Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерної інженерії та управління
(повна назва)

Кафедра Безпеки інформаційних технологій
(повна назва)

# АТЕСТАЦІЙНА РОБОТА
# Пояснювальна записка

рівень вищої освіти другий (магістерський)

Метод виявлення шкідливого коду з використанням технологій інтелектуального аналізу даних
(тема)

Виконав: Смирнов Л.М.
(прізвище, ініціали)
студент 2 курсу, групи БІКСм-18-1
Спеціальність 125 Кібербезпека
(код і повна назва спеціальності)
Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)
Освітня програма «Безпека інформаційних і комунікаційних систем»
(повна назва освітньої програми)

Керівник доц. Петренко О.Є.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Халімов Г.З.
(підпис) (прізвище, ініціали)

2019 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерної інженерії та управління
(повна назва)

Кафедра Безпеки інформаційних технологій
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 125 Кібербезпека
(код і повна назва)

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

Освітня програма «Безпека інформаційних і комунікаційних систем»
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 20 ___ р.

# ЗАВДАННЯ
НА АТЕСТАЦІЙНУ РОБОТУ

студентові *Смирнову Леву Михайловичу*
(прізвище, ім'я, по батькові)

1. Тема роботи *Метод виявлення шкідливого коду з використанням технологій інтелектуального аналізу даних*

затверджена наказом по університету від "04" листопада 2019 р. № 1649Ст

2. Термін подання студентом роботи до екзаменаційної комісії _____

3. Вихідні дані до роботи *Теоретичні дані про методи машинного навчання*

4. Перелік питань, що потрібно опрацювати в роботі

*1.Особливості реалізації та застосування методів машинного навчання в сфері інформаційної безпеки*

*2. Загальна структура виконуваних файлів і можливості з їх дослідження*

*3. Аналіз результатів реалізації методу виявлення шкідливого коду*

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій *Презентаційний матеріал у вигляді слайдів*

**КАЛЕНДАРНИЙ ПЛАН**

| № | Назва етапів роботи | Термін виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Отримання завдання | *20.10.18* | |
| 2 | Пошук літератури | *21.10.18-15.06.19* | |
| 3 | Аналіз зібраних даних | *16.06.19-27.09.19* | |
| 4 | Розробка методів машинного навчання для виявлення шкідливого коду | *28.09.19-23.10.19* | |
| 5 | Аналіз отриманих результатів | *24.10.19-15.11.19* | |
| 6 | Оформлення пояснювальної записки | *16.11.19-15.12.19* | |

Дата видачі завдання _____ _____20__ р.

Студент _____
(підпис)

Керівник роботи (проекту) _____ _____доц. Петренко О.Є._____
(підпис) (посада, прізвище, ініціали)

# РЕФЕРАТ

Пояснювальна записка атестаційної роботи: 66 с., 20 рис., 4 табл., 1 дод., 19 джерел.

ШКІДЛИВЕ ПЗ, УРАЗЛИВОСТІ, МАШИННЕ НАВЧАННЯ, АНТИВІРУС, ОБФУСКАЦІЯ.

Шкідливі програми (malware) стають все більш і більш складними. Сучасні віруси мають здатність мутувати, змінюватися в процесі життєдіяльності, що призводить до зростання кількості варіантів шкідливих програм. Традиційні підходи, засновані на пошуку сигнатур файлів перестають бути ефективними. На зміну приходить автоматизація аналізу файлів для виявлення підозрілих файлів.

Метою атестаційної роботи є дослідженню методів машинного навчання для завдання виявлення шкідливих програм PE формату для ОС Windows і побудови додатку на основі машинного навчання, здатного з високою точністю детектувати шкідливий код до його виконання. А саме, проводиться:

- Дослідження існуючих підходів до статичного аналізу PE файлів
- Відбір значущих для аналізу атрибут PE файлів
- Порівняння алгоритмів машинного навчання для даного завдання

# ABSTRACT

Master’s thesis: 66 pages, 20 figures, 4 tables, 1 appendix, 19 sources.

MALWARE, VULNERABILITIES, MACHINE LEARNING, ANTIVIRUS, OBFUSCATION.

The major goal of this thesis is is to investigate machine learning methods applied to identifying the malware files of PE format for Windows and to build a machine-based application capable of detecting malicious code with high accuracy. Namely, following is carried out:

- Investigation of existing approaches to static analysis of PE files
- Selection of PE file attribute that is important for analysis
- Comparison of machine learning algorithms for this task

ЗМІСТ

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

HMM – приховані марковські моделі

ELF – виконуваний файл лінукс системи

SVM – метод опорних векторів

HPC – лічильник продуктивності процесора

ML – машинне навчання

ПК – персональний комп'ютер

OS – операційна система

DT – дерево рішень

# ВСТУП

Сьогодні комп'ютерні програми і додатки розробляються з великою швидкістю. З розвитком комп'ютерної техніки та інтернету, все більше людей починають турбуватися про їхню безпеку та безпеку своїх даних. Розвиток шкідливого ПЗ призводить до розвитку більш сучасних механізмів захисту. За даними McAfee Lab в четвертому кварталі 2014 року з'явилися понад 350 мільйонів повністю унікальних шкідливих програм (зростання на 17% відносно третього кварталу 2014 роки) [1]. За дослідженням Symantec більш 44.5 мільйонів шкідливих ПЗ було створено за травень 2015 [2]. За 2017 рік світова громадськість познайомилася з WannaCry, Petya, NotPetya. Аналіз такої кількості даних вимагає від антивірусних компаній постійно зростаючого зусилля. Віруси стає все більш складно виявити. Сучасне шкідливе ПЗ здатне як впроваджуватися і заражати чисті файли системи і інших програм, так і самостійно поширюватися і міняти своє тіло виконання в процесі життєдіяльності. Застосовуються різні способи приховування шкідливого коду. Використовуються інструменти шифрування секцій, коду, даних. Шкідливе ПЗ може не тільки вкрасти або пошкодити дані користувача, вимагати гроші, уповільнити роботу комп'ютера, але і перетворити комп'ютер користувача в шпигуна, заволодіти його управлінням.

Традиційний підхід до виявлення шкідливих програм заснований на зіставленні сигнатур досліджуваних файлів [3]. Процедура полягає в наступному:

1) Новий вірус / шкідливе ПЗ починає поширюватися;

2) Експерти антивірусних компаній отримують зразки для дослідження поведінки вірусу;

3) Експерти присвоюють вірусу унікальну сигнатуру, що представляє собою послідовність інструкцій;

4) Сигнатура додається в базу даних сигнатур шкідливих програм;

5) Клієнти повідомляються про оновлення бази сигнатур;

6) Клієнти оновлюють бази сигнатур, таким чином стаючи захищеними від даного виду шкідливого ПЗ.

Варто відзначити, що сигнатури дозволяють гарантовано визначити тип вірусу. Це дозволяє додатково вносити в базу сигнатур способи лікування заражених файлів, вірусів.

Даний підхід при всій своїй простоті володіє ключовими недоліками:

1) Можливість захищати клієнта тільки від відомих вірусів. Не можна отримати сигнатуру абсолютно нового вірусу, не дослідивши його. Тут варто зробити застереження, що сигнатури, як правило, створюються так, щоб покривати не один, а якомога більшу кількість вірусів, сімейство вірусів. Однак завжди існує така зміна тіла вірусу, при якому сигнатура перестає виявлятися;

2) Постійне зростання бази даних сигнатур. 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 вірусів навчилися успішно обходити пошук сигнатур за допомогою обфускації тіла вірусу. Поліморфні і метаморфічні шкідливі програми змінюють свій зовнішній вигляд в процесі життєдіяльності. Все це спонукало антивірусні компанії розробляти альтернативні способи захисту. А саме, можна виділити два великих напрямки досліджень [4]:

- Статичний аналіз (аналіз структури бінарного файлу, його атрибутів, логічних структур, потоку виконання і даних)
- Динамічний аналіз (відстеження дій програми при виконанні,

побудова її профілю)

Кожен з методів має свої переваги і недоліки. Як правило, для кращого детектування шкідливих програм вони використовуються одночасно. У кожному з цих методів існує ймовірність помилково визначити наявність у файлі вірусу, коли насправді файл чистий. У таких випадках передбачуване лікування може зіпсувати файл, що спричинить до втрати інформації.

В роботі буде розглянуто статичний аналіз PE файлів для операційної системи Windows. Вибір пов'язаний з величезною популярністю цієї системи. PE формат же є стандартом файлів, що виконуються і бібліотек. Потрібно розуміти, що шаблон, алгоритму проектування статичного аналізу шкідливого ПЗ застосовується і до інших платформ.

Задача полягає в створенні методу на основі машинного навчання для статичного аналізу PE файлів для операційної системи Windows з подальшою інтеграцією в програмний продукт. Також, важливо зазначити, що метод машинного навчання повинен володіти високою швидкістю роботи. Для слабких персональних комп'ютерів час сканування системного диска має укладатися в кілька хвилин.

Для вирішення даної задачі потрібно:

1. дослідити існуючі підходи;
2. виявити найбільш важливі ознаки для визначення шкідливості файлу в умовах обмежених ресурсів і часу;
3. порівняти сучасні алгоритми машинного навчання стосовно до даної задачі.

# 1 ВИКОРИСТАННЯ МЕТОДІВ МАШИННОГО НАВЧАННЯ В СФЕРІ ІНФОРМАЦІЙНОЇ БЕЗПЕКИ

## 1.1 Дуельні марковські моделі для аналізу вірусів

Вонг і Штамп показали, що інструменти, засновані на прихованих марковських моделях (HMM), ефективні при виявленні метаморфічних комп'ютерних вірусів. У цьому документі досліджуються ці інструменти більш глибоко, щоб краще зрозуміти значення прихованих станів у цих моделях. В інших областях стани HMM були пов'язані з деякими фундаментальними аспектами проблеми. Наприклад, Кейв та Нойвірт виявляють, що HMM з двома прихованими станами для англійської (письмової) мови - відповідає голосним і приголосним. Ця стаття намагається розкрити деталі про приховані стани та визначити, яку інформацію вони можуть дати про загальний код збірки, зокрема про вірусний код.

Основне розуміння полягає в тому, що набори для побудови вірусів та метаморфічний код - це по суті інший тип компіляторів. Наші тести будують моделі для чотирьох різних компіляторів, для рукописного (доброякісного) асемблерного коду, для трьох комплектів створення вірусів та для двох сімей метаморфічних шкідливих програм. Ми визначаємо чіткі моменти наших моделей, відзначаючи, чим розбір рукописних записів відрізняється від скомпільованого коду та як доброякісний код відрізняється від коду вірусу.

Ми використовуємо це розуміння різних моделей щоб більш ефективно виявляти комп'ютерні віруси. Традиційний підхід використовує приховану марковську модель вірусного коду і позначає код зараженим, якщо він перевищує заданий поріг. Натомість ми перевіряємо фільм на декілька різних HMM та фіксуємо файл як вірус, лише якщо вірусна HMM повідомляє про найбільшу ймовірність спостереження за даним файлом. Ми називаємо цей підхід дуельні HMM [1], наводячи думку про те, що різні HMM конкурують

один з одним. Наші результати показують, що стратегія поєднання HMM досягає вищих результатів у порівнянні до порогової методики і часто є ефективною при виявленні вірусів. Незважаючи на те, що в інших областях, таких як виявлення вторгнення, було застосовано дуельні HMM, цей підхід раніше не застосовувався до ідентифікації вірусу.

Цей документ є розширенням попередньої роботи конференції, включаючи аналіз додаткових джерел доброякісного коду і додаткові сімейства вірусів, включаючи MWOR, який спеціально розроблений для ухилення від методики виявлення, використовуваної Вонгом та Штампом. Ми також показуємо, як пороговий підхід може поєднуватися з дуельною стратегією HMM для зниження накладних витрат на поєднання HMM, не знижуючи точності результатів. Внесок цієї статті полягає в наступному:

— досліджуємо смисловий сенс прихованого стану прихованої моделі Маркова.

— демонструємо ефективність HMM в розрізненні між різними компіляторами.

— розробляємо стратегію дуельних HMM, нову технологію використання декількох HMM в ідентифікації вірусів.

— розробляємо багаторівневу стратегію HMM, яка поєднує пороговий підхід та стратегію дуелінгу HMM, отримуючи переваги обох методів

### 1.1.1 Виявлення метаморфічних зловмисних програм

Метаморфні віруси важко виявити за допомогою традиційних підходів до сканування. Код вірусу затьмарений, а не просто зашифрований. У цьому розділі ми розглядаємо файли, заражені вірусом MetaPHOR [2], відомим своїми складними метаморфічними методами. Крім того, ми перевіряємо стратегію проти MWOR, метаморфного черв'яка, спеціально розробленого для ухилення від детекторів на основі HMM.

Вірус Win32 / Simile, іноді відомий як Win32 / Etap, є одним з найбільш

просунутих метаморфічних вірусів. Він побудований за допомогою середи MetaPHOR, створеної програмістом вірусів, відомим лише як "the Mental Driller". Він заражає 32-бітні файли Windows, хоча пізніші версії також заражають Linux ELF-файли. Хоча він не включає руйнівний механізм, він може бути модифікований для цього. Приблизно 90% коду вірусу відноситься до його метаморфічної поведінки.

**Observation Probabilities**

| 0 | 1 | 2 | 3 |
|---|---|---|---|
| MOV 65% | J* 59% | CALL 31% | TEST 31% |
| PUSH* 16% | JMP 23% | CMP 26% | SUB 27% |
| POP* 11% | LEA 18% | ADD 20% | XOR 19% |
| Other 8% | | AND 14% | LEAVE 15% |
| | | Other 9% | Other 8% |

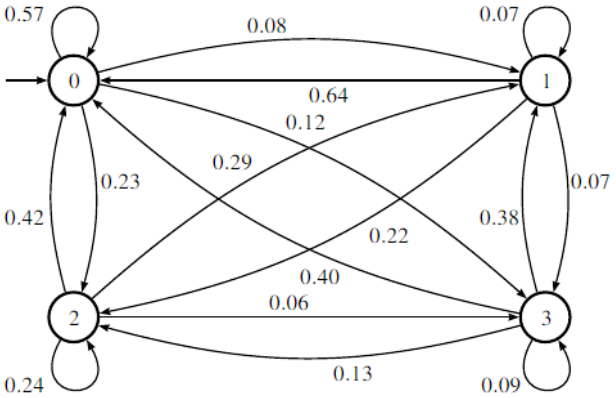


Рисунок 1.1 - HMM для файлів заражених MetaPHOR

Наші початкові дані тренінгу складаються з 49 програм, скомпільованих MinGW та заражених MetaPHOR. На рисунку 1.1 показана модель, сформована з інфікованих MetaPHOR файлів. Він має деякі помітні подібності з моделлю MinGW. В обох моделях існує стан CALL з високою ймовірністю спостереження за кодами CMP та ADD. У стані 3 домінують спостереження TEST, SUB та XOR в обох HMM. Як і модель MinGW, метафорічна модель починається у стані 0 зі 100% вірогідністю. Основне розмежування між двома моделями полягає в спостереженні стрибкових інструкцій. Модель MinGW має чіткий стан MOV та стан PUSH / POP, тоді як модель MetaPHOR поєднує ці два стани та розбиває інструкції зі стрибків у їх власні стани. За цією особливістю він більше нагадує HMM для написаних вручну сбірок.

Результати помітно покращуються при тестуванні за допомогою стратегії дуельних HMM, як показано в таблиці 1.1. Лише 4 з 300 доброякісних фільмів помилково ідентифіковані як заражені, і всі окрім 9 з

60 заражених файлів правильно ідентифіковані. У цих тестах не було використаних ухилів, хоча можливі перекоси результатів для досягнення бажаного балансу між помилковими негативними та хибними позитивами.

Таблиця 1.1 – Порівняння стратегій виявлення

| | Dueling | | | | Threshold | | | |
|---|---|---|---|---|---|---|---|---|
| Virus Family | False Positives | False Negatives | Accuracy (%) | Total Time (ms) | False Positives | False Negatives | Accuracy (%) | Total Time (ms) |
| G2 | 0/370 | 0/50 | 100.00 | 1030.57 | 62/370 | 4/50 | 84.29 | 548.22 |
| MPCGEN | 0/370 | 0/50 | 100.00 | 981.13 | 89/370 | 0/50 | 78.81 | 495.47 |
| NGVCK | 2/370 | 75/200 | 86.49 | 1155.61 | 166/370 | 16/200 | 68.07 | 550.97 |
| MetaPHOR | 4/370 | 9/60 | 96.98 | 1113.47 | 198/370 | 10/60 | 34.88 | 538.66 |
| MWOR (PR 1) | 0/370 | 0/100 | 100.00 | 2019.35 | 0/370 | 0/100 | 100.00 | 866.27 |
| MWOR (PR 2) | 0/370 | 0/100 | 100.00 | 2413.05 | 1/370 | 0/100 | 99.79 | 964.16 |
| MWOR (PR 3) | 0/370 | 0/100 | 100.00 | 2897.53 | 4/370 | 0/100 | 99.15 | 1152.99 |
| MWOR (PR 4) | 0/370 | 0/100 | 100.00 | 3433.05 | 4/370 | 0/100 | 99.15 | 1341.37 |

### 1.1.2 Ідентифікація MWOR

MWOR [3] - це вдосконалений метаморфічний черв'як, спеціально призначений для ухилення від детекторів на основі HMM. Він додає мертвий

код, взятий з утиліти Linux, щоб поєднатися з доброякісним кодом. Маденур Шрідхара та Штамп [3] показують, що MWOR здатний ухилятися від виявлення від порогових детекторів HMM, коли доданий мертвий код більше ніж у 2,5 рази перевищує код черв'яка.

Наші експерименти перевіряють різні коефіцієнти заповнення; тобто кількість включених мертвих кодів варіюється в різних тестах. Кожна тестова група включає 100 виконуваних файлів MWOR, оцінених в 5-кратній перехресній валідації. Відношення заповнення 1 (PR 1) вказує відношення 1: 1 мертвого коду до коду MWOR; 4 (PR 4) вказує на співвідношення мертвого коду 4: 1 до коду MWOR.

Таблиця 1.1 показує, що стратегія дуелінга HMM досягає вищих результатів аніж пороговий підхід. Стратегія досягає досконалості, навіть при використанні найбільшого коефіцієнта заповнення (PR 4).

### 1.1.3 Багаторівнева стратегія HMM: поліпшення ефективності стратегії дуелінгу HMM

Хоча наші результати показують, що стратегія дуелінгу HMM досягає хороших результатів у виявленні зловмисного програмного забезпечення, яке використовує передові метаморфічні методи, воно вимагає значної кількості накладних витрат порівняно з пороговим підходом.

У цьому розділі ми досліджуємо, як дуельні HMM можна поєднувати з пороговим підходом. Ця багаторівнева стратегія HMM істотно знижує накладні витрати дуельної стратегії HMM, зберігаючи ефективність оригіналу.

При багаторівневому підході потенційне зловмисне програмне забезпечення повинне пройти кілька шарів підготовлених HMM. Мета полягає в застосуванні порогового підходу, щоб допомогти зменшити кількість тих файлів, що насправді передаються на шар дуелінгу, тим самим збільшуючи ефективність. Сутність багаторівневої стратегії полягає у

скороченні часу, необхідного для класифікації файлу; За допомогою порогової моделі ми виокремлюємо «дефінітивно хороший» та «дефінітно поганий», залишаючи невелику кількість файлів в сірій зоні, яку потрібно оцінити за допомогою дуельного підходу HMM.

Приховані моделі Маркова демонструють обіцянку як інструмент для виявлення вірусу, особливо для виявлення метаморфічних вірусів. Ми ілюструємо, як кілька HMM можуть бути використані у поєднанні зі стратегією HMM, покращуючи ефективність детекторів на основі HMM. Ми також показуємо, як підхід, орієнтований на поріг, та стратегія дуельних HMM можуть поєднуватися у різних рівнях, щоб мінімізувати низку часову ефективність стратегії дуельних HMM. Ми також розкриваємо деякі деталі щодо прихованих станів HMM-моделей, що дозволяє глибше розуміти критичні властивості базових моделей. Використовуючи інформацію, доступну в цих моделях, ми сподіваємось покращити нашу здатність виявляти метаморфічні шкідливі програми.

## 1.2 Виявлення та аналіз спаму зображень

Спам можна визначити як небажану масову електронну пошту. За даними Symantec [4], протягом 2015 року спам становив приблизно 60% всієї вхідної електронної пошти, і, отже, фільтрація спаму є важливою для постійної життєздатності електронних повідомлень. Спам зображень, що складається з тексту спаму, вбудованого у зображення, спамерами використовували як засіб ухилення від текстових фільтрів спаму. Ми використовуємо термін «хем», щоб відрізнити законні повідомлення від спаму. Таким чином, проблема виявлення спаму зображень полягає в розрізненні зображень хем та спаму.

Ми застосуємо аналіз основних компонентів (PCA) та аналіз метода опорних векторів (SVM) до проблеми виявлення спаму зображень. Ми показуємо, що обидві методи можуть з хорошою точністю виявити поточне

покоління спаму зображень. Далі ми аналізуємо нашу техніку SVM для визначення оптимальної підмножини ознак, яка дозволяє зробити виявлення значно ефективнішим. Нарешті, ми розробляємо новий і вдосконалений (з точки зору спамера) набір даних спама зображень. Ми показуємо, що цей покращений набір даних не може бути виявлений з розумною точністю за допомогою аналогічних методів PCA або SVM. Цей вдосконалений набір даних може слугувати складним випадком для майбутніх досліджень з метою покращення поточного стану в області виявлення спаму зображень.

### 1.2.1 Результати PCA

Після тренувань, використовуючи 500 зображень із стандартного набору даних, ми перевірили отриманий результат, використовуючи 414 зображень з кожного класу спаму та доброякісних класів на основі п’яти домінуючих власних векторів. Потім ми побудували криву ROC на основі цих результатів, з яких ми отримали AUC 0,99 і точність 97%. Ми також провели аналогічні експерименти, використовуючи $m$ домінуючих власних векторів для кожного $m$ = 1, 2,. . . , 500. Для кожного з цих 500 експериментів ми визначили AUC. Отримані значення AUC наведені на рисунку 1.2. З цих результатів ми спостерігаємо, що максимальний успіх досягається, коли кількість домінуючих власних векторів становить від 3 до 10. Крім того, AUC поступово зменшується для більшої кількості власних векторів. Оскільки ефективність балів більша для менших значень $m$, вибір приблизно $m$ = 5 власних векторів був би оптимальним у цьому випадку.

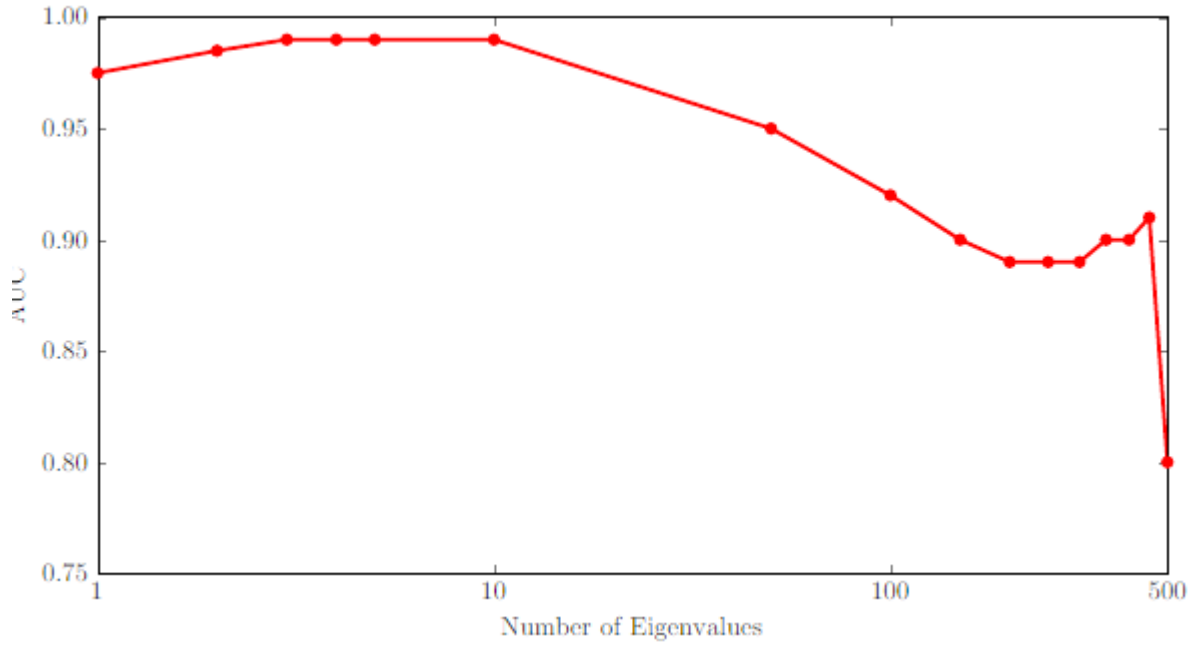


Рисунок 1.2 - AUC як функція числа власних значень

### 1.2.2 Експерименти та результати SVM

Порівняємо SVM на основі ознак, вибраних алгоритмом RFE, з наївним підходом до вибору ознак, що базуються виключно на ранжируванні ознак в оригінальній 21-ознаковій SVM (використовуючи ваги SVM). Ми бачимо, що використання RFE для вибору функцій працює набагато краще, ніж наївний підхід, особливо, коли використовується невелика кількість ознак. Це важливо, оскільки чим менше необхідних ознак, тим ефективнішим буде бал. 13 ознак, вибраних методом RFE, - Comp, Edges, Edgelen, SNR, Noise, LBP, Color, HOG, Variance 1, Variance 2, Variance 3, Skew 2 та Kurtosis 2. Зауважте, що без вибору особливостей ми отримали точність 96%. Однак, вибравши 13 з 21 функцій за допомогою RFE, точність підвищується до 97,25%. Крім того, коли використовуються лише три ознаки, ми можемо досягти точності, яка по суті така ж, як і для більш дорогої 21-ознакової моделі. Цей приклад ілюструє перевагу RFE для проблеми виявлення спаму зображень.

Для нашого SVM-аналізу ми витягли широкий набір з 21 ознаки зображення. Ми визначили силу кожної окремої особливості та порівняли їх

із SVM, який використовував усі 21 функції. Найсильніші з окремих ознак показали несподівано добрі результати. Потім ми проаналізували комбінації ознак, використовуючи рекурсивне усунення ознак [5]. Ми виявили, що чудові результати можна отримати лише з трьома ознаками, а оптимальні результати отримані з 13 ознаками. Використання невеликого підмножини ознак зробить детектор на основі SVM надзвичайно ефективним на практиці.

## 1.3 Використання ансамблевих методів для виявлення зловмисного програмного забезпечення

Зловмисне програмне забезпечення (Зловмисне програмне забезпечення) - це програма, спеціально розроблена з метою крадіжки або пошкодження даних та / або припинення функціонування систем без згоди користувача. Відповідно до звіту про загрози McAfee за 2017 рік [5], у третьому кварталі 2017 року було зафіксовано 57,6 мільйонів нових зразків шкідливих програм, що за весь час найбільше число, що збільшилось на 10% порівняно з другим кварталом. Нещодавнє розповсюдження обчислювальних пристроїв у мобільних та IoT доменах також зробило ефективне виявлення шкідливих програм важливою проблемою для вирішення. Традиційні методи виявлення зловмисних програм, такі як виявлення на основі підписів та виявлення аномалії на основі семантики, розглядаються як програмні рішення, які часто несуть значні обчислювальні витрати. У відповідь рішення, що підтримуються обладнанням, демонструють багатообіцяючі результати, зменшуючи затримку процесу виявлення на порядок із невеликими апаратними витратами.

Рішення на основі машинного навчання відіграють важливу роль в автоматизованому виявленні зловмисних програм. Такий спосіб виявлення зловмисного програмного забезпечення може бути реалізований у мікропроцесорному обладнанні зі значно низькими накладними витратами порівняно з програмними методами завдяки швидкому виявленню всередині

обладнання [6]. Ці класифікатори навчаються за допомогою функцій низького рівня, таких як дані лічильників продуктивності процесора (HPC), які фіксуються під час виконання, щоб відповідним чином відображати поведінку програми. HPC - це набір реєстрів спеціального призначення, побудованих в сучасних мікропроцесорах для збору кількості подій, пов'язаних з обладнанням, які широко використовуються для прогнозування потужності, продуктивності та енергоефективності обчислювальних систем. Останні дослідження показали, що поведінку зловмисного програмного забезпечення можна диференціювати від доброякісних програм, класифікуючи аномалії в просторах функцій низького рівня, таких як мікро архітектурні події, зібрані регістрами HPC, доступними в сучасних процесорах. Виявлення зловмисного програмного забезпечення за допомогою HPC стало перспективною альтернативою традиційним методам виявлення зловмисних програм проти збільшення загроз безпеці.

Попередня робота проводила обмежене дослідження щодо виявлення зловмисного програмного забезпечення, передбачаючи наявність великої кількості (наприклад, 16 або 32) та різноманітних HPC, тоді як сучасні процесори, особливо у вбудованих мобільних та IoT-доменах, мають обмежену кількість HPC, що становить від 2 до 8 Тому в реальних системах для збору різноманітних даних HPC для досягнення високої точності за допомогою загальної моделі ML, представленої в попередній роботі [6], потрібно запустити програму кілька разів, оскільки апаратне забезпечення може підраховувати лише невелику підмножину подій одночасно що робить непрактичним для виявлення шкідливих програм під час запуску. Крім того, попередні дослідження в основному зосереджуються на конкретних класифікаторах, ігноруючи аналіз різних типів рішень ML.

У цій роботі за допомогою систематичного підходу ми спочатку аналізуємо велику кількість HPC та виявляємо найважливіші функції низького рівня, пов’язані з виявленням шкідливих програм. Далі ми ретельно оцінюємо та характеризуємо різні типи машинного навчання для

класифікації доброякісних та шкідливих програм під час виконання. Крім того, ми досліджуємо ефективність ансамблевого навчання в підвищенні продуктивності класифікаторів ML. Ми вивчаємо різні загальні та комплексні методи навчання з точки зору точності, надійності, продуктивності та витрат на впровадження обладнання. На відміну від попередніх досліджень запропонований апарат на базі машинного навчання ефективно виконує виявлення зловмисного програмного забезпечення під час виконання з використанням обмеженої кількості HPC. На основі показників досягнутої ефективності, площі та затримки ми надаємо цінні відомості, які допомагають у виборі точних та відповідних обладнанням класифікаторів ML для виявлення зловмисних програм. Цей всебічний аналіз допомагає дизайнерам зрозуміти і орієнтуватися на компроміси між декількома параметрами проектування, запропонованими кожним класифікатором ML.

У цій роботі ми запропонували апаратну систему виявлення зловмисного програмного забезпечення, яка ефективно виявляє шкідливе програмне забезпечення з обмеженою кількістю HPC. Ми реалізували різні моделі навчання, включаючи загальні класифікатори ML та ансамблевий класифікатор AdaBoost та ретельно оцінили їх з точки зору точності, надійності, продуктивності та апаратних витрат. Ми показали, що використання восьми HPC може забезпечити достатню точність майже 94% для ефективного виявлення шкідливих програм під час роботи. Результати показують, що без врахування витрат на обладнання, складні класифікатори ML, такі як MLP та BayesNet, є переможцями з огляду на їх високу ефективність. Однак після обліку витрат на впровадження вони виявляються найгіршими щодо продуктивності / площі та затримки порівняно зі значно простішими, але дещо менш точними класифікаторами, такими як JRip та J48. Також методи, засновані на правилах і на деревах, показуючи до 13% підвищення продуктивності, виграють більше від застосування ансамблевого навчання. Ці легкі класифікатори в поєднанні з ансамблевим навчанням можуть відповідати надійності загальних сильних класифікаторів, що

виключає необхідність використання дорогих класифікаторів для кращої продуктивності.

## 1.4 Виявлення кібератак за допомогою ланцюгів Маркова

Кібератака розпочинається за допомогою ряду комп'ютерних дій для пошкодження безпеки (наприклад, доступності, цілісності та конфіденційності) комп'ютерної та мережевої системи. Виявлення кібератак спрямоване на визначення кібератак, коли вони діють на комп'ютерній та мережевій системі. Існують два загальні підходи до виявлення кібератак:

• розпізнавання шаблонів

• виявлення аномалії

, які доповнюють один одного при виявленні кібератаки.

Методи розпізнавання шаблонів ідентифікують і зберігають шаблони відомих атак. Потім вони порівнюють спостережувану поведінку випробуваного з тими відомими шаблонами атаки та сигналізують атаку, коли є відповідність. Технології розпізнавання шаблонів використовуються у багатьох комерційних програмних та дослідницьких прототипах. Сигнатури атаки позначаються як строки, послідовності подій, графіки активності та сценарії атаки (що складаються з послідовностей подій, їх передумов та цільових компрометованих станів). Правила, діаграми зміну станів, кольорові моделі Петрі-сітки та дерева рішень [7] використовувались для представлення та визначення шаблонів в сигнатурах атак. Хоча технології розпізнавання шаблонів є ефективними при виявленні відомих атак, вони не можуть виявити нові або незвичні атаки, чиї підписи невідомі. Крім того, сховище шаблонів підписів атаки повинно постійно оновлюватися, щоб залишатися корисним у системному середовищі, що динамічно змінюється, наприклад, конфігурації, протоколи, архітектура та сценарії атак.

Для предмета (наприклад, користувача, файлу, привілейованої програми, хост-машини чи мережі ), техніка виявлення аномалії складається

з встановлення профілю норми (профілю нормальної діяльності суб'єкта), спостереження за діяльністю суб'єкта та сигналізації атак, коли спостережувальна діяльність суб'єкта помітно відрізняється від його нормального профілю. Методи виявлення аномалії враховують відхилення спостережуваних дій суб'єкта від його нормального профілю, тобто симптоми аномалій, як напади. Деннінг [8] виправдовує підхід до виявлення аномалії щодо виявлення кібератаки. Методи виявлення аномалії можуть виявляти як нові, так і відомі напади, якщо вони демонструють істотні відмінності від профілю норми. Оскільки методи виявлення аномалій розглядають усі аномалії як атаки, помилкові тривоги передбачаються, коли аномалії є наслідком неправильної поведінки замість кібератак. Техніки розпізнавання шаблонів та методи виявлення аномалії можуть примножувати можливості виявлення аномалії при спільному використанні.

У цьому документі представлена робота над підходом виявлення аномалії до виявлення кібератаки та зосереджується на стійкій техніці Марковських ланцюгів МС. Існує декілька методів виявлення аномалій, які відрізняються представленням профілю норми та виведенням відхилення від профілю норми.

Методи виявлення аномалії на основі специфікацій описують політику безпеки та дозволену діяльність суб’єкту, такого як привілейована програма або мережевий сервер, у вигляді формальної логіки, правил та / або графіків. Однак важко перерахувати та вказати всі можливі нормальні дії або політики безпеки суб'єкта, особливо масштабного суб'єкта, такого як хост-машина або мережа, де взаємодія декількох об'єктів і дій неминуча. Крім того, діяльність багатьох суб'єктів в комп'ютерній та мережевій системі (наприклад, користувачі та файли) передбачає невизначеності та зміни, які не можуть бути виражені формальною логікою, правилами та / або графіками.

Методи виявлення аномалії на основі статистики [8] будують статистичний профіль (а саме універсальний або багатоваріантний статистичний розподіл) звичайної діяльності суб'єкта з історичних даних. Ці

статистичні профілі не враховують порядок, в якому відбуваються події суб'єкта. Хоча ці методи виявлення аномалії на основі статистичних даних продемонстрували перспективні показники для виявлення кібератаки щодо швидкості виявлення та помилкової частоти тривоги, не ясно, чи можна досягти подальшого поліпшення ефективності виявлення атак шляхом впорядкування подій (переходи подій). Довідка [7] показує, що виявлення атаки, використовуючи декілька послідовних подій протягом заданого періоду, дає кращу ефективність, ніж використання однієї події в один момент часу. Оскільки для багатьох кібератак потрібна низка пов'язаних подій для здійснення, можливо покращити ефективність виявлення атак шляхом включення впорядкування подій.

Існує кілька методів виявлення аномалії, які досліджують впорядкованість подій для виявлення кібератаки. Як нейромережі, що повторюються, так і персептрон використовувались для прогнозування наступної події (наприклад, виклик системи командора) з ряду минулих подій. Методи виявлення аномалії на основі імунології зберігають великий набір послідовних подій як нормальну поведінку суб'єкта, і використовують негативний або позитивний алгоритм відбору для виявлення послідовностей, що надходять, що відрізняються від послідовностей подій у профілі норми. Методи виявлення аномалій на основі оцінки параметрів Байєса для побудови моделі Маркова для тестування профілю профілю та коефіцієнта ймовірності відношення, байєсівських мереж, прихованих моделей маркова, або аналізу основних компонентів, також досліджуються. Однак навчання всіх цих прийомів з урахуванням впорядкованості подій обчислювально інтенсивне. Висновок аномалій у багатьох вищезазначених методиках також є обчислювально інтенсивним. Методи виявлення аномалії на основі імунології потребують великої кількості пам'яті для зберігання великого набору наслідків подій. Методи виявлення аномалії на основі моделі Маркова обчислювально інтенсивні завдяки їх використанню оцінки параметрів Байєса для вивчення того чи іншого профілю та тестування

коефіцієнта ймовірності для виведення аномалій. У комп'ютерній та мережевій системі ці методи не застосовуються для виявлення кібератак у режимі реального часу.

У статті представлено обчислювально ефективний метод застосування моделі МС для виявлення комп'ютерних вторгнень. У цій статті представлено подальше вивчення цього методу, зокрема щодо стійкості техніки МС до шуму під час впровадження в систему.

## 1.5 Виявлення зловмисного програмного забезпечення на мобільних пристроях

Значно збільшилося використання мобільних пристроїв для передачі даних в додаток до голосових послуг. Нова мережева інфраструктура орієнтована на поліпшення послуг передачі даних цим пристроям. Трансляція засобів масової інформації на мобільних пристроях та оплата мобільними кредитними картками є прикладами таких послуг. По мірі того, як інфраструктура мобільної мережі продовжує зростати, зростає і спектр функціональних можливостей мобільної трубки. Мобільні платформи, такі як Symbian Operating System, WindowsMobile тощо, дозволяють виконувати складні програми, яких раніше не було помічено на таких пристроях. Однак усі ці розробки зробили мобільні пристрої привабливою ціллю для хакерів та авторів вірусів. Доступ / знищення приватної інформації, що зберігається на мобільних пристроях, сьогодні серйозно турбує користувачів. Крім цього, розповсюдження черв'яків може мати серйозний вплив на продуктивність мобільної мережі. Це пояснюється нелегальним використанням пропускної здатності, що може бути помічено, оскільки наявна смуга пропускання все ще є дещо обмеженою. Таким чином, зловмисне програмне забезпечення в мобільному домені може призвести до фінансових втрат як для користувача, так і для оператора мережі.

Останні два роки мобільне шкідливе програмне забезпечення постійно

збільшується. Навіть незважаючи на те, що наявне програмне забезпечення порівняно просте порівняно з ПК, очікується, що майбутні атаки можуть призвести до більшого збитку. Отже, важливо вирішити проблему зловмисного програмного забезпечення для мобільних пристроїв, перш ніж досягти тривожних масштабів. Оскільки галузь проти зловмисного програмного забезпечення для ПК досить зріла, існує чимало загальних методик, розроблених для боротьби зі зловмисними програмами. Але важливо відзначити основні відмінності між ПК та мобільним пристроєм, особливо обмеження в обчислювальній потужності, пам'яті та обмежених ресурсах акумулятора. Цільова експлуатація зловмисного програмного забезпечення для мобільних пристроїв також суттєво відрізняється від функцій на ПК через різницю в операційних системах та апаратних засобах. Наприклад, більшість мобільних пристроїв базуються на архітектурі ARM [9]. Отже, нам потрібно приділяти належну увагу, використовуючи методи, засновані на ПК для мобільних пристроїв. Деякі критерії виявлення зловмисного програмного забезпечення на мобільних пристроях включають:

- Метод виявлення повинен ефективно використовувати пам'ять та обчислювальні ресурси, а не розряджати акумулятор пристрою.

- Метод виявлення повинен мати низький показник помилкової тривоги, тобто вважати, що не зловмисне програмне забезпечення є зловмисним.

- Метод виявлення повинен бути простим / економічно ефективним для оновлення по бездротовій мережі.

Існує два загальних способи захисту мобільного пристрою. Один полягає в тому, щоб забезпечити захист на рівні пристрою, а другий - забезпечити захист на мережевому рівні шляхом перевірки пакетів, призначених для пристрою. Захист на рівні пристрою виявляє та очищає зловмисне програмне забезпечення, включаючи віруси, трояни та шпигунські програми, встановлені на пристрої, тоді як захист на рівні мереж виявляє та запобігає вторгненням з мережи. Для цього дослідження ми розглядаємо

захист, заснований на пристроях, який сканує файли пристрою на наявність шкідливих програм. Оскільки виявлення зловмисного програмного забезпечення на рівні ПК є добре вивченою проблемою, деякі методи були розроблені для анти-шкідливих програм на базі ПК, включаючи:

- виявлення на основі сигнатури;
- евристичне сканування;
- емуляція

Виявлення на основі сигнатури ґрунтується на пошуку заздалегідь визначених сигнатур вірусів у вхідних файлах. Сигнатура віруса - це послідовність байтів, унікальна для вірусу, і не зустрічається у нормальних файлах. Така сигнатура зазвичай витягується вірусознавцем після аналізу. Вилучення сигнатури вручну може часом бути трудомістким процесом. Однак деякі системи автоматичного вилучення сигнатур [10] успішно розроблені з метою прискорити процесу. Виявлення сигнатур - це широко використовуваний метод на домені ПК через його простоту та його здатність видавати низьку помилкову тривогу. Однак виявлення сигнатур має той недолік, що він не може виявити нові віруси, для яких сигнатури не були вилучені. Загальні сигнатури можуть у певній мірі виявити варіанти існуючих вірусів. Для цього сигнатури повинні бути вилучені з вмісту, загального для всіх варіантів.

На відміну від виявлення підписів, евристичне сканування базується на виявленні загальних функціональних особливостей / особливостей вірусів. Теорією евристичного сканування є виявлення нових вірусів на основі його загальних особливостей із існуючими. Однак він має недолік, що іноді схильний до помилкових сигналів. Оскільки мобільні віруси з'явилися лише нещодавно, є дуже мало зразків мобільних вірусів порівняно з кількістю доступних зразків у домені ПК. Відсутність достатньої кількості вірусних зразків у мобільному домені ускладнює видобуток надійних загальних особливостей та розробку евристичних методів виявлення на основі наявних даних. Якщо методи евристичного сканування розроблені з використанням

обмежених даних, це може зробити їх більш схильним до помилкових сигналів тривоги або налаштовуватися на дані тренувань, тобто воно може виявляти лише віруси навчального набору і не дуже добре узагальнювати. Це також може вимагати великих змін або оновлень для набору ознак дуже часто, оскільки набір ознак не дуже стабільний. Отже, ми не враховуємо використання евристичного сканування на мобільних пристроях на цьому етапі.

Емуляція - це техніка, коли ми дозволяємо вірусу запускатись у віртуальному середовищі, щоб безпечно встановити схему виконання вірусу. Основна проблема адаптації цієї техніки для мобільних пристроїв полягає в тому, що вона потребує значної кількості обчислювальної потужності та пам'яті. Цей метод спочатку був розроблений для вірусів на основі ПК, де обмеження на пам’ять та обчислювальні ресурси не такі великі, як на мобільних пристроях. Отже, на обмежених пристроях, таких як мобільні телефони, ми не розглядаємо цей підхід. Цей метод все ще може ефективно застосовуватися для аналізу зловмисних програм для мобільних пристроїв у віддалених лабораторіях, щоб встановити, чи є новий зразок вірусом чи ні. Інші методи можуть бути використані для виявлення та очищення цього вірусу на фактичному пристрої.

Враховуючи міркування щодо виявлення зловмисного програмного забезпечення на мобільному пристрої, ми розробляємо метод виявлення на основі сигнатур. Слід також зазначити, що виявлення сигнатур залишатиметься частиною антивірусних мобільних систем, навіть якщо будуть розроблені інші методи, такі як евристичне сканування, оскільки це сприяє швидкому виявленню відомих типів вірусів. Тут нашою метою є розробити метод відповідності сигнатур таким чином, щоб він мав високу швидкість сканування та низьке використання пам'яті, що робить його придатним для мобільних пристроїв. Зокрема, ми використовуємо хеш-таблицю для зберігання хеш-значень підписів для збільшення швидкості. Крім того, ми прагнемо усунути невідповідність при зіставленні не

шкідливих файлів, використовуючи ймовірність появи байтів сигнатур в нешкідливому вмісті. З метою оцінювання ми порівнюємо продуктивність нашого сканера з відомим сканером Clam-AV, який використовує алгоритм багатостороннього узгодження Aho-Corasick для виявлення на основі сигнатур. Результати на мобільному пристрої показують, що наш сканер використовує до 50% менше пам’яті зі швидкістю сканування трохи кращою, ніж у Clam-AV.

Symbian OS - найпопулярніша мобільна платформа для смартфонів. Це також платформа, на яку націлилися вірусові автори і має найбільшу кількість мобільних вірусів. Отже, ми проводимо тестування та оцінку нашого алгоритму відповідності на пристроях Symbian OS. На даний момент у нас є близько 82 сигнатур вірусів, найменша довжина яких дорівнює 32 байтам, а довге рівне - 128 байт. Ці підписи здатні виявити 362 мобільні зловмисні програми, які є найбільшою частиною поточних мобільних вірусів. Для того, щоб оцінити ймовірність появи, ми використовуємо статистику, отриману з набору 1000 нешкідливих мобільних додатків. Наша оптимізація процесів відповідності підпису базується на фактичних даних, які часто використовуються в не шкідливих даних, ніж інші. Це проілюстровано на рисунку 1.3, який показує ймовірність виникнення для кожного можливого значення одного байта від 0 до 255. Ці значення отримуються шляхом збору частоти появи для всіх можливих значень байтів у нешкідливих даних тренінгу. Як ми можемо спостерігати, байти як 0 більше домінуючі в нешкідливих даних, ніж інші байти. Ми використовуємо ці знання для мінімізації кількості порівнянь під час підпису.

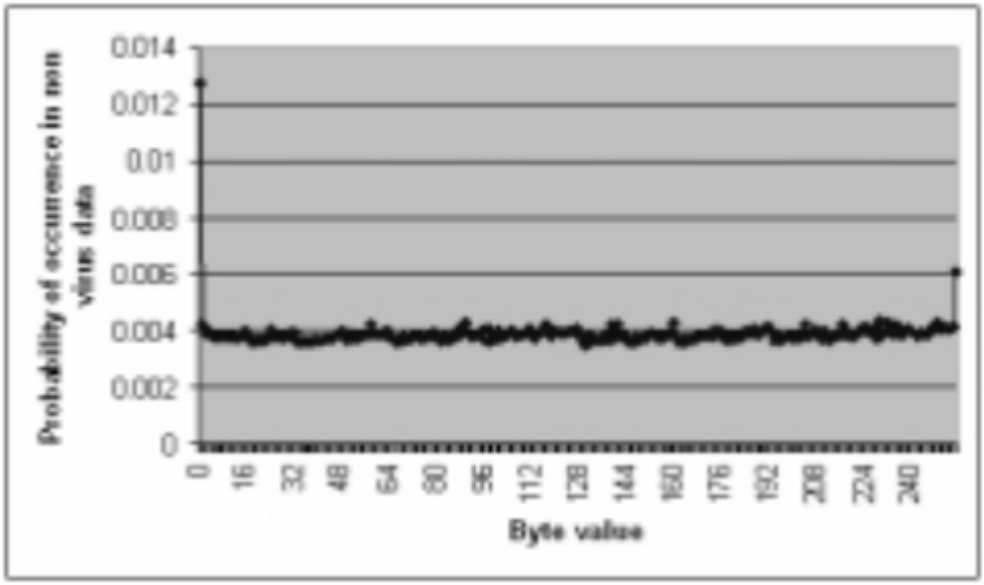


Рисунок 1.3 - Виникнення байтів у нешкідливих даних

## 1.6 Використання N-грам для виявлення зловмисного програмного забезпечення

Термін "зловмисне програмне забезпечення" було введено для того, щоб назвати будь-яку комп'ютерну програму зі шкідливими намірами, наприклад, віруси, черв'яки або троянські коні. Паралельно із зростанням Інтернету кількість потужних та різноманітних зловмисних програм збільшується з кожним роком, а також їх здатність уникати будь-яких бар'єрів безпеки. Класичний метод виявлення цих загроз полягає в очікуванні зараження певної кількості комп’ютерів, визначенні сигнатури файлу для вірусу і нарешті пошуку конкретного рішення для нього. Таким чином, спираючись на список сигнатур (також відомий як база даних сигнатур), програмне забезпечення для виявлення зловмисних програм може забезпечити захист від відомих вірусів (тобто тих, які в цьому списку). Цей підхід виявився ефективним, коли загрози заздалегідь відомі, і є найбільш розширеним рішенням антивірусного програмного забезпечення.

Більше того, при появі нового вірусу і до отримання відповідної сигнатури файлу мутації вихідного вірусу, можуть привести до обігу

виявлення на основі старої сигнатури.

Ці факти призвели до ситуації, коли автори шкідливих програм розробляють нові віруси та різні способи приховування свого коду, в той час як дослідники розробляють нові інструменти та стратегії їх виявлення. Така еволюція дуже ускладнює розробку універсального детектора зловмисних програм. Таким чином, завдання дослідника - досягти дуже хороших результатів проти відомих зловмисних програм та ускладнити спробу написання нового невизнаного вірусу. Як правило, існує декілька показників для оцінки ефективності нової системи виявлення шкідливих програм. По-перше, ми повинні переглянути коефіцієнт виявлення зловмисного програмного забезпечення (тобто кількість вірусів у зразку, який програмне забезпечення виявляє). По-друге, ми повинні дивитися на хибнопозитивне співвідношення (тобто кількість нешкідливих програм, помилково класифікованих як зловмисне програмне забезпечення), оскільки воно визначає, наскільки практичним є метод комерціалізації: нова система, здатна виявити багато шкідливих програм, але ціною високого показника помилкових позитивних результатів не є практичною у реальному світі, коли система повинна мати справу з безліччю доброякісних програм, які не повинні класифікуватися як зловмисне програмне забезпечення. Тому антивірусні компанії, як правило, віддають перевагу скромному коефіцієнту виявлення з низьким (або в ідеалі нульовим) хибнопозитивним співвідношенням, а не помітним виявленням з високим також хибнопозитивним співвідношенням.

Розпізнавання мови - це дослідницька область, яка вирішила подібну проблему, оскільки їм також доводиться мати справу з отриманням інформації, яку важко помітити на перший погляд. Найпоширенішою технікою проти цієї проблеми було використання так званих n-грамів. Наприклад, у китайській класифікації документів або класифікації тексту. N-грам - це всі підрядки більшого рядка довжиною n. Рядок просто розбивається на підрядки фіксованої довжини n. Наприклад, рядок

"MALWARE" можна розділити на кілька 4-х грам: "MALW", "ALWA", "LWAR", "WARE" тощо. По-перше, ми розглянемо тут нову методологію виявлення зловмисного програмного забезпечення, засновану на використанні n-грамів для створення підписів файлів. По-друге, ми вирішуємо питання розгляду помилкових позитивних даних, використовуючи параметр з назвою d, щоб контролювати, наскільки жорсткий метод поводить класифікацію примірника як шкідливого програмного забезпечення або доброякісного програмного забезпечення, щоб уникнути помилкових позитивних результатів.

В експериментах ми побудували підписи файлів, які складаються з набору n-грамів для кожного файлу у вибірці (2000 файлів), для n = 2, n = 4, n = 6 і n = 8. Ці файли містять n-грамів текстів, отриманих із виконуваних файлів. Ми розділимо ці файли, що містять підписи файлів, на два окремих файли на кожне значення n. Один з двох файлів містить n-грам файлів, які будуть використовуватися як навчальний набір, а інший містить n-грам файлів, які будуть використовуватися як тестовий набір. На малюнку 1 показано отримане помилкове відношення та коефіцієнт виявлення щодо параметрів n, k і d. Точніше, для n = 2 коефіцієнт виявлення є досить низьким, досягаючи максимального коефіцієнта виявлення 69,66%, тому 2-грам, здається, не підходить для підписів файлів. У цьому експерименті ми домоглися найкращих результатів для коефіцієнта виявлення для n = 4, отримавши максимальне відношення виявлення 91,25% для k = 7 і d = 2. Для наступних n значень коефіцієнт виявлення нижчий, ніж для n = 4, однак, другий найкращий результат досягається для значення n 8. Крім того, самі помилкові позитивні коефіцієнти та коефіцієнт виявлення не показують потенціал системи. Якщо ми зосередимось на найкращих результатах, коли коефіцієнт помилкового позитивного значення 0%, ми досягнемо найкращого результату при n = 4, k = 17 і d = 17, зберігаючи 74,37% коефіцієнта виявлення.

Таким чином, наш метод використовує n-грамові підписи для

досягнення виявлення невідомої шкідливої програми. У нашому експерименті ми вибрали набір декомпільованого коду зловмисного програмного забезпечення та текстів, які є результатом моніторингу виконання цього файлу, щоб діяти як невідомі загрози, а інший набір діяв як відомих загроз. Під цими приміщеннями ми витягуємо n-грамів відомих файлів і n-грамів невідомих. Для будь-якого екземпляра невідомого набору ми будемо класифікувати шкідливі програми чи доброякісні програми, ґрунтуючись на збігах n-грамів між набором n-грамів відомих файлів та набором n-грамів невідомих. Ми продемонстрували, що методології сигнатур на основі n-грамів можуть досягти виявлення нових або невідомих зловмисних програм. Крім того, із включенням параметра d у метод класифікації ми досягли налаштування того, наскільки сувора буде система для класифікації невідомого примірника як зловмисного програмного забезпечення; таким чином, це спосіб контролювати хибнопозитивне співвідношення. Цей метод забезпечує хороший коефіцієнт виявлення і можливість контролювати хибнопозитивне відношення. Таким чином, цей метод може бути застосований для виявлення стільки зловмисного програмного забезпечення, скільки він може, або його можна налаштувати, щоб дати 0% хибне позитивне співвідношення.

# 2 МЕТОДИ МАШИННОГО НАВЧАННЯ ВИКОРИСТАНІ В РОБОТІ

## 2.1 Прихована марковська модель

Прихована марковська модель (HMM – Hidden Markov Model) є статистичної марковської моделлю, в якій модельована система вважається марковским процесом з неспостережуваними (прихованими) станами. Прихована марковська модель може бути представлена як найпростіша динамічна баєсова мережа.

У простіших марковских моделях (наприклад, ланцюги Маркова) стан безпосередньо видимий спостерігачеві, тому ймовірності переходу стану є єдиними параметрами, а в прихованій марковської моделі стан не видно безпосередньо, але вихід (у вигляді даних або «токена»), що залежить від стану, видно. Кожне стан має розподіл ймовірності по можливим вихідним токенам. Тому послідовність токенів, що генеруються HMM, дає деяку інформацію про послідовність станів.

Прикметник приховане відноситься до послідовності станів, через яку проходить модель, а не до параметрів моделі; модель називається прихованої марковської моделлю, навіть якщо ці параметри (ймовірності переходу) відомі точно.

Важливим прикладом ланцюгів Маркова є дискретні марковські моделі. Кожен стан $s_t$ може приймати один з дискретної безлічі станів, а ймовірність переходу з одного стану в інший визначається таблицею ймовірностей для всієї послідовності станів. Більш конкретно, $s_t \in \{1, ..., K\}$ для деякого кінцевого K, и для всіх t, $P(s_t = j \mid s_{t-1} = i) = A_{ij}$, де A - параметр моделі, який є фіксованою матрицею допустимих ймовірностей (таких, що $A_{ij} \geq 0$ и $\sum_{j=1}^{K} A_{ij} = 1$). Щоб повністю охарактеризувати модель, також потрібен розподіл станів для першого кроку часу: $P(s_1 = i) = a_i$.

Прихована марковська модель (HMM) моделює часові ряди

спостережень $y_{1:T}$, які визначаються прихованим дискретним ланцюгом Маркова $s_{1:T}$. Зокрема, передбачається, що вимір $y_t$ визначається розподілом «випромінювання», яке залежить від прихованого стану в момент часу t: p ($y_t$ | $s_t$ = i). Як правило, $s_t$ кодує основну структуру часових рядів, в той час як $y_t$ відповідає вимірам, які фактично спостерігаються.

Спільний розподіл по спостережуваним і прихованим значенням:

$$p(s_{1:T}, y_{1:T}) = P(y_{1:T} \vee s_{1:T})P(s_{1:T}) \quad (2.1)$$

де,

$$P(s_{1:T}) = P(s_1)\prod_{t=1}^{T} P(s_t \vee s_{t-1}) \quad (2.2)$$

$$P(y_{1:T}|s_{1:T}) = \prod_{t=1}^{T} p(y_t \vee s_t) \quad (2.3)$$

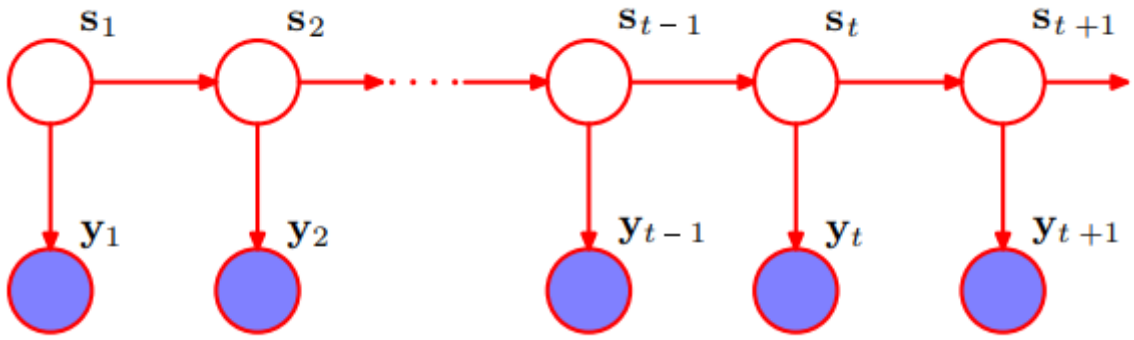


Рисунок 2.1 – Змінні в HMM і їх умовні залежності (відображені стрілками).

Повна HMM складається з наступних параметрів: a, A і параметрів розподіла Ґауса «випромінювань»$\mu_{1:K}$ та $\Sigma_{1:K}$ [11].

## 2.2 Метод опорних векторів

Метод опорних векторів – набір алгоритмів навчання з учителем, що використовуються для задач класифікації та регресійного аналізу. Належить до сімейства лінійних класифікаторів. Особливою властивістю методу опорних векторів є безперервне зменшення емпіричної помилки класифікації і збільшення зазору, тому метод також відомий як метод класифікатора з

максимальним зазором.

Основна ідея методу - переклад вихідних векторів в простір більш високої розмірності і пошук розділяючої гіперплощини з максимальним зазором в цьому просторі. Дві паралельні гіперплощини будуються по обидва боки гіперплощини, що розділяє класи. Розділяючою гіперплощиною буде гіперплощина, що максимізує відстань до двох паралельних гіперплощин[12].

Припустимо, що точки мають вигляд:

$$\{(x_1, c_1), (x_2, c_2), \ldots, (x_n, c_n)\} \quad (2.4)$$

де $c_i$ приймає значення 1 або –1, в залежності від того, якому класу належить точка $x_i$. Кожне $x_i$ – це p-мірний дійсний вектор, зазвичай нормалізований значеннями [0,1] або [–1,1]. Якщо точки не будуть нормалізовані, то точка з великими відхиленнями від середніх значень координат точок занадто сильно вплине на класифікатор. Для кожного елемента вже заданий клас, до якого він належить. Необхідно, щоб алгоритм методу опорних векторів класифікував їх таким же чином. Для цього будується розподіляюча гіперплощина, яка має вигляд:

$$w \cdot x - b = 0 \quad (2.5)$$

Вектор w – перпендикуляр до розподіляючої гіперплощини. Параметр b дорівнює по модулю відстані від гіперплощини до початку координат. Якщо параметр b дорівнює нулю, гіперплощина проходить через початок координат.

Необхідно визначити опорні вектора і гіперплощини, паралельні оптимальної і найближчі до опорних векторів двох класів. Можна показати, що ці паралельні гіперплощини можуть бути описані наступними рівняннями (з точністю до нормування):

$$w \cdot x - b = 1 \quad (2.6)$$

$$w \cdot x - b = -1 \quad (2.7)$$

Якщо навчальна вибірка лінійно роздільна, можна вибрати гіперплощини таким чином, щоб між ними не лежала жодна точка навчальної

вибірки і потім максимізувати відстань між гіперплощинами. Ширина смуги між ними дорівнює$\frac{2}{\|w\|}$. Щоб виключити всі точки зі смуги, необхідно переконатися для всіх i, що

$$c_i(w \cdot x - b) \geq 1, \quad 1 \leq i \leq n \tag{2.8}$$

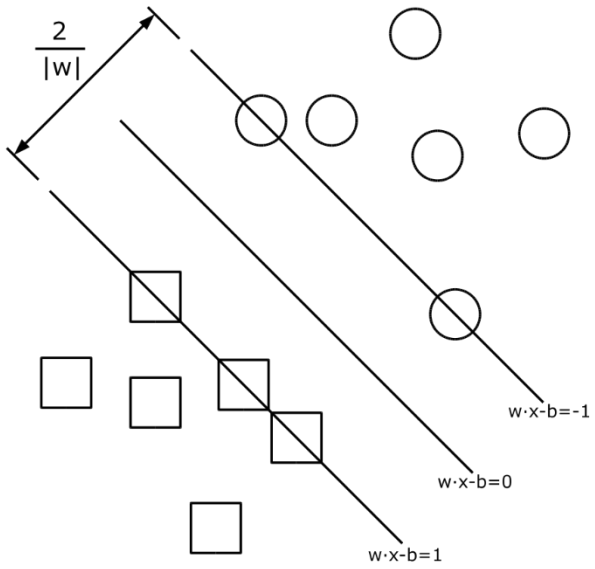


Рисунок 2.2 – Оптимальна розподіляюча гіперплощина для методу опорних векторів, побудована на точках з двох класів.

## 2.3 Наївний баєсів класифікатор

Наївний баєсів класифікатор є простим імовірнісним класифікатором, заснованим на застосуванні теореми Баєса зі строгими (наївними) припущеннями про незалежність.

Наївний Баєс - це простий спосіб побудови класифікаторів: моделі, які присвоюють мітки класів екземплярам, представленим як вектори значень ознак, де мітки класів вибираються з деякого кінцевого набору. Існує сімейство алгоритмів для навчання таких класифікаторів, засноване на загальному принципі: всі класифікатори наївного Баєса припускають, що значення певної ознаки не залежить від значення будь-якої іншої ознаки

примірника.

Проблема інших класифікаторів стосується великої кількості ознак, необхідних для вивчення правдоподібною моделі. В умовних моделях класів Ґауса з d-мірними вхідними векторами необхідно оцінити математичне сподівання і коваріаційну матрицю для кожного класу. Математичне сподівання буде представлено d-мірним вектором, але число невідомих в коваріаційній матриці зростає квадратично з d. Тобто дисперсія є матрицею d × d.

Наївний Баєс спрямован на спрощення завдання оцінки, припускаючи, що різні вхідні ознаки (наприклад, різні елементи вхідного вектора) умовно незалежні. Математично для ознак x = ($x_1$, …,$x_n$) и класів $C_k$ баєсів класифікатор це функція, яка присвоює мітку класу:

$$y = \underset{k\in\{1,\dots,K\}}{max} p(C_k) \prod_{i=1}^{d} p(x_i \vee C) \qquad (2.9)$$

Виходячи з цього припущення, замість оцінки однієї d-мірної щільності, оцінюються d 1-мірних щільності. Це важливо, тому що кожен 1D розподіл Ґауса має тільки два параметри, його математичне сподівання і дисперсію, обидва з яких є скалярами. Таким чином, модель має 2d невідомих. У разі розподіла Ґауса, модель наївного Байеса ефективно замінює загальну коваріаційну матрицю d × d діагональною матрицею. На діагоналі коваріаційної матриці є d записів; i-й запис - це дисперсія $x_i|C$.

Незважаючи на простий дизайн і спрощені припущення, класифікатори наївного Баєса показали задовільні результати у багатьох складних ситуаціях. Проте, всебічне порівняння з іншими класифікаційними алгоритмами в 2006 році засвідчило, що класифікація Баєса поступається іншим підходам, таким як boosted trees або random forests.[13]

## 2.4 Випадковий ліс

Випадковий ліс - це класифікатор, що складається з набору деревовидних класифікаторів {h(x, $\Theta_k$), k = 1, ...}, де {$\Theta_k$} – незалежні,

однаково розподілені випадкові вектори, і кожне дерево віддає один голос за найпопулярніший клас на вході x. Помилка узагальнення для лісів сходиться до границі з ростом кількості дерев в лісі і залежить від сили окремих дерев в лісі і кореляції між ними.

Значні поліпшення в точності класифікації стали результатом зростання ансамблю дерев і дозволу їм голосувати за найпопулярніший клас. Для вирощування цих ансамблів часто генеруються випадкові вектори, які визначають зростання кожного дерева в ансамблі. Раннім прикладом є беггінг, де для вирощування кожного дерева виконується випадковий вибір (без заміни) з прикладів в навчальному наборі. Іншим прикладом є вибір випадкового поділу, де на кожному вузлі поділ вибирається випадковим чином з числа K кращих розділень. Ще один підхід до навчання полягає в тому, щоб вибрати набір тренувань з випадкового набору вагів із прикладів в навчальному наборі.

Спільним елементом у всіх цих процедурах є те, що для k-го дерева генерується випадковий вектор $\Theta_k$, що не залежить від минулих випадкових векторів $\Theta_1$, ..., $\Theta_{k-1}$, але з тим же розподілом; і дерево вирощується з використанням навчального набору і $\Theta_k$, в результаті чого виходить класифікатор h (x, $\Theta_k$), де x - вхідний вектор. Наприклад, при беггінге випадковий вектор Θ генерується як кількість відліків в N полях, отриманих з N дротиків, кинутих навмання в поля, де N - кількість прикладів в навчальному наборі. У випадковому поділі вибір Θ складається з ряду незалежних випадкових чисел від 1 до K. Характер і розмірність Θ залежать від його використання в побудові дерева.[14]

## 2.5 Методи градієнтного спуску

Градієнтний спуск [15] - загальна парадигма, яка лежить в основі багатьох алгоритмів машинного навчання та пошуку знань. У нейронних мережах оновлення вагового значення вихідних та прихованих вузлів є

формою градієнтного спуску. Важливою перевагою методу градієнтного спуску є те, що він може обробляти набір вхідних даних по одному рядку, так що потреба в пам'яті є низькою, оскільки під час обчислення потрібно зберігати лише вектори рядків і ваги.

### 2.5.1 Стандартний по відношенню до стохастичного градієнта

Дано набір зразків даних $D = \{(x_i, y_i) \mid i = 1, 2, \ldots, N\}$ де кожен $x_i = \{x_{i,1}, x_{i,2}, \ldots, x_{i,m}\}$ являє собою набір m атрибутів і-ої вибірки, а $y_i$ відповідає цільовому атрибуту, ми хочемо визначити цільову функцію $f(x_i)$, яка дає найменшу помилку при прогнозуванні значення у невідомого зразка. Це еквівалентно знаходженню набору значень ваги $\{\omega_1, \omega_2, \ldots, \omega_m\}$ таким чином, що загальна помилка прогнозування зведена до мінімуму.

Градієнтний спуск підходить для пошуку простору функцій помилок для отримання оптимальної функції, яка мінімізує помилку передбачення стосовно даного набору зразків даних. Якщо набір зразків даних лінійно відокремлюється, підходи градієнтного спуску здатні визначити оптимальний набір вагових значень для вагового вектора. Загалом, метод градієнтного спуску виконує низку ітерацій для сходження до оптимального вагового вектора. Він використовує початковий вектор ваги для визначення похибки прогнозування. Якщо похибка не є задовільно низькою, ваговий вектор модифікується для зменшення похибки. При кожній ітерації ваговий вектор спрямовує пошук функції помилок у напрямку самого крутого спуску. Цей процес триває до досягнення глобальної мінімальної точки помилки.

При стандартному градієнтному спуску всі зразки даних обробляються на кожній ітерації для визначення найбільш крутого спуску. Якщо зразки даних лінійно відокремлюються, то глобальна мінімальна точка досяжна. Однак якщо зразки даних не є лінійно відокремленими, то стандартний метод ніколи не збігатиметься, оскільки оптимальна точка не існує. Стандартний метод градієнтного спуску має ще одну слабкість. Що стосується складності

обчислень, метод вимагає, щоб усі вибірки даних були оброблені при кожній ітерації. Це має практичні обмеження, коли набір даних дуже великий.

На практиці застосовується стохастичний варіант градієнтного спуску. При кожній ітерації стохастична версія обробляє лише один зразок даних для визначення найбільш крутого спуску та оновлення вагового вектора. Важливою перевагою стохастичного підходу є те, що метод буде конвергуватися (і значно швидше) навіть тоді, коли вибірки даних не є лінійно відокремленими. Однак, у цьому випадку, буде досягнуто лише місцевого мінімуму.

При використанні стохастичного варіанту градієнтного спуску, швидкість конвергенції не залежить від частки даних, що утримуються кожною стороною, оскільки стохастичний градієнтний спуск обробляє дані послідовно.

Зауважимо, що функція помилок у методах градієнтного спуску забезпечує засоби для спуску, а функція оновлення вагового вектора визначає спосіб виконання найбільш крутого спуску. Загалом, поки функція помилки є будь-якою формою частково диференційованої функції, функція оновлення ваги може бути отримана для ефективного виконання градієнтного спуску. Це стосується алгоритмів машинного навчання, які включають кілька функцій помилок (і, відповідно, кілька функцій оновлення), таких як багатошарові нейронні мережі.

## 2.6 Бустінг та алгоритм Adaboost

Boosting — це потужній алгоритм машинного навчання для виконання класифікацій об’єктів. Дано кілька об’єктів, які належать до одного з двох заданих класів. Алгоритм підвищення рівня полягає в класифікації об'єктів у кожному з двох класів шляхом дослідження їх характеристик. Boosting об'єднує результати декількох слабких класифікаторів для побудови сильного класифікатора. А слабкий класифікатор трохи краще, ніж

випадковий класифікатор. Однак сильний класифікатор достатньо близький до ідеального класифікатора.

Boosting розвиває лінійну комбінацію вхідного набору слабких класифікаторів, щоб розробити сильний класифікатор. Це задає більшою вагою для точнішого слабкого класифікатора і зменшується – для менш точного слабкого класифікатора. Рис. 2.3 ілюструє конфігурацію слабких класифікаторів, сильних класифікатор та алгоритм Boosting.

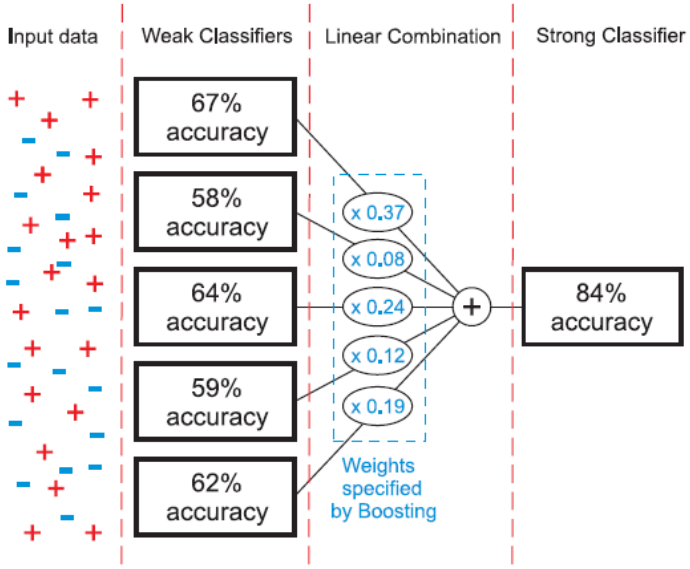


Рисунок 2.3 – Слабкі та сильні класифікатори в алгоритмі Boosting

### 2.6.1 Алгоритм AdaBoost

AdaBoost – це скорочення від Adaptive Boosting. Це відхилення від алгоритму Boosting, який визначає вагу адаптивно. Алгоритм AdaBoost, презентували у 1995 році Фрейндом та Шапіром, який вирішив багато практичних труднощів від ранніх алгоритмів Boosting. Нижче приведена спрощена версія AdaBoost алгоритму.

Алгоритм приймає в якості вхідного навчального набору $(x_1, y_1)$, $\ldots(x_m, y_m)$, де кожен $x_i$ є об'єктом у вхідних навчальних даних, що належать до класу $y_i$, і кожне $y_i$ має значення +1 або −1. У цій роботі ми не розширюємо

AdaBoost для багато класових проблем. Слабкі класифікатори $C_1$; … $C_T$ також передбачені для AdaBoost. Кожен $C_t$ це функція, яка приймає об'єкт $x_i$ і видає здогадку $\tilde{y}_i$. Визначенний клас $\tilde{y}_i$ повинен бути трохи відрізнятися від справжнього класу $y_i$. AdaBoost викликає слабкі класифікатори $C_t$ в серії раундів t = 1; … ; *T*, і обчислює вагу $\alpha_t$ для $C_t$ відповідно до його похибки класифікації $\epsilon_t$.

Алгоритм також зберігає ваги $\omega_1$; … $\omega_m$ для об'єктів навчання. Спочатку всі ваги встановлюються однаково, але на кожному раунді, ваги неправильно класифікованих предметів збільшуються так, що слабкі класифікатори в майбутніх раундах змушені зосередитися на важких прикладах у навчальному наборі. Псевдо-код для AdaBoost задається наступним чином:

- Вхідні дані

  $(x_1, y_1)$, …$(x_m, y_m)$ де $y_i \in \{+1,-1\}$ і $x_i \in \{objectspace\}$ (2.10)

  $C_1$; … $C_T$ є слабкими класифікаторами

- Тіло

  Ініціалізувати $\omega_i = \frac{1}{m}$

  Для *t* = 1, …T

  Розрахувати $\tilde{y}_i = C_t(x_t)$ для *i* = 1, …,*m*

  $$\epsilon_t. = \sum \omega_i (y_i \neq \tilde{y}_i) \quad (2.11)$$

  $$\alpha_t = \frac{1}{2} ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right) \quad (2.12)$$

  $$\omega_i = \omega_i exp\,(-\alpha_t y_t \tilde{y}_t)/Z_t \text{ для i = 1, …, } m \quad (2.13)$$

  $Z_t$ є фактором нормалізації для збереження $\omega$ розподілу

- Вихідні дані

  Кінцева гіпотеза (сильний класифікатор):

  $$H_x = sign(\sum_{t=1}^{T} \alpha_t C_t(x)) \quad (2.14)$$

Кінцева гіпотеза *H(x)* є або +1, або –1, що є прогнозуванням сильного класифікатора та значення $|\sum_{t=1}^{T} \alpha_t C_t(x)|$ забезпечує рівень довіри для класифікації. Рис.2.4 показує приклад двох переваг. Стимулювання щодо інших методів класифікації полягає в тому, що дискримінація порогів може

змінюватися. Для того щоб змінити поріг дискримінації до B, можна додати зміщення B до остаточного порогу дискримінації.

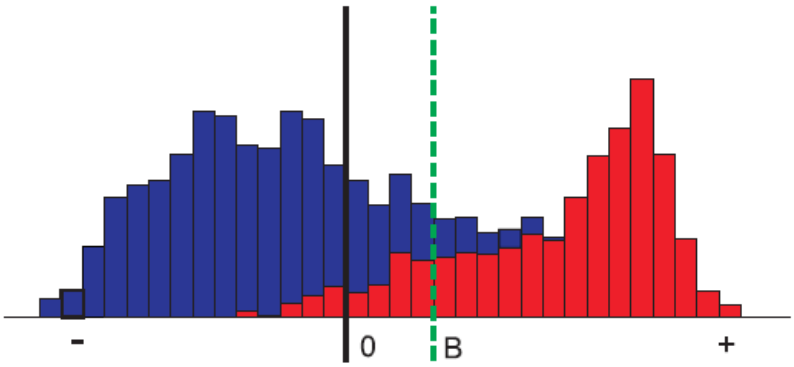


Рисунок 2.4 – Дискримінація в AdaBoost

### 2.6.2 Застосування Adaboost в проблемі оцінки безпеки

Вирішення проблеми оцінки безпеки енергосистеми з підходом AL засобами класифікації. У цій проблемі для класифікації множини робочих станів (OS), стабільності різних OS потужності визначається система після виникнення визначеної непередбаченої ситуації шляхом автономного моделювання. Цей набір ділиться випадковим чином на дві підмножини під назвою набір для навчання (LS) та набір для тренування (TS). Зазвичай розмір LS в 4–5 разів більший, ніж у TS [16].

По-перше, для спрощення приймається лише двійкова класифікація, тому набір OS класифікується на дві категорії: стабільний і нестабільний. Чітка межа (наприклад, час критичного очищення в перехідній стабільності) вибирається для поділу набору OS на дві підмножини. Цей критерій повинен бути чітким, щоб уникнути додаткові зайві складності. Нечітка межа може бути використана у подальших дослідженнях для отримання кращих показників.

Як було сказано вище, попередньо класифіковані OS діляться випадковим чином для генерації LS та TS. LS використовується для навчання системи. Кожен член LS складається з різних записів, які називаються

атрибутами. Ці атрибути є електричними станами OS як кут і величина фазової напруги, активний і реактивний потік електроенергії по лініях електропередачі, топологічний стан мережі енергосистеми та інші. Як показано ці атрибути можуть мати безперервний час або дискретні час. Алгоритм AdaBoost використовує функції для цих атрибутів на кожному члені LS.

Після цього AdaBoost класифікує LS за допомогою функцій. Час навчання залежить від розміру LS. Важливо вибрати правильний розмір для LS, щоб уникнути не навчання та зайвий додатковий час на навчання. Час тренувань зазвичай менше, ніж більшість методів AL. Точність методу може бути вимірювана шляхом застосування TS до навчальної мережі. Класифікація проводиться шляхом призначення різних ваг до кожної функції. Слід зазначити, що ці ваги є цінні тому що вони представляють функціональний внесок у стабільність системи щодо зазначеної непередбаченої ситуації.

## 2.7 Дерева рішень

DT - це інструмент підтримки прийняття рішень, який використовує двійкове дерево або модель для прогнозування можливих наслідків. Алгоритм обміну даними, що використовується у DT у запропонованому підході, є CART (Дерева класифікації та регресії), який вперше був розроблений Breiman et al. у 1980-х роках [17]. Він широко застосовується в багатьох галузях, таких як фінансовий аналіз, ідентифікація хімічних компонентів та медична діагностика, і був впроваджений у сферу енергосистем Wehenkel et al.in 1989.

Як показано на рис. 2.5, враховуючи випадок, представлений набором вимірювань (тобто *A, B, C*, ...) для конкретного ОС, клас (тобто безпечний або незахищений) випадку можна передбачити, відкинувши вимірювання випадку вниз від кореневого вузла до кінцевого вузла DT. Вектор

предикторів може складатися як з числових змінних (наприклад, $A$), так і з категоричних змінних (наприклад, $B$). Змінні називаються числовими змінними, якщо їх вимірювання є реальними числами, тоді як категоричними змінними називаються, ті. що беруть значення із скінченної множини (наприклад, $S$), яка може не мати натурального впорядкування.

База даних, що складається з кількості випадків, необхідна для навчання ДТ, які випадковим чином поділяються на навчальний набір (LS) та тестовий набір (TS). LS використовується для вирощування серії DT при збільшенні розмірів, в той час як TS використовується для оцінки їх точності для визначення оптимальної DT. DT вирощується шляхом рекурсивного розподілення навчальних випадків на його вузлах. Основна ідея вибору кожного розподілення така, що навчаючі випадки у кожному нащадковому вузлі чистіші за батьківський вузол. Оптимальний вибір правил розподілення може бути обчислений за допомогою повторних спроб мінімізувати загальний індекс нечистості GINI. [17]

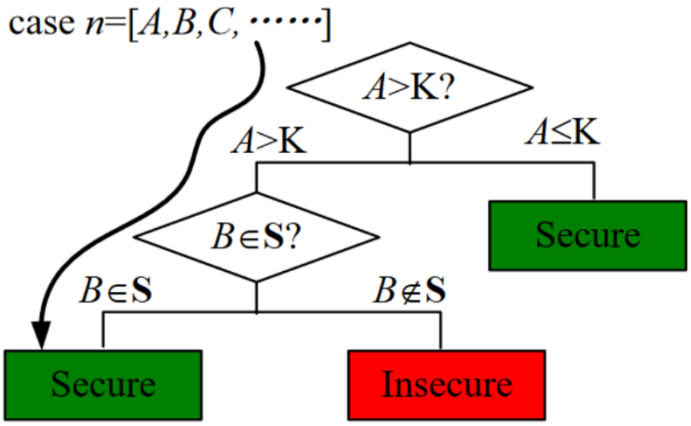


Рисунок 2.5 – Простий ілюстративний DT.

Попередньо скоригована ймовірність ефективно використовується для контролю правил розподілення щодо компромісу між чистотою класу та точністю класу. Ймовірність того, що справа потрапляє у вузол $t$, є

$$p_t = \sum_{i=1}^{J} \tag{2.15}$$

Де $\pi_i$, $N_i$ та $n_i (i = 1, \dots J)$ це попередні ймовірності, кількість випадків у

LS, кількість випадків, що містяться у вузлі для класу *i*, відповідно. Тоді умовна ймовірність класу *i* з урахуванням того, що випадок, досягнув вузла *t*, визначена у (2)

$$p_i(t) = \tag{2.16}$$

Таким чином ймовірності $p_L(t)$ та $p_R(t)$ того, що випадки у вузлі *t*, підуть до лівого нащадкового вузлу $t_L$, і до правого нащадкового вузлу $t_R$ визначаються як (3) і (4) відповідно

$$p_L(t) = p_t \; /p_t^{parent} \tag{2.17}$$

$$p_L(t) = p_t \; /p_t^{parent} \tag{2.18}$$

Як було сказано раніше, оптимальне правило розщеплення $\delta$ вибирається для максимального збільшення чистоти після розподілені, щоб мінімізувати загальну нечистоту, як визначено в (2.19), де *i(t)*, $i_L(t)$, $i_R(t)$ - показник домішки батьківського вузла, лівого нащадкового вузла та правого нащадкового вузла відповідно.

$$max\{\Delta(\delta, y)\} = max\{i(t) - p_L(t)i(t_L) - p_R(t)i(t_R)\} \tag{2.19}$$

Отже, скоригувавши попередні ймовірності $\pi_i$, можна скорегувати правила розподілення, щоб знайти область перекриття, як показано жовтим кольором на рис.2.6. Відповідно, області червоного та синього кольорів мають ймовірність винятку нижче $\pi_b$ та $\pi_r$ відповідно.

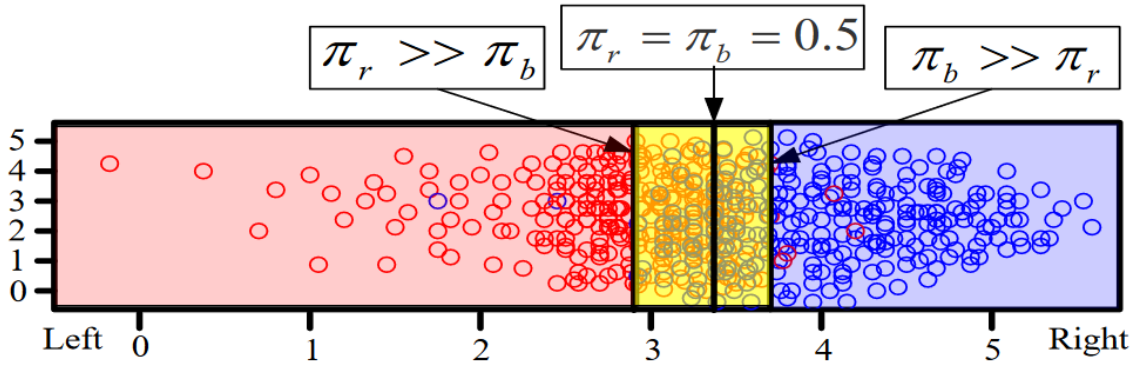


Рисунок 2.6 – Пороги DT w.r.t. попереднє коригування ймовірності

Для підвищення точності класифікації прийняті інші розширені методи, такі як випадкові ліси (RF). Випадковий ліс - це безліч некорельованих DT, так що кожне дерево залежить від випадкового

субвектора, вибраного з повного вектора предикторів. Вихід (безпечний або незахищений) моделі RF - це результат голосування від великої кількості DT, який може отримати вигоду від зменшення дисперсії на основі агрегації. Bootstrap sampling використовується в RF, щоб допомогти краще оцінити розподіл вихідного набору даних. Для кожного DT в RF приблизно одна третина випадків залишається поза bootstrap вибіркою, що називаються даними поза пакетом (OOB), які можуть бути використані для тестування моделі. Точність RF та CART порівнюється пізніше у цій роботі (результати див. у Таблиці I).

## 2.8 Штучні нейронні мережі

Штучний нейрон (далі – нейрон) є основою будь-якої штучної нейронної мережі. Нейрони являють собою відносно прості, однотипні елементи, що імітують роботу нейронів мозку. Кожен нейрон характеризується своїм поточним станом за аналогією з нервовими клітинами головного мозку, які можуть бути порушені і загальмовані.

Штучний нейрон, також як і його природний прототип, має групу синапсів (входів), які з'єднані з виходами інших нейронів, а також аксон - вихідний зв'язок даного нейрона - звідки сигнал збудження або гальмування надходить на синапси інших нейронів.

Кожен синапс характеризується величиною синаптичного зв'язку або вагою $w_i$, яка за своїм фізичним змістом еквівалентна електричної провідності. Поточний стан нейрона визначається як зважена сума його входів:

$$v = \sum_{i=1}^{n} x_i w_i, \tag{2.20}$$

де $x_i$ – вхід нейрона, а $w_i$ – відповідна цьому входу вага.

Выхід нейрона є функцією його стану:

$$y = \varphi(s) \tag{2.21}$$

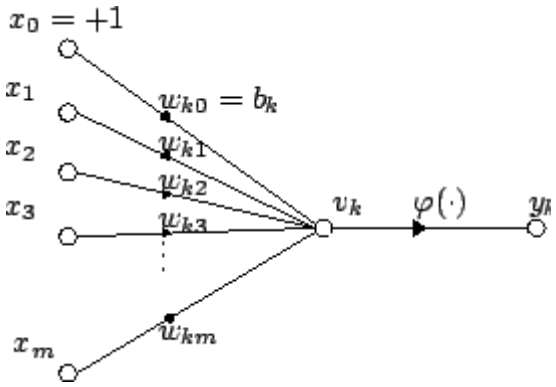


Рисунок 2.7 – Базова структура штучного нейрона.

Нелінійна функція φ (s) називається передавальною, функцією що стискає або функцією збудження нейрона. В якості передавальної функції часто використовується сигмоїдальна (s-образна або логістична) функція.

Функціонування нейронної мережі, а саме дії, які вона здатна виконувати, залежить від величин синоптичних зв'язків. Тому, задавшись структурою нейронної мережі, що відповідає певній задачі, розробник повинен знайти оптимальні значення для всіх вагових коефіцієнтів w.

Цей етап називається навчанням нейронної мережі, і від того, наскільки якісно він буде виконаний, залежить здатність мережі вирішувати під час експлуатації поставлені перед нею задачі. Найважливішими параметрами навчання є: якість підбору вагових коефіцієнтів і час, який необхідно витратити на навчання. Як правило, два цих параметра пов'язані між собою прямою залежністю і їх доводиться вибирати на основі компромісу. В даний час всі алгоритми навчання нейронних мереж можна розділити на два великі класи: з учителем і без вчителя.[18]

# 3 ДОСЛІДЖЕННЯ МЕТОДІВ МАШИННОГО НАВЧАННЯ ДЛЯ ДЕТЕКТУВАННЯ ШКІДЛИВОГО ПРОГРАМНОГО КОДУ

## 3.1 Отримання тестових даних

Для того, щоб навчити класифікатор потрібно мати розмічені дані. Для нашої задачі потрібно отримати набір чистих і шкідливих файлів PE формату. Проблеми отримання чистих файлів немає. Можна використовувати файли системи Windows відразу після її установки. Шкідливі ж файли можливо отримати зі сторонніх джерел. Був обраний сервіс VirusTotal. Сервіс дозволяє отримати вердикти 109 на сьогоднішній день антивірусів по файлам, що досліджувались. Варто зазначити, що для файлу не по всіх антивірусів може бути отриманий вердикт. Це відбувається з кількох причин. Наприклад, файл виявився в базі зовсім не давно і ще не був оброблений всіма антивірусами. Сервіс надає можливість як завантажити свій файл, так і отримати раніше завантажені файли з аналізом антивірусів, використовуючи платний API. На Рисунку 3.1 показана статистика по завантажених файлів. Крім PE формату, можливо завантажувати текст в різних форматах, архіви. На сьогоднішній день в базі міститься більше мільйона файлів PE формату.

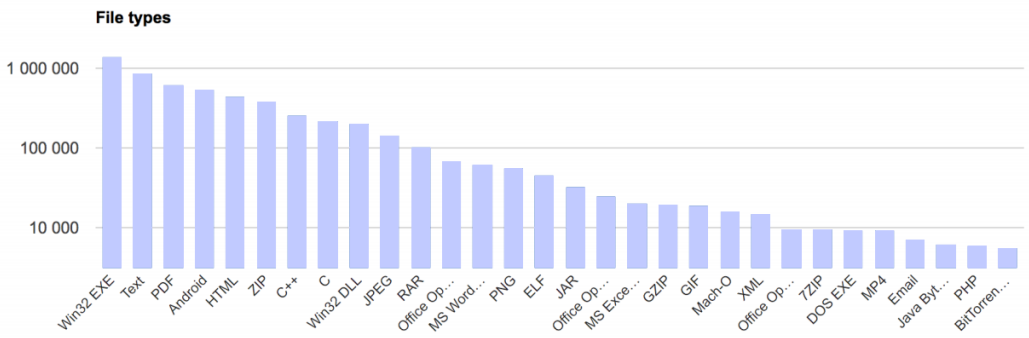


Рисунок 3.1 – Статистика сервісу VirusTotal по завантажених файлах

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

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

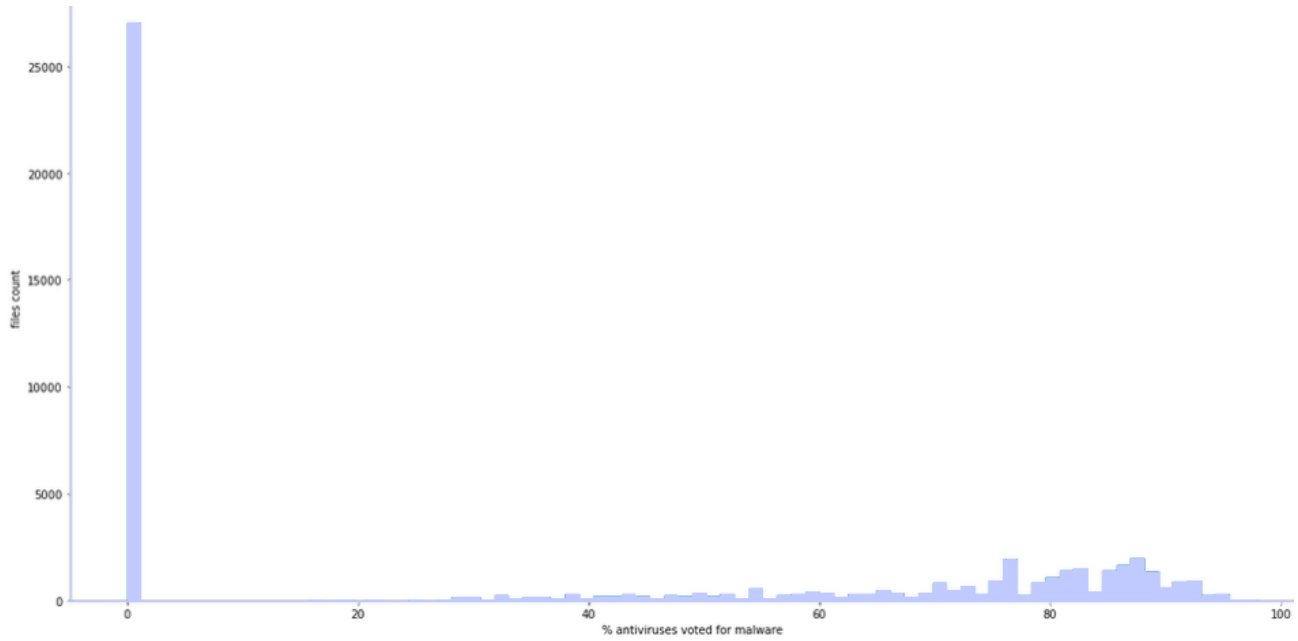


Рисунок 3.2 – Розподіл файлів по голосам від всіх антивірусів. По осі X відношення числа антивірусів тих, хто проголосував за шкідливість файлу до всіх антивірусів, котрі голосували

Антивіруси влаштовані по-різному і видають не завжди один вердикти на файлах, що видно на рисунку 3.2. Тобто, з одного боку, є антивіруси, які помилилися, вважаючи чисті файли вірусами (false positive), і, існують ті, хто

помилився, вважаючи вірус чистим файлом (false negative). За ідеальних виявляючих алгоритмах антивірусів на гістограмі не було б «хвоста» правіше 100%. Виділяється проблема - які файли вважати шкідливими при навчанні алгоритму машинного навчання.

Для найбільш точного передбачення були обрані топ-8 популярних і надійних антивірусів (Paloalto, SentinelOne, McAfee, BitDefender, Kaspersky, CrowdStrike, Avast, Symantec). Індекс шкідливості підраховується тільки для цих антивірусів. Відповідна гістограма представлена на рисунку 3.3. При такому підході близько 5000 файлів як і раніше залишаються сумнівними (сіра зона) - близько половини з надійних антивірусів проголосували за шкідливість. Визначимо файл шкідливим, якщо хоча б один з топ-8 антивірусів визначив його таким. При такому підході ми намагаємося максимально посилити детектування вірусів, зменшуючи кількість хибно негативних спрацьовувань. Зауважимо, 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, отримуємо наступні дані (розмір підібраний так, щоб дані були збалансовані) в таблиці 3.1.

Таблиця 3.1 – Статистика за зібраними файлами

| | |
|---|---|
| Всі файли | 55432 |
| Чисті файли | 27049 |
| Шкідливі файли | 27049 |

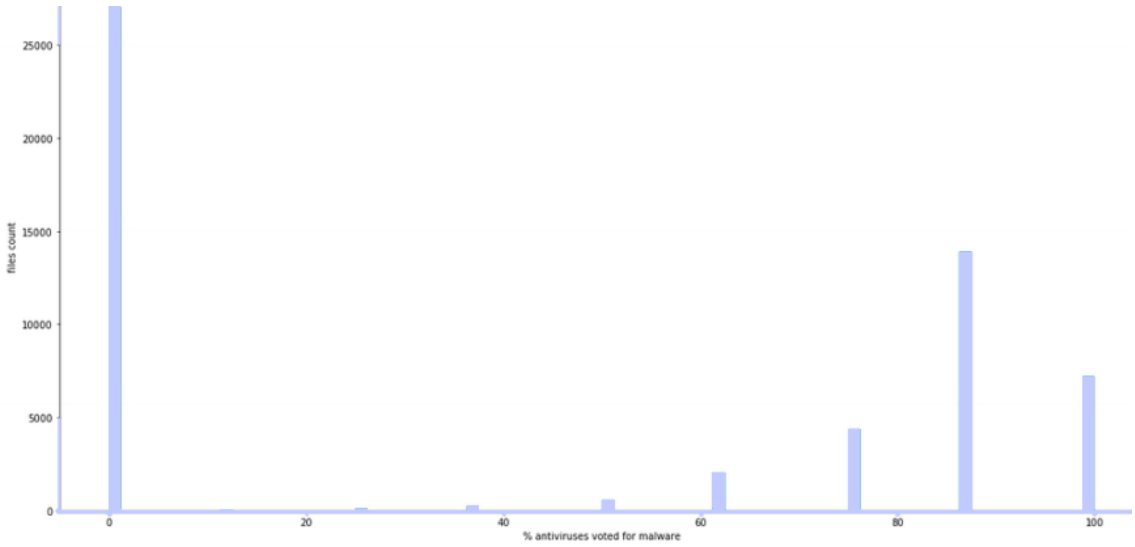


Рисунок 3.3 – Розподіл файлів по голосам від всіх антивірусів. По осі X відношення числа антивірусів тих, хто проголосував за шкідливість файлу до всіх антивірусів, котрі голосували

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

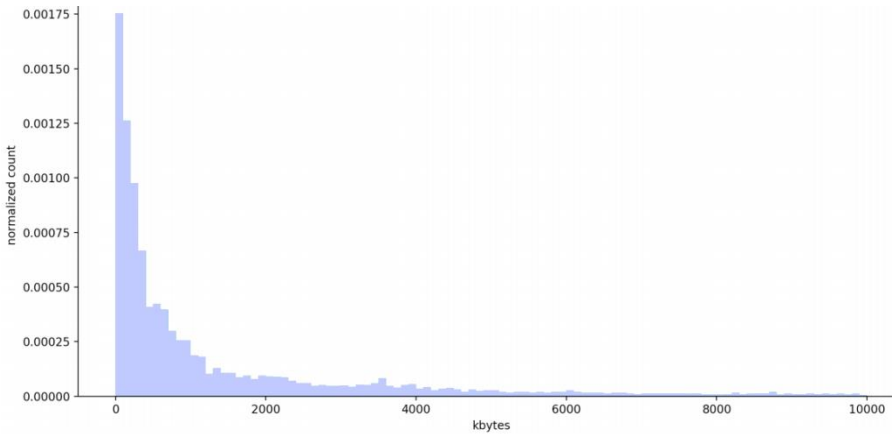


Рисунок 3.4 – Гістограма розподілу розмірів отриманих файлів. По осі X розмір файлу в кілобайтах

## 3.2 План проведення експериментів

Опишемо процес проведення експериментів. Перед проведенням експериментів розіб'ємо дані (файли) на навчальну вибірку і тестову щодо 5 до 1 збалансовано (з однаковим ставленням кількості шкідливих файлів до чистих). Підбір ознак, гіперпараметрів будемо виконувати на навчальній вибірці. Порівняння алгоритмів машинного навчання проводитимемо на результатах з тестової вибірки, що дозволить найбільш точно провести експеримент, уникаючи перенавчання на тестових даних.

Будь-який алгоритм машинного навчання містить гіперпараметри - деякий набір параметрів, який впливає на модель, і, в кінцевому рахунку, якість навчання. Так, наприклад, для 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-валідація 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 залишилася. Усереднити отримані значення. Приклад для k = 10 представлений на рисунку 3.5. Для кожного алгоритму з найкращим набором гіперпараметрів будемо обчислювати accuracy (точність) і false positive rate (частку хибно позитивних спрацьовувань) на тестовій (відкладеної) 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, fl міру або площа під AUC кривої. 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.

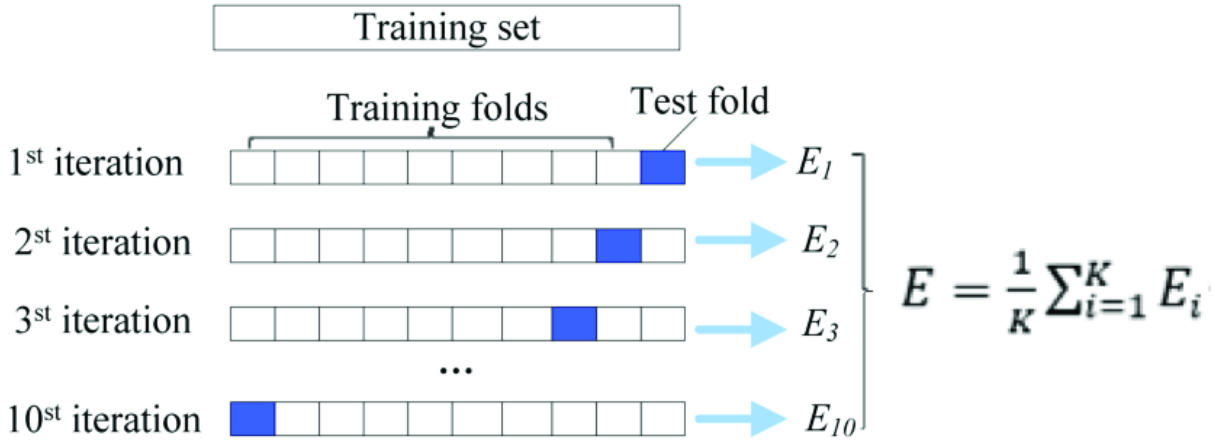


Рисунок 3.5 – Приклад k-fold крос-валідації при k = 10

## 3.3 Відбір ознак PE-файлу

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

Додатково до атрибутів будемо обчислювати ентропію секцій. Ентропія розраховується наступним чином (ентропія Шеннона):

$$H = -\sum_{i=1}^{n} p_i log\, p_i \tag{3.1}$$

Основу логарифма беремо рівним 256 - число можливих значень байту. Таким чином, ентропія H буде приймати значення від 0 до 1.

Висока ентропія секції може сигналізувати про стиснення секції, використанні пакувальників, шифрувальників для приховування шкідливого

коду. Зібрані атрибути з файлів:

- чисельні значення атрибут, взяті з полів PE файлу безпосередньо
- прапори присутності api, секцій та їх характеристики. Значення: {0, 1}
- ентропія секцій

Після роботи жадного алгоритму отримали набір з 135 ознак, на основі яких можливо визначити шкідливі програми.

Збираємо прямі атрибути структури PE файлу з MS-DOS заглушки, coff заголовка, опціонального заголовка. Для фіксованої кількості dll бібліотек і імпортованих api 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 то, на що і вказують. Такі відхилення рішення 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 вилучення. 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 рисунку. 3.6 представлена значимість груп ознак. За графіком видно, що ентропія секцій разом з ознаками опціонального заголовка грає вирішальну роль в визначенні вірусу, зараження файлу. Найменш важлива інформація про DOS заголовку, coff заголовку і характеристиках секцій.

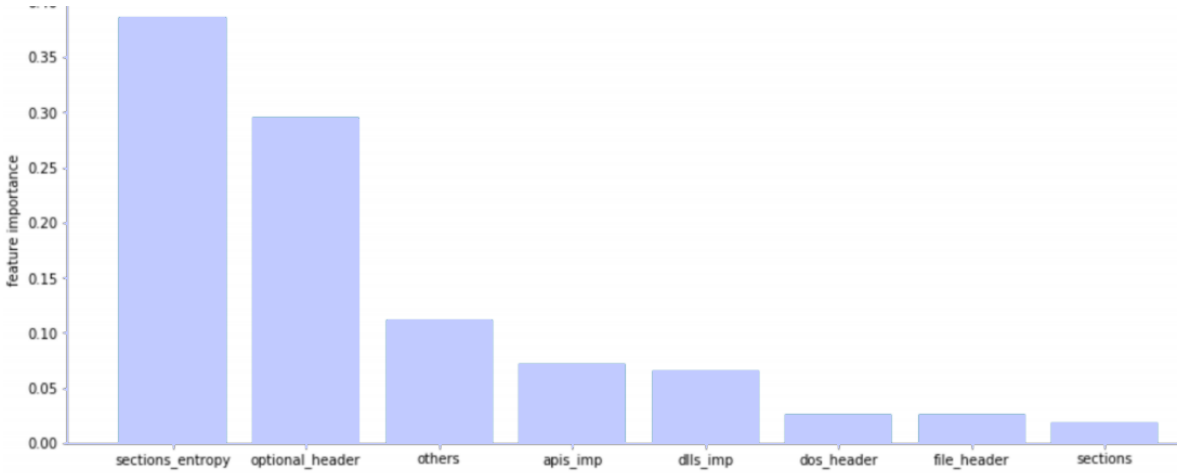


Рисунок 3.6 – Значимість ознак, розрахована за випадковим лісом

## 3.4 Порівняння алгоритмів машинного навчання

Після того, як знайдені ознаки і зафіксовані дані потрібно знайти найкращу модель, кращий алгоритм класифікації. Будемо розглядати 4 обраних моделі: випадковий ліс (Random Forest), XGBoost, LightGBM, CatBoost. Всі експерименти проводимо на мові Python в середовищі ipython notebook з відповідними інструментами і бібліотеками. Для кожної моделі пошук гіперпараметрів здійснюємо по 3-fold кроссвалідації. Після підбору гіперпараметрів навчаємо модель на всій наданої тренувальної вибірці і обчислюємо accuracy і false positive rate на тестових даних. Утворені результати наведені в таблиці 3.2.

Таблиця 3.2 – Порівняння алгоритмів

| Алгоритм | Accuracy % | False positive rate % |
|---|---|---|
| Random Forest | 95.9 | 1.9 |
| XGBoost | 96.3 | 1.4 |
| LightGBM | 96.5 | 1.4 |
| CatBoost | 96.3 | 1.5 |

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

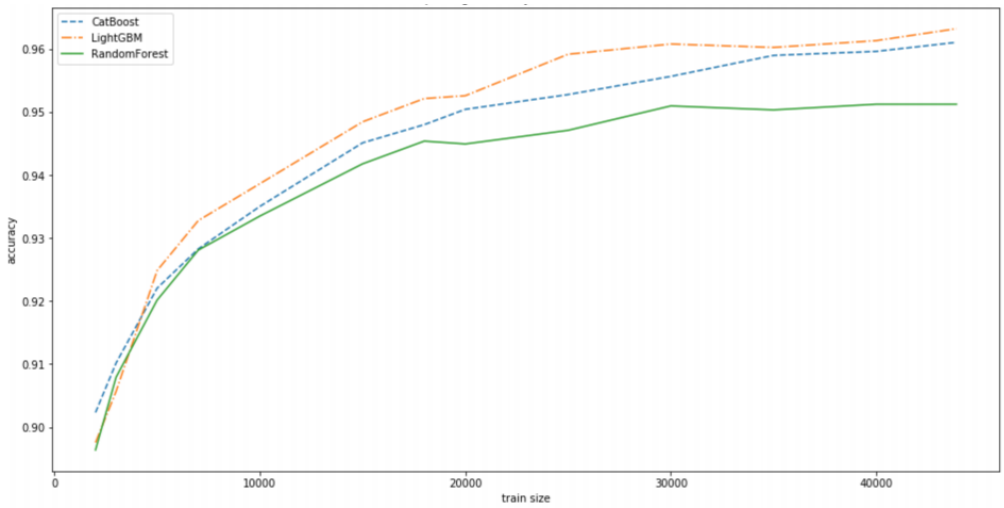


Рисунок 3.7 – Залежність якості алгоритмів від розміру навчальної вибірки

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 гіперпараметрів не опинився кращим.

Взявши за основу алгоритм градієнтного бустінга з LightGBM реалізацією було реалізовано додаток. Крім якості роботи алгоритму інша важлива характеристика - швидкість його роботи. Була виміряна швидкість роботи движка для перевірки задоволення вимог в реальних умовах. Результати представлені в таблиці 3.3. Швидкість обчислювалася на 50 тисячах файлах, розташованих на HDD диску. При запуску на SSD диску швидкість роботи класифікатора не змінилася, а PE парсер став працювати в середньому за 5 мс на одному файлі.

На рисунку 3.8 представлено розподіл часу обробки додатком 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.

Таблиця 3.3 – Середній час роботи складових частин додатка

| | Час роботи, мс |
|---|---|
| PE-парсер | 16 |
| Класифікатор | 0.5 |
| Загальний час | 16.5 |

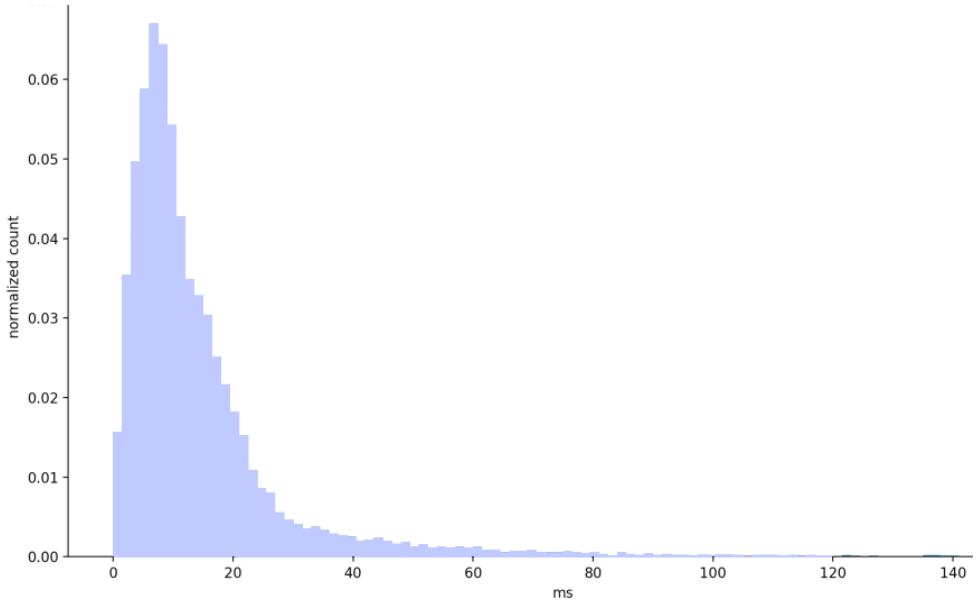


Рисунок 3.8 – Розподіл часу роботи програми для файлів різного розміру

На рисунку 3.9 представлена залежність швидкості роботи движка 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. Логічне припущення, 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

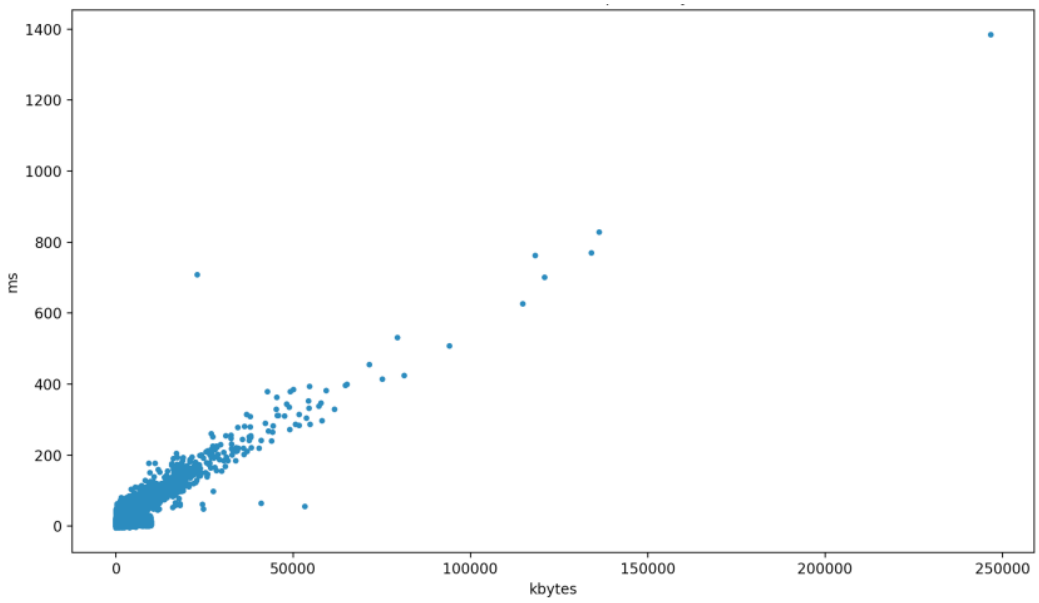


Рисунок 3.9 – Залежність швидкості роботи додатка від розміру файлу

Утворені результати оптимістичні і показують можливість застосування методу на практиці. Також цікаво подивитися за такою характеристикою, як false negative rate. Раніше вважалося, що файл є вірусом, якщо за це проголосував хоча б один з надійних антивірусів. Тепер побудуємо розподіл індексу шкідливості від всіх антивірусів для false negative файлів. Гістограма приведена на рисунку 3.10. Можна порівняти даний розподіл до представленого раніше для всіх файлів (рисунок 3.2).

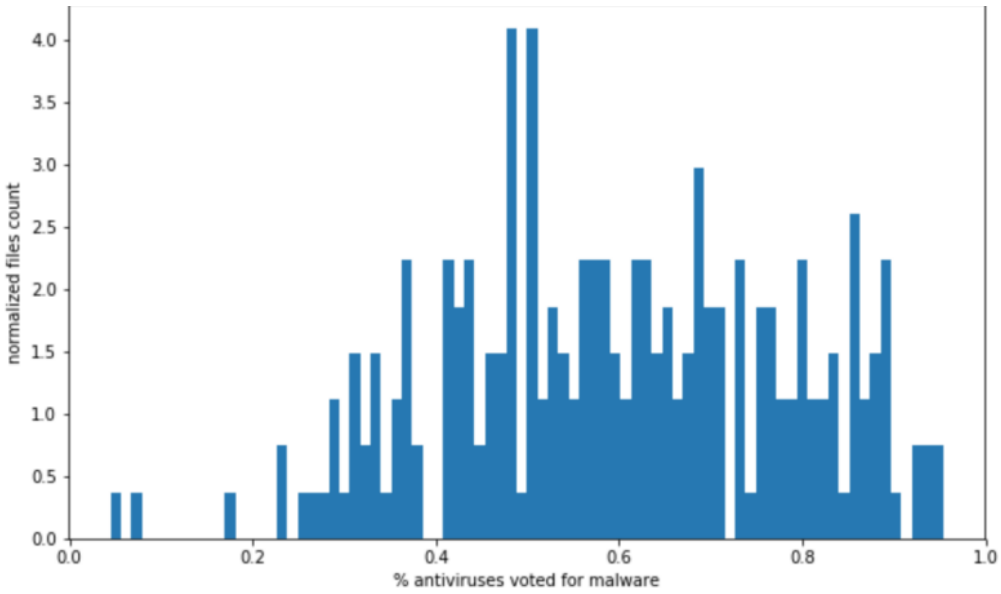


Рисунок 3.10 – Розподіл голосів антивірусів по false negative файлів

На рисунку 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.

# ВИСНОВКИ

В ході роботи було отримано ряд результатів:

Виконано огляд методів статичного аналізу визначення шкідливих програм. Область активно розвивається в даний час, відбувається перехід від евристик до застосування методів машинного навчання. Виділено основні принципи побудови статичного аналізу. Існує велика кількість робіт зі створення евристик для детекції шкідливого коду. У багатьох роботах не висвітлюється практичне застосування алгоритмів машинного навчання.

Отримано дані для аналізу (більше 50000 файлів), побудови статичного аналізу. Представлений метод розбиття файлів на чисті і шкідливі.

Проведений відбір найбільш значущих ознак.

Проведено порівняння сучасних алгоритмів машинного навчання в рамках даного завдання в умовах обмежених ресурсів і часу.

Реалізовано додаток детектування шкідливого коду для кінцевих користувачів.

Проектування і створення додатку виконувалося на основі результатів проведених досліджень з відбору найбільш значущих ознак, вибору підходящого алгоритму. Вимоги до швидкості роботи моделі накладали обмеження на вибір алгоритму, кількість ознак. В ході роботи було отримано уявлення про значущість окремих ознак і груп ознак для детектування шкідливого ПЗ.

Отримані результати можна використовувати для побудови більш складних систем детектування вірусів.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. A. Kalbhor, T. H. Austin, E. Filiol, S. Josse, and M. Stamp, Dueling Hidden Markov Models for Virus Analysis, Journal in Computer Virology, 11(2):103-118, 2015.

2. Driller, T.M.: Metamorphic permutating high-obfuscating reassembler source. http://vx.netlux.org/29a/29a-6/29a-6.602, accessed December 2011.

3. 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.

4. Symantec trend report, https://www.symantec.com/security_response/publications/monthlythreatreport.jsp #Spam, Accessedon April 15, 2016.

5. McAfee Labs. Infographic: Mcafee labs threats report. December 2017.

6. N. Patel et al., “Analyzing hardware-based malware detectors”, In DAC’17, June 2017.

7. N. Ye, X. Li, and S. M. Emran, “Decision trees for signature recognitionand state classification,” in Proc. 2000 IEEE SMC Information Assurance and Security Workshop.

8. D. E. Denning, “An intrusion-detection model,” IEEE Trans. SoftwareEng., vol. SE-13, no. 2, pp. 222–232, Feb. 1987.

9. D. Jaggar, ARM Architecture and Systems, IEEE Micro 17(4) (1997), 9–11.

10. J.O. Kephart and W.C. Arnold, Automatic extraction of Computer Virus Signatures, in: 4th Virus Bulletin InternationalConference, 1994, 178–184.

11. Aaron Hertzmann — Machine Learning and Data Mining. Lecture Notes. — 2012.

12. Владимир Вьюгин — Математические основы теории машинного обучения и прогнозирования. — 2013.

13. Caruana R. — An empirical comparison of supervised learning algorithms. — 2006.

14. Leo Breiman — Random Forests. — 2001.

15. T. Mitchell. Machine Learning. McGraw-Hill, 1997.

16. L. Wehenkel and M. Pavella, ”Decision trees and transient stability of power systems” IFAC), Automatica, Vol. 27, No.1, pp. 115-134. 1991.

17. L. Breiman, J. Friedman, R. A. Olshen, and C. J. Stone, Classification and Regression Trees. Belmont, CA, Wadsworth, 1984.

18. L. Breiman, J. Friedman, R. A. Olshen, and C. J. Stone, Classification and Regression Trees. Belmont, CA, Wadsworth, 1984.

19. П.Г. Круг — Нейронные сети и нейрокомпьютеры. — 2002.