виявлення аномалій з використанням правил, пророкуючих системні виклики, для коротких послідовностей трас програми.

Для кожної програми був використаний список унікальних послідовностей довжини 10, на підставі яких були створені зразки для тренування RIPPER. Кожна послідовність була перетворена в приклад RIPPER перетворенням всіх системних викликів, за винятком останнього, в атрибути.

На підставі останнього виклику визначається клас, відповідний послідовності. Такі пари атрибут/мета створюються для тестових трас. При цьому використовуються всі послідовності системних викликів, а не тільки приклади унікальних трас. RIPPER створює гіпотези на основі тренувальних зразків – список правил, що описують нормальну поведінку системи.

Для кожного правила RIPPER визначає оцінку порушення на основі кількості ситуацій, в яких правило було застосовано коректно в тренувальних даних. Наприклад, для правила, чиї умови були задоволені M раз під час тренування, і передбачення було зроблено коректно T раз, оцінку порушення можна визначити як 100 * M/T. Середнє значення оцінки порушень може бути використано для обчислення рангу траси (це не може бути застосовано для online виявлення вторгнень). Отже, доцільно використовувати відхилення від імовірної траси (використовувалося значення 80%) як «пропуск» з використанням механізму, описаного для інших методів.

1.5. Методи машинного навчання для виявлення шкідливого ПО в мережах.

Методи машинного навчання також застосовуються при виявленні аномалій в мережах.

В [25] автори запропонували заміну стандартному модулю виявлення в системі Snort деревами рішень. Експерименти були проведені на наборі даних DARPA [26] і показали збільшення швидкості обробки pcap-файлів, що використовуються для аналізу мережевих пакетів, в середньому на 40,3 % в порівнянні зі стандартним модулем.

Одним з найбільш часто використовуваних підходів для виявлення вторгнень є байєсовські мережі. Байєсова мережа – це модель, яка кодує імовірнісні відносини між розглянутими подіями (перемінними) і надає певний механізм для обчислення умовних ймовірностей їх настання [27].

Окремий випадок цієї моделі – наївний байєсовський класифікатор (байєсовський метод) зі строгими припущеннями про незалежність вхідних змінних. Класифікатор дозволяє оцінити апостеріорну ймовірність приналежності примірника заданому класу на основі безумовної теореми Байєса.

В [28] застосовуються псевдобайєсовські оціночні функції для визначення апріорних і апостеріорних ймовірностей нових атак. Наївний байєсовський класифікатор використовується для класифікації мережевих зразків. В [28] стверджується, що внаслідок властивостей запропонованого методу системі не потрібні попередні знання про шаблони нових атак.

Запропонована в [28] система ADAM складається з трьох частин: модуля передобробки, інтелектуального модуля і модуля класифікації.

Перший модуль збирає дані з TCP/IP трафіку і витягує інформацію з кожного з'єднання у відповідності з наступною схемою (time_stamp, src_ip, src_port, dst_ip, dst_port, conn_status), де time_stamp – тимчасова мітка, src_ip – IP-адреса джерела, src_port – порт джерела, dst_ip – IP-адреса призначення, dst_port – порт призначення, conn_status – стан з'єднання.

Інтелектуальний модуль застосовує правила асоціації X → Y до записів з'єднань, де X і Y відповідно передумова та постумова правил, описаних усередині ядра системи. Цей модуль працює в двох режимах: режимі навчання і режимі виявлення. У першому режимі відбувається побудова профілів нормального поведінки користувачів і системи, і генеруються правила асоціації, які будуть використовуватися для навчання модуля класифікації. У другому режимі інтелектуальний модуль отримує раніше не зустрічавші правила асоціації, які відрізняються від профілю. Представлені три рівня роботи інтелектуального модуля: відокремлений рівень, рівень домену, а також рівень вибірки ознак. Перший рівень має два режими: статичний і динамічний аналіз. Перший режим використовується під час нормальної роботи системи, коли створюється профіль для поведінки системи. Другий режим використовує метод ковзаючого вікна для поетапного аналізу правил асоціації. На рівні домену система відстежує IP-адреси відправника і одержувача.

З'єднання вважається підозрілим, якщо ці адреси належать одній і тій же підмережі. На рівні ознак система збирає зліпки мережевого поведінки кожні три секунди для їх подальшого аналізу. Також існує більш тривалий процес вибірки ознак, який повинен кожні 24 години виявляти повільно відбуваючі і тривалі за часом аномалії.

Модуль класифікації співвідносить нові правила асоціації до нормальних або аномальних подій. Деякі з аномальних подій можуть згодом бути класифіковані як атаки. В роботі використовується функція щільності розподілу Діріхле. Вона дозволяє оцінити значення для таблиць контингентності з великою кількістю нулів. В даних таблицях колонки відповідають атрибутам зразків з'єднань, а рядки вказують на клас навчальних даних, який є або нормальним з'єднанням, або типом атаки. Таблиця будується таким чином, що значення в комірці на перетині і-го рядка і j-ой колонки вказує на кількість зразків в навчальній вибірці, для яких представлені і-ий клас з'єднання і j-ий атрибут. Крім всіх класів, представлених в навчальній вибірці, додається рядок в таблиці спряженості для позначення додаткового класу, що представляє новий тип атак. Значення в комірках нового рядка заповнюються нулями. Після чого застосовуються метод Діріхле, що згладжує особливості в новому рядку, і наївний байєсовський класифікатор.

Автори відзначають дві основні переваги їх системи – це робота в режимі реального часу і виявлення аномалій.

Метод на основі MAP-сплайнів дозволяє побудувати досить точну апроксимацію поведінки звичайного користувача або зловмисника по заданих параметрах. З цією метою вибирається набір базисних функцій, і знаходяться коефіцієнти в лінійному розкладі по заданому базису і навчальним векторах. В [29] автори пропонують будувати класифікатори на основі MAP-сплайнів для розпізнавання 5 обраних типів мережевих з'єднань, а також наводять показники точності виявлення на даних з набору DARPA 1998 для MAP-сплайнів, нейронних мереж і методу опорних векторів. Для тестування використовувалася вибірка потужністю 6890 записів, для навчання застосовувалося 5092 записи.

З наведених результатів можна зробити висновок, що MAP-сплайни і метод опорних векторів показують велику ефективність у розпізнаванні 4 класів сполук в порівнянні з нейронними мережами.

Серед інших підходів машинного навчання варто відзначити алгоритми кластеризації [30, 31] і регресії [32].

1.6. Висновки до розділу 1

Проведений загальний огляд методик і підходів до задачі статичного аналізу шкідливого програмного забезпечення показав, що для реалізації метода в програмному продукті з використанням сучасних алгоритмів машинного навчання необхідно узагальнення отриманих результатів досліджень та отримання найбільш важливих атрибутів для завдання виявлення шкідливого ПЗ на типовому для користувачів наборі файлів в умовах обмежених ресурсів і часу.

РОЗДІЛ 2

АЛГОРИТМИ МАШИННОГО НАВЧАННЯ

2.1. Дерево рішень

Алгоритм «дерево рішень» (decision tree) тут згадаємо лише тому, що на ньому грунтуються більш складні, представлені тут алгоритми, і, слід розуміти принцип побудови базових алгоритмів для використання їх в системі.

Дерево рішень, в загальному випадку, є способом подання правил в ієрархічній послідовній структурі, де кожному об'єкту відповідає один вузол, що дає рішення.

Як правило розглядають бінарне дерево рішень.

Введемо позначення:

V – множина вузлів в дереві. βv – предикат, функція в вершині v, яка повертає {0, 1}. Y – множина класів, в нашому випадку – 2 (pure – чистий файл, infected – шкідливий).

Вибір класу виконується наступним чином:

$\forall v \in V_{внутр} \rightarrow$ предикат $\beta v : X \rightarrow \{0, 1\}, \beta \in B$

$\forall v \in V_{лист} \rightarrow$ ім’я класу $cv \in Y$

На практиці використовуються одномірні предикати, які порівнюють значення одного з ознак з порогом.

Для побудови дерева існує багато алгоритмів. Конкретний метод побудови вирішального дерева визначається:

1) видом предикатів в вершинах;
2) функціоналом якості (для прийняття рішення про влаштування навчальної вибірки в розглянутій вершині. Як правило використовується критерій Джині або ентропійний критерій);
3) критерієм зупинки.
4) методом обробки пропущених значень;

5) методом стрижки (видалення деяких вершин з метою зниження складності та підвищення узагальнюючої здатності).

Процес пошуку вузла представлений на рис. 2.1. з відповідним лістингом – рис. 2.2.

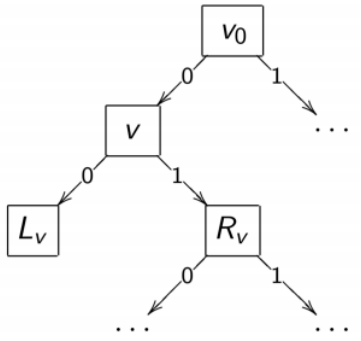

Рис. 2.1. Дерево рішень

```
while v ∈ V_внутр do
    if β_v(x) = 1 then
        go to right;
        v := R_v;
    else
        go to left;
        v := L_v;
    end
    return c_v
end
```

Рис. 2.2. Пошук вузла

Можливість обробляти пропущені значення корисна і в нашій задачі. Вибрані імена секцій або імпортовані назви бібліотек, функцій можуть бути не знайдені. В такому випадку, для такого файлу частина значень, ознак буде пропущена. Вирішальне дерево може бути навчене і на таких даних. А саме (тут U – множина об'єктів навчання, G_{ain} – деякий вибраний критерій для максимізації в кожній вершині, S_v – функція, при при 0 – L_v, 1 – R_v)

На стадії навчання:

$b_v(x_i)$ не визначене => xi виключається з U для $Gain(b_v, U)$

$q_{vk} = \frac{|U_k|}{|U|}$ – оцінка ймовірності k-й гілки, $v \in V_{внутр}$

$P(y|x,v) = \frac{1}{|U|}\sum_{x_i \in U}[y_i = y]$ для всіх $v \in V_{ліст}$

На стадії класифікації:

$b_v(x)$ визначено => з дочірньої $s = S_v(b_v(x))$ взяти $P(y|x, v) = P(y|x, s)$

$b_v(x)$ не визначене => пропорційний розподіл:

$$P(y|x,v) = \sum_{k \in D_v} q_{vk} P(y|x, S_v(k))$$

Остаточне рішення – найбільш ймовірний клас:

$a(x) = arg\ max(y \in Y)\ P(y|x, v_0)$

Для побудови дерева, як правило, використовується жадібний алгоритм з оптимізацією в кожному вузлі заданого функціонала якості (зазвичай розглядають критерій Джині, ентропійний критерій). Вибір функціоналу якості також сильно впливає на кінцевий результат.

Як готовий алгоритм, вирішальне дерево сьогодні не використовується. Алгоритм схильний до перенавчання, сильний вплив на шуми. Також, не завжди дерево будує найкращий розподіл (знаменитий приклад із завданням побудови XOR функції). На рис. 2.3 відповідний приклад.

Велика увага приділяється ансамблям дерев рішень. Кожне з дерев здатне до перенавчання, до сильного впливу на шумові об'єкти, але використання їх ансамблів (бустінг, беггінг) перетворює в один з найбільш успішних і застосовних на практиці інструментів машинного навчання.

Побічний ефект від побудови дерева рішень полягає в можливості оцінки важливості ознак. Важливість ознаки розраховується на підставі того, наскільки поліпшується обрана метрика при поділі вибірки за ознакою в черговому вузлі. Це може дати уявлення які ознаки для побудови дерева виявилися найбільш вирішальними, а які не були використані. Потрібно розуміти, що важливість ознак оціночна.

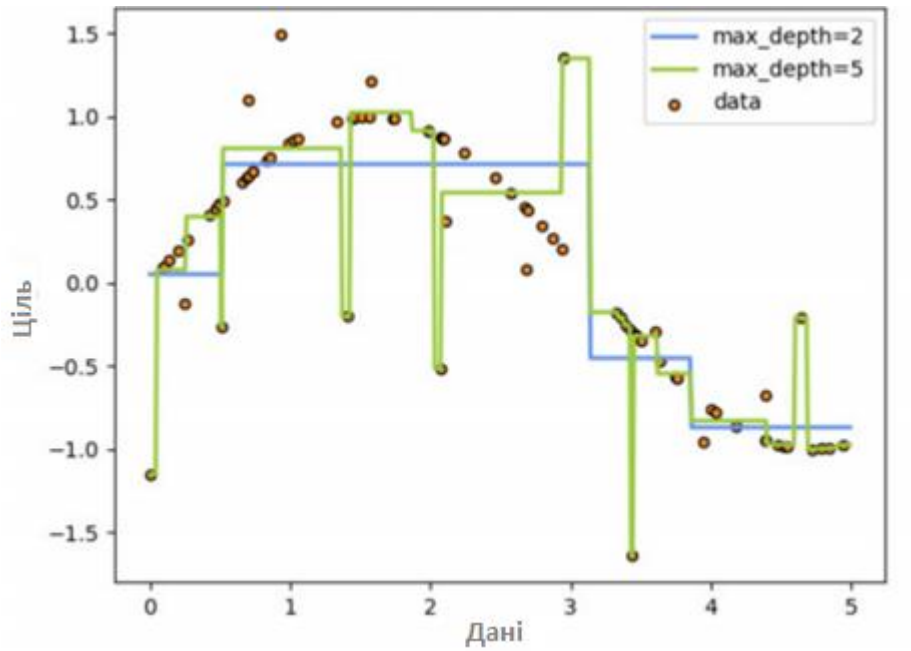

Рис. 2.3. Схильність до перенавчання вирішального дерева

Далі будуть наведені алгоритми, засновані на деревах рішень. Багато властивостей, зазначені тут, вірні і для ансамблів. Наприклад, ми як і раніше можемо отримати важливість ознак через усереднення по кожному дереву.

2.2. Випадковий ліс

Випадковий ліс (Random forest) являє собою ансамбль дерев рішень, які навчаються незалежно [35]. Кожен з вирішальних дерев, як правило, роблять простим. Для прийняття остаточного рішення вибирається значення, за яке проголосували більшість дерев. Кожне дерево в окремо дає низьку якість, але за рахунок їх великої кількості результат виходить хорошим. На рис. 2.4. можна бачити приклад випадкового лісу з двох дерев.

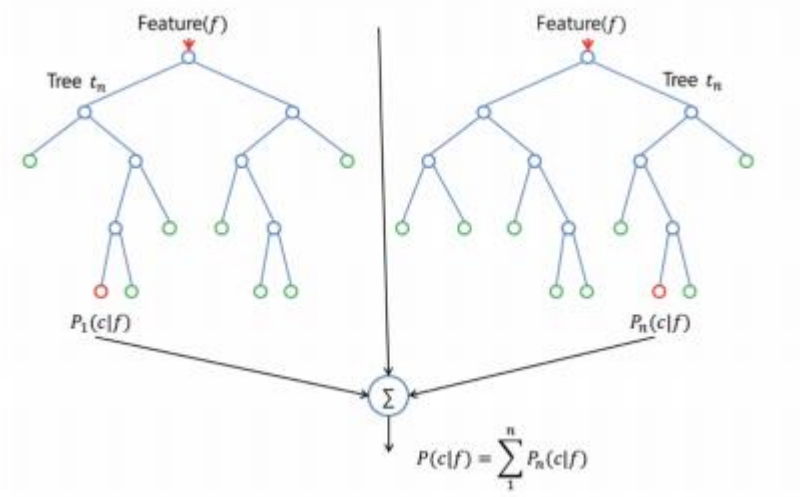

Рис. 2.4. Випадковий ліс з 2 дерев

У побудові дерев на даних є свої особливості. Зазначимо традиційний підхід в побудові дерева. Нехай N – кількість прикладів на навчанні. тоді:

1) вибирається підвибірка навчальної вибірки розміру N (можливо, з поверненням)

2) будується дерево. При пошуку розбиття при побудові кожного вузла розглядаються всі ознаки, а тільки частина з них (sklearn реалізація – корінь з кількості ознак)

3) дерево будується до повного вичерпання підвибірки, або на основі будь-яких інших критеріїв

Кожне з дерев намагаються зробити якомога простіше. При прийнятті рішення вибирається клас, за який проголосувало більшість дерев рішень.

Випадковий ліс дає порівняно погані результати при поділі вибірки в метричному просторі. У тому випадку, коли очікується деякий «правильний» вигляд розділяючої гіперплощини. Проте, у випадкового лісу є велике число переваг, які можуть використовуватися в розглянутій задачі визначення шкідливості файлу. Алгоритм не оперує метриками, що дозволяє вільно працювати з ознаками різної природи. Можливо обробляти дані з пропущеними значеннями ознак. Висока швидкість роботи алгоритму, незалежна побудова вирішальних дерев.

2.3. Градієнтний бустінг

При побудові бустінга, базові алгоритми будуються послідовно, а не паралельно, компенсуючи помилку на попередній ітерації [36]. У загальному вигляді алгоритм навчання градиентного бустінга представлений нижче (рис. 2.5.). Тут L (a, y) – довільна функція втрат, b_i – і-ий базовий алгоритм. f_i являє собою і-е наближення.

Data: навчальна вибірка X^l, параметр T;

Result: базові алгоритми і їх ваги $\alpha_t b_t$, $t = 1, \ldots, T$;

ініціалізація: $f_i := 0$, $i = 1, \ldots, l$;

for $t \leftarrow 1$ **to** T **do**

базовий алгоритм, що наближує градієнт:

$b := \arg\min_b \sum_{i=1}^{l} (b(x_i) + L'(f_i, y_i))^2$;

задача одновимірної мінімізації:

$a_t := \arg\min_{a>0} \sum_{i=1}^{l} L(f_i + a b_t(x_i), y_i)$;

оновлення вектора значень на об'єктах вибірки:

$f_i := f_i + \alpha_t b_t(x_i)$;

$i = 1, \ldots, l$;

end

Рис. 2.5. Алгоритм градієнтного бустінга

Маємо лінійну комбінацію базових алгоритмів: $a(x) = \sum_{t=1}^{T} a_t b_t(x)$. Це і є алгоритм градієнтного бустінга. Являє лінійну комбінацію базових алгоритмів. Весь фокус полягає в тому, як побудувати такий набір алгоритмів і підібрати ваги.

Функціонал якості з довільною функцією втрат L(a, y):

$$Q(a,b,X^l) = \sum_{i=1}^{l} L(\sum_{t=1}^{T-1} a_t b_t(x_i) + ab(x_i), y_i) \to min_{a,b}$$

На кожній ітерації, при пошуку чергового базового алгоритму, мінімізується даний функціонал Q(α, b; X^l)

У разі градиентного бустінга на вирішальних деревах, очевидно, базові алгоритми являють собою дерева рішень. Параметр T – кількість дерев, що вибудовуються.

Найбільш популярні реалізації градієнтного бустінга на вирішальних деревах: XGBoost [17], LightGBM [37], CatBoost [38].

Xgboost представляє лише найбільш вдалу реалізацію ідей, запропонованих раніше.

LightGBM алгоритм реалізовувався з метою прискорити існуючі реалізації бустінга, а саме, XGBoost.

Основний час на побудову алгоритму йде на пошук кращої точки розбиття тренувальних даних у вузлі за певною ознакою. У XGBoost реалізовані 2 механізму:

1) для знаходження кращого розбиття ознаки пресортуются і знаходиться найкраще розбиття;

2) алгоритм на основі гістограми. ознаки об'єктів розбиваються на групи.

LightGBM пропонує 2 техніки для багаторазового прискорення швидкості роботи алгоритму:

1) gradient-based One-Side Sampling (GOSS). Для побудови чергового дерева використовуються не всі об'єкти. Видаляються з розгляду об'єкти з малими градієнтами;

2) exclusive Feature Bundling (EFB). Упаковка ознак. При великій кількості ознак дані з високою ймовірністю розріджені і є можливість зменшити кількість ефективних ознак.

LightGBM пропонує 2 техніки для багаторазового прискорення швидкості роботи алгоритму:

3) gradient-based One-Side Sampling (GOSS). Для побудови чергового дерева використовуються не всі об'єкти. Видаляються з розгляду об'єкти з малими градієнтами;

4) exclusive Feature Bundling (EFB). Упаковка ознак. При великій кількості ознак дані з високою ймовірністю розріджені і є можливість зменшити кількість ефективних ознак.

У статті про LightGBM було проведено порівняння алгоритму з XGBoost як за швидкістю роботи, так і за якістю на декількох наборах даних. Розбиралися завдання класифікації і ранжування. Було продемонстровано від 2-20 кратне збільшення швидкості роботи алгоритму бустінга при тій же точності на відкладеній вибірці.

CatBoost ставить своїм завданням ефективну роботу з категоріальним ознаками (catboost == categorial boosting), а також пропонує нову схему для підрахунку значень у вузлах, що дозволяє зменшити перенавчання. Ще одне нововведення в тому, що при побудові чергового дерева поділ в вузлі може відбуватися за сукупністю двох категоріальних ознак. Підтримка GPU. У статті проводиться порівняння з XGBoost, LightGBM, H2O реалізаціями. Показано, що CatBoost навіть з параметрами за замовчуванням дозволяє домогтися кращої якості (Logloss) на 8 розглянутих наборах даних. Показані результати порівняння часів передбачення алгоритмів CatBoost, XGBoost, LightGBM. CatBoost виявився на порядок швидше як в однопотоковому режимі, так і в багатопотоковому. CatBoost знаходиться у вільному доступі і активно розвивається.

В силу того, що в задачі присутні категоріальні ознаки, доцільно досліджувати також і цю реалізацію.

2.4. Метод опорних векторів

Метод опорних векторів (Support Vector Machine – SVM) відноситься до групи граничних методів. Ця група визначає класи за допомогою кордонів областей.

За допомогою даного методу розв’язуються завдання бінарної класифікації. В основі методу лежить поняття площин рішень. Площина (plane) рішення розділяє об'єкти з різною класової приналежністю.

На рис. 2.6. наведено приклад, в якому беруть участь об'єкти двох типів. Розділяюча лінія задає кордон, праворуч від якої – всі об'єкти типу brown (коричневий), а зліва – типу yellow (жовтий).

Новий об'єкт, що потрапляє направо, класифікується як об'єкт класу brown або – як об'єкт класу yellow, якщо він розташувався ліворуч від поділяючої прямою. У цьому випадку кожен об'єкт характеризується двома вимірами.

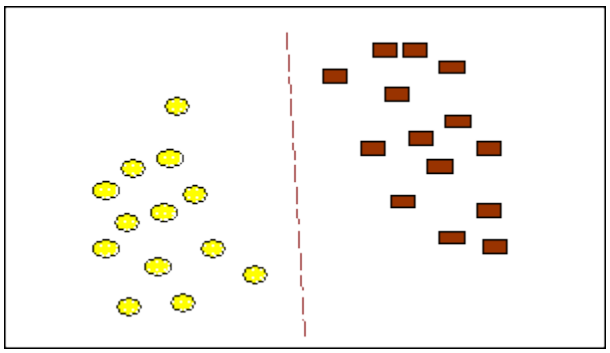

Рис. 2.6. Поділ класів прямою лінією

Мета методу опорних векторів – знайти площину, що розділяє дві множини об'єктів; така площина показана на рис. 2.7. На цьому малюнку множину зразків поділено на два класи: жовті об'єкти належать класу A, коричневі – класу B.

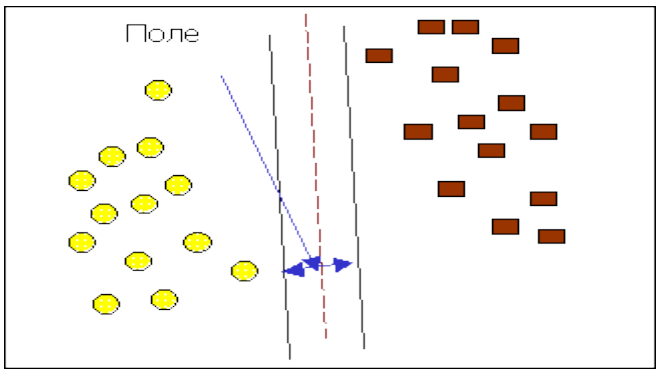

Рис. 2.7. До визначення опорних векторів

Метод відшукує зразки, що знаходяться на кордонах між двома класами, тобто опорні вектора, вони зображені на рис. 2.8.

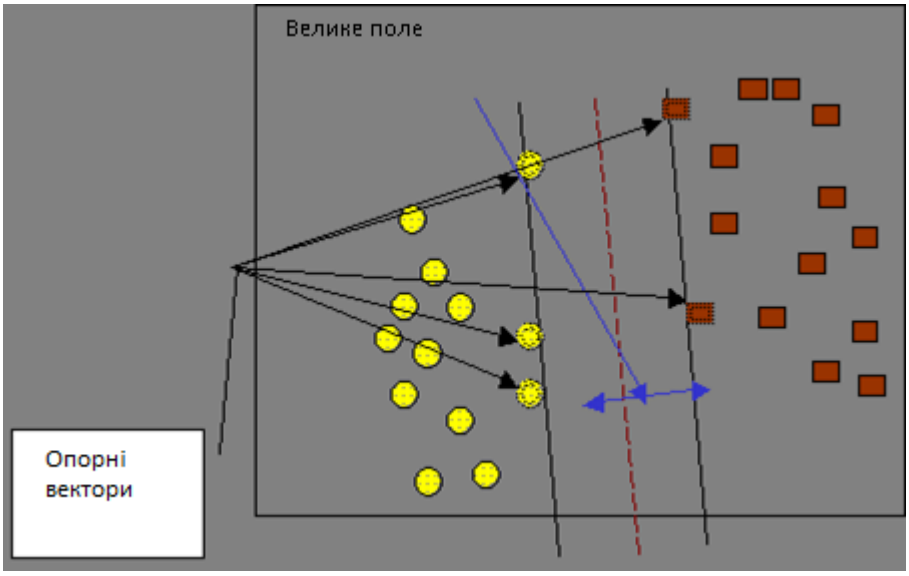

Рис. 2.8. Опорні вектори

Опорними векторами називаються об'єкти множини, що лежать на кордонах областей. Класифікація вважається хорошою, якщо область між кордонами порожня. На рис. 2.8 показано п'ять векторів, які є опорними для даної множини.

Рішення задачі бінарної класифікації за допомогою методу опорних векторів полягає в пошуку деякої лінійної функції, яка правильно поділяє набір даних на два класи. Розглянемо задачу класифікації, де число класів дорівнює двом.

Завдання можна сформулювати як пошук функції f(x), що приймає значення менше нуля для векторів одного класу і більше нуля – для векторів іншого класу. В якості вихідних даних для вирішення поставленого завдання, тобто пошуку класифікуючої функції f(x), дано тренувальний набір векторів простору, для яких відома їх приналежність до одного з класів. Сімейство класифікуючих функцій можна описати через функцію f(x). Гіперплоскость визначена вектором a й значенням b, тобто f(x) = ax + b. Рішення даного завдання проілюстровано на рис. 2.9.

В результаті рішення задачі, тобто побудови SVM-моделі, знайдена функція, що приймає значення менше нуля для векторів одного класу і більше нуля – для векторів іншого класу. Для кожного нового об'єкта негативне або позитивне значення визначає приналежність об'єкта до одного з класів.

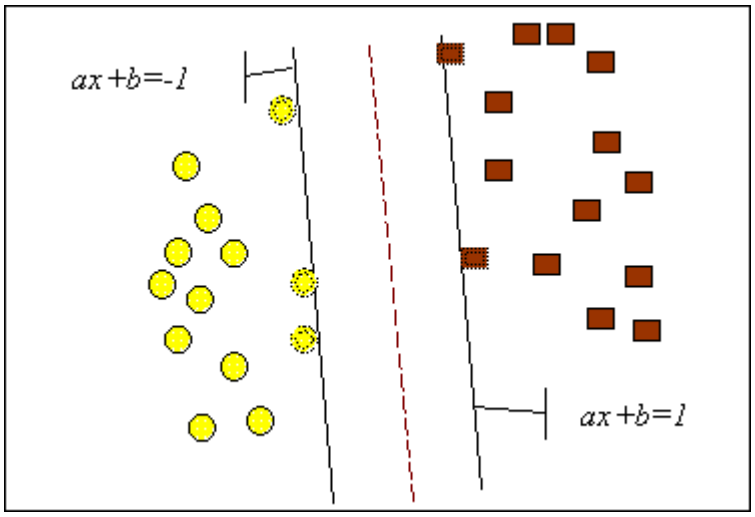

Рис. 2.9. Лінійний SVM

Найкращою функцією класифікації є функція, для якої очікуваний ризик мінімальний. Поняття очікуваного ризику в даному випадку означає очікуваний рівень помилки класифікації.

Безпосередньо оцінити очікуваний рівень помилки побудованої моделі неможливо, це можна зробити за допомогою поняття емпіричного ризику. Однак слід враховувати, що мінімізація останнього не завжди призводить до мінімізації

очікуваного ризику. Цю обставину слід пам'ятати при роботі з відносно невеликими наборами тренувальних даних.

Емпіричний ризик – рівень помилки класифікації на тренувальному наборі.

Таким чином, в результаті рішення задачі методом опорних векторів для лінійно поділюваних даних отримуємо функцію класифікації, яка мінімізує верхню оцінку очікуваного ризику.

Однією з проблем, пов'язаних з вирішенням завдань класифікації досліджуваних методом, є та обставина, що не завжди можна легко знайти лінійну границю між двома класами.

У таких випадках один з варіантів – збільшення розмірності, тобто перенесення даних з площини в тривимірний простір, де можливо побудувати таку площину, яка ідеально розділить безліч зразків на два класи. Опорними векторами в цьому випадку будуть служити об'єкти з обох класів, які являються екстремальними.

Таким чином, за допомогою додавання так званого оператора ядра і додаткових розмірностей, знаходяться границі між класами у вигляді гіперплощин.

Однак слід пам'ятати: складність побудови SVM-моделі полягає в тому, що чим вище розмірність простору, тим складніше з ним працювати. Один з варіантів роботи з даними високої розмірності – це попереднє застосування будь-якого методу зниження розмірності даних для виявлення найбільш істотних компонент, а потім використання методу опорних векторів.

Як і будь-який інший метод, метод SVM має свої сильні і слабкі сторони, які слід враховувати при виборі даного методу.

Недолік методу полягає в тому, що для класифікації використовується не вся множина зразків, а лише їх невелика частина, яка знаходиться на границях.

Перевага методу полягає в тому, що для класифікації методом опорних векторів, на відміну від більшості інших методів, досить невеликого набору даних. При правильній роботі моделі, побудованої на тестовій множині, цілком можливе застосування даного методу на реальних даних.

Метод опорних векторів дає змогу отримати функцію класифікації з мінімальною верхньою оцінкою очікуваного ризику (рівня помилки класифікації), використовувати лінійний класифікатор для роботи з нелінійно розділяючими даними, поєднуючи простоту з ефективністю.

2.5. Метод "найближчого сусіда"

Слід відразу зазначити, що метод "найближчого сусіда" ("nearest neighbour") відноситься до класу методів, робота яких грунтується на зберіганні даних в пам'яті для порівняння з новими елементами. При появі нового запису для прогнозування знаходяться відхилення між цим записом і подібними наборами даних, і найбільш подібна (або ближній сусід) ідентифікується.

Наприклад, при розгляді нового клієнта банку, його атрибути порівнюються з усіма існуючими клієнтами даного банку (дохід, вік і т. д.). Безліч "найближчих сусідів" потенційного клієнта банку вибирається на підставі найближчого значення доходу, віку і т. д.

При такому підході використовується термін "k-найближчий сусід" ("k-nearest neighbour"). Термін означає, що вибирається k "верхніх" (найближчих) сусідів для їх розгляду в якості множини "найближчих сусідів". Оскільки не завжди зручно зберігати всі дані, іноді зберігається тільки множина "типових" випадків. В такому випадку використовуваний метод називають міркуванням за аналогією (Case Based Reasoning, CBR), міркуванням на основі аналогічних випадків, міркуванням по прецедентах.

Прецедент – це опис ситуації в поєднанні з детальним зазначенням дій, що вживаються в даній ситуації.

Підхід, заснований на прецедентах, умовно можна поділити на такі етапи:

1) збір докладної інформації про поставлену задачу;

2) зіставлення цієї інформації з деталями прецедентів, що зберігаються в базі, для виявлення аналогічних випадків;

3) вибір прецеденту, найбільш близького до поточної проблеми, з бази прецедентів;

4) адаптація обраного рішення до поточної проблеми, якщо це необхідно;

5) перевірка коректності кожного знову отриманого рішення;

6) занесення детальної інформації про новий прецедент в базу прецедентів.

Таким чином, висновок, заснований на прецедентах, являє собою такий метод аналізу даних, який робить висновки щодо даної ситуації за результатами пошуку аналогій, що зберігаються в базі прецедентів.

Даний метод по своїй суті відноситься до категорії "навчання без вчителя", тобто є "самонавчальною" технологією, завдяки чому робочі характеристики кожної бази прецедентів з плином часу і накопиченням прикладів поліпшуються. Розробка баз прецедентів по конкретної предметної області відбувається на природному для людини мовою, отже, може бути виконана найбільш досвідченими співробітниками компанії – експертами або аналітиками, які працюють в даній галузі.

Однак це не означає, що CBR-системи самостійно можуть приймати рішення. Останнє завжди залишається за людиною, даний метод лише пропонує можливі варіанти вирішення і вказує на самий "розумний" з її точки зору.

Переваги методу

1) простота використання отриманих результатів.

2) рішення не унікальні для конкретної ситуації, можливе їх використання для інших випадків.

3) метою пошуку є не гарантовано вірне рішення, а найкраще з можливих.

4) недостатки метода "ближайшего соседа"

5) даний метод не створює будь-яких моделей або правил, узагальнюючи попередній досвід, – у виборі рішення вони ґрунтуються на всьому масиві доступних історичних даних, тому неможливо сказати, на якій підставі будуються відповіді.

6) існує складність вибору міри "близькості" (метрики). Від цього заходу головним чином залежить обсяг безлічі записів, які потрібно зберігати в пам'яті для досягнення задовільної класифікації або прогнозу. Також існує висока залежність результатів класифікації від обраної метрики.

7) при використанні методу виникає необхідність повного перебору навчальної вибірки при розпізнаванні, наслідок цього – обчислювальна трудомісткість.

8) типові завдання даного методу – це завдання невеликої розмірності за кількістю класів і змінних.

2.6. Байєсова класифікація

Альтернативні назви: байєсовське моделювання, байєсівська статистика, метод байєсівських мереж.

Ознайомитися детально з байєсовською класифікацією можна в [39]. Спочатку байєсівська класифікація використовувалася для формалізації знань експертів в експертних системах, зараз байєсівська класифікація також застосовується в якості одного з методів Data Mining.

Так звана наївна класифікація або наївно-байєсовський підхід (naive-bayes approach) є найбільш простим варіантом методу, що використовує байєсовські мережі. При цьому підході вирішуються завдання класифікації, результатом роботи методу є так звані "прозорі" моделі.

"Наївна" класифікація – досить прозорий і зрозумілий метод класифікації. "Наївною" вона називається тому, що виходить з припущення про взаємну незалежність ознак.

Властивості наївної класифікації:

1) використання всіх змінних і визначення всіх залежностей між ними.

2) наявність двох припущень щодо змінних:

– всі змінні є однаково важливими;

– всі змінні є статистично незалежними, тобто значення однієї змінної нічого не говорить про значення іншої.

Більшість інших методів класифікації припускають, що перед початком класифікації ймовірність того, що об'єкт належить тому чи іншому класу, однакова; але це не завжди вірно.

Припустимо, відомо, що певний відсоток даних належить конкретного класу.

Виникає питання, чи можемо ми використовувати цю інформацію при побудові моделі класифікації? Існує безліч реальних прикладів використання цих апріорних знань, які допомагають класифікувати об'єкти. Типовий приклад з медичної практики. Якщо доктор відправляє результати аналізів пацієнта на додаткове дослідження, він відносить пацієнта до якогось певного класу. Яким чином можна застосувати цю інформацію? Ми можемо використовувати її в якості додаткових даних при побудові класифікаційної моделі.

Відзначають такі переваги байєсовських мереж як методу Data Mining:

1) в моделі визначаються залежності між усіма змінними, це дозволяє легко обробляти ситуації, в яких значення деяких змінних невідомі;

2) байєсовські мережі досить просто інтерпретуються і дозволяють на етапі прогностичного моделювання легко проводити аналіз за сценарієм "що, якщо";

3) байесовский метод дозволяє природним чином поєднувати закономірності, виведені з даних, і, наприклад, експертні знання, отримані в явному вигляді;

4) використання байесовских мереж дозволяє уникнути проблеми перенавчання (overfitting), тобто надмірного ускладнення моделі, що є слабкою стороною багатьох методів (наприклад, дерев рішень і нейронних мереж).

Наївно-байесовский підхід має наступні недоліки:

1) перемножать умовні ймовірності коректно тільки тоді, коли всі вхідні змінні дійсно статистично незалежні; хоча часто даний метод показує досить хороші результати при недотриманні умови статистичної незалежності, але

теоретично така ситуація повинна оброблятися більш складними методами, заснованими на навчанні байесовских мереж;

2) неможлива безпосередня обробка безперервних змінних – потрібно їх перетворення до інтервальної шкали, щоб атрибути були дискретними; однак такі перетворення іноді можуть призводити до втрати значущих закономірностей;

3) на результат класифікації в наївно-байєсовському підході впливають тільки індивідуальні значення вхідних змінних, комбінований вплив пар або трійок значень різних атрибутів тут не враховується. Це могло б поліпшити якість класифікаційної моделі з точки зору її прогнозуючої точності, однак, збільшило б кількість перевірених варіантів.

2.7. Висновки до розділу 2

Аналізувавши всі алгоритми машинного навчання, було обрано такі алгоритми: випадковий ліс (Random Forest), XGBoost, LightGBM, CatBoost. Саме ці алгоритми були обрані враховувати різнорідну природу ознак, одержуваних з бінарних файлів. Частина ознак являє собою прапори присутності, частина – статистики. Такі, наприклад, як ентропії секцій, а частина – поля PE структур. Також, ми беремо до уваги швидкість роботи алгоритму. Існуючі дослідження в області детектування шкідливого програмного забезпечення також підтверджують цей вибір.

РОЗДІЛ 3

АНАЛІЗ ТА ПОРІВНЯННЯ РЕЗУЛЬТАТІВ ЗНАХОДЖЕННЯ ВІРУСІВ ЗА ДОПОМОГОЮ МЕТОДІВ МАШИННОГО НАВЧАННЯ

3.1. Отримання даних

Для того, щоб обучити класифікатор потрібно мати розмічені дані. Для нашого завдання потрібно отримати набір чистих та шкідливих файлів PE формату. Проблеми в отриманні чистих файлів немає. Можна використовувати всі файли системи Windows відразу після її установки. Шкідливі ж файли можливо отримати із інших джерел. Був обраний сервіс VirusTotal. Цей сервіс дозволяє отримати рішення 109 (на сьогоднішній день) антивірусів по досліджуваному файлу. Варто зазначити, що для досліджуваного файлу не по всіх антивірусам може бути отриманий вердикт. Це відбувається з декількох причин. Наприклад, файл з'явився в базі зовсім не давно і ще не був оброблений всіма антивірусами. Сервіс дає можливість як завантажити свій файл, так і отримати раніше завантажені файли з аналізом антивірусів, використовуючи платний API. На рис. 3.1 показана статистика по завантажених файлах.

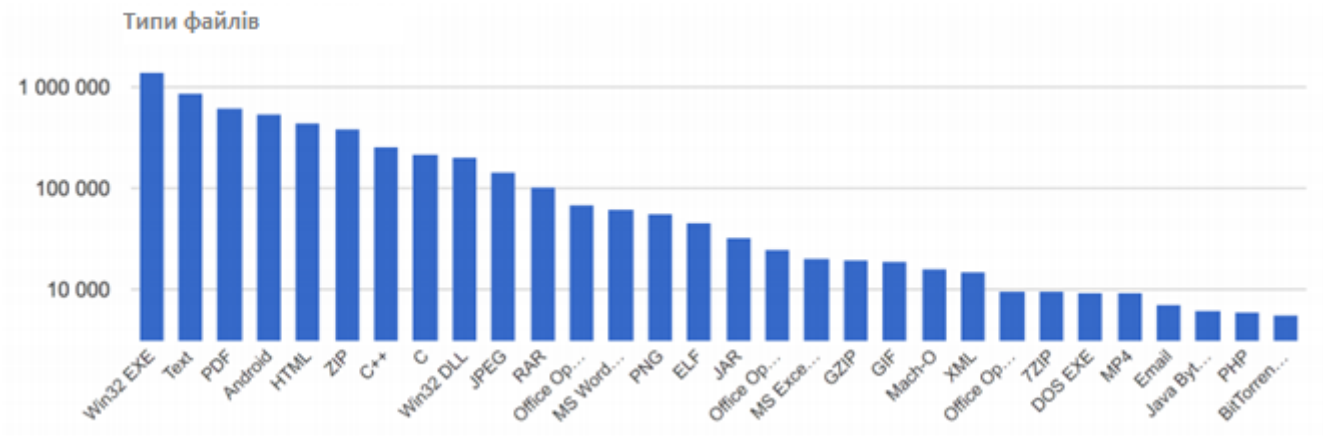

Рис. 3.1. Статистика сервісу VirusTotal по завантажених файлах

Крім PE формату, можливо завантажувати багато інших файлів, таких як текст в різних форматах, архіви. На сьогоднішній день в базі є більше мільйона файлів PE формату.

Було відзначено, що для досліджуваного файлу не завжди є рішення по всім антивірусам. Тому введемо таке поняття, як «індекс шкідливості». «Індексом шкідливості» будемо називати відношення числа антивірусів тих, хто проголосував за шкідливість файлу до кількості проголосувавших антивірусів по цьому файлу. Таким чином, «індекс шкідливості» приймає значення від 0 до 1, де 1 означає, що всі антивіруси, які аналізували файл, вважають його вірусом.

За допомогою програми VirusTotal було отримано 55432 PE-файлів (54299 exe файлів, 1133 dll). Для кожного з них отримані вердикти по антивірусам. На рис. 3.2 показано розподіл рішень антивірусів для отриманих файлів. Для кожного з файлів було підраховано «індекс шкідливості». Розподіл файлів по індексу шкідливості ми можемо побачити на гістограмі.

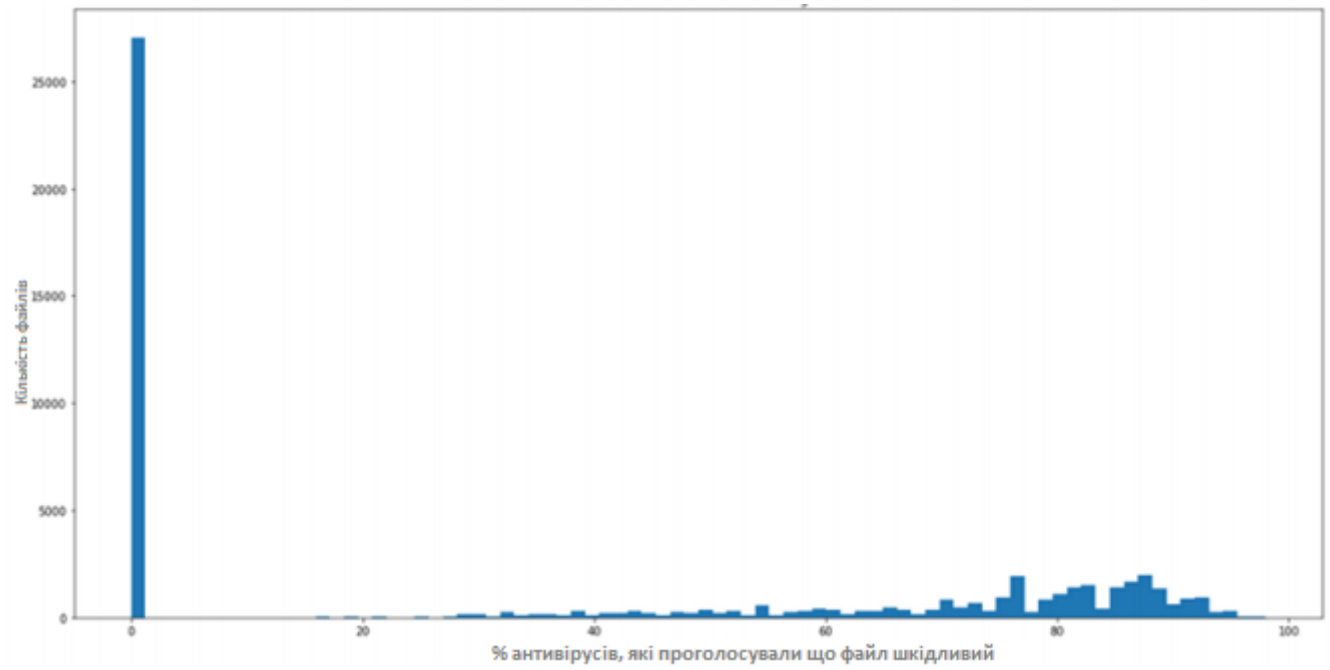

Рис. 3.2. Розподіл файлів по голосам від всіх антивірусів. По осі X відношення числа антивірусів тих, хто проголосував за шкідливість файлу до всіх антивірусів, які голосували

Антивіруси влаштовані по-різному і видають не завжди узгоджуючі вердикти на досліджуваних файлах, що видно на рис. 3.2. Тобто, з одного боку, існують антивіруси, які помилилися, вважаючи чисті файли вірусами (false positive), і, є ті, хто помилився, вважаючи вірус чистим файлом (false negative). На ідеальних детектирующих алгоритмах антивірусів на гістограмі не було б

«хвоста» правіше 100 %. З’являється проблема – які файли вважати шкідливими при навчанні алгоритму машинного навчання.

Для більш точного передбачення були вибрані топ-8 популярних і надійних антивірусів (McAfee, Paloalto, SentinelOne, BitDefender, Avast, Kaspersky, CrowdStrike, Symantec). Індекс шкідливості підраховується тільки для цих антивірусів. Можна побачити відповідну гістограму на рис. 3.3.

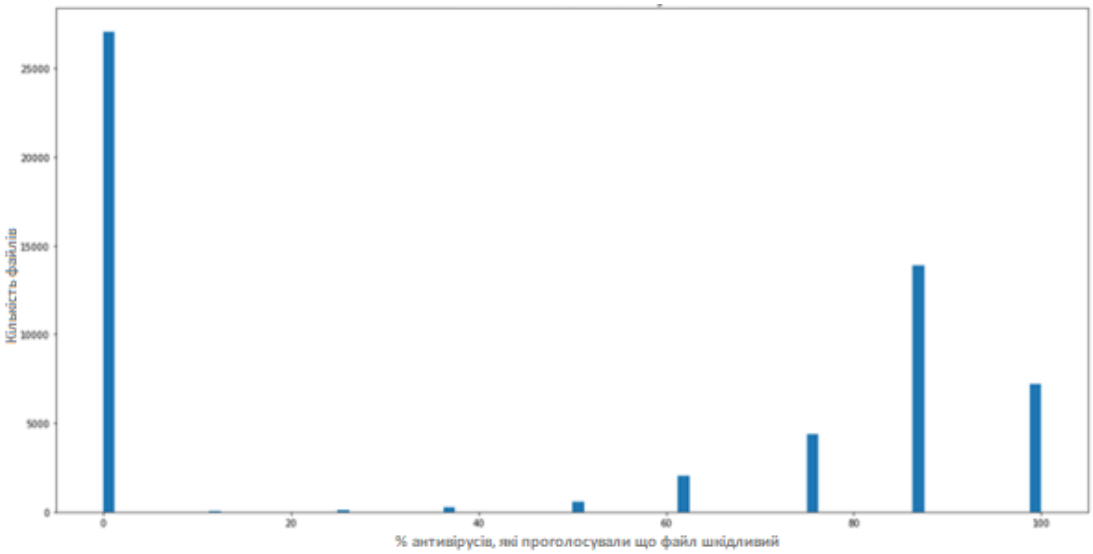

Рис. 3.3. Розподіл файлів по голосам від надійних (топ-8) антивірусів. По осі X відношення числа антивірусів тих, хто проголосував за шкідливість файлу до всіх антивірусів, які голосували

При такому підході близько 5000 файлів як і раніше залишаться сумнівними (сіра зона) – близько половини з найкращих антивірусів проголосували за шкідливість. Вирішимо файл шкідливим, якщо хоча б один з топ-8 антивірусів визначив його таким.

При цьому підході ми намагаємося максимально посилити знаходження вірусів, зменшуючи кількість хибно негативних спрацьовувань. Потрібно зауважити, що більшість алгоритмів класифікації підтримує можливість крім індексу передбаченого класу видавати ймовірність віднесення об'єкта до класу, тобто, ми маємо ще один інструмент впливу на передбачення вже після навчання

алгоритму (вважати шкідливим файлом, якщо ймовірність більше 30%, а не більше 50%).

Таким чином, ми отримуємо наступні дані (розмір підібраний так, щоб дані були збалансовані) в табл. 3.1.

Таблиця 3.1

Статистика за зібраними файлами

Всі файли	55432
Чисті файли	27049
Шкідливі файли	28383

На рис. 3.4 приведена гістограма розподілу розмірів файлів завантажених з VirusTotal як для чистих, так і для шкідливих файлів.

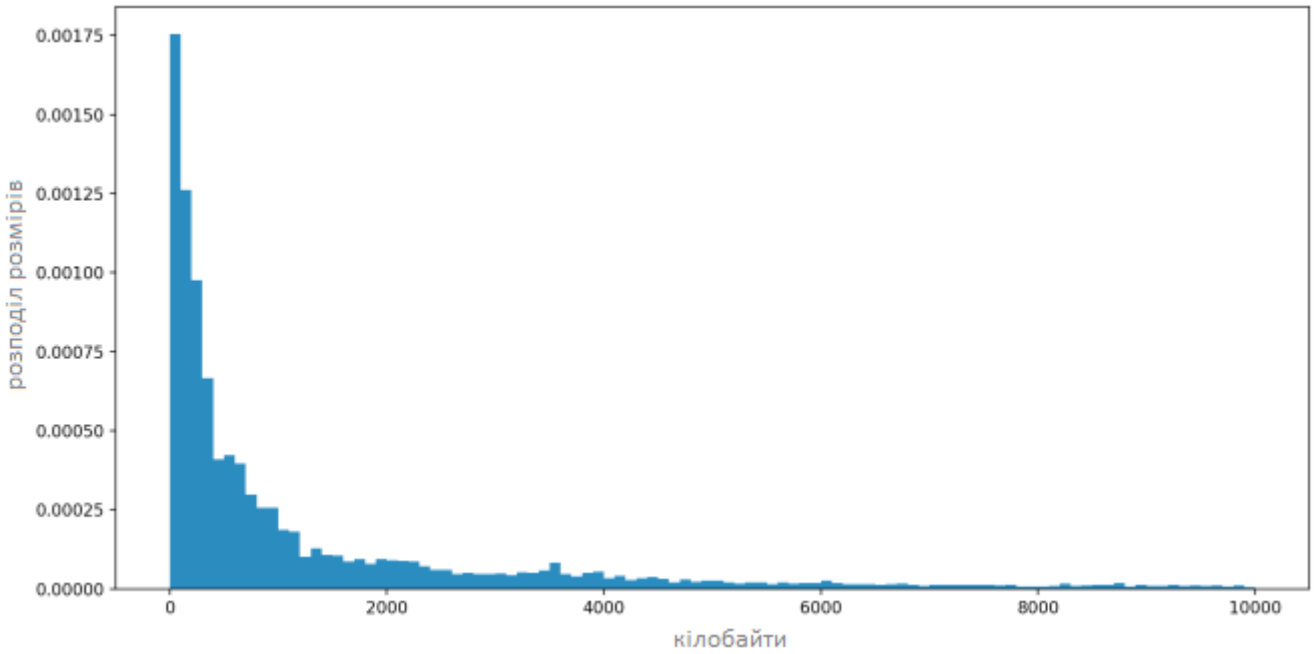

Рис. 3.4. Гістограма розподілу розмірів отриманих файлів. По осі X розмір файлу в кілобайтах

Бачимо, що в основному файли мають розмір в 500-1000 кілобайт і дуже мало файлів мають розмір близько 10 мегабайт. Також, варто відзначити, що шкідливе ПЗ, як правило, має менший розмір файлів.

3.2. План експериментів

Опишемо процес проведення експериментів. До проведення експериментів розіб'ємо дані (файли) на навчальну вибірку і вибірку для тестування по відношенню 5 до 1 збалансовано (з однаковим відношенням кількості файлів з шкідливим ПЗ до чистих). Підбір ознак, гіперпараметрів будемо виконувати на навчальній вибірці. Порівняння алгоритмів машинного навчання проводитимемо на результатах вибірки для тестування, що дозволить найбільш правильно провести експеримент, уникаючи перенавчання на даних для тестування.

Будь-який алгоритм машинного навчання містить гіперпараметри – набір параметрів, який впливає на модель, і, в кінцевому рахунку, на якість навчання. Так, наприклад, для алгоритму випадкового лісу це кількість дерев у лісі, глибина дерев, кількість ознак, що розглядаються при розбитті вибірки в черговому вузлі дерева та інші. Для пошуку гіперпараметрів скористаємося 3-fold крос-валідацією на навчальній вибірці. Тобто, розіб'ємо всю навчальну вибірку на 3 збалансовані частини, тричі навчимо алгоритм на 2 частинах і обчислимо значення метрик на одній, та що залишилася. Усереднимо отримані значення. Приклад для k = 10 ознак на рис. 3.5. Для кожного алгоритму з найкращим набором гіперпараметрів будемо обчислювати accuracy (точність) і false positive rate (частку хибно позитивних спрацьовувань) на тестовій (відкладеній) вибірці для порівняння моделей між собою. В силу того, що нам вдалося отримати збалансовані дані, accuracy представляється вдалою метрикою для оцінки якості алгоритмів. Інакше, варто було б вибрати інші способи оцінки, так, наприклад, f1 міру або площа під AUC кривої.

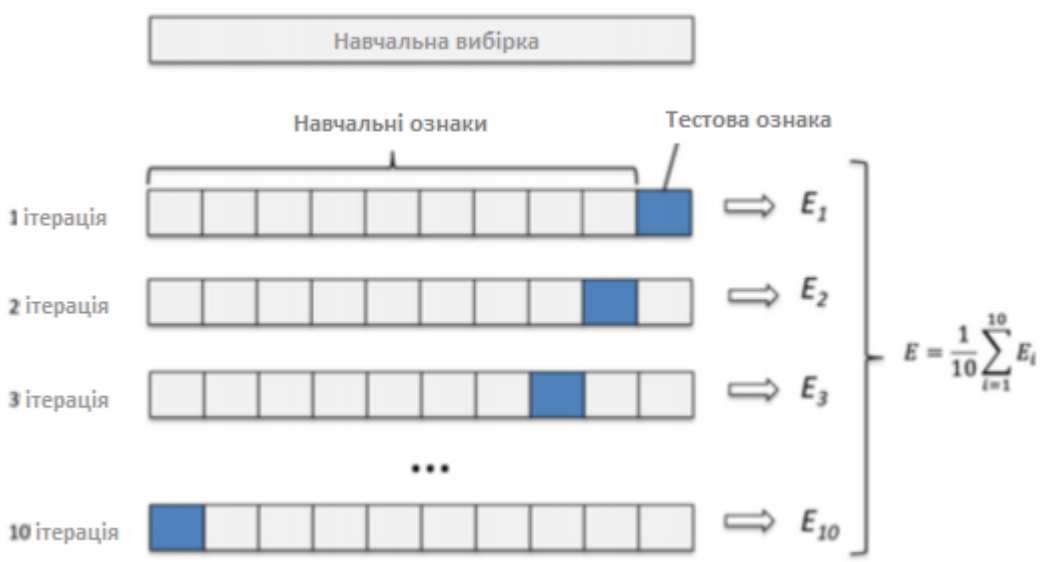

Рис. 3.5. Приклад k-fold крос-валідації при k = 10

Мотивація розрахунку false positive rate (FPR) полягає в тому, що нам важливо знати кількість чистих файлів, які помилково віднесені до шкідливих. Таких файлів має бути якомога менше. Таке помилкове визначення може призвести до псування файлів, втрати даних.

3.3. Відбір ознак

Раніше (PE формат) ми визначили яким чином і якого типу атрибути потрібно отримувати з PE файлу. Виберемо максимально можливе число ознак з файлу, а потім відбір будемо проводити жадібним алгоритмом. Тобто, на кожному кроці будемо додавати до вже існуючих той атрибут, який дає найбільший приріст точності (accuracy) моделі / класифікатору. Так будемо робити до тих пір, поки якість перестане рости. Виконуємо відбір на навчальній вибірці 3-fold крос-валідацією. Для відбору ознак в якості алгоритму будемо використовувати випадковий ліс.

Додатково до атрибутів будемо обчислювати ентропію секцій.

Ентропія розраховується наступним чином (ентропія Шеннона) [40]:

$$H = -\sum_{i=1}^{n} p_i \log p_i$$

Основу логарифма беремо рівною 256 – число можливих значень байта. Таким чином, ентропія H буде приймати значення від 0 до 1.

Висока ентропія секції може сигналізувати про стиснення секції, використання пакувальників, шифрувальників для приховування шкідливого коду.

Зібрані атрибути з файлів:

1) чисельні значення атрибут, взяті з полів PE файлу безпосередньо;

2) прапори присутності API, секцій та їх характеристики. Значення: {0, 1};

3) ентропія секцій.

Після роботи жодного алгоритму ми отримали набір з 135 ознак, на основі яких можливо детектувати шкідливі програми.

Повний перелік зібраних атрибут файлів представлений в Додатку А. Збираємо прямі атрибути структури PE файлу з MS-DOS заглушки, coff заголовка, опционального заголовка. Для фіксованої кількості dll бібліотек і імпортованих арі були виділені запропоновані в додатку назви. Було вирішено виділити 5 секцій з назвами: text, data, rsrc, rdata, reloc і отримати характеристики – права секції (shared, execute, read, write) і їх ентропію. Раніше обмовлялося, що назви секцій для завантажувача не мають значення, проте дані, «стандартні» секції зустрічаються у більшості файлів і містять, як правило те, на що і вказують. Такі відхилення метод буде відловлювати. Разом з безпосередньо витягнутими параметрами буде збирати загальну інформацію по самим структурам. Наприклад, кількість імпортованих бібліотек, назв функцій, секцій, символів. Назвемо категорію таких ознак – general.

Можна виділити групи атрибутів в залежності від структури їх видобування. Окремо будемо розглядати характеристики секцій і їх ентропії. Після виділення таких груп можливо побудувати графік їх важливості.

Важливість групи вважається як сума важливість ознак, що входять в групу. На рис. 3.6 представлена значимість груп ознак.

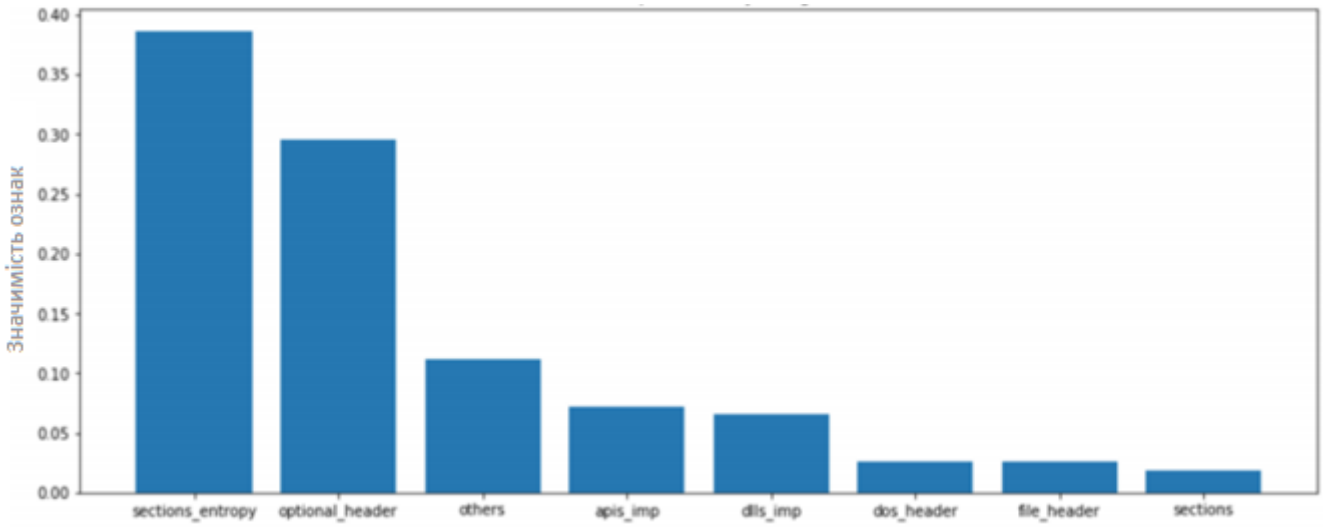

Рис. 3.6. Важливість ознак, розрахована випадковим лісом

За графіком видно, що ентропія секцій разом з ознаками опціонального заголовка грає вирішальну роль в детекції вірусу, зараження файлу. Найменш важлива інформація про DOS заголовок, coff заголовок і характеристику секцій.

3.4. Порівняння алгоритмів

Після того, як знайдені визначальні ознаки і зафіксовані дані потрібно знайти найкращу модель і найкращий алгоритм класифікації. Буде розглянено 4 обраних моделі: випадковий ліс (Random Forest), LightGBM, XGBoost, CatBoost. Всі експерименти проводяться на мові Python в середовищі ipython notebook з відповідними бібліотеками і інструментами. Для кожної моделі пошук гіперпараметров здійснюємо по 3-fold крос валідації. Після підбору гіперпараметров навчаємо модель на всій наданої тренувальної вибірці і обчислюємо accuracy і false positive rate на тестових даних. Утворені результати наведені в таблиці 3.

На додачу до зведеної таблиці якості алгоритмів наведемо залежність якості алгоритмів від розміру навчальної вибірки – рис. 3.7. По кривим можна помітити (апроксимація), що збільшення навчаних даних може поліпшити якість детектування вірусів (криві не досягнули насичення).

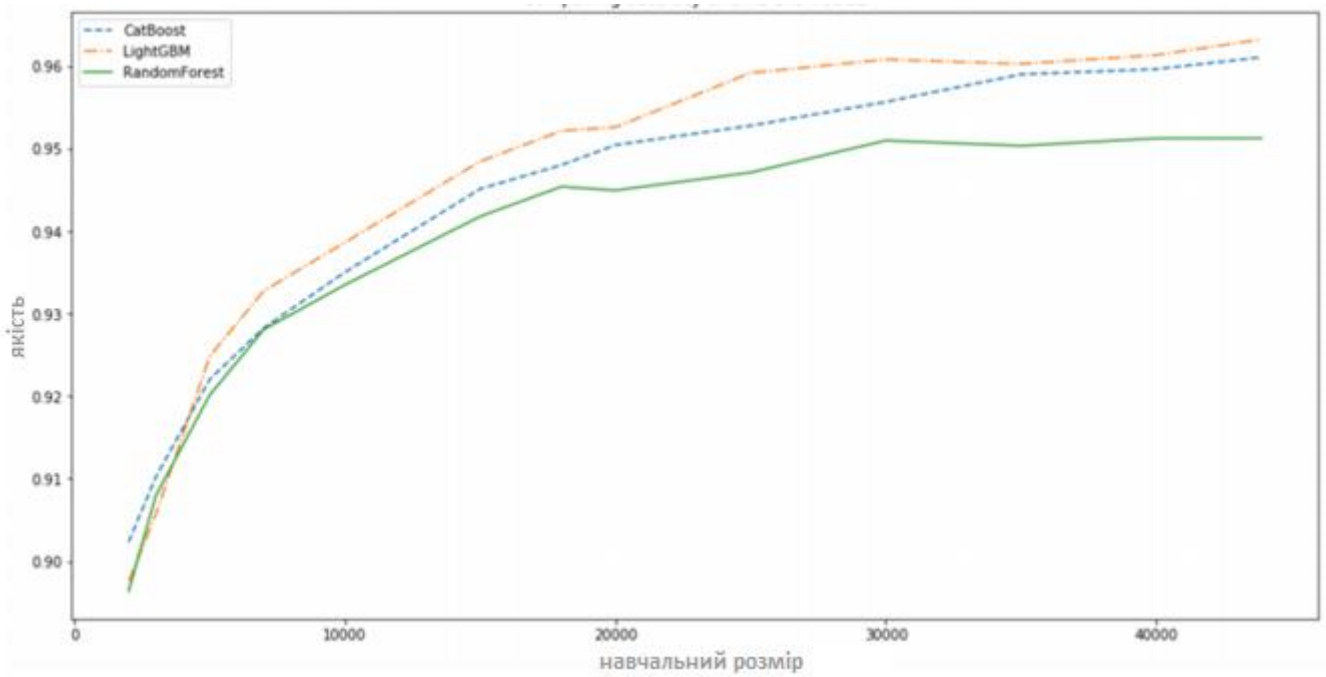

Рис. 3.7. Залежність якості алгоритму від розміру даних

Кращим з розглянутих алгоритмів в застосуванні до даної задачі виявився градієнтний бустінг з LightGBM реалізацією. Вдалося домогтися кращої, ніж необхідна точності (accuracy) і хорошою false positive rate. CatBoost навіть з підбором гіперпараметров не опинився кращим (табл. 3.2).

Взявши за основу алгоритм градієнтного бустінга з LightGBM реалізацією був реалізований метод на C++ (дивитися додаток Б) і інтегрований в продукт. Крім якості роботи алгоритму інша важлива характеристика – швидкість його роботи.

Таблиця 3.2

Порівняння алгоритмів

Алгоритм	**Accuracy %**	**False positive rate %**
Випадковий ліс	95.9	1.9
XGBoost	96.3	1.4
LightGBM	96.5	1.4
CatBoost	96.3	1.5

Була виміряна швидкість роботи методу для перевірки задоволення вимог в реальних умовах. Результати представлені в табл. 3.3. Швидкість обчислювалася на 50 тисячах файлах, розташованих на HDD диску. При запуску на SSD диску швидкість роботи класифікатора не змінилася, що логічно, а PEпарсер став працювати в середньому за 5 мс на одному файлі.

Таблиця 3.3

Середній час роботи складових частин методу

	Час роботи, ms
PE-парсер	16.0
Класифікатор	0.5
Загальний час	16.5

На рис. 3.8 представлено розподіл часу обробки методом файлів. Бачимо, що, в основному, файл обробляється менш ніж за 20 мс. Також проаналізуємо частку файлів, що обробляються за необхідний час. А саме, частку файлів, що обробляються за час менше 30 мс, якби обмеження на час було жорстким і застосовувалося до кожного файлу. В цьому випадку будуть оброблені 90 % всіх файлів, а 10 % – проігноровані. Варто розуміти, що в реальних умовах обмеження вводяться не на один файл, а на кілька, виходячи з середнього часу, а також з того, що більшість вірусів, інфікованих файлів мають малий розмір

файлу і, відповідно, малий час роботи алгоритму. Середній час обробки файлу – 21 мс, що є хорошим показником і задовольняє вимогу.

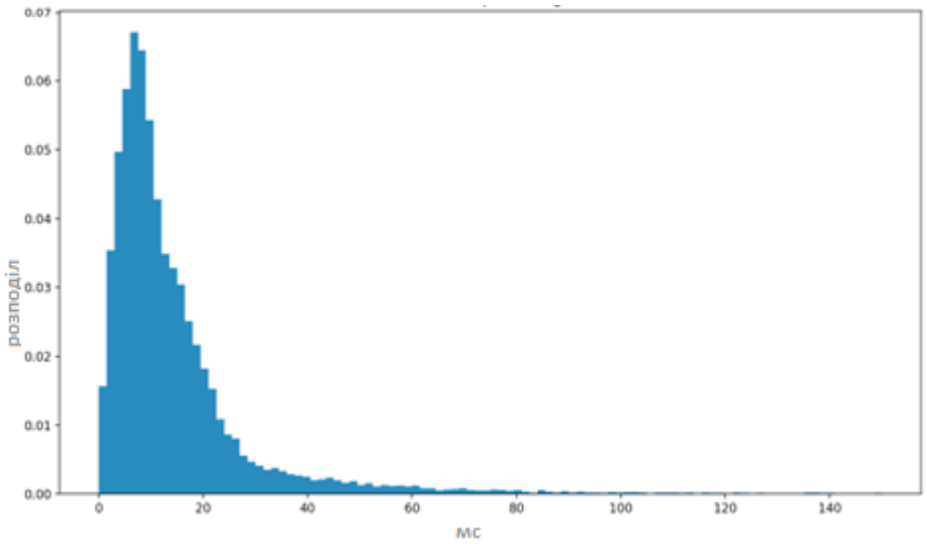

Рис. 3.8. Розподіл часу роботи метода для файлів різного розміру

На рис. 3.9 представлена залежність швидкості роботи методу від розміру файлу.

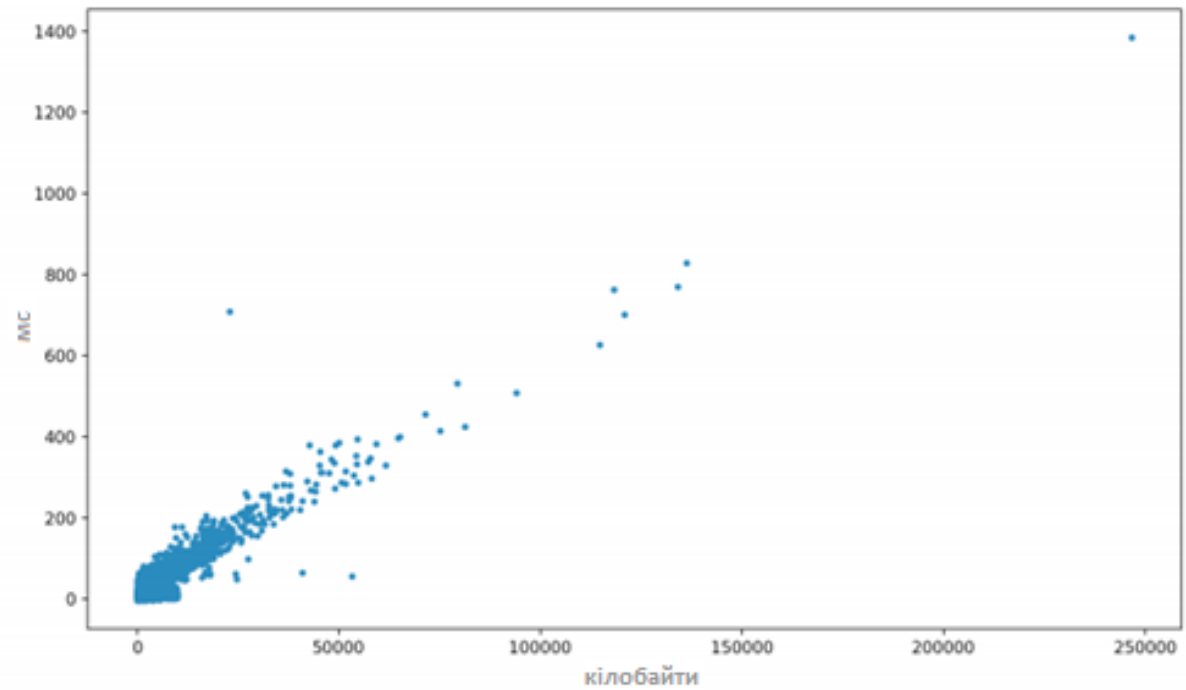

Рис. 3.9. Залежність швидкості роботи методу від розміру файлу

Бачимо, що залежність часу обробки файлу від його розміру лінійна. Під час вилучення атрибутів з PE файлу ми повинні пройтися по таблиці імпорту, по секціях для отримання ентропій. Логічно припустити, що чим розмір файлу більше, тим більше стають секції і таблиці імпорту, що містять назви dll і api. Маючи розмір досліджуваних файлів, можна приблизно оцінити час їх обробки. Так, на 100 gb даних потрібно близько 10 хвилин.

Отримані результати оптимістичні і показують можливість застосування методу на практиці. Також цікаво подивитися за такою характеристикою, як false negative rate. Частка вірусів, які метод не зміг виявити. Раніше вважалося, що файл є вірусом, якщо за це проголосувала хоча б один з надійних антивірусів. Тепер побудуємо розподіл індексу шкідливості від всіх антивірусів для false negative файлів. Гістограма наведена на рис. 3.10. Можна порівняти даний розподіл до представленого раніше для всіх файлів (рис. 3.2).

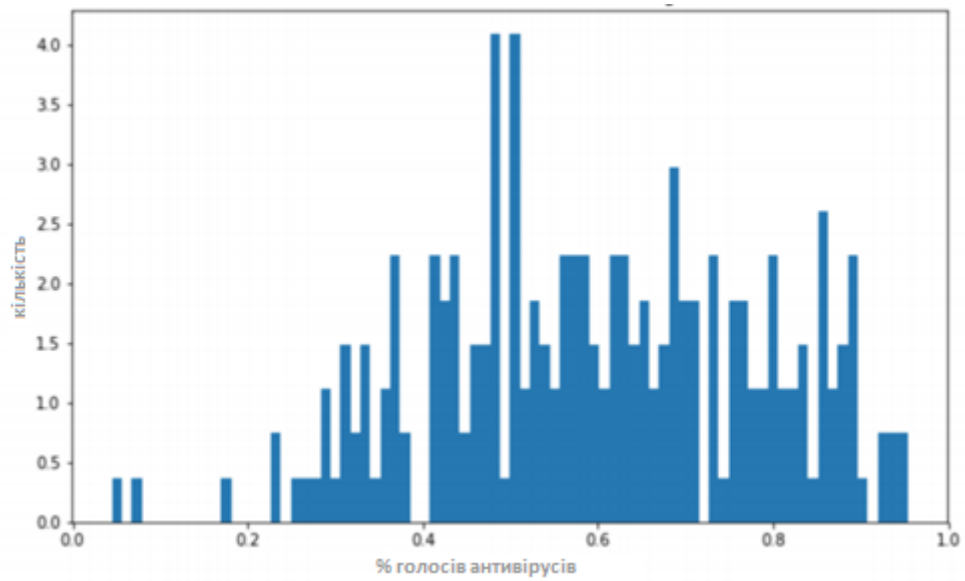

Рис. 3.10. Розподіл голосів антивірусів по false negative файлів

На рис. 3.10 розподіл більш рівномірний. незважаючи на загальне зміщення вправо (центр мас приблизно у 0.6), багато антивірусів виявилися нездатні правильно визначити ці випадки. Для поліпшення якості алгоритму можливо сконцентруватися на таких сірих файлах і визначити їх особливості.

3.5. Висновки до розділу 3

Реалізовано найкращий метод детектування шкідливого коду в рамках програмного продукту на базі машинного навчання, що задовольняє встановленим вимогам швидкості роботи моделі, які накладали обмеження на вибір алгоритму, кількість видобутих ознак. Отриману експертизу можна+ використовувати для побудови більш складних систем детектування вірусів.

ВИСНОВКИ

Під час роботи було досягнуто низку результатів.

1. Здійснено огляд методів, що застосовуються для статичного аналізу виявлення зловмисних програм, проведено вибір найбільш важливих ознак для детектування шкідливого програмного забезпечення та порівняння сучасних алгоритмів машинного навчання в рамках даного завдання в умовах обмежених ресурсів і часу,

2. Порівняно сучасні алгоритми машинного навчання стосовно до можливості їх використання для побудови більш складних систем детектування вірусів та обрано найкращий алгоритм по метриці точності (accuracy), що задовольняє висунутим вимогам якості.

3. Реалізовано найкращий метод детектування шкідливого коду в рамках програмного продукту на базі машинного навчання, що задовольняє встановленим вимогам швидкості роботи моделі, які накладали обмеження на вибір алгоритму, кількість видобутих ознак. Отриману експертизу можна використовувати для побудови більш складних систем детектування вірусів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ТА ЛІТЕРАТУРИ

1. Samani R. McAfee Labs Threats Report / R. Samani, C. Beek. // McAfee. – 2019.
2. Madenur Sridhara, S., Stamp, M.: Metamorphic worm that carriesits own morphing engine. J. Comput. Virol. 9(2), 49–58 (2013).DOI 10.1007/s11416-012-0174-z. URL http://dx.doi.org/10.1007/s11416-012-0174-z.
3. Symantec trend report, https://www.symantec.com/security_response/publications/monthlythreatreport.jsp #Spam, Accessedon April 15, 2016.
4. Hahn K. Robust Static Analysis of Portable Executable Malware // HTWK Leipzig. — 2014.
5. Novel Feature Extraction, Selection and Fusion for Effective Malware Family Classification / Ahmadi M. [et al.] // In Proceedings of the 6 ACM Conference on Data and Application Security and Privacy / ACM. — 2016. — Pp. 183-194.
6. An intelligent PE-malware detection system based on association mining / Ye Y. [et al.] // Journal in Computer Virology. — 2008. — Vol. 4, no. 4. — Pp. 323-334.
7. PE-Miner: Mining Structural Information to Detect Malicious Executables in Realtime / Shafiq M. Z. [et al.] // Recent Advances in Intrusion Detection, Lecture Notes in Computer Science — 2009 — Vol. 5758 — Pp. 121-141.
8. Tabish S. M., Shafiq M. Z., Farooq M. Malware detection using statistical analysis of byte-level file content // In Proceedings of the ACM SIGKDD Workshop on CyberSecurity and Intelligence Informatics / ACM. — 2009. — Pp. 23-31.
9. Automatic generation of string signatures for malware detection / Griffin K. [et al.] // In Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection. — 2009. — Pp. 101-120.

10. Hu X., Chiueh T.-c., Shin K. G. Large-scale malware indexing using functioncall graphs // In Proceedings of the 16th ACM Conference on Computer and Communications Security / ACM. — 2009. — Pp. 611-620.

11. Malware detection based on mining api calls / Sami A. [et al.] // In Proceedings of the 2010 ACM Symposium on Applied Computing / ACM. — 2010. — Pp. 1020-1025.

12. Manjunath. Malware images: Visualization and automatic classification / Nataraj L. [et al.] // In Proceedings of the 8th International Symposium on Visualization for Cyber Security / ACM. — 2011. — Pp. 41-47.

13. A static, packer-agnostic filter to detect similar malware samples / Jacob G. [et al.] // In Proceedings of the 9th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. — 2013. — Pp. 102-122.

14. Opcode sequences as representation of executables for data-mining-based unknown malware detection / Santos I. [et al.] // Information Sciences. — 2013. — Vol. 231 — Pp. 64-82.

15. Novel active learning methods for enhanced PC malware detection in windows OS / Nissim N. [et al.] // Expert Systems with Applications. — 2014. — Vol. 41, no. 13. — Pp. 5843-5857.

16. DLLMiner: structural mining for malware detection / Narouei M. [et al.] // Security and communication networks. — 2015. — Vol. 8, no.18. — Pp. 3311- 3322.

17. Chen T., C. Guestrin C. XGBoost: A Scalable Tree Boosting System // Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. — 2016. — Pp. 785-794.

18. Breiman L. Bagging predictors // Machine Learning. — 1996. — Vol. 24, no. 2 — Pp. 123-140.

19. Microsoft Corporation. PE Format specification. — URL: https: //msdn.microsoft.com/library/windows/desktop/ms680547(v=vs.85) .aspx?id=19509

20. Portable Executable Wiki. — URL: https://en.wikipedia.org/wiki/ Portable_Executable

21. Microsoft Corporation. What is a DLL? — 2007 — URL: https://support.microsoft.com/kb/815065/EN-US

22. Kruegel C., Toth T. Using Decision Trees to Improve Signature-Based Intrusion Detection. Recent Advances in Intrusion Detection. 2003. pp. 173–191.

23. DARPA Intrusion Detection Data Sets. Available at: https://www.ll.mit.edu/ideval/data/ (accessed: 22.03.2016).

24. Heckerman D. A Tutorial on Learning with Bayesian Networks. Innovations in Bayesian Networks: Theory and Applications. 2008. vol. 156. pp. 33–82.

25. Barbara D., Wu N., Jajodia S. Detecting Novel Network Intrusions Using Bayes Estimators. Proceedings of the First SIAM International Conference on Data Mining. 2001. pp. 1–17.

26. Mukkamala S., Sung A.H., Abraham A., Ramos V. Intrusion Detection Systems Using Adaptive Regression Splines. Sixth International Conference on Enterprise Information Systems. 2006. pp. 211–218.

27. Ranjan R., Sahoo G. A new clustering approach for anomaly intrusion detection. International Journal of Data Mining & Knowledge Management Process (IJDKP). 2014. vol. 4. no. 2. pp. 29–38.

28. Wang Y. A multinomial logistic regression modeling approach for anomaly intrusion detection. Computers & Security. 2005. vol. 24. Issue 8. pp. 662–674.

29. A survey on Heuristic Malware Detection Techniques / Bazrafshan Z. [et al.] // 5th Conference on Information and Knowledge Technology. — 2013. — Pp. 113-120.

30. Ucci D., Aniello L., Baldoni R. Survey on the Usage of Machine Learning Techniques for Malware Analysis // arXiv preprint arXiv:1710.08189. — 2018.

31. LightGBM: A Highly Efficient Gradient Boosting Decision Tree / Ke G. [et al.] // Advances in Neural Information Processing Systems 30. — 2017.

32. Dorogush A. V., Ershov V., Gulin A. CatBoost: gradient boosting with categorical features support. — 2017.

СПИСОК ВИКОРИСТОВУВАНИХ ОЗНАК

1. MS-DOS Stub/e_lfanew
2. MS-DOS Stub/e_crlc
3. Coff Header/NumberOfSections
4. Coff Header/NumberOfSymbols
5. Coff Header/PointerToSymbolTable
6. Coff Header/SizeOfOptionalHeader
7. Coff Header/Characteristics
8. Optional Header/SizeOfInitializedData
9. Optional Header/SizeOfUninitializedData
10. Optional Header/BaseOfCode
11. Optional Header/DllCharacteristics
12. Optional Header/SizeOfStackReserve
13. Optional Header/SizeOfHeapReserve
14. Optional Header/SectionAlignment
15. Optional Header/SizeOfHeapCommit
16. Optional Header/FileAlignment
17. Optional Header/SizeOfImage
18. Optional Header/NumberOfRvaAndSizes
19. Optional Header/SizeOfHeaders
20. Optional Header/SizeOfStackCommit
21. DLL Referred/Gdiplus.dll
22. DLL Referred/Rpcrt4.dll
23. DLL Referred/Comdlg32.dll
24. DLL Referred/Wtsapi32.dll
25. DLL Referred/Imm32.dll
26. DLL Referred/Mscoree.dll
27. DLL Referred/Gdi32.dll
28. DLL Referred/Oleacc.dll
29. DLL Referred/Version.dll

30. DLL Referred/Userenv.dll
31. DLL Referred/Msvcrt.dll
32. DLL Referred/Shell32.dll
33. DLL Referred/Wintrust.dll
34. DLL Referred/Uxtheme.dll
35. DLL Referred/Wininet.dll
36. DLL Referred/Iphlpapi.dll
37. DLL Referred/Winspool.drv
38. DLL Referred/Winmm.dll
39. DLL Referred/Setupapi.dll
40. DLL Referred/Advapi32.dll
41. DLL Referred/Ole32.dll
42. DLL Referred/Shlwapi.dll
43. DLL Referred/Ntdll.dll
44. DLL Referred/Mpr.dll
45. DLL Referred/Netapi32.dll
46. DLL Referred/User32.dll
47. DLL Referred/Oleaut32.dll
48. DLL Referred/Urlmon.dll
49. DLL Referred/Comctl32.dll
50. DLL Referred/Crypt32.dll
51. DLL Referred/Ws2_32.dll
52. DLL Referred/Psapi.dll
53. DLL Referred/Msimg32.dll
54. API Referred/Createfilew
55. API Referred/Createfilea
56. API Referred/Getmodulehandlew
57. API Referred/Getmodulehandlea
58. API Referred/Getmodulehandleexw
59. API Referred/Heapsetinformation

60. API Referred/Loadlibrarya
61. API Referred/Rtllookupfunctionentry
62. API Referred/Virtualalloc
63. API Referred/Openprocesstoken
64. API Referred/Openprocess
65. API Referred/Rtlcapturecontext
66. API Referred/Setunhandledexceptionfilter
67. API Referred/Getenv
68. API Referred/Createprocessasusera
69. API Referred/Createprocessasuserw
70. API Referred/Duplicatehandle
71. API Referred/Loadicona
72. API Referred/Virtualprotect
73. API Referred/Virtualprotectex
74. API Referred/Writeprocessmemory
75. API Referred/Createdirectorya
76. API Referred/Createdirectoryw
77. API Referred/Rtlvirtualunwind
78. API Referred/Openfile
79. API Referred/Reopenfile
80. API Referred/Ntcreatefile
81. API Referred/Ntsetinformationfile
82. API Referred/Setfileinformationbyhandle
83. API Referred/Movefilea
84. API Referred/Movefilew
85. API Referred/Movefileexa
86. API Referred/Movefileexw
87. API Referred/Copyfilea
88. API Referred/Copyfilew
89. API Referred/Copyfileexw

90. API Referred/Copyfileexa
91. API Referred/Ntopenfile
92. API Referred/Deletefilew
93. API Referred/Deletefilea
94. API Referred/Findfirstfilea
95. API Referred/Findfirstfilew
96. API Referred/Findfirstfileexa
97. API Referred/Findfirstfileexw
98. API Referred/Replacefilea
99. API Referred/Replacefilew
100. API Referred/Writefile
101. API Referred/Writefileex
102. API Referred/Suspendthread
103. API Referred/Ntsuspendprocess
104. API Referred/Ntsuspendthread
105. Section/text : характеристики * 4, энтропия
106. Section/data : характеристики * 4, энтропия
107. Section/rsrc : характеристики * 4, энтропия
108. Section/rdata : характеристики * 4, энтропия
109. Section/reloc : характеристики * 4, энтропия
110. General/NumberOfReferredDLLs
111. General/NumberOfReferredAPIs
112. General/NumberOfReferredAPIOrdinals
113. General/NumberOfSections
114. General/NumberOfExportTableSymbols
115. General/NumberOfRelocSectionItems