Міністерство освіти і науки України
Київський Національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ПОЯСНЮВАЛЬНА ЗАПИСКА
Дипломної роботи

магістра
(назва освітньо-кваліфікаційного рівня)

галузь знань *12 Інформаційні технології*
(шифр і назва галузі знань)

спеціальність *125 Кібербезпека*
(код і назва спеціальності)

освітній рівень ***магістр***
(назва освітнього рівня)

кваліфікація
(код і назва кваліфікації)

на тему: *Розробка елементів системи Breach and Attack Simulation*

Виконавець: студент ***2*** **курсу, групи** ***КБм-21***

(підпис) ***Пономарьов Сергій Максимович***
(прізвище ім'я по-батькові)

	Прізвище, ініціали	**Оцінка**	**Підпис**
Науковий керівник	*Лукова-Чуйко Н.В.*		
Рецензент			
Нормоконтроль			

Київ
2021

Міністерство освіти і науки України
Київський Національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ЗАТВЕРДЖЕНО:
завідувач кафедри
кібербезпеки та захисту інформації
_____ Лукова-Чуйко Н.В.
«_____»_____20___року

ЗАВДАННЯ
на виконання дипломної роботи

спеціальності *125 Кібербезпека*
(код і назва спеціальності)

студенту *КБм-21* (група) *Пономарьову Сергію Максимовичу* (прізвище ім'я по-батькові)

Тема дипломного роботи *Розробка елементів системи Breach and Attack Simulation*

1. ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Рішення засідання кафедри кібербезпеки та захисту інформації факультету інформаційних технологій протокол №_____ від _____

2. МЕТА ТА ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Об'єкт досліджень *Процес створення елемента Breach and Attack Simulation з використанням технологій машинного навчання*

Предмет досліджень *Елементи технології Breach and Attack Simulation для пошуку вразливостей з використанням методів машинного навчання*

Мета *створення елементу, реалізуючого можливість пошуку вразливостей на основі інформації про інші вразливості системи*

Вихідні дані для проведення роботи

3. ОЧІКУВАНІ НАУКОВІ РЕЗУЛЬТАТИ

Наукова новизна *Вдосконалення існуючих методів пошуку вразливостей системи Breach and Attack Simulation*

Практична цінність *Збільшується якість пошуку вразливостей з використанням системи Breach and Attack Simulation*

4. ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

5. ЕТАПИ ВИКОНАННЯ РОБОТИ

Найменування етапів робіт	Строки виконання робіт (початок-кінець)
1. Уточнення поставленої задачі.	12.10.2020 – 16.10.2020
2. Аналіз літератури.	19.10.2020 – 10.12.2020
3. Збір даних. Обґрунтування вибору рішення.	25.01.2021 – 12.02.2021
4. Дослідження процесу проведення проактивного пошуку загроз.	15.02.2021 – 26.02.2021
5. Пошук шляхів вдосконалення проактивного пошуку загроз.	05.04.2021 – 16.04.2021
6. Аналіз результатів проведеного дослідження.	19.04.2021 – 07.05.2021
7. Оформлення пояснювальної записки. Підготовка до захисту роботи	11.05.2021 – 17.05.2021

6. РЕАЛІЗАЦІЯ РЕЗУЛЬТАТІВ ТА ЕФЕКТИВНІСТЬ

Економічний ефект *Підвищує здатність системи симуляції зламів і атак виявляти вразливості інформаційних систем*

Соціальний ефект *Збільшується вірогідність своєчасного виявлення та усунення вразливостей інформаційної системи, що знижує вірогідність вдалої атаки на систему*

7. ДОДАТКОВІ ВИМОГИ

Завдання видав _____ _____
(підпис) (прізвище,ініціали)

Завдання прийняв до виконання _____ _____
(підпис) (прізвище,ініціали)

Дата видачі завдання: _____

Термін подання дипломної роботи до ЕК _____

УДК [004.4+004.65](004.91)

РЕФЕРАТ

Пояснювальна записка до дипломної роботи «Розробка елементів системи Breach and Attack Simulation»: 101 сторінок, 18 рисунків, 3 додатки та 28 таблиць, 41 літературне джерело.

Об’єкт дослідження – Процес створення елемента Breach and Attack Simulation з використанням технологій машинного навчання.

Мета роботи – створення элементу, реалізуючого можливість пошуку вразливостей на основі інформації про інші вразливості системи

Предмет дослідження – Елементи технології Breach and Attack Simulation для пошуку вразливостей з використанням методів машинного навчання.

В даній дипломній роботі були розглянуті та проаналізовані відомі засоби тестування безпеки, та новий, набуваючий популярності метод симуляції зламів і атак. Визначено стандарти оцінки вразливостей та їх метрики. Проаналізовано технології машинного навчання та методи обробки текстової інформації. Після аналізу обрано алгоритм за яким було створено програмний код для пошуку вразливостей на основі інормації про інші вразливості.

Практична значимість: Збільшено якість пошуку вразливостей з використанням системи Breach and Attack Simulation.

Актуальність теми: Система Breach and Attack Simulation являється новою технологією в сфері оцінки стану захищеності інформаційних систем, але не розкрила повного потенціалу. Однією з можливостей покращення роботи даної системи є використання методів машинного навчання, задля знаходження уразливостей інформаційної системи.

Напрямок подальших досліджень: втілення інтеграції, розробленого у роботі, елемента в систему симуляції зламів і атак.

Ключові слова: Breach and Attack Simulation, CVSS, CWE, NVD, машинне навчання, тести на проникнення, Red Team, системи виявлення вторгнень, методи обробки текстової інформації.

ЗМІСТ

ВСТУП

Необхідність перевірки стану захищеності інформаційної системи – актуальне питання, що постає перед будь-якою організацією, яка піклується про власну безпеку та безпеку своїх користувачів.

Використовуючи сучасні технологічні рішення для надання послуг, потрібно забезпечити відповідний рівень безпеки і, навіть після впровадження політики безпеки та систем захисту, аналіз їх стану та вразливостей займає важливе місце у житті компанії.

Існує декілька засобів тестування безпеки мережі, всі вони користуються попитом і мають широке розповсюдження. Надалі мною буде розглянуто види цих засобів, їх переваги та недоліки, а також – більш новий, набуваючий актуальності засіб тестування.

Тестування безпеки це тип тестування програмного забезпечення, який виявляє уразливості, загрози, ризики в програмному забезпеченні і запобігає атакам від зловмисників. Метою тестів безпеки є виявлення всіх можливих слабких місць у системі, які можуть привести до втрати інформації, доходів, репутації з боку співробітників або сторонніх осіб.

Метою тестування безпеки є виявлення загроз в системі і оцінка її потенційних вразливостей, щоб система не перестала функціонувати. Це також допомагає у виявленні можливих загроз безпеки в системі і допомагає розробникам в усуненні цих проблем.

РОЗДІЛ 1
АНАЛІЗ ЗАСОБІВ ТЕСТУВАННЯ БЕЗПЕКИ

1.1 Тести на проникнення

Тести на проникнення (або пентести) виконуються вручну співробітниками компанії або зовнішніми консультантами, які намагаються оцінити захищеність інфраструктури організації шляхом її безпечного злому. Для цього можуть використовуватися уразливості операційних систем, служб або додатків, неправильні конфігурації або недостатньо обережне поводження користувачів.

Іншими словами, проводиться атака на мережу, додатки, пристрої та співробітників організації з метою перевірити, чи можуть хакери здійснити такий злом. За результатами тестування також стає ясно, наскільки глибоко зміг би проникнути зловмисник і який обсяг даних він міг би вкрасти або використовувати в своїх цілях.

Процес тестування на проникнення поділяється на п’ять етапів:

1. Планування та розвідка

Перший етап передбачає:

- Визначення обсягу та цілей тесту, включаючи системи, на які слід звернутись, та методи тестування, які слід використовувати.
- Збір інформації (наприклад, мережевих і доменних імен, поштового сервера), щоб краще зрозуміти, як працює ціль та її потенційні вразливості.

2. Сканування

Наступним кроком є розуміння того, як ціль реагуватиме на різні спроби вторгнення. Зазвичай це робиться за допомогою таких методів:

- Статичний аналіз - перевірка системи для оцінки поведінки під час роботи.
- Динамічний аналіз – перевірка системи в робочому стані. Це більш практичний спосіб сканування, оскільки він забезпечує перегляд у реальному часі продуктивності системи.

3. Отримання доступу

На цьому етапі для виявлення вразливостей цілі використовуються атаки веб-додатків, такі як міжсайтові сценарії, використання SQL-ін'єкцій та бекдор. Потім тестувальники намагаються використати ці уразливості, як правило, шляхом нарощування привілеїв, викрадення даних, перехоплення трафіку тощо, щоб зрозуміти шкоду, яку вони можуть завдати.

4. Зберігання доступу

Метою цього етапу є з'ясувати, чи можна використовувати вразливість для постійної присутності в системі, що експлуатується - досить довго, щоб тестувальник отримав поглиблений доступ. Ідея полягає в імітації передових постійних загроз, які часто залишаються в системі місяцями, щоб викрасти найбільш конфіденційні дані організації.

5. Аналіз

Потім результати тесту на проникнення складаються у звіт, де детально викладається:

- Конкретні вразливості, які були використані
- Конфіденційні дані, до яких здійснювався доступ
- Протягом якого часу тестер міг залишатися в системі не виявленим

Цю інформацію аналізує персонал служби безпеки, щоб допомогти налаштувати параметри WAF на підприємстві та інші рішення безпеки додатків для виправлення вразливостей та захисту від майбутніх атак

Методи тестування на проникнення:

- Зовнішнє тестування

Зовнішні тести на проникнення націлені на активи компанії, які є видимими в Інтернеті, наприклад, веб-програма, веб-сайт компанії або сервери електронної пошти та доменних імен (DNS). Мета - отримати доступ та здобути цінні дані.

- Внутрішнє тестування

Під час внутрішнього тесту виконавець, який має доступ до програми за його брандмауером, імітує атаку зловмисного інсайдера. Це не обов'язково повинна бути симуляція зловмисного працівника. Типовим початковим сценарієм може бути працівник, чиї облікові дані були вкрадені через фішингову атаку.

- Сліпе тестування

Під час сліпого тесту тестувальнику надається лише назва підприємства, на яке націлена атака. Це надає співробітникам служби охорони час для підготовки, до того, як відбуватиметься фактичний напад.

- Подвійне сліпе тестування

Під час подвійного сліпого тесту співробітники служби безпеки не мають попередніх знань про атаку. Як і в реальному світі, у них не буде часу підкріпити захист до спроби вторгнення в систему.

- Цільове тестування

У цьому випадку і тестувальник, і персонал служби безпеки працюють разом і інформують один одного про свої рухи. Це цінна навчальна вправа, яка надає команді безпеки зворотній зв’язок у реальному часі з точки зору хакера.

Переваги та недоліки такого методу викладені у таблиці 1.1

Таблиця 1.1

Переваги та недоліки тестів на проникнення

Переваги	**Недоліки**
● Виявляються слабкі місця, котрі не виявляє сканування на	● Результати залежать від знання та досвіду тестувальника та не дають

вразливості. ● Виявляються деякі критичні вразливості інфраструктури. ● Пентестери можуть використовувати будь-які технології атаки, включаючи ті, що з'явились в той же день. ● Результуючий звіт допомагає ліквідувати вразливості. ● Являється хорошим випробуванням для засобів забезпечення безпеки пережі	повної картини, оскільки вручну неможливо перевірити всі аспекти системи. ● Обмежене тестове середовище не дозволяє використовувати всі можливості, які мають реальні хакери. ● Тестувальник не здатен перевірити абсолютно всі відомі йому технології атаки. ● Звіту за результатами тестування доводиться довго чекати. ● Результати пентеста відображають стан системи в певний період часу. Висока вартість такого тестування не дозволяє проводити його часто.

1.2 Сканування на наявність вразливостей

Для сканування на наявність вразливостей існують як платні рішення, так і додатки з відкритим вихідним кодом. Вони дозволяють знайти уразливості, які вже відомі розробникам ПО, і слабкі місця, якими кіберзлочинці вже користувалися. Комп'ютери та мережі організації перевіряються на наявність тисяч вразливостей, таких як дефекти в програмному забезпеченні, відсутність виправлень операційної системи, вразливі служби, небезпечні конфігурації і уразливості веб-додатків.

Сканування вразливостей може виконуватися як зовні, так і зсередини мережі або сегмента мережі, що оцінюється. Організації можуть виконувати зовнішнє сканування поза межами свого мережевого периметру, щоб визначити вплив атак серверів та програм, доступних безпосередньо з Інтернету. Тим часом, сканування внутрішньої вразливості має на меті виявити недоліки, які хакери можуть використати для експлуатації різних систем та серверів, якщо вони отримають доступ до локальної мережі.

Легкість доступу до частин внутрішньої мережі залежить від того, як мережа налаштована і сегментована. Через це будь-яка програма управління вразливістю повинна починатися з картографування та інвентаризації систем організації та класифікації їх важливості на основі доступу, який вони надають, і даних, якими вони володіють.

Деякі галузеві стандарти, такі як Стандарт безпеки даних платіжних карток (PCI-DSS), вимагають, щоб організації щоквартально виконували сканування як зовнішньої, так і внутрішньої вразливості, а також кожного разу, коли встановлюються нові системи або компоненти, змінюється топологія мережі або брандмауер, змінюються або модернізуються різні програмні продукти. Зовнішнє сканування повинно виконуватися за допомогою інструментів, схвалених PCI Approved Scanning Vendor (ASV).

З огляду на широке впровадження хмарної інфраструктури, процедури сканування вразливості повинні бути адаптовані, щоб включати також розміщені в хмарні ресурси. Зовнішні сканування особливо важливі в цьому контексті, оскільки неправильно налаштовані та небезпечні розгортання баз даних та інших служб у хмарі становлять небезпеку.

Сканування вразливості не може існувати опосередковано, тому його слід доповнити тестуванням на проникнення. Це різні процеси, які мають спільну мету виявити та оцінити слабкі місця в безпеці. Сканування вразливостей - це автоматизована діяльність, яка спирається на базу даних

відомих вразливостей, таких як CVE / NVD. Це призводить до більш точної оцінки ризику, який становлять різні вразливі місця.

Результати такого сканування застосовуються в процесах автоматизованого аудиту безпеки IT-середовища компанії. Найчастіше сканери вразливостей, перевіряючі мережі і веб- сайти на наявність тисяч загроз безпеки, стають ядром системи інформаційної безпеки, а список знайдених вразливостей - основою для подальших дій по виправленню.

Переваги та недоліки такого методу викладені у таблиці 1.2.

Таблиця 1.2

Переваги та недоліки сканувань на наявність вразливостей

Переваги	**Недоліки**
● Автоматизоване, просте, може виконуватись за розкладом. ● Виявляє відомі вразливості. ● Результати можуть бути отримані всього за декілька годин. ● Не потребує спеціальних знань. ● База відомих вразливостей постійно оновлюється ● Може бути більш економічно вигідним рішенням у порівнянні з пентестами. ● Можливість одночасного виконання декількох сканувань	● Відсутність відомостей про динаміку процесів; ● Видно тільки миттєвий знімок стану безпеки. ● Погано виявляє вразливості, котрі ще не були описані. Організація залишається вразливою в проміжках часу між оновленнями. ● Високий коефіцієнт хибних спрацювань (30%-60%) ● Не враховується можливість протидії зі сторони загроз. ● Може збільшити навантаження на виробничу середу і призводити до збоїв. ● Призначено для допоміжних, а не ключових систем, працюючих у

	реальному часі.

1.3 Тестування red team

Термін Red Team прийшов з військової середовища і визначає «дружню» атакуючу команду. На противагу їй існує команда захисників - Blue Team.

Відмінність Red Team операцій від класичного пентеста в першу чергу в регламенті дій і попередженні сторони, що захищається. Також, при «класичному» пентесте часто використовуються «білі списки», обмеження за часом проведених робіт, рівню взаємодії з системою. При проведенні Red Team операцій немає практично ніяких обмежень, проводиться реальна атака на інфраструктуру: від атак зовнішнього периметра, до спроб фізичного доступу та соціотехнічних методів.

Завдання Blue Team - забезпечувати захист інфраструктури наосліп: команду захисників не попереджають про проведення атаки або про її відмінності від реальних зловмисників – це один з кращих факторів перевірити як захисні системи, так і здатність фахівців виявляти і блокувати атаки, а згодом проводити розслідування інцидентів. Після завершення операції необхідно порівняти відпрацьовані вектори атак з зафіксованими інцидентами для поліпшення системи захисту.

Підхід Red Team найближче співвідноситься з таргетованою атакою - APT (Advanced Persistent Threat). Команда Red Team повинна складатися з досвідчених професіоналів, з багатим досвідом як побудови IT / ІБ інфраструктури, так і досвідом компрометації систем.

Використання конкретного інструментарію в окремому випадку може бути обумовлено специфікою того чи іншого додатку або сервісу і слабко відрізняється від звичайного тестування на проникнення. При проведенні Red Team операцій постає питання командної взаємодії та систематизації отриманих

результатів - це і звіти різних інструментальних засобів аналізу і уразливості виявлені в ручному режимі - все це представляє з себе величезний обсяг інформації, в якому без належного порядку і системного підходу можна загубити щось важливе. Також існує необхідність зведення звітів та їх нормалізація і приведення до єдиного вигляду.

Зазвичай Red Team операції покривають досить об'ємні інфраструктури, які вимагають застосування спеціалізованого інструментарію:

- Сканери та утиліти для проведення інвентаризації периметра, з можливістю поділу робочих зон.
- Системи обробки даних при проведенні тестування на проникнення.
- Використання засобів аналізу і управління уразливостями.
- Системи проведення соціотехнічних кампаній.

Переваги та недоліки такого методу викладені у таблиці 1.3.

Таблиця 1.3

Переваги та недоліки тестування Red Team

Переваги	**Недоліки**
● Імітує тактику, технології і процедури справжніх хакерів. ● Дозволяє підготуватися до реальних атак за схожими сценаріями. ● Випереджуючий підхід. ● Більш економічно вигідне рішення у порівнянні з пентестами. ● Виявляє невідомі раніше проблеми в нестандартних місцях. ● Дозволяє оцінювати ефективність інфраструктури безпеки	● Імітації необхідно проводити регулярно. ● Потребує підготовленого штатного чи позаштатного персоналу. ● Результати тестувань може бут важко порівнювати, оскільки вони можуть виконуватись за різними правилами та умовами. ● Задіяні значні ресурси. ● Через недостатню автоматизацію тестування важно

	повторювати одноманітно. ● Важко оцінювати вплив змін в інформаційному середовищі і відстежувати динаміку ефективності захисту.

1.4 Breach And Attack Simulation як засіб автоматичного тестування

Платформа Breach and Attack Simulation (BAS) розвиває ідею імітації цілеспрямованих атак і оцінює фактичну готовність організації до відбиття кіберзагроз. Такий метод дозволяє виявляти критичні вразливості інфраструктури, проводячи кібератаки з кількох векторів так, як це робили б реальні зловмисники.

Пробні атаки здійснюються по шаблонах реальних хакерських угрупувань, державних кібервійськ і навіть від особи уявних неблагонадійних співробітників.

Моделювання порушення може імітувати атаки шкідливих програм на кінцеві точки, вилучення даних, атаки шкідливого програмного забезпечення та складні атаки APT, які рухаються побічно через мережу, орієнтуючись на найцінніші активи. Поєднуючи командні техніки червоного та синього (практика, відома як “фіолетове об’єднання”) та автоматизуючи їх, платформи пробиття та атаки забезпечують постійне покриття.

Ці моделювання можна виконувати цілодобово, без вихідних, 365, що забезпечує організаціям набагато глибший огляд справжнього стану своєї оборонної готовності. Це критично важливо, оскільки зловмисники можуть перемогти будь-які установки безпеки, отримавши достатньо часу, роблячи безперервне тестування найефективнішим способом зменшення ризику.

На додаток до переваг, пов'язаних з автоматизацією та постійним моніторингом, моделювання зламів та атак також дозволяє командам безпеки змінити спосіб, яким вони забезпечують охорону. Замість того, щоб реагувати, чекати результатів сканування або випуску патчів, дана система дозволяє захисникам прийняти спосіб мислення зловмисника. Вони можуть проявляти ініціативу та активно досліджувати вразливі місця, а не сподіватися, що поточні заходи безпеки виявляться достатніми.

Моделювання зламів та атак приносить ще одну перевагу щодо звичайної перевірки безпеки: ця модель не настільки залежить від людських навичок. Пентестери та червоні чи сині команди складаються з людей з певними наборами навичок та рівнями досвіду - і обидва ці фактори можуть значно відрізнятися від людини до людини або команди до команди. Людська помилка, спричинена недосвідченістю, недоглядом або неправильним судженням, може вплинути на результат ручного тестування. Автоматизоване моделювання порушень усуває цю змінну, крім підвищення ефективності та зниження витрат.

Узагальнити можливості BAS можна в таблиці 1.4.

Таблиця 1.4

Можливості BAS

До зламу	1. Поштові шлюзи - Тестує стійкість до актуальних зараз кіберзагрозам. 2. Веб-шлюзи - Тестує стійкість до атак з заражених веб-сайтів по протоколах HTTP / HTTPS. 3. Брандмауер веб-додатків - Тестує стійкість до атак з заражених веб-сайтів по протоколах HTTP / HTTPS
Після зламу	1. Крадіжка даних - Тестує захист від несанкціонованої передачі конфіденційних даних за межі корпоративної мережі. 2. Поширення по мережі - Тестує захист Windows-домена організації за допомогою складного алгоритму поширення по

	мережі.
Злам	1. Безпека кінцевих точок - Перевіряє, чи здатні інструменти захисту кінцевих точок відбити атаки з різних векторах. 2. Стійкість до фішингу - Перевіряє навички співробітників по протидії фішингу за допомогою комплексних настроюються імітацій.
Особливі вектори захисту	1. Актуальні загрози - Вектор, що включає в себе поштові та веб шлюзи, а також кінцеві точки. Тестує стійкість до актуальних зараз кіберзагроз. 2. Повний цикл APT-атаки - Запускає цілеспрямовані APT-атаки повного циклу Kill Chain.

Особливості BAS:

- Моделювання APT

  Симулятори зламів та атак оцінюють та перевіряють найновіші та вдосконалені тактики, техніки та практики, що циркулюють по всьому світу. Зокрема, передові стійкі загрози становлять значний ризик для організацій через соціальну інженерію, вразливість нульового дня та здатність залишатися непоміченою та невиявленою. Жоден інструмент не гарантує зупинки кожної атаки. Тим не менше, система BAS може створити загрозу для виявлення вразливостей нульового дня та представити потенційні шляхи атак для зловмисних акторів, що рухаються по мережі.

- Автоматизація проти ручного тестування

  Для тестування на проникнення, організації червоних команд або внутрішнього аудиту безпеки, організації та сторонні підрядники безпеки відповідають за ручне проектування та виконання кожного проходу. Незалежно від того, чи було сканування націлене на

критичний актив, чи проводилось оцінка вразливості всієї мережі, тестування мережі вручну є вичерпним з будь-якою частотою.

Рішення BAS мають технологічну майстерність для пом'якшення цієї проблеми за рахунок автоматизації розгортання спеціальних сканувань та атак, що стосуються конкретної мережі, що інформується щодо джерел загроз та екосистеми галузі.

- Статистика в реальному часі

  Зловмисникам все одно, скільки часу організація виділяє на підтримку периметру охорони, і вони із задоволенням скористаються навіть невеликим віконцем можливостей. З огляду на це, малі та середні підприємства та організації підприємств знають, що цілодобовий моніторинг необхідний. Залучивши BAS, фірми можуть економити внутрішні ресурси. Адміністратори мережі можуть бути впевнені, що знайдене порушення повинне викликати своєчасне повідомлення. Під час атаки, негайне повідомлення та своєчасні дії можуть зупинити зловмисника на шляху до причинення будь-яких додаткових збитків.

- Гнучка для розвитку інфраструктури

  Оскільки організації все частіше переходять до хмарник сервісів або розглядають альтернативи локальній інфраструктурі, їм потрібне рішення, що охоплює все. Як новіша технологія, моделювання зломів та атаки може застосовуватися до більшості інфраструктур або сегментів мережі, включаючи організації, що рухаються до гібридної хмари або SD-WAN.

РОЗДІЛ 2
ОГЛЯД СТАНДАРТІВ ОЦІНКИ ВРАЗЛИВОСТЕЙ

У галузі комп'ютерної безпеки вразливість - це слабкість, яку може використати зловмисник. Слабкістю може бути будь-який дефект комп'ютерної системи, який може призвести до порушення інформаційної безпеки. Вразливість, яка невідома сторонам, відповідальним за їх виправлення, називається вразливістю нульового дня.

Звіти про дефекти часто надсилаються до системи, де відповідальні сторони можуть вивчати, відтворювати, визначати пріоритети та контролювати стан виправлення дефектів. Ці звіти короткі, кілька речень описують дефекти програмного забезпечення. Деякі звіти про дефекти можуть надавати інформацію про потенційні вразливі місця, які слід враховувати при визначенні пріоритетів або загальній видимості. Деякі системи повідомлення про дефекти є загальнодоступними.

За даними Arnold et al. [2009] та Wijayasekara et al. [2012] висновки вимагає більше часу для включення та розповсюдження програмних виправлень, не пов’язаних із безпекою, ніж тих, які були виявлені як уразливі місця, коли про них було повідомлено. Райт [2013] та ін. дійшов висновку, що після вивчення бази даних помилок для програмного забезпечення бази даних MySQL було виявлено значну кількість раніше невідомих уразливостей.

Національна база даних про вразливості (NVD) містить інформацію про описи вразливостей, контрольні списки безпеки, недоліки програмного забезпечення, пов'язані з безпекою, помилкові конфігурації, назви продуктів та показники впливу. База даних NVD підтримується урядом США, і дані є у вільному доступі на їхніх каналах даних. [] Канали даних оновлюються щонайменше щодня. Описи вразливості - це порівняно короткі пропозиції про вразливості. Ці речення можна використовувати для виявлення потенційних

вразливостей у будь-якому іншому тексті, включаючи повідомлення про дефекти в системах відстеження дефектів. Фахівці з безпеки оцінюють ступінь вразливості та основні причини, і ця інформація також доступна серед даних NVD.

Ступінь вразливості виражається за допомогою загальної системи оцінки вразливості (CVSS), яка базується на декількох класифікаціях. Основні причини вразливості виражаються за допомогою загального переліку слабкості (CWE), який є ієрархічним деревом із сотень типів слабких місць. Кожна вразливість ідентифікується за допомогою ідентифікаційного номера загальних вразливостей та ризиків (CVE). Органи нумерації CVE - це організації у всьому світі, яким дозволено призначати ідентифікаційні номери CVE.

Common Platform Enumeration (CPE) - це структурована схема імен для систем, програмного забезпечення та пакетів. Інформація CPE включається до іншої інформації про вразливість через канали даних NVD. У цій дипломній роботі дані NVD використовуються для виявлення вразливостей із тексту. Для дослідження було вибрано кілька баз даних про дефекти, щоб знайти звіти, що стосуються безпеки, з усіх звітів. Дані NVD також використовуються для вивчення та прогнозування вибраних алгоритмів машинного навчання для класифікації класифікацій CVSS та CWE.

Бібліотека Scikit-learn була обрана для здійснення класифікації та вимірювання продуктивності алгоритму машинного навчання. Scikit-learn - це безкоштовна бібліотека машинного навчання для мови програмування Python.

В даній роботі було проведено дослідження щодо інструменту, який виявляє та класифікує вразливі місця, використовуючи лише коротку письмову текстову інформацію. Найбільш складним завданням для досягнення цього є перетворення людського письмового тексту, придатного для алгоритмів машинного навчання. Ці завдання попередньої обробки перетворюють текст на функції, які можуть обробляти алгоритми машинного навчання.

Для вирішення цього було порівняно різноманітні векторизатори Scikit-learn. Інша проблема полягає у виявленні вразливостей із тексту, знаючи лише описи вразливостей. Це називається однокласовою або одинарною проблемою класифікації. Традиційна класифікація статистичного або машинного навчання стосується двійкової класифікації, яка вимагає як позитивних, так і негативних зразків у даних навчання. Стверджується, що однокласна класифікація успішно застосовується в численних сферах академічних досліджень та промислових застосувань [Wang et al., 2017]. Результати показують, що класифікатор OneClassSVM від Scikit-learn здатний виявляти вразливості так само добре, як і класифікатор на основі ключових слів, який був реалізований під час цієї роботи. Проблема все ще залишається відносно великою кількістю помилкових спрацьовувань.

Класифікація CVSS в основному є багатокласовою проблемою класифікації, але деякі метрики можуть мати лише два класи. Результати показують, що класифікація CVSS є успішною з використанням лінійних алгоритмів класифікації з оцінкою F1 близько 0,816. Проблема полягає в тому, що існує кілька показників для класифікації, щоб отримати оцінку CVSS. Автоматична класифікація CWE також життєздатна з використанням тих самих методів, що і класифікація CVSS, але проблема полягає в тому, що існує десятки окремих класів і в багатьох класах є низька кількість зразків. У цьому дослідженні було використано підхід до пошуку кореневих категорій цих класів CWE.

2.1 Common Vulnerability Scoring System

Загальна система оцінки вразливостей (CVSS) - це відкритий галузевий стандарт для оцінки серйозності вразливостей безпеки. Система підтримується організацією Forum of Incident Response and Security Teams. Система зіставляє оцінки серйозності вразливостей, що дозволяє фахівцям у галузі безпеки

визначати пріоритети реагування та ресурси відповідно до загрози. Бали розраховуються використанням формули для кількох показників, які приблизно визначають простоту використання і вплив експлойта. Діапазон балів - від 0 до 10, де 10 - найбільш серйозна вразливість. CVSS складається з трьох груп показників: базових, тимчасових і контекстно залежних. Кожна група формується набором показників, який виражають у вигляді вектора. Кожна метрика скорочена і розділена двокрапкою, як показано на рисунку 1 як приклад векторів CVSS. Базова оцінка являє собою вроджені характеристики вразливості, тимчасова оцінка - метрики, які змінюються з часом через події, зовнішні, по відношенню до уразливості. Оцінка стану навколишнього середовища змінює вплив в залежності від середовища, в якій було виявлено уразливість. Бази даних вразливостей зазвичай надають базові оцінки. Варто зауважити, що вони не є тимчасовими або контекстно залежними.

Рисунок 2.1 - Приклади векторів CVSS (ліворуч: CVSS3, праворуч: CVSS2)

2.2 Основні метрики CVSS2

Специфікація CVSS версії 2 була опублікована в червні 2007 року. Основні показники складаються з шести окремих показників, де кожен має заздалегідь визначені класифікації, які використовуються у формулі для обчислення фактичної оцінки тяжкості. Показники та класифікації CVSS2 наведені в таблиці 1. Метрики вектора доступу, складності доступу та автентифікації оцінюють як здійснюється доступ до вразливості та чи потрібні додаткові умови для її використання.

Три показники впливу вимірюють вплив вразливості на IT-актив. Впливи незалежні між собою і описують втрату конфіденційності, цілісності та доступності. (CVSS2, 2007).

Таблиця 2.1

Показники та класифікації CVSS2

Метрика	**Опис**	**Класифікація**
Вектор атаки (AV)	«Відображає, як вразливість використовується».	Network (N) Adjacent (A), Local (L)
Складність атаки (AC)	«Вимірює складність атаки»	High (H) Medium (M), Low (L)
Аутентифікація (A)	«Вимірює кількість випадків, коли зловмисник повинен пройти автентифікацію, щоб використати вразливість.»	None (N) Single (S), Multiple (M)
Вплив на конфіденційність(C)	«Вимірює вплив на конфіденційність.»	Complete (C), Partial (P), None (N)
Вплив на цілісність(I)	«Вимірює вплив на цілісність.»	Complete (C), Partial (P), None (N)
Вплив на	«Вимірює вплив на	Complete (C),

доступність(A)	доступність.»	Partial (P), None (N)

Оцінка CVSS2 обчислюється на основі шести метрик з класифікаціями, як показано в таблиці 1. Базова оцінка CVSS2 складається з метрик експлуатаційності та впливу, обчислюється наступним чином:

$$\text{Impact}_{\text{conf}} = \text{case ConfidentialityImpact of N: 0.0, P: 0.275, C: 0.660}$$

$$\text{Impact}_{\text{Integ}} = \text{case IntegrityImpact of N: 0.0, P: 0.275, C: 0.660}$$

$$\text{Impact}_{\text{Avail}} = \text{case AvailabilityImpact of N: 0.0, P: 0.275, C: 0.660}$$

$$\text{Impact} = 10.41 \times (1 - (1 - \text{Impact}_{\text{conf}}) \times (1 - \text{Impact}_{\text{Integ}}) \times (1 - \text{Impact}_{\text{Avail}}))$$

$$\text{f(impact)} = 0 \text{ if Impact} = 0, 1.176 \text{ otherwise}$$

$$\text{AV} = \text{case AccessVector of L: 0.395, A: 0.646, N: 1.0}$$

$$\text{AC} = \text{case AccessComplexity of H: 0.35, M: 0.61, L: 0.71}$$

$$\text{Au} = \text{case Authentication of M: 0.45, S 0.56, N: 0.704}$$

$$\text{Expl} = 20 \times \text{AV} \times \text{AC} \times \text{Au}$$

$$\text{BaseScore} = \text{round\_to\_1\_decimal}(((0.6 \times \text{Impact}) + (0.4 \times \text{Expl}) - 1.5) \times \text{f(Impact)})$$

2.1.2 Основні метрики CVSS3

Специфікація CVSS версії 3 була опублікована в червні 2015 року. Основна відмінність від попередньої версії полягає в тому, що існує вісім окремих показників, які класифікуються для обчислення фактичного балу. Базові показники та класифікації CVSS версії 3 наведені в таблиці 2.2 (CVSS3, 2015).

Таблиця 2.2

Показники та класифікації CVSS3

Метрика	Опис	Класифікація	Показники
Вектор атаки (AV)	Відображає контекст, за допомогою якого можливе використання вразливості	Network (N), Adjacent (A), Local (L), Physical (P)	0,85, 0,62, 0,55, 0,2
Складність атаки(AC)	Вимірює складність атаки	High (H), Low (L)	0,44 0,77
Необхідні привілеї(PR)	Описує рівень привілеїв, якими повинен володіти зловмисник перед успішним використанням вразливості	High (H), Low (L), None (N)	0,27 / 0,5, 0,62 / 0,68, 0,85
Взаємодія з користувачем(UI)	Включає вимогу до користувача, брати участь в компрометації вразливого компонента	Required (R), None (N)	0,62 0,85
Сфера дії(S)	Чи дозволяє експлуатація вразливості порушити конфіденційність,	Changed (C), Unchanged (U)	Змінює привілеї, необхідні у разі зміни сфери

	цілісність і доступність будь-якого іншого компонента системи.		
Вплив на конфіденційність(C)	Вимірює вплив на конфіденційність	High (H), Low (L), None (N)	0,56, 0,22, 0
Вплив на цілісність(I)	Вимірює вплив на цілісність	High (H), Low (L), None (N)	0,56, 0,22, 0
Вплив на доступність(A)	Вимірює вплив на доступність	High (H), Low (L), None (N)	0,56, 0,22, 0

Чисельні значення в таблиці 2.2 використовуються у рівняннях для обчислення фактичного балу. Базова оцінка є функцією рівнянь підрахування впливу та експлуатації.

Показник оцінки модифікує числові значення класифікацій, а також рівняння щодо розрахунків впливу та BaseScore. Базова оцінка CVSS3 обчислюється так:

$$\text{Impact}_{\text{Base}} = 1 - [(1 - \text{Impact}_{\text{Conf}}) \times (1 - \text{Impact}_{\text{Integ}}) \times (1 - \text{Impact}_{\text{Avail}})]$$

$$\text{Exploitability} = 8.22 \times \text{AV} \times \text{AC} \times \text{PR} \times \text{UI}$$

$$\text{If(Scope} = \text{Unchanged):}$$

$$\text{Impact} = 6.42 \times \text{Impact}_{\text{Base}}$$

$$\text{If(Scope} = \text{Changed):}$$

$$\text{Impact} = 7.52 \times [\text{Impact}_{\text{Base}} - 0.029] - 3.25 \times [\text{Impact}_{\text{Base}} - 0.02]^{15}$$

$$\text{BaseScore} = \text{If}(\text{Impact} \leq 0), 0 \text{ else:}$$

$$\text{If}(\text{Scope} = \text{Unchanged}):$$

$$\text{BaseScore} = \text{Roundup}(\text{Minimum}[(\text{Impact} + \text{Exploitability}), 10])$$

$$\text{If}(\text{Scope} = \text{Changed}):$$

$$\text{BaseScore} = \text{Roundup}(\text{Minimum}[1.08 \times (\text{Impact} + \text{Exploitability}), 10])$$

В даній дипломній роботі, для обчислення фактичних оцінок CVSS2 та CVSS3 був використаний метод Calcu_vector cvsslib.

2.2 CWE

Common Weakness Enumeration - це ієрархічний перелік типів слабких сторін програмного забезпечення. CWE підтримується некомерційною організацією MITRE. Остання версія 3.2 була опублікована в січні 2019 року. Нова версія публікується щорічно. Ієрархічні списки поділяються на основі концептуальних поглядів, таких як: дослідження, розробоки та архітектурні погляди відповідно.

Погляд на концепцію дослідження описує слабкі та залежні місця між собою з метою виявлення теоретичних прогалин в рамках CWE. Погляд на концепцію розробки організовує слабкі місця, пов'язані з розробкою програмного забезпечення. Погляд архітектурної концепції організовує слабкі місця відповідно до загальних тактик архітектурної безпеки. Його мета - виявити потенційні помилки, які можуть бути допущені в процесі розробки програмного забезпечення. У цій дипломній роботі концепція дослідження була обрана як основа для вирішення питань CWE. Приклад ієрархії досліджень наведено на малюнку 2.2.

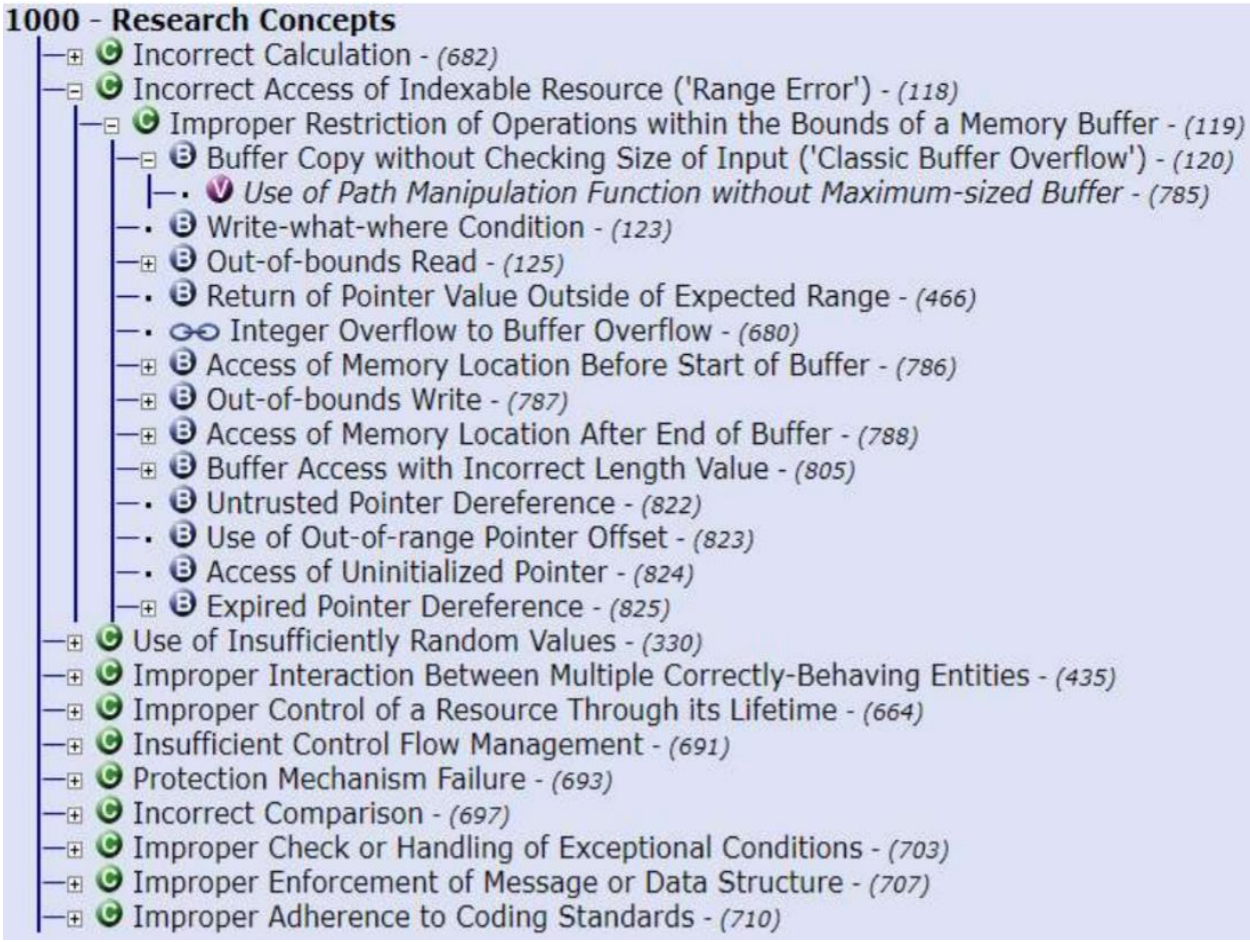

Рисунок 2.2. Ієрархія CWE за даними концепції дослідження

Усі класи кореневого рівня показані на рисунку 2, а саме: деревоподібні зв’язки між слабкими місцями, які існують на різних рівнях абстракції. На найвищому рівні існують класи для групування слабких сторін. Класи - це слабкі сторони, які, на відміну від основних, описані на більш абстрактному рівні.

Варіант слабкості описується на дуже низькому рівні деталізації, як правило, обмежується певною мовою або технологією. Також категорія типу CWE існує, але не в поданій концепції дослідження. Дана – розглядає лише 806 записів CWE із загальної кількості 1131 записів.

2.3 Common Platform Enumeration

CPE - це структурована схема імен для систем, програмного забезпечення та пакетів. CPE включає формальний формат імені, метод перевірки імен та формат опису для прив'язки тексту до імені. Словник CPE розміщується та підтримується організацією NIST, яка є частиною Security Content Automation Protocol (SCAP).[] Остання версія 2.3 була опублікована в 2013 році. CPE вважається галузевим стандартом, спочатку визначеним організацією MITRE. Прив'язка рядків у форматованому CPE зображена на рисунку 3. Усі одинадцять значень атрибутів повинні відображатися у відформатованому прив'язку рядків.

```
cpe:2.3:<part>:<vendor>:<product>:<version>:<update>:<edition>:<language>:<sw
_edition>:<target_sw>:<target_hw>:<other>
```

Рисунок 3. Прив'язка рядків у форматованому CPE

Формат визначає, в якій платформі були виявлені вразливості. Більш детальне пояснення щодо формату:

- part: Тип системи, один із наступних:
  - a = Застосунок
  - h = Апаратне забезпечення
  - o = операційна система
- Vendor: Назва організації, яка розробила продукт.
- Product: назва товару, вказана постачальником.
- Version: ідентифікатор версії продукту.
- Update: оновіть назву версії, визначеної постачальником, наприклад, “R2” для Windows 2012.

- Edition: Видання програмного забезпечення, вказане постачальником, наприклад, “сервер” або “x86”.
- Language: мова програмного забезпечення, наприклад, англійська.
- sw_edition: Видання програмного забезпечення, визначене постачальником, для адаптації певного ринку або класу кінцевих користувачів.
- target_sw: Програмне обчислювальне середовище, в якому працює продукт.
- target_hw: Архітектура набору інструкцій, якими ідентифікується продукт, наприклад, «x86».
- Other: будь-яка інша описова інформація, яка не вміщується в жодному іншому атрибуті.

Повний словник CPE знаходиться у вільному доступі на веб-сайті NVD[], який оновлюється щодня. CPE, які зіставлені з уразливостями, перераховані серед каналів даних NVD. У цій дисертації було використано прив’язку рядків у форматі CPE у строчках даних NVD.

РОЗДІЛ 3
ТЕХНОЛОГІЇ МАШИННОГО НАВЧАННЯ

Класифікація тексту або категоризація тексту - це завдання присвоєння одного або декількох заздалегідь визначених класів неструктурованим текстовим документам відповідно до їх змісту. Класифікація тексту використовується для різних цілей у багатьох різних сферах наприклад: новини можуть бути упорядковані за темами, наукові праці зазвичай класифікують за технічними сферами, а звіти пацієнтів у галузі охорони здоров'я індексуються у декількох категоріях. Фільтр спаму може розподіляти повідомлення електронної пошти як спам або не спам. Класифікація тексту може бути ручною, на основі простих правил чи слів, або ж вона використовує методи машинного навчання.

Структуровані дані стосуються інформації з високим ступенем організованості, яка легше обчислюється та обробляється комп'ютером. Неструктуровані дані - це інформація, яка не має заздалегідь визначеної моделі даних, тобто організація даних не була визначена напередодні. Напівструктуровані дані - це форма структурованих даних, яка не відповідає формальній структурі моделей даних або форм, але містить теги або інші маркери для відокремлення семантичних елементів як забезпечення ієрархії записів та полів даних.

Методи машинного навчання при автоматичній класифікації тексту поділяють на: схожість із полями розпізнавання шаблонів, статистики та аналізу даних. Розпізнавання зразків - це галузь машинного навчання, яка фокусується на розпізнаванні зразків та закономірностей даної інформації.

Статистика - це розділ математики, що займається збором, аналізом, інтерпретацією, поданням та організацією даних. Видобуток даних - це

обчислювальний процес виявлення закономірностей у великих наборах даних, що ставить собі за мету виявлення нової інформації.[]

Машинне навчання можна умовно розділити на дві широкі категорії: навчання під наглядом або без нагляду, залежно від навчального сигналу. Навчання під наглядом - це вивчення завдань із позначених навчальних даних, а без нагляду - це завдання описати приховану структуру в немаркованих навчальних даних. Іншим напрямком видобутку інформації є виявлення аномалій. Ідея полягає в тому, що аномалії, порівняно зі звичайними вихідними даними, можна якось виявити. Виявлення аномалій може використовувати контрольовані або некеровані методи машинного навчання. Посібник користувача Scikit-learn розподіляє виявлення аномалій на виявлення відхилень та новинок. При виявленні чужорідних даних навчальні дані містять відхилення, далекі від інших, і оцінювачі ігнорують відхилені спостереження. При виявленні новинок дані навчання не забруднюються викидами [Scikit, 2019].

У термінології машинного навчання класифікація зазвичай розглядається як примірник навчання під контролем. Відповідна процедура без нагляду відома як кластеризація. У класифікації категорії відомі заздалегідь і надаються для кожного навчального документа. При кластеризації шукаються групи зразків, які природно існують разом. [Witten & Frank, 2005]. Бінарна або біноміальна класифікація - це завдання класифікації елементів даної множини на дві групи. Багатокласова або багаточленна класифікація - це проблема класифікації екземплярів на три або більше класів. У класифікації з декількома мітками передбачається наявність кількох міток для кожного зразка.

В однокласовій класифікації або унарній класифікації об’єкти конкретного класу намагаються ідентифікувати з усіх об’єктів шляхом вивчення навчального набору, що містить лише об’єкти цього класу. Ефективність унарного класифікатора залишається відносно стабільною, коли дисбаланс класу набору даних збільшується, продуктивність двійкового

класифікатора відповідно зменшується [Bellinger, 2012]. Збалансований набір даних - це той, який містить однакову або майже однакову кількість вибірок кожного класу.

При аналізі даних, набір даних - це матриця, яка складається з рядків зразків та стовпців об'єктів. У класифікації тексту кількість ознак зазвичай становить сотні або тисячі. Цей тип набору називається високорозмірним набором даних [Kantardzic, 2011]. Велика кількість функцій часто призводить до розрідженості, що означає, що для більшості випадків значення має нуль.

Методи класифікації тексту повинні вміти обробляти розрідженість даних. Розрідженість може бути врахована на етапі вилучення ознак, застосовуючи техніку зменшення розмірів. Загальна модель класифікатора машинного навчання наведена на рисунку 3.1. Вона ілюструє, як неструктурований текст перетворюється на ознаки, спочатку на етапі навчання, а потім - прогнозування.

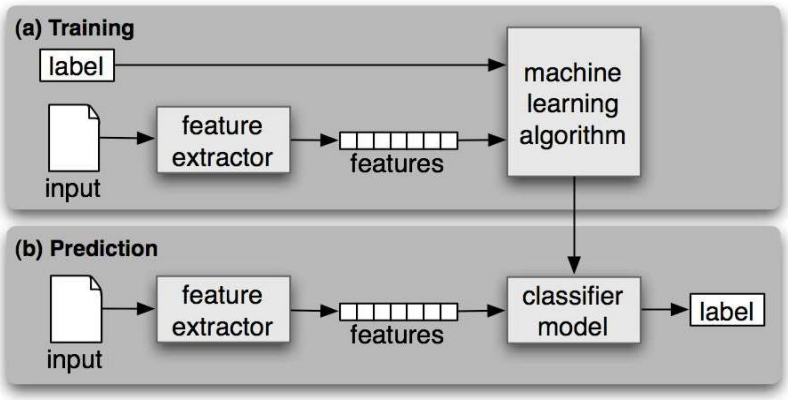

Рисунок 3.1. Модель класифікатора машинного навчання

Під час навчання свої особливості надає алгоритм машинного навчання з позначками правильних результатів класифікації. Алгоритм внутрішньо будує модель, яка здатна розрізняти мітки, що зазначені в навчальних даних.

При прогнозуванні, на основі навченої моделі класифікатор може прогнозувати мітки в нових, небачених даних, але здатен передбачати лише такі

мітки, які видно в навчальних даних. Модель на риунку 4 ілюструє парадигму навчання під контролем. У випадку однокласової класифікації, дані, надані на етапі навчання, містять зразки лише з одного класу.

Однокласовий класифікатор може передбачити, наскільки точка даних порівнюється з даними навчання. У цьому сенсі підхід можна визначити як напівконтрольне виявлення новизни [Scikit, 2019]. У цій роботі були використані лише традиційні методи машинного навчання, але нові методи, засновані на нейронних мережах, існують і для вирішення проблем класифікації природних мов.

3.1 Алгоритми машинного навчання

Наразі існує багато алгоритмів машинного навчання, залежно від типу проблеми, яку алгоритм повинен вирішити. Kantardzic (2011) розподіляє типи проблем на шість основних завдань з видобутку даних: класифікація, регресія, кластеризація, узагальнення, моделювання залежності та виявлення змін та відхилень.

Як було зазначено у передуючих розділах, класифікація намагається привести предмети до попередньо визначених класів, оскільки кластеризація прагне визначити саме кінцевий набір категорій. Регресія намагається зіставити елементи даних із змінними прогнозу реальної вартості. Узагальнення - це завдання, яке включає методи пошуку компактного опису набору даних.

Моделювання залежності призначене для пошуку локальної моделі, яка описує значні залежності між змінними даних. Виявлення змін та відхилень використовується для виявлення найбільш значущих змін у наборі даних. Посібник користувача [Scikit-learn] містить поради для вибору бажаного алгоритму для проблеми машинного навчання, як показано на рисунку 5.

Шпаргалка допомогає обрати підхід до вирішення проблеми машинного навчання. Підходи являються, скоріше, вказівками, щоб вести роботу в

правильному напрямку. У даній роботі були вивчені підходи Naïve Bayes, Stochastic Gradient Descent, Linear Support Vector Machine, and K-nearest Neighbors. Бібліотека Scikit-learn також пропонує підходи до вирішення проблем які стосуються виявлення аномалій. Були вивчені наступні підходи: One-class Support Vector Machine, IsolationForest та LocalOutlierFactor.

У цій главі алгоритми роздглядаються та коротко пояснюються з точки зору використання інтерфейсів програмування, а не реалізації фактичних алгоритмів.

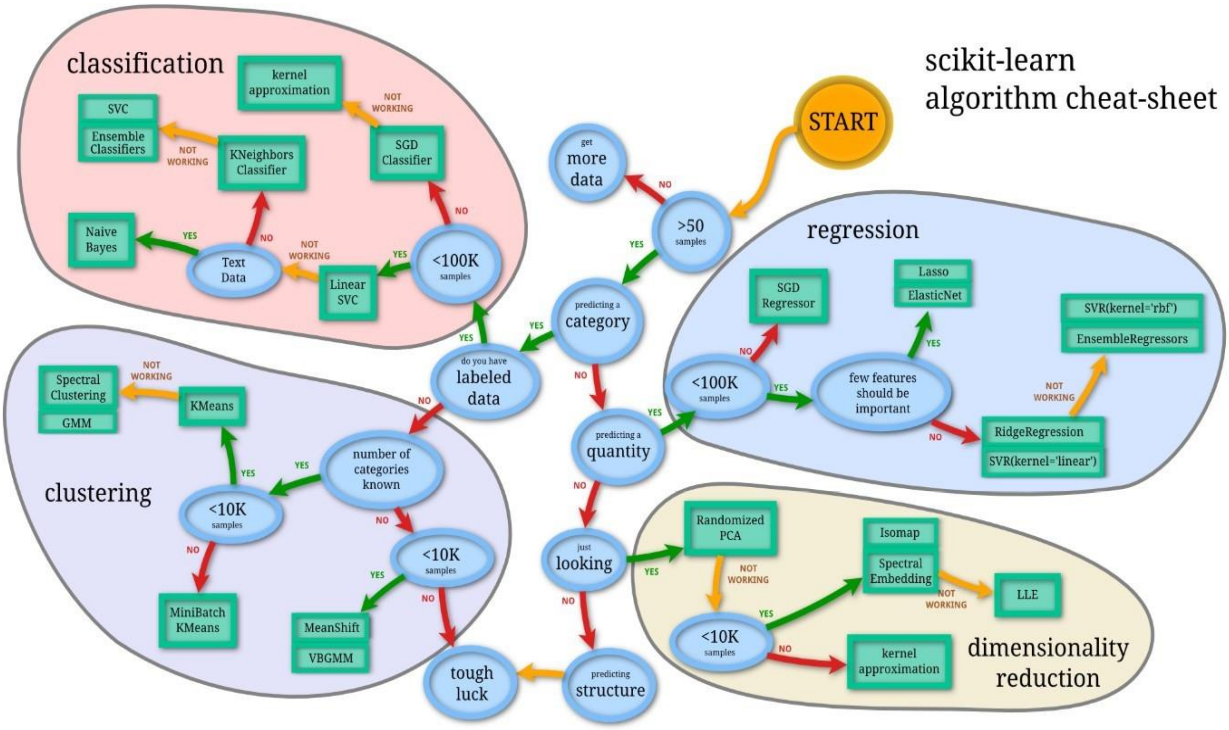

Рисунок 3.2 - Поради з вибору для алгоритму Scikit-learn

3.2 ONE-CLASS SUPPORT VECTOR MACHINE

Schölkopf [2001] представив однокласову машину опорних векторів (OCSVM) для вирішення проблеми виявлення новизни. При виявленні на етапі

прогнозування класифікатор намагається визначити, чи можна точку даних відрізнити від вихідних точок даних, які спостерігаються на етапі навчання.

OCSVM - це розширення стандартної двійкової машини опорних векторів (SVM). Wang (2017) вказували на важливість вибору гіперпараметрів класифікатора, який має значний вплив на його ефективність. У машинному навчанні гіперпараметр - це параметр, значення якого встановлюється до початку навчального процесу. За даними Wang (2017), найважливішими гіперпараметрами, які потрібно правильно налаштувати, є наступні: «Коефіцієнт регуляризації ν та ширина ядра Гауса σ. N контролює верхню межу відхилених цільових даних, яка часто налаштовується на відхилення шуму під час тренування, тоді як σ контролює плавність межі рішення. Надмірно великий або малий σ призведе до недооснащення та переоснащення відповідно. Неправильний ν призведе до спотворення межі рішення через шумні цільові дані або відхилення надмірних цільових даних.». У машинному навчанні недооцінка - це межа прийняття класифікатора, яка занадто проста, щоб пояснити дисперсію даних. Надмірне оснащення є складною межею прийняття рішень. Обидві обставини призводять до поганої класифікації.

Бібліотека Scikit-learn забезпечує реалізацію OCSVM у класі OneClassSVM модуля svm. Реалізація заснована на libsvm.

Libsvm - це бібліотека машинного навчання з відкритим кодом, написана на C ++. Найважливіші гіперпараметри OneClassSVM представлені в таблиці 3.1, повний список гіперпараметрів можна знайти в посібнику користувача номер 4.

Таблиця 3.1

Повний список гіперпараметрів

Гіперпараметр	Опис	Можливі значення
Kernel	Вказує тип ядра, який буде використовуватися	‘linear’,

	в алгоритмі. Якщо викликається, він використовується для попереднього обчислення матриці ядра	‘rbf’, ‘poly’, ‘sigmoid’, ‘precomputed’, callable
gamma	Коефіцієнт ядра для ‘rbf’, ‘poly’ і ‘sigmoid’. Якщо gamma='scale' він використовує 1 / (n_features * X.var()) як значення gamma.	float
tol	Толерантність до критерію зупинки	float
nu	Частка помилок у навчанні та векторах підтримки. Має бути в інтервалі [0, 1]. За замовчуванням буде прийнято 0,5	float

Параметр ядра регулює функцію обчислення гіперплощини вектора підтримки. Значення за замовчуванням - «rbf», що означає радіальну базову функцію. У цій дипломній роботі під час налаштування гіперпараметрів було перевірено різні значення параметрів.

3.3 IsolationForest

IsolationForest - це метод керування без нагляду, який виконує виявлення нестабільних об'єктів у високовимірних наборах даних. Метод ізолює зразки шляхом випадкового вибору ознак та розподілу значень між максимальним та мінімальним значенням обраних. Він може бути представлений деревоподібною структурою[Scikit, 2019]. Метод виявляє аномалії, засновані на концепції ізоляції, тобто без використання вимірювання відстані або щільності. Метод повертає оцінку аномалії кожного зразка. Бібліотека Scikit-learn забезпечує безліч гіперпараметрів, що впливають на функціональність. Деякі

параметри пояснюються в таблиці 3.2, повний список гіперпараметрів можна знайти в посібнику користувача.

Таблиці 3.2

Список гіперпараметрів

Hyperparameter	Опис	Можливі значення
contamination	«Частка викидів у наборі даних. Використовується під час навчання для визначення граничного значення функції прийняття рішення ».	float in (0., 0.5), optional (default=0.1)
n_estimators	«Кількість базових оцінювачів».	int, optional (default=100)
bootstrap	«Чи відповідають окремі дерева випадковим підмножинам навчальних даних, відібраних із заміною.»	boolean, optional (default=False)

3.4 Local Outline Factor

Local Outlier Factor (LOF) - це некерований алгоритм, запропонований Breunig [2000] для пошуку аномальних точок даних шляхом вимірювання локального відхилення даної точки від її сусідів. Алгоритм може ефективно виявляти локальні відхилення від великих наборів даних. Оцінка аномалії кожної вибірки називається Local Outline Factor. Вона ілюструє, наскільки ізольований об'єкт порівнюється з оточуючими її сусідами. Оточення

визначається найближчими сусідами, а відстань використовується для оцінки місцевої щільності. Бібліотека Scikit-learn забезпечує численні гіперпараметри для LOF. Деякі гіперпараметри представлені в таблиці 3.3, повний перелік гіперпараметрів можна знайти в посібнику користувача.

Таблиця 3.3

Гіперпараметри LocalOutlierFactor

Гіперпараметр	Опис	Можливі значення
n_neighbors	«Кількість сусідів, яку слід використовувати за замовчуванням для запитів k-neighbors. Якщо n_neighbors перевищує кількість наданих зразків, будуть використані всі зразки.»	int, optional (default=20)
algorithm	«Алгоритм, який використовується для обчислення найближчих сусідів: «ball_tree» використовуватиме BallTree, «kd_tree» використовуватиме KDTree. «Brute» використовуватиме brute-force. «Auto» намагатиметься визначити найбільш підходящий алгоритм на основі знань навчання».	{‘auto’, ‘ball_tree’, ‘kd_tree’, ‘brute’}, optional
contamination	«Частка викидів у наборі даних. Використовується під час навчання для визначення	float in (0., 0.5), optional (default=0.1)

	порогу функції прийняття рішення.»	
novelty	За замовчуванням LocalOutlierFactor призначений лише для виявлення сторонніх факторів. Встановіть для новинки значення True, якщо ви хочете використовувати LocalOutlierFactor для виявлення новинки	boolean, default False

3.5 NAÏVE BAYES

Згідно з посібником користувача Scikit-learn, методи Naive Bayes - це набір керованих алгоритмів навчання, заснованих на застосуванні теореми Байєса. Класифікатор Байєса припускає, що наявність певної ознаки не пов’язана з наявністю будь-якої іншої ознаки. Класифікатори Байєса надзвичайно добре показали свою роботу в реальних ситуаціях, таких як: класифікація документів та фільтрація спаму. Стверджується, що класифікато Байєса вимагає невеликої кількості навчальних даних для оцінки необхідних параметрів.

Учні Байєса та класифікатори можуть бути швидшими порівняно з більш досконалими методами, проте ці класифікатори також відомі як поганий оцінювач, тому результати вірогідності не згарантують достовірний результат. Бібліотека Scikit-learn пропонує безліч варіантів реалізації Naive Bayes. У цій роботі було обрано класифікатор багаточленового класифікатора Байєса (MultinomialNB), оскільки класифікатор використовувався при класифікації тексту. [Scikit, 2019].

3.6 STOCHASTIC GRADIENT DESCENT

Stochastic Gradient Descent (SGD) - це контрольоване дискримінаційне навчання лінійних класифікаторів під опуклими функціями втрат. Посібник користувача Scikit-learn вказує, що SGD застосовується для виявлення та обробки масштабних та розріджених проблем машинного навчання, які часто зустрічаються при класифікації тексту, а також при обробці природних мов.

Класифікатор може масштабуватися до дуже великих матриць з більш ніж 10^5 навчальних зразків і більше 10^5 особливостей. SGDClassifier Scikit-learn підтримує багатокласову класифікацію, поєднуючи кілька двійкових класифікаторів в схемі один проти всіх (OVA). [Scikit, 2019].

Фактичний класифікатор можна вибрати як гіперпараметр. У таблиці 3.4 представлені деякі гіперпараметри, повний перелік яких можна знайти в посібнику користувача.

Таблиця 3.4

Гіперпараметри SGDClassifier

Гіперпараметр	Опис	Можливі значення
los	Функція втрат, яку слід використовувати. «hinge» дає лінійний SVM. Втрата «log» дає логістичну регресію, імовірнісний класифікатор. « perceptron » - це лінійні втрати, що використовуються алгоритмом персептрона. Інші	tr, default: ‘hinge’, ‘log’, ‘modified_huber’, ‘squared_hinge’, ‘perceptron’

	втрати призначені для регресії замість класифікації ".	
lph	Константа, яка множить термін регуляризації. Також використовується для обчислення швидкості навчання, коли встановлено «optimal»	float, default=1 4
tol	Критерій зупинки. Якщо це не значення None, ітерації зупиняться, коли (loss> best_loss - tol) для n_iter_no_change послідовних епох	float or None, optional (default=1e-3) bool, optional
shuffle	«Чи слід перемішувати дані навчання після кожної епохи (ітерації)».	Default: True

3.7 LINEAR SUPPORT VECTOR MACHINE

Support Vector Machine (SVM) - це сукупність контрольованих методів навчання, що використовуються для класифікації, регресії та виявлення вибуху. У посібнику користувача Scikit-learn [2019] було вказано, що SVM ефективний навіть у випадку великої кількості функцій, в порівнянні з кількістю зразків, але класифікатор може зазнати

перевантаження. SVM також ефективно використовує пам'ять. SVM підтримує вказівку різних функцій ядра для функції прийняття рішення.

Перевантаження можна уникнути, вибравши ядро та термін регуляризації. SVM безпосередньо не надають оцінки ймовірності. У цій роботі лінійне ядро було обрано для подальшого вивчення. LinearSVC від Scikit-learn забезпечує лінійний SVM. Реалізація базується на бібліотеці liblinear, яка є більш ефективною, ніж реалізація на основі бібліотеки libsvm.

LinearSVC підтримує багатокласову класифікацію за схемою OVA. У таблиці 3.5 представлені деякі важливі гіперпараметри LinearSVC, повний перелік гіперпараметрів можна знайти в посібнику користувача.

Таблиця 3.5.

Гіперпараметри LinearSVC

Гіперпараметр	Опис	Можливі значення
penalty	«Вказує норму, яка застосовується під час штрафування. Штраф «l2» - це стандарт, який використовується у SVC. «L1» веде до розріджених векторів коефіцієнтів.»	string, «l1» or «l2» (default='l2')
loss	«Визначає функцію втрат. «hinge» - це стандартна втрата SVM, тоді як	string, «hinge» or «squared_hinge» (default=»square d_hinge»)

	«squared_hinge» - квадрат втрати hinge»	
dual	«Виберіть алгоритм для вирішення подвійної або первинної задачі оптимізації. Віддайте перевагу dual = False, коли n_samples> n_features»	bool, (default=True)
tol	Терпимість до критеріїв зупинки	float, optional (default=1e-4)
C	Параметр штрафу C терміну помилки	float, optional (default=1.0)

Реалізація використовує генератор випадкових чисел для вибору функцій під час підгонки моделі. Припускається, що це може призвести до різних результатів для тих самих вхідних даних. І якщо це трапиться, потрібно спробувати зменшити значення параметра tol.

3.8 K-Nearest Neighbors

Посібник користувача Scikit-learn (2019) називає класифікацію найближчих сусідів методом навчання на основі екземпляру або незагальненим методом навчання. Метод не намагається сформувати загальну внутрішню модель, але, замість цього, зберігає екземпляри навчальних даних. Класифікація обчислюється як проста більшість голосів найближчих сусідів кожного пункту.

Потім точці запиту присвоюється клас даних, який має найбільшу

кількість представників у найближчих сусідах точки. Алгоритм простий, але метод був успішно застосований у ряді проблем класифікації та регресії, таких як: рукописні цифри або сцени супутникових зображень.

Методи найближчих сусідів можуть бути корисними в ситуаціях класифікації, в таких втпадках, коли межа прийняття рішення є дуже нерегулярною. (Scikit, 2019).

3.9 Інтерфейси програмування Scikit-learn

У бібліотеці Scikit-learn оцінювач класифікації - це клас Python, який реалізує методи підгонки та прогнозування. Встановлення оцінювача означає, що дані навчання подаються як параметр. Дані тестування надаються як параметр, щоб мати можливість передбачити класи, до яких належать невидимі зразки. Як правило кодування, оцінювачі Scikit-learn дотримуються певних вказівок, щоб зробити свою поведінку більш передбачуваною та зручною в роботі і відповідно значно спростити задачу для розробників.

Бібліотека забезпечує клас Pipeline для послідовного застосування списку перетворень та остаточного оцінювача. Проміжні етапи повинні реалізовувати методи підгонки та перетворення. Тільки остаточний оцінювач повинен реалізувати метод прогнозування. Мета конвеєра - полегшити кроки обробки, які можна перехресно перевірити.

Для реалізації нового оцінювача, сумісного з Scikit-learn, клас повинен успадковувати BaseEstimator Scikit-learn, а також один із класів типу оцінювача. Класи типів - ClassifierMixin, RegressorMixin, ClusterMixin та TransformerMixin. У випадку оцінювача для класифікації методи відповідності, прогнозування та оцінки потрібно замінити. Подальшу інформацію про впровадження оцінювачів можна знайти в посібнику розробника Scikit-learn.[9]

GridSearchCV - це корисний клас для налаштування гіперпараметрів або параметрів оцінювача в конвеєрі. Клас виконує вичерпний пошук за вказаними значеннями параметрів для оцінювача. GridSearchCV використовує методи підбору і оцінки. Він також може використовувати методи прогнозування, передбачення проблеми, рішення функції, перетворення та зворотної трансформації, якщо вони реалізовані в використовуваному оцінювачі. Він виконує перехресний перевірений пошук по сітці параметрів. Інший варіант - використовувати клас ParameterGrid безпосередньо передаючи кожен елемент сітки як параметр методу set_params оцінювача.

Пакет метрик містить корисні методи для обчислення оцінок класифікації, наприклад, методи f1_score та roc_auc_score. Пакет також містить метод Classification_report для друку звіту про результати класифікації та метод confusion_matrix, який повертає матрицю результатів класифікації. Scikit-learn також пропонує безліч реалізацій векторизатора для попередньої обробки текстових даних.

РОЗДІЛ 4
МЕТОДИ ОБРОБКИ ТЕКСТОВОЇ ІНФОРМАЦІЇ

Історія обробки природних мов (НЛП) почалася в 1950 році, коли Алан Тьюрінг займався цією темою у своїй роботі "Обчислювальні машини та інтелект". З тих пір НЛП зайняв своє місце в галузі інформатики, штучного інтелекту та обчислювальної лінгвістики. НЛП складається з процесів, необхідних для взаємодії між комп'ютерною та людською мовами. Іншими словами, NLP - це інженерія функцій для перетворення неструктурованого тексту в функції та алгоритми машинного навчання для подальшої обробки.

Кантарджич [2011] розділяє інженерію функцій на фази підготовки даних та зменшення даних. За його підрахунками, на цих етапах у процесі видобутку даних використовується понад 50% зусиль. Підготовка даних стосується трансформації необроблених даних, нормалізації, згладжування даних, обробки відсутніх даних та роботи з викидами. Зменшення даних стосується зменшення функцій, зменшення розмірності та зменшення вартості. Зниження характеристик важливо, оскільки більшість програм реального світу для видобутку даних характеризуються великими розмірами даних, де не всі функції важливі.

Зменшення розмірності відноситься до математичних перетворень для зменшення особливостей, наприклад, аналіз основних компонентів Кархунена – Лоева. Зниження значення - це зменшення кількості дискретних значень для даної ознаки. Це також відомо як дискретизація функцій. У цій дипломній роботі реалізовано зменшення значення, перетворюючи показники CVSS на чотири рівні тяжкості.

У НЛП синтаксичний аналіз використовується для оцінки відповідності природної мови граматичним правилам. Деякі широко відомі прийоми НЛП: лематизація - це перетворення різних флексованих форм слова в єдину

форму. Морфологічна сегментація - це поділ слів на окремі одиниці, які називаються морфемами. У сегментації слів великий шматок суцільного тексту поділяється на окремі одиниці. Позначення частини мови використовується для визначення частини мови для кожного слова. Синтаксичний аналіз передбачає проведення граматичного аналізу поданого речення.

Стеммінг - це скорочення складених слів до їх кореневої форми. Токенізація - це розбиття тексту на послідовність N-грамових лексем. N-грам означає поєднання послідовності N-слів у лексеми. Наприклад, 2-грамовий маркер містить два наступних слова. Стоп-слово - це загальновживане слово, яке може не містити жодної корисної інформації, наприклад англійські слова „the", „a", „an", „in" тощо. Розпізнавання іменованих сутностей - це техніка, яка намагається зіставити елементи в тексті з власними іменами. Багато впроваджених методів реалізовано в наборі інструментів Stanford CoreNLP (Manning et al, 2014). Іншим набором інструментів, який був використаний у цій дисертації, є Інструментарій природничих мов (NLTK). Це платформа для побудови програм Python для роботи з написаним природним текстом. На рисунку 6 наведено Wijayasekara et al. (2014) техніка обробки та класифікації звітів про помилки до звичайних помилок та помилок, пов'язаних із безпекою. Ці етапи попередньої обробки також можуть бути використані в інших цілях класифікації тексту.

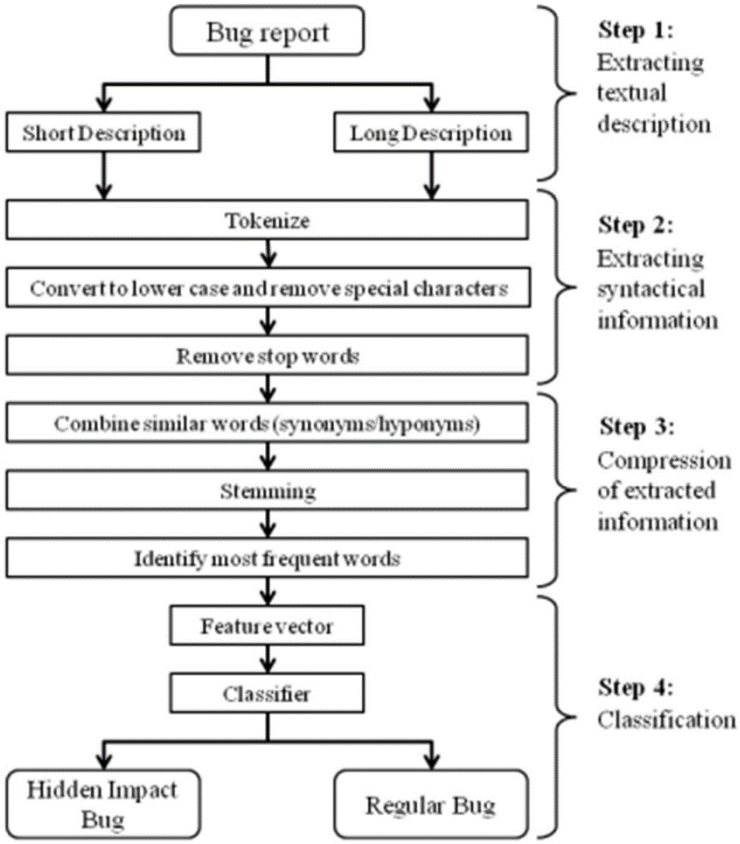

Рисунок 4.1. Етапи обробки природної мови

На рисунку крок 1 представляє основний неструктурований витяг тексту. Короткий опис - це заголовок, який може складати 5-10 слів. Довгий опис - це повідомлення про дефект, кілька речень про помилку. На кроці 2 синтаксична інформація витягується у вигляді окремих унікальних слів, які перетворюються на малі регістри. У процесі вилучення видаляються слова та символи, які можуть не містити значної кількості інформації. Це робиться шляхом маркерування опису в поданні мішка слів, а потім видалення стоп-слів. Проста форма подання "мішка слів" полягає в тому, щоб вести підрахунок, скільки разів кожне слово з’являється в документі. На кроці 3 подібні слова поєднуються та проводяться основи, щоб визначити найбільш часті слова. Крок 4 ілюструє, як ознаки передаються класифікатору, який робить прогноз щодо тексту, поданого як вхідні дані для всього процесу. Етапи 2 і 3 представляють етапи підготовки та скорочення даних при

видобутку даних. [Wijayasekara] (2012) використовував Wordnet для ідентифікації синонімів для їх поєднання. WordNet - це велика лексична база даних англійської мови. Вони використовували Портер, що сформулював, щоб скоротити слова в їх основну форму. Поєднання слів, що несуть подібну інформацію, ще більше зменшує кількість ознак у матриці

4.1 Частота терміна - зворотна частота документа

Термін "Частота - зворотна частота документа" (TFIDF) - це числова статистична схема зважування, яка відображає, наскільки важливим є слово для документа в колекції чи корпусі. У мовознавстві корпус - це великий і структурований набір текстів. TFIDF - це дуже популярна схема зважування, яка використовується в цифрових бібліотечних системах. TFIDF можна розрахувати наступним чином:

$$d\ (i)\ =\ TF\ (wi, d)\ \times\ IDF\ (wi)\quad IDF\ (w)\ =\ log\ (\frac{|D|}{DF(w)})$$

Йоахімс [1996] пояснив, як працює TFIDF, так: «Термін частота $TF\ (wi,\ d)$ - кількість випадків, коли слово w зустрічається в документі d. Частота документів $DF\ (w)$ - це кількість документів, у яких слово w зустрічається принаймні один раз. Обернену частоту документа $IDF\ (w)$ можна обчислити за частотою документа. | D | - загальна кількість документів. Зворотна частота документа слова є низькою, якщо вона зустрічається у багатьох документах, і найвищою, якщо слово зустрічається лише один раз. Потім значення $d\ (i)$ ознаки wi для документа d обчислюється як добуток. $d\ (i)$ називається вагою слова wi в документі d. Ця схема зважування говорить, що слово wi є важливим терміном індексації для документа d, якщо воно часто зустрічається в ньому. З іншого боку, слова, які трапляються у багатьох документах, оцінюються менш важливими через низьку зворотну частоту

документів». Його дослідження зосереджувало категоризацію тексту за допомогою класифікаторів Байєса та TFIDF.

4.2 Стеммінг та лематизація

Стеммінг - це грубий евристичний процес, який обрізає кінці слів з надією правильно перетворити слова в їх кореневу форму. Оригінальний алгоритм стемінгу був введений в 1979 році в Комп'ютерній лабораторії, Кембридж, Англія. Алгоритм Porter Stemming написаний і підтримується його автором Мартіном Портером. У цій дипломній роботі було використано бібліотеку NLTK Snowball Stemmer, яка реалізує алгоритм стереографії Портера (Porter, 1980).

Іншим підходом до визначення основи слова є лематизація. Лемматизація - це процес визначення словникової форми слова на основі задуманого значення. Він може використовувати словник для відображення слів або підходів на основі правил. Згідно з дослідженнями Камачо-Колладоса та Пілехвара (2018), проста токенізація працює однаково або краще, ніж лематизація або групування багатослів для класифікації тексту за допомогою нейромережевих класифікаторів. У цій дипломній роботі було використано WordnetLemmatizer бібліотеки NLTK.

Векторизація тексту

Бібліотека Scikit-learn пропонує багато корисних занять з вилучення функцій з тексту. У Scikit-learn процес перетворення текстових документів на числові ознаки називається векторизацією. Витяг функцій відрізняється від вибору функцій: вилучення функції перетворює довільні дані в числові функції, які можна використовувати для машинного навчання. Вибір функцій - це техніка машинного навчання, яка

застосовується до цих функцій. [Scikit, 2019]. CountVectorizer від Scikit-learn забезпечує базове представлення функцій, що містять багато слів. Клас TfidfVectorizer забезпечує представлення функцій TFIDF. У таблиці 4.1 представлені деякі важливі параметри векторизатора, повний перелік параметрів доступний у посібнику користувача.

Таблиця 4.1

Параметри векторизатора

Параметр	Опис	Можливі значення
stop_words	«Якщо «англійська», використовується вбудований список зупинок слів для англійської мови. Якщо у списку передбачається, що цей список містить слова зупинки, усі вони будуть видалені з отриманих маркерів. Якщо ні, то слова зупинки використовуватись не будуть. Для параметра max_df можна встановити значення в діапазоні [0,7, 1,0] для	string {‘english’}, list, or None (default)

	автоматичного виявлення та фільтрування стоп-слів на основі частоти термінів у внутрішньому документі.»	
lowercase	«Перетворіть усі символи в малі регістри»	boolean, True by default
ngram_range	«Нижня і верхня межа діапазону значень для різних N-грам, що підлягають вилученню. Будуть використані всі значення n такі, що min_n <= n <= max_n.»	tuple (min_n, max_n) Default: (1, 1)
min_df	«При побудові словникового запасу ігноруйте терміни, частота документів яких суворо нижча за заданий поріг.»	float in range [0.0, 1.0] or int, default=1
token_pattern	«Регулярний вираз, що являється маркером. За замовчуванням він	string, Default: r'\b[^\d\W]+ \b'

	вибирає маркери з 2 або більше буквено-цифрових символів. Пунктуація повністю ігнорується і завжди трактується як роздільник символів.»	

Стоп-слова вважаються неінформативними і їх можна видалити, щоб уникнути їх використання при прогнозуванні. Списки слів можуть містити слова, які можуть містити важливу інформацію, наприклад, «вразливість». Списки списків слів - це простий інструмент управління шумом. [Nothman, 2018]. У цій роботі регулярний вираз шаблону лексеми змінено за замовчуванням, так що об'єкти виділяють щонайменше три символи буквено-цифрових слів.

Вбудовані векторизатори Scikit-learn не враховують потенційні орфографічні помилки або виведення слів, але функціонал можна додати, налаштувавши маркер або аналізатор. Посібник користувача Scikit-learn [2019] вказує, що замість передачі налаштованих методів як параметрів конструктора більш доцільним є успадкування класу та перевизначення методів-членів: build_preprocessor, build_tokenizer та build_analyzer. Основні функціональні можливості методів-членів описані більш докладно в посібнику користувача.

РОЗДІЛ 5
ПРАКТИЧНА РЕАЛІЗАЦІЯ

Якоб Дж. Тьо [2016] у своїй дисертації зробив висновок, що ефективність кожного класифікатора сильно різнилась між наборами даних, але класифікатор Байєса перевершував усі інші класифікатори у всіх випадках, тоді як класифікатори SVM завжди були одними з найбільш ефективних. У ході дослідження три окремі бази даних про дефекти NASA використовувались як набори даних. Дослідження було зосереджене на виявленні прихованих помилок впливу за допомогою контрольованих та некерованих підходів машинного навчання. Помилку прихованого впливу можна визначити як вразливість, виявлену як таку після розкриття помилки. Результатом цього дослідження було те, що, хоча підхід без нагляду добре працював, він не був настільки ефективним, як метод під наглядом, досяг G-балу лише 0,715, де найкращий контрольований підхід досяг G-Score 0,903. Дослідження пропонує продовжувати вивчати узагальнення профілів вразливості, і слід проводити емпіричні дослідження бази даних помилок з відкритим кодом. Як альтернативу, дослідження пропонує постійний підхід до виявлення аномалій до багатокласової класифікації, що розглядає кожен клас багатокласової проблеми як однокласну проблему виявлення аномалії.

Міямото та ін. [2015] порівняв наступні алгоритми: класифікатор Байєса, Latent Dirichlet Allocation, приховане семантичне індексування та Supervised Latent Dirichlet Allocation (SLDA) для оцінки оцінки CVSS. Вони використовували описи вразливості NVD як набір даних. Вони виявили, що алгоритм SLDA є найбільш ефективним, і довели ще кращі результати, додавши річну вагу алгоритму. Врешті решт дійшли висновку, що їх метод не може виявити абсолютно нові вразливості. Вони також зауважили, що якби інформація про CWE була присвоєна вразливості, можливим стало б і

підвищення точності. Отже, вони припустили, що, можливо, можна створити двоступеневий метод, який спочатку оцінює CWE-ID з опису, а потім оцінює базові показники CVSS, використовуючи їх метод.

Ламкамфі та ін. [2011] виявили, що мультиноміальний класифікатор Наїв Байєса найшвидше може досягти стабільної точності, маючи лише близько 250 повідомлень про помилки кожного ступеня важкості на етапі навчання. Вони вивчали бази даних про дефекти GNOME та Eclipse, щоб передбачити серйозність повідомлень про помилки, але не у сенсі вразливостей або безпеки. Вони вивчають підходи класифікаторів Байєса, K-nearest neighbour та машини опорних векторів (SVM).

Бозоргі та ін. [2010] найбільше досліджував бази даних OSVDB та Mitre CVE для визначення найбільш ймовірно використаних вразливостей. Вони використовували класифікатор SVM для розмежування даних на два класи. Вони виявили, що багато функцій не мають значення, але багато інформації міститься в текстових полях. Вони використовували представлення "bag-of-words" для кожного текстового поля, щоб перетворити текст на елементи.

Петерс та ін. [2017] розробив інструмент для пошуку звітів про помилки, пов'язані з безпекою, використовуючи кілька баз даних про дефекти проекту з відкритим кодом. Вони використали методи видалення стоп-слів та попередньої обробки TFIDF та порівняли Random Forest, Naive Bayes.

Алгоритми машинного навчання логістичної регресії, багатошарового персептрону та K-найближчих сусідів. Їх результати показують, що вибір алгоритму машинного навчання різниться між базами даних. Інструмент вирішує проблему дисбалансу класу. Дані страждають від проблеми дисбалансу класів, коли розподіл класів сильно незбалансований. У цьому контексті багато алгоритмів навчання класифікації мають низьку точність прогнозування для рідкісних класів.

Хан та ін. [2017] вивчав прогнозування балів CVSS2 на основі описів вразливості NVD. Вони розділили вразливість на чотири класи за ступенем важкості від критичної до низької. У своєму підході, замість того, щоб покладатися на ручну розробку функцій, вони використовували вбудовування слів та одношарову неглибоку згорткову нейронну мережу (CNN), щоб автоматично фіксувати дискримінаційні особливості слів та речень описів вразливості для прогнозування тяжкості вразливості. Вони прийняли стратегію недовибірки, щоб збалансувати набори даних про навчання та тестування. Вони порівняли свій метод із TFIDF + SVM, вбудованими словами + SVM, вбудованими словами + 2-шаровим CNN та вбудованими словами + CNN з довгостроковою короткочасною пам'яттю (LSTM). Їх метод перевершив конкуруючі методи, маючи оцінку F1, усереднену за класами тяжкості 0,816.

Вбудовування слів - це загальна назва методів, коли слова чи фрази зі словникового запасу відображаються у векторах дійсних чисел. Він передбачає математичне вбудовування з простору з багатьма розмірами на слово в неперервний векторний простір з набагато нижчим розміром.

CNN та LSTM - це нейромережеві підходи в машинному навчанні. Спочатку CNN був розроблений для зіставлення даних зображень із вихідною змінною. LSTM був запропонований в 1997 році Зеппом Хохрейтером та Юргеном Шмідхубером для вирішення проблем вибуху та зникнення градієнтів. Спочатку CNN був розроблений для зіставлення даних зображень із вихідною змінною.

Іншим способом пошуку вразливостей на додаток до сканування текстів дефектів є сканування вихідного коду. Оскільки звіти про дефекти можуть містити фрагменти коду в тексті, Wijayasekara et al. [2012] використовував статичний аналізатор коду для створення функцій на додаток до методів попередньої обробки тексту. Лі та ін. [2018] розробив інструмент, що використовує нейромережеві методи для виявлення

потенційних вразливостей у вихідному коді. Інструмент обмежений лише мовами програмування C / C ++.

Sanguino та Uetz [2017] проаналізували словник CPE та канали CVE. Вони розробили метод, який рекомендує визначити пріоритетний список ідентифікаторів CPE для даного програмного продукту. На основі їх спостережень є чотири основні проблеми: записи CVE без посилань на CPE, програмні продукти без призначених CPE, друкарські помилки та відсутність синхронізації між обома наборами даних можуть призвести до неправильних результатів виведення систем управління вразливостями. Вони пропонують, щоб NIST, відповідальний за підтримку цих сховищ, визначив механізм подолання цих проблем.

Ruohonen and Leppänen [2018] вивчали методи пошуку текстової інформації для зіставлення CVE до CWE. На основі інформації про NVD вони намагалися призначити CWE для вразливостей у чотирьох програмних продуктах, які можна знайти в базі даних Snyk. Вони застосували традиційні методи попередньої обробки тексту та чотири системи зважування, включаючи TFIDF, 1-3 грами та прихований семантичний аналіз, щоб зіставити CWE з уразливістю. І, зрештою, отримали погані результати точності, але їх не можна узагальнити для всіх репозиторіїв чи мов програмування. Також ними було виявлено, що прості пошуки ключових слів на основі ідентифікаторів CVE та CWE є більш надійними. Ruohonen and Leppänen [] зауважили, що вибір серед певних корпорацій, пов'язаних з безпекою, швидше за все, сильно вплине на відображення вразливості - CWE, і CWE не дуже схожі між собою. Незважаючи на те, що уніграми дали найкращі результати, наприклад, триграми «відмова в обслуговуванні» та «відмінювання покажчика NULL» повинні досягати вищих ваг, ніж будь-які інші N-грами.

РОЗДІЛ 6
МЕТОДИ ТА ДАНІ

База даних NVD була використана для вивчення класифікації CVSS та CWE, а також для вивчення однокласних класифікаторів. Для вивчення одинарної класифікації було зібрано кілька баз даних про дефекти проектів з відкритим кодом. Проблеми, пов'язані з безпекою, були позначені фахівцем із безпеки вручну, щоб підтвердити, що ключові слова про вразливість можна знайти у звітах про дефекти. Автоматична класифікація CVSS та CWE виконувалась лише з використанням даних NVD.

6.1 NVD DATA

NVD надає канали даних про вразливості у форматі JSON, зазначеному в схемі NVD JSON 1.0. Канали даних доступні у стислому форматі, де файли розподіляються щороку. Крім того, нещодавно додані та нещодавно змінені вразливості доступні в окремих файлах. У цій дипломній роботі були завантажені останні вразливості та всі дані про вразливість з 2018 року до кінця квітня 2019 року. 18755 вразливостей використовувались як набори даних в експериментах. На рисунку 6.1 наведено приклад одного елемента даних про вразливість, який завантажується та аналізується для подальшого використання. За даними Міямото та співавт. (2015), присвоєння вищих ваг нещодавно опублікованим CVE дає кращі результати класифікації.

Крім того, звужуючи розмір набору даних, це може заощадити обчислювальну потужність.

```
['CVE-2018-5357',
 'ImageMagick 7.0.7-22 Q16 has memory leaks in the ReadDCMImage function in coders/dcm.c.',
 'CWE-399',
 'AV:N/AC:M/Au:N/C:N/I:N/A:P',
 4.3,
 'CVSS:3.0/AV:N/AC:L/PR:N/UI:R/S:U/C:N/I:N/A:H',
 6.5,
 ['imagemagick', 'imagemagick', 'canonical', 'ubuntu_linux']]
```

Рисунок 6.1 - Приклад аналізу даних NVD

Представлена структура даних - це список Python, в якому елементи представлені нижче, перераховані зверху: ідентифікатор CVE, текст вразливості, клас CWE, вектор CVSS2, оцінка CVSS2, вектор CVSS3, оцінка CVSS3 та назви постачальників та компонентів CPE. Тексти вразливості були попередньо оброблені та передані алгоритмам машинного навчання для прогнозування класифікацій. Для перевірки результатів класифікатора використовувались вектори CVSS та класи CWE. Імена CPE, з них 6571, були вивчені, щоб порівняти, чи має виняток чи включення їх кращі результати. За даними Han та співавт. [2017] Тексти вразливості стислі, середня довжина становить лише 37,5 ± 15,4 слова. Не всі вразливості мають CVSS2, CVSS3 та CWE, наведені у цілому наборі даних. На рисунку 6.2 представлено розподіл даних метрик CVSS2.

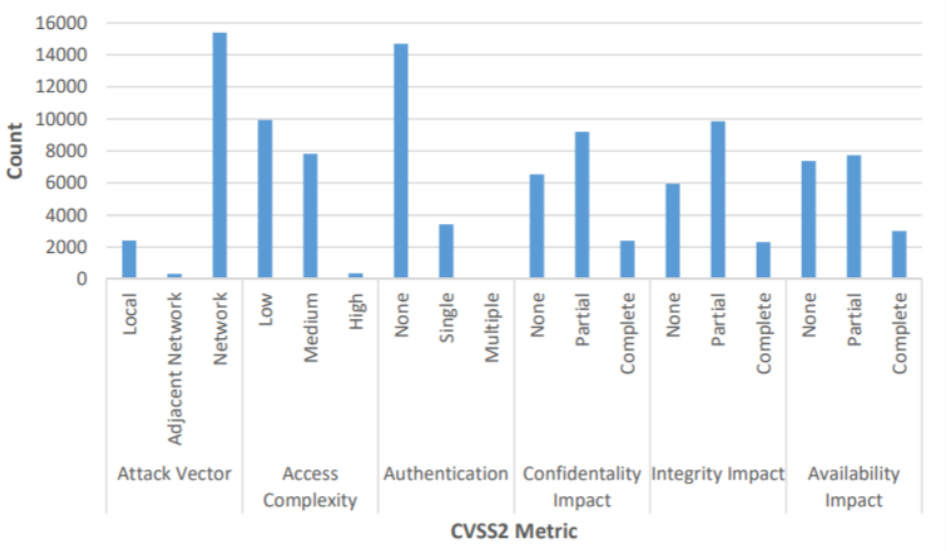

Рисунок 6.2 Розподіл даних CVSS3 за векторами

В даній роботі підхід до класифікації намагається передбачити вектори метрики за метрикою, а потім обчислити оцінку. У CVSS2 існує шість прогнозів класифікації, які слід зробити на основі метрик для обчислення фактичного балу. Як показано на малюнку 8, існує дисбаланс високого класу у векторі атаки, складності доступу та показниках автентифікації. Клас "Множинність" метрики автентифікації був повністю відкинутий, оскільки існує лише 4 спостереження за ними. Щодо метрики вектора атаки та складності доступу є понад 300 спостережень за класами малої кількості. Основна кількість вразливостей може бути використана з мережі, маючи низьку складність доступу, не вимагаючи автентифікації зловмисником. Часткова втрата конфіденційності, цілісності та доступності є основними наслідками вразливостей.

На рисунку 6.3 показано розподіл даних CVSS3 за векторами. Всього є 17928 вразливих місць, вектор CVSS3 подано у цілому наборі даних. Класифікаційний підхід робить вісім прогнозів до того, як можна розрахувати фактичний бал CVSS3. Як вказує рисунок, є деякі класи, які мають невелику кількість спостережень, але жодне не повинно бути відкинуте. На векторі атаки є лише 177 фізичних вразливостей, які можна використати із сусідньої мережі лише трохи більше 300 спостережень. При низькому впливі на доступність існує лише близько 300 вразливих місць. Більшість уразливостей можна використовувати в мережі, маючи низьку складність атаки, не вимагаючи привілеїв від зловмисника та не потребуючи взаємодії з користувачем.

В основному, оскільки область дії не змінюється, права доступу не підвищуються вразливими компонентами. Найбільший вплив має великий рівень конфіденційності, цілісності та доступності для оцінюваних вразливостей CVSS3. Цей тип середньої вразливості дає базовий бал 9,8.

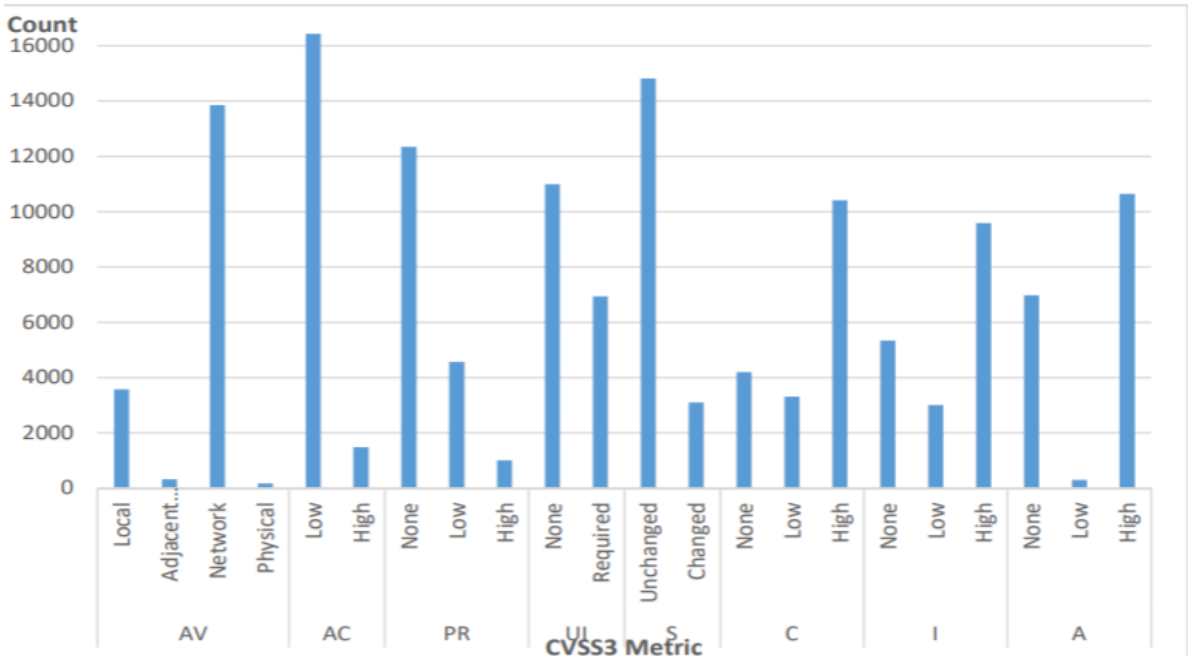

Рисунок 6.3 - Розподіл даних CVSS3 за векторами

Хан та ін. [2017] класифікував уразливості на чотири групи за ступенем тяжкості, як зазначено в таблиці 6.1. Поради щодо безпеки Atlassian класифікують вразливості на чотири рівні серйозності на основі оцінок CVSS. Вони оцінювали ступінь вразливості на основі цих категорій. У цій дисертації категорії важкості використовуються в класифікації після обчислення фактичного балу.

На рисунку 6.4 вразливості за ступенем серйозності показані в балах CVSS2 та CVSS3.

Таблиця 6.1

Вразливості за ступенем важкості

Діапазон	**Опис**	**Категорія**
9.0 – 10.0	Вразливості цього діапазону зазвичай використовуються	Critical

	прямо, оскільки зловмисникові не потрібні спеціальні дані автентифікації або знання про окремих жертв	
7.0 – 8.9	“Вразливості, як правило, важко використати. Використання таких уразливих місць може призвести до підвищених привілеїв, значного простою або порушення конфіденційності, цілісності або доступності	High
4.0 – 6.9	Вразливості, як правило, вимагають від зловмисника знаходження в одній локальній мережі з жертвою або маніпулювання окремими жертвами за допомогою соціальної інженерії. Вплив таких	Medium

	вразливостей, як правило, пом'якшується такими факторами, як привілеї користувача, вимоги до автентифікації	
0.1 – 3.9	Вразливості на рівні «low» дуже мало впливають на бізнес підприємства. Для використання завжди потрібен доступ до локальної або фізичної системи	Low

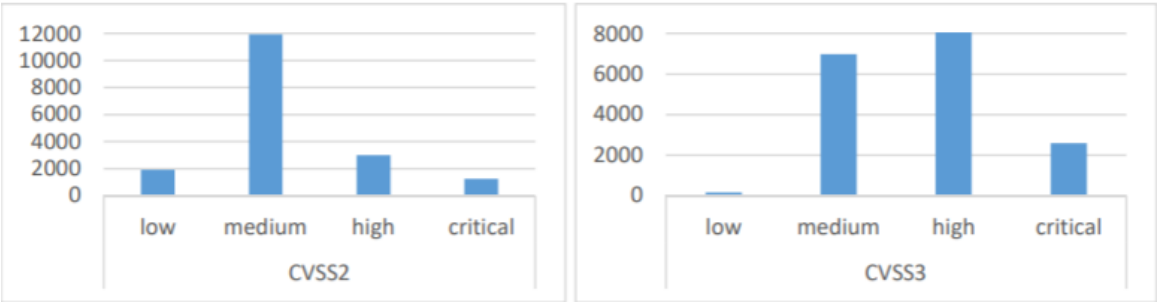

Рисунок 6.4 - Розподіл кількості CVSS на основі серйозності вразливості

В даній роботі в рейтингу CVSS2 11946 вразливостей потрапили в середню категорію, але решта вразливостей становила від 1000 до 3000. У CVSS3 бали були 163 вразливості в низькій категорії, 6984 вразливості в середній, 8187 вразливості у високій та 2590 вразливостей у критичній важкості.

Існує 18755 вразливостей та 105 різних CWE. На рисунку 6.5

наведено розподіл CWE. Є 38 CWE, які призначаються менше 10 разів. У 5 найкращих CWE є 8228 вразливостей з усіх 18775.

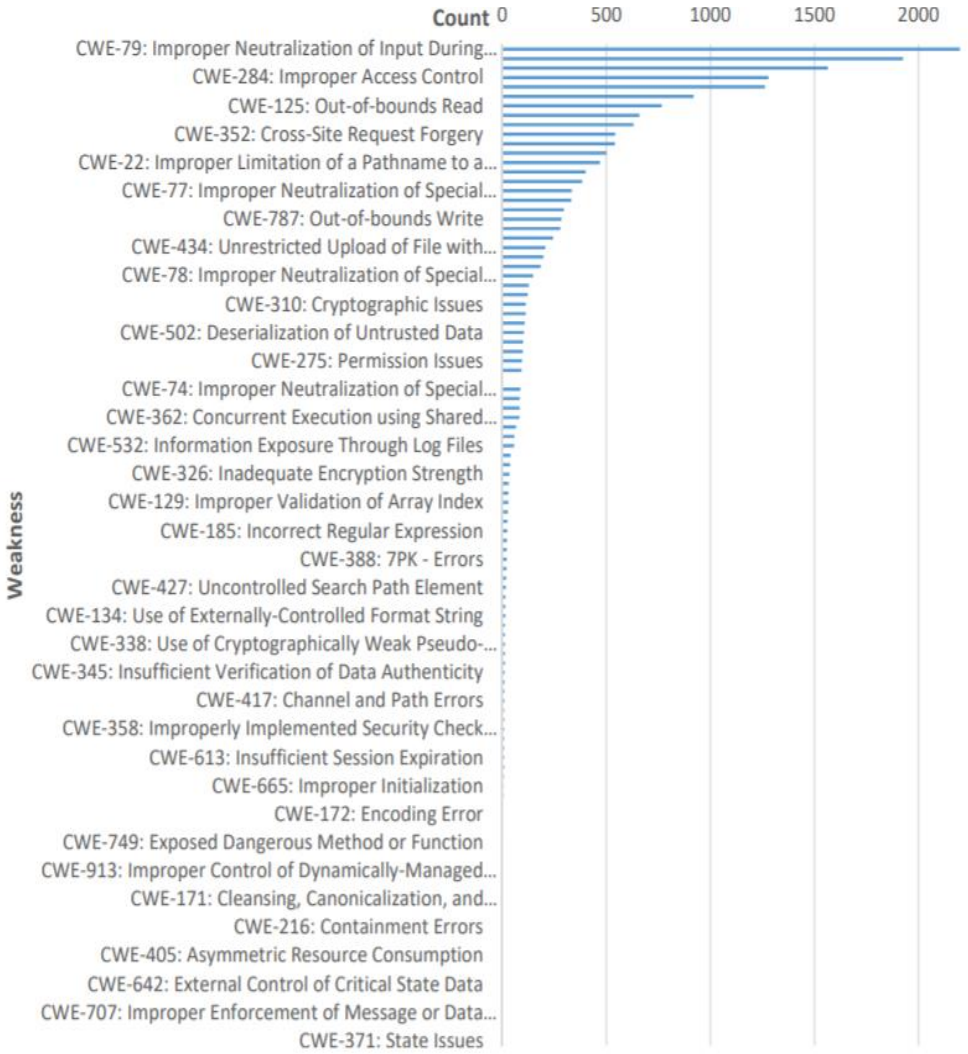

Рисунок 6.5 - Розподіл вихідного набору даних CWE

Перші 5 оригінальних наборів даних CWE складаються з неправильної нейтралізації вводу під час генерації веб-сторінок, неправильного обмеження операцій в межах буфера пам'яті, неправильної перевірки вводу, неправильного контролю доступу та експозиції інформації. У цій дипломній роботі була зроблена спроба збалансувати дані CWE шляхом пошуку кореневих категорій малої кількості CWE. Якщо CWE було менше

десяти разів, робилася спроба пошуку кореневої категорії. Це реалізовано в класі CweFinder, який можна знайти в Додатку 1. Ієрархія CWE доступна у різних форматах, розділених у трьох різних поданнях. Подання концепції дослідження було використано лише в цій дипломній роботі. На малюнку 6.6 наведено розподіл кореневих CWE із представленого повного набору даних.

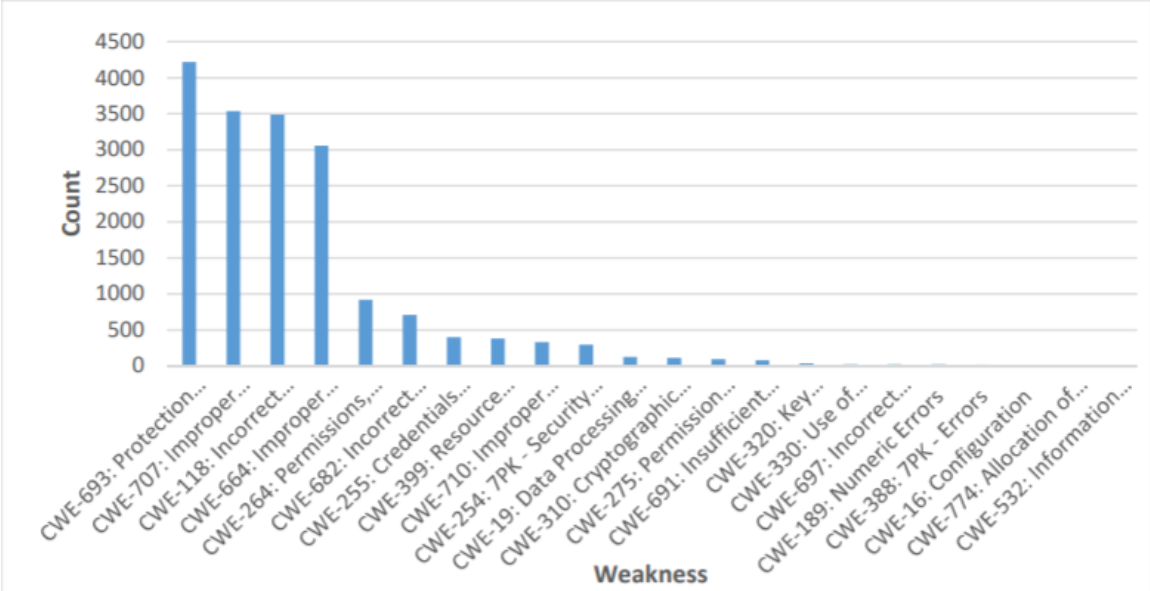

Рисунок 6.6 - Розподіл кореневих CWE на повному наборі даних

У поданні Концепції дослідження корінні CWE зосереджені в основному навколо чотирьох основних типів слабкості, як показано на рисунку 6.6:

- CWE-693: Збій механізму захисту
- CWE-707: Неправильне застосування структури повідомлень або даних
- CWE-118: Неправильний доступ до індексованого ресурсу
- CWE-664: Неправильний контроль над ресурсом протягом усього життя.

Після знаходження кореневих CWE для повного набору даних, два кореневі елементи були призначені лише два рази: вплив інформації через файли журналів та розміщення дескрипторів файлів. Після повного відкидання числа CWE, менших або рівних десяти, чисельність даних у

підсумку мала 18753 вразливості в 70 різних CWE. Для 10-кратного методу перехресної перевірки потрібно щонайменше десять спостережень у кожному класі. Розподіл даних представлений на малюнку 6.7, який був використаний в експериментах. Повністю відкинуті CWE показані в Таблиці 6.4. Більш детально розглянувши відкинуті CWE, він виявляє багато застарілих категорій або пропонується до припинення в наступних версіях. Було 631 порожніх CWE та 121 CWE з маркуванням "NVD-CWE-noinfo". Вони повністю відкидаються, а також не описують причин уразливості.

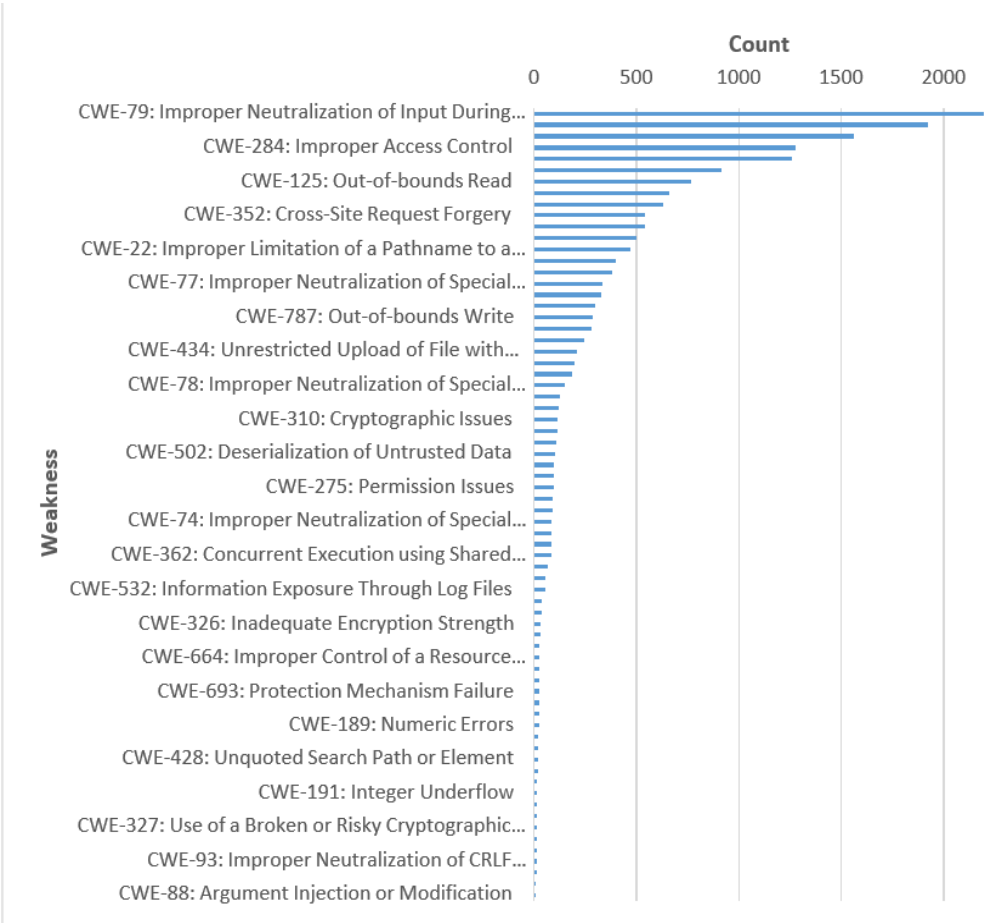

Рисунок 6.7 - Розподіл CWE вибраних кореневих елементів

Було 32 вразливості, які були відхилені з якихось причин, наприклад: відхилені CNA або дублікати, які не містять корисної інформації для прогнозування. Ці вразливості було відкинуто із набору даних. Крім того,

була 41 вразливість, оскаржена або відхилена CNA, яка, однак, містить корисні описи, але це єдине слово було видалено. Ці описи вразливості починаються з «** DISPUTED**» або «** REJECT **». Небажані загальновживані слова були вилучені з описів вразливості, щоб краще розрізняти дефект та вразливість. На додаток до стандартних слів зупинки Scikit-learn видалено такі слова: “проблема”, “дефект”, “помилка”, “недолік”, “версія”, “система”, "Тому що", і "раніше".

Таблиця 6.2

Інформація про вилучені та застарілі CWE

ID	Ім’я	Опис
CWE-18	«CWE CATEGORY: Source Code»	«Цей запис розглядається як застарілий ».
CWE-398	«CWE CATEGORY: 7PK – Code Quality»	«Погана якість коду призводить до непередбачуваної поведінки. З точки зору користувача, часто виявляється як погана зручність використання.»
CWE-371	«CWE CATEGORY: State Issues»	«Слабкі місця в цій категорії пов'язані з неправильним управлінням станом системи.»
CWE-199	«CWE:CATEGORY: Information Management»	«Слабкі сторони цієї категорії пов’язані з неправильним поводженням із конфіденційною інформацією.»
CWE-534	«DEPRECATED: Information Exposure Through Debug Log Files»	«Цей запис застарілий, оскільки його абстракція була надто низькою. Див. CWE-532»

CWE-769	«DEPRECATED: Uncontrolled File Descriptor Consumption»	«Цей запис застарілий, оскільки він був копією CWE-774. Весь вміст передано на CWE-774»
CWE-171	«CWE CATEGORY: Cleansing, Canonicalization, and Comparison Errors»	«Слабкі сторони цієї категорії пов'язані з неправильною обробкою даних у механізмах захисту, які намагаються нейтралізувати ненадійні дані.»
CWE-754	«Improper Check for Unusual or Exceptional Conditions»	«Програмне забезпечення не перевіряє або неналежним чином перевіряє наявність незвичних або виняткових умов, які, як очікується, не трапляються часто під час повсякденної роботи програмного забезпечення.»
CWE-17	«CWE CATEGORY: Code»	«Цей запис розглядається як застарілий.»
CWE-91	«XML Injection»	«Можливо, запис потрібно буде припинити або перетворити на загальну категорію»
CWE-358	«Improperly Implemented Security Check for Standard»	«Програмне забезпечення не реалізує або неправильно реалізує одну або декілька перевірок, що мають відношення до безпеки, як зазначено в розробці стандартизованого алгоритму, протоколу або техніки.»
CWE-444	«Inconsistent	«Коли неправильно сформовані

	Interpretation of HTTP Requests ('HTTP Request Smuggling')»	або аномальні HTTP-запити інтерпретуються однією або кількома сутностями в потоці даних між користувачем та веб-сервером, наприклад проксі-сервером або брандмауером, вони можуть трактуватися непослідовно, дозволяючи зловмисникові "переправити" запит на один пристрій без того, щоб інший пристрій про це знав.»
CWE-682	«Incorrect Calculation»	«Програмне забезпечення виконує обчислення, що генерує неправильні результати, які згодом використовуються для прийняття важливих для безпеки рішень або управління ресурсами.»
CWE-417	«CWE CATEGORY: Channel and Path Errors»	«Цей запис розглядається як застарілий.»
CWE-346	«Origin Validation Error»	«Програмне забезпечення не перевіряє належним чином джерело даних або зв'язку.»

6.2 DEFECT DATABASE

Бази даних про дефекти використовувались для вимірювання результатів однокласової класифікації. На етапі навчання використовувались описи

вразливості NVD, а для прогнозування використовувались бази даних дефектів, які були використані в цій роботі.

Таблиця 6.3

Список баз даних звітів про помилки

Проект	Галузь	Дата початку	Дата завершення	∑
Chromium	«Веб-браузер під назвою Chrome»	30.08.2008	11.06.2010	41940
Wicket	«Компонентна веб-додаток для програмування на Java.»	20.10.2006	09.11.2014	1000
Ambari	«Веб-інтерфейс управління Hadoop, що підтримується RESTful API»	27.09.2011	08.08.2014	1000
Camel	«Механізм маршрутизації»	08.07.2007	18.09.2013	1000
Derby	«Реляційна система управління базами даних»	28.09.2004	17.09.2014	1000

Бази даних містять усі разом 351 звіт про дефекти, позначені безпекою, а решта - не пов'язані із безпекою, звичайні звіти про дефекти. Щоб переконатися, що звіти, пов’язані з безпекою, містять справжні ключові слова про вразливість, спеціаліст із безпеки вручну позначив ці

елементи, а 148 елементів - як потенційно вразливі. 200 випадково відібраних зразків на базу даних, звіти про дефекти, не пов'язані з безпекою, використовувались як набір для тестування, а поверх цього були додані потенційно вразливі елементи. В результаті тестові тести розміром 1148 повідомлень про дефекти були використані в однокласних класифікаційних експериментах. На рисунку 6.8 наведено приклад маркованого вручну дефекту, який є потенційно вразливим.

```
"Provide way to optionally enable two-way SSL for Server-Agent communication
The two-way SSL mechanism used during server-agent registration exists to prote
ct communication. This is useful in production environments but in typical 'fir
st use' or POC scenarios  having this level of security is not necessary. As we
ll  certificate generation can be problematic causing failures.We need to provi
de a way to make this mechanism optional:1) By default  ship with Server-Agent
Two-Way SSL off.2) At any time post install  a user should be able to turn on T
wo-Way SSL and turn it back off  etc.   "
```

Рисунок 6.8 - Приклад потенційно вразливої вади

6.3 Метрики

Показники, представлені в цьому розділі, є популярними показниками для вимірювання та перевірки ефективності класифікації. Є чотири різні результати, при яких одинарна та бінарна класифікація може призвести до класифікації дефектів [Witten & Frank, 2005, с. 162]:

- Справжній позитив (ТП) правильно класифікує потенційно вразливий дефект.
- Справжній негатив (TN) правильно класифікується як нормальний дефект.
- Хибнопозитивний (FP) неправильно класифікує потенційно вразливий дефект.
- Помилково негативний (FN) - неправильно класифікований нормальний

дефект.

В оптимальному випадку всі прогнози класифікуються правильно, і не мають неправильно класифікованих зразків. Гейтс і Тейлор [2007] стверджували, що система може бути абсолютно марною через велику кількість помилкових спрацьовувань, навіть якщо знайдені всі правильно класифіковані зразки. Навіть незважаючи на те, що їх дослідження стосувалось виявлення аномальних вторгнень, тут застосовується та сама аналогія: коефіцієнт хибнопозитивних результатів не повинен бути вище 1%.

Результати класифікаційних заходів можуть бути використані для обчислення дійсного негативного показника (TNR або специфічність), істинного позитивного коефіцієнта (TPR, відкликання чи чутливість), коефіцієнта хибнопозитивного (FPR), точності, точності, оцінки F1 (F1), G -оцінка (G), коефіцієнт кореляції Меттьюса (MCC) та площа під кривою робочої характеристики приймача (AUC), як показано нижче:

$$TNR = \frac{TN}{TN + FP}$$

$$TPR = recall = \frac{TP}{TP + FN}$$

$$FPR = \frac{TP}{FP + TN}$$

$$precision = \frac{TP}{TP + FP}$$

$$\text{accuracy} = \frac{TN + TP}{TN + FP + TP + FN}$$

$$F1 = \frac{2 * recall * \text{precision}}{recall + precision}$$

$$G = \sqrt{precision * recall}$$

$$MCC = \frac{TP * TN - F * FN}{\sqrt{(TP + F)(TP + F)(TN + FP)(TN + F)}}$$

$$\text{AUC} = \int_{x=0}^{1} TPR\left(FPR^{-1}(x)\right)dx$$

Точність, частка правильних класифікацій серед усіх класифікацій, є дуже простим та інтуїтивно зрозумілим заходом, але при класифікації та виявленні аномалій це може призвести до занадто оптимістичних результатів. У цьому сенсі відкликання та точність є кращими мірками для опису продуктивності.

Оскільки переслідується одне значення продуктивності, оцінка F1 є гармонічним засобом точності та відкликання, що підходить для двійкової та багатокласової класифікації. F1-бал досягає свого найкращого значення при 1 і найгіршого при 0. G-бал - це середнє геометричне значення точності та відкликання. Однак у літературі показник F1 є більш поширеним. F1-бал не враховує дисбаланс класу, і випадкове значення F1-балу може змінюватися. Щоб зробити результати порівнянними з іншими дослідженнями, MCC та AUC можуть бути кращими варіантами.

MCC - це показник якості класифікації, який враховує справжні та хибні позитивні та негативні сторони, і, як правило, розглядається як збалансований показник, який можна використовувати, навіть якщо класи мають дуже різні розміри. Коефіцієнт +1 являє собою ідеальний прогноз, 0 не кращий за випадковий, а -1 означає повну незгоду. Крива робочої характеристики приймача - це графічний графік TPR проти FPR, який ілюструє діагностичну здатність двійкового або унарного класифікатора, оскільки поріг його дискримінації змінюється. AUC - це площа під кривою.

Значення AUC коливається від 0 до 1. Випадкове значення становить 0,5. AUC є широко використовуваним показником ефективності для класифікації, але не має автоматичного розширення для багатокласового випадку. MCC може бути використаний як показник ефективності в багатокласових задачах. [Hand,

2009; Jurman et al., 2012]. Бібліотека Scikit-learn також встановлює обмеження щодо використання метрик: Параметр оцінки методу cross_val_score підтримує обмежений набір параметрів вимірювання.12 Scikit-learn (2019) надає опції для усереднення зважування кожного класу в багатокласовій класифікації. У таблиці 6.4 представлені варіанти усереднення балів F1.

Таблиця 6.4

Варіанти усереднення балів F1

Опція	Опис
binary	«Результати лише для вказаного класу. Застосовується тільки, якщо цілі двійкові.»
micro	«Обчислює показники глобальні показники, підраховуючи загальну кількість справжніх позитивних, помилкових негативних і помилкових позитивних результатів.»
macro	«Обчислює показники для кожної мітки та знайходить їх незважене середнє значення.»
weighted	«Обчислює показники для кожної мітки та знаходить їх середнє значення, зважене за кількістю справжніх екземплярів для кожної мітки. Це може призвести до оцінки F, яка не відрізняється від точності та відкликання.»

samples	«Обчислює показники для кожного екземпляра та знаходить їх середнє значення.»

У даній роботі унарна класифікація вимірюється за допомогою AUC-оцінки, а багатокласова класифікація вимірюється за допомогою усередненої мікроскопії F1-оцінки. Результати можуть бути відображені у вигляді двовимірної матриці заплутанності з рядком і стовпцем для кожного класу. Кожен елемент матриці показує кількість тестових прикладів, для яких фактичним класом є рядок, а передбачуваним класом - стовпець. Усі правильні передбачення розташовані по діагоналі таблиці, що дозволяє легко візуально перевірити таблицю на наявність помилок передбачення [Witten & Frank, 2005]. Матриці плутанини класифікації CVSS можна знайти в Додатку 3.

Перехресна перевірка - це стандартна методика перевірки статистичної моделі для оцінки того, як результати статистичного аналізу будуть узагальнюватися до незалежного набору даних. Мета полягає в тому, щоб перевірити, як прогнозована модель буде діяти на практиці. Результат перехресної перевірки є надійним лише у тому випадку, якщо набори даних для навчання та тестування є окремими, а окремий екземпляр даних належить набору даних для навчання чи тестування, але не обом. Наступні методи перехресної перевірки можна знайти в літературі [Arlot & Celisse 2010; Kohavi 1995; Kantardzic 2011]:

- **Holdout** - це метод, коли набір даних ділиться на дві частини. Один набір даних використовується для навчання, а інший для тестування. Співвідношення двох наборів даних може бути довільним, але зазвичай дві третини використовуються для навчання, а решта для тестування. Цей метод можна сказати про перехресну перевірку, лише якщо він був перевірений більше

одного разу. Потім використовуються середні результати кількох раундів.

- **K-fold cross-validation** - це метод, коли набір даних ділиться на K наборів даних однакового розміру. Один набір даних використовується для тестування, а решта для навчання. Перевірка виконується K разів, так що кожен частковий набір даних використовується для тестування на кожному раунді.
- **Leave-one-out** є особливим випадком K-кратного перехресної перевірки, оскільки K вибирається рівним зразкам набору даних. Один зразок використовується для тестування, а інші зразки - для навчання. Перевірка повторюється K разів. Цей метод обчислювально дорогий.

У цій дисертації був використаний метод десятикратного повторення. Scikit-learn забезпечує метод train_test_split для розділення набору даних. За замовчуванням дві третини даних використовуються для навчання. Крім того, у багатокласовій класифікації було використано 10-кратну перехресну перевірку. Метод cross_val_score Scikit-learn забезпечує параметр cv для реалізації цього.

Надвибіркова та недодискретизація є широко відомими методами усунення дисбалансу класів у наборах даних. При передискретизації точки даних класу меншості відтворюються, оскільки при недодискретизації береться лише частина точок даних класу більшості.

Існують методи синтезу точок даних на даних меншин. У цій роботі було обрано стратегію недодискретизації, оскільки передискретизація може призвести до наявності тих самих даних у процесі навчання та тестування, що надалі призводить до занадто оптимістичних результатів.

РОЗДІЛ 7
РЕЗУЛЬТАТИ

Цей розділ показує результати виявлення вразливості з баз даних про дефекти з відкритим кодом та оцінки CVSS2, оцінки CVSS3 та багатокласової класифікації CWE.

Мета виявлення вразливості - класифікувати, чи містить текст ключові слова, що використовуються в описах вразливостей, чи ні.

Результати багатокласової класифікації пояснюють, наскільки добре текст вразливості можна використовувати для оцінки результатів CVSS та слабкості CWE.

Набір даних, що використовується в експериментах, містить загалом 18755 вразливостей, що мають описи вразливостей з початку 2018 року до квітня 2019 року. Результати показують порівняння ефективності класифікації з різними класифікаторами та методами.

7.1 Виявлення вразливості

Було протестовано три різних алгоритми машинного навчання, які виявляють описи потенційних вразливостей із тексту, доступних у Scikit-learn: OneClassSVM, LOF та IsolationForest. Дані NVD використовувались при навчанні, а тексти дефектів - при прогнозуванні.

Результати представлені в таблиці 7.1, де наведено показник AUC кожного алгоритму. Також минулий час вимірювали під час експериментів.

Таблиця 7.1

Ефективність виявлення вразливості

Алгоритм	**AUC-бал**	**Час**
OneClassSVM	**0,682**	18.5 с
IsolationForest	0,5	8.6 с
LocalOutlierFacto	0,492	9.8 с

Експерименти проводились із параметрами кожного типового алгоритму. IsolationForest і LOF швидші, але продуктивність не краща, ніж випадкова. Для подальшого дослідження було обрано класифікатор OneClassSVM. Випробувались різні векторизатори з класифікатором, щоб знайти найбільш ефективну комбінацію. Також було застосовано класифікатор на основі ключових слів, щоб побачити різницю між підходом машинного навчання та простим пошуком тексту. Вихідний код для виявлення вразливості та класифікатор, заснований на ключових словах, знаходиться в Додатку 1.

Класифікатор використовує SciVit-learn CountVectorizer для внутрішнього представлення описів вразливостей, що містять слова. Випробовувалися різні комбінації N-грамів. Для класифікації тексту до вразливості він повинен містити принаймні два N-грамових маркери описів вразливості. Штампування та лематизація досліджувалися, щоб виміряти, яка техніка попередньої обробки дає найкращі показники. У таблиці 14 представлені результати різних комбінацій діапазонів N-грамів векторизатора в класифікаторі на основі ключових слів. Параметр min_df, мінімальна частота документа, залишився за значенням за замовчуванням.

Параметр змушує векторизатор ігнорувати терміни, частота документів яких суворо нижча заданого порогу. Показано показник AUC,

кількість функцій векторизатора та затрачений час.

Таблиця 7.2

Ефективність класифікатора на основі ключових слів із діапазонами N-грамів

N-грамовий діапазон	**AUC-бал**	**Кількість функцій**	**Час**
1-1	0,5039	8065	121.7 с
1-2	0,503	44187	124.5 с
1-3	0,50435	93456	127.3 с
2-2	**0,7402**	37200	123.1 с
2-3	0,74	84650	124.7 с
3-3	0,52	48041	127.7 с

Результати показують, що підрахунок 2-грамів і не врахування окремих слів дає набагато кращу продуктивність. Маючи додатково 3-грами, це збільшує кількість функцій, але не продуктивність. Тільки 2-грамові були обрані для подальших досліджень у класифікаторі на основі ключових слів. Класифікатор із стемінгом та лематизацією порівнювали з різними комбінаціями векторизаторів з OneClassSVM. Ці результати представлені на рисунку 7.1. Також показані різні розміри наборів даних у процесі навчання.

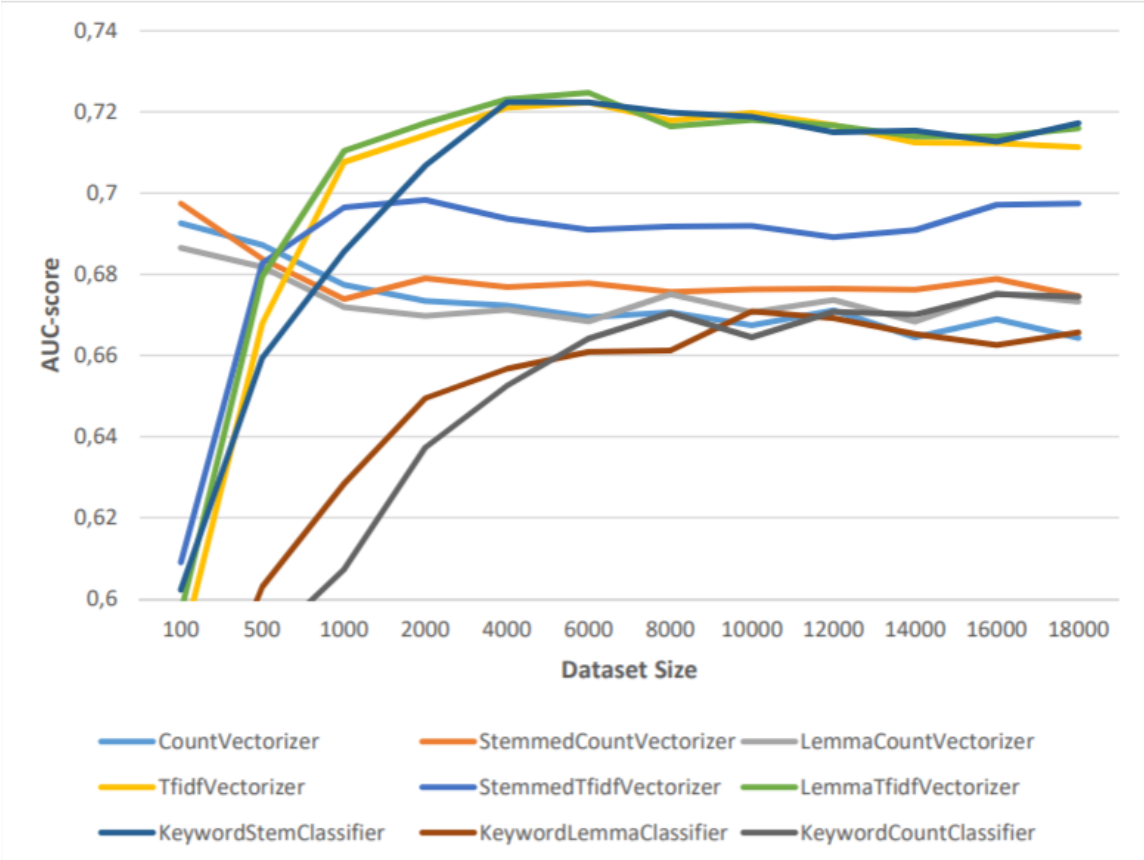

Рисунок 7.1 - Показник AUC OneClassSVM та розмір набору даних

Результати вказують, що використання 1000 - 4000 описів вразливості було б достатньо для досягнення стабільної продуктивності. Звичайний CountVectorizer та його методи зменшення особливостей дають найкращі результати при дуже низькій кількості зразків, але жодна з цих комбінацій не перевищує показник AUC 0,7. Класифікатор на основі ключових слів із простим CountVectorizer або лематизацією дає не кращі результати, ніж 0,68, але із стримуванням він дає 0,72 набору даних, що становить 4000 вразливостей. OneClassSVM працює краще із зважуванням TFIDF, ніж звичайне подання «bug-of-words».

OneClassSVM із стемінгом не перевищує показник AUC 0,7, але звичайний TFIDF працює майже так само добре, як TFIDF, при цьому лематизація досягає оцінки 0,72 при 4000 вразливостях. Векторизатор

TFIDF з лематизацією було обрано для подальших експериментів OneClassSVM.

У таблиці 7.3 показані показники AUC різних діапазонів N-грамів за допомогою класифікатора OneClassSVM з векторизацією та лематизацією TFIDF. Також представлено кількість функцій векторизатора та затрачений час.

Таблиця 7.3

OCSVM + TFIDF + лематизуюча ефективність з діапазонами N-грамів

N-грамовий діапазон	**AUC-бал**	**Кількість функцій**	**Час**
1-1	0,699	10401	14,3 с
1-2	**0,703**	59834	16,7 с
1-3	0,69	122703	19,1 с
2-2	0,566	49556	10,9 с
2-3	0,562	122686	13,6 с
3-3	0,536	64357	9,3 с

Результати показують, що використання окремих слів як ознак майже настільки ж добре, як 2-грами разом.

Кількість функцій значно збільшується, коли діапазон N-грамів збільшується, але не впливає на продуктивність.

У таблиці 7.4 наведено ефективність класифікатора на основі ключових слів із різними числами 2-грамових токенів.

Найкраща ефективність досягається, коли текст містить принаймні два вразливі 2-грамові маркери.

Таблиця 7.4

AUC-показник 2-грамового підрахунку класифікатора на основі ключових слів

	1	**2**	**3**	**4**	**5**
Класифікація на основі ключових слів	0,691	**0,731**	0,694	0,686	0,632

Векторизатори надають параметр min_df для відкидання слів, яких немає в мінімальній кількості документів. У таблиці 7.5 показано вплив різних параметрів min_df. Збільшуючи значення параметра, це негативно впливає на кількість функцій.

Також можна спостерігати ефект минулого часу OneClassSVM. Ефективність OneClassSVM та класифікатора на основі ключових слів є найвищою і має min_df = 1.

У Scikit є два загальних варіанти – навчитися знаходити оптимальну комбінацію гіперпараметрів класифікатора: вичерпно тестуючи всі можливі комбінації за допомогою GridSearchCV або випадковим чином вибираючи вибірку з заданої кількості кандидатів із простору параметрів за допомогою RandomizedSearchCV. У цій роботі було обрано вичерпний підхід.

В одинарній класифікації вичерпний пошук повинен бути реалізований за допомогою ParameterGrid, оскільки для передачі іншого методу даних до методу підгонки та методу прогнозування необхідний інший набір даних. На рисунку 16 показано результати вичерпного пошуку.

Таблиця 7.5

Вплив параметра мінімальної частоти документа

min_ df	**OCSVM AUC-бал**	**KeywordStem AUC-бал**	**OCSVM Кількість функцій**	**KeywordStem Кількість функцій**	**OCSVM Час**
1	**0,711**	**0,728**	58943	37028	17,2
2	0,708	0,705	15532	8217	13,8
3	0,703	0,69	8573	3995	12,9
4	0,702	0,671	5961	2545	13,7
5	0,688	0,657	4665	1963 рік	13,7
10	0,673	0,618	2425	887	11,6
20	0,688	0 587	1296	451	11,3
30	0,686	0,577	951	334	11,4
40	0,683	0,577	758	257	11,0
50	0,682	0,563	618	199	10,6

```
0.7334054054054056: {'clf__kernel': 'linear', 'clf__nu': 0.1, 'clf__shrinking': True, 'clf__tol': 1e-05}
0.7334054054054056: {'clf__kernel': 'linear', 'clf__nu': 0.1, 'clf__shrinking': True, 'clf__tol': 1e-06}
0.7334054054054056: {'clf__kernel': 'linear', 'clf__nu': 0.1, 'clf__shrinking': True, 'clf__tol': 1e-07}
0.7334054054054056: {'clf__kernel': 'linear', 'clf__nu': 0.1, 'clf__shrinking': False, 'clf__tol': 1e-05}
0.7334054054054056: {'clf__kernel': 'linear', 'clf__nu': 0.1, 'clf__shrinking': False, 'clf__tol': 1e-06}
0.7334054054054056: {'clf__kernel': 'linear', 'clf__nu': 0.1, 'clf__shrinking': False, 'clf__tol': 1e-07}
0.7319864864864865: {'clf__kernel': 'linear', 'clf__nu': 0.2, 'clf__shrinking': True, 'clf__tol': 1e-05}
0.7319864864864865: {'clf__kernel': 'linear', 'clf__nu': 0.2, 'clf__shrinking': True, 'clf__tol': 1e-06}
0.7319864864864865: {'clf__kernel': 'linear', 'clf__nu': 0.2, 'clf__shrinking': True, 'clf__tol': 1e-07}
0.7319864864864865: {'clf__kernel': 'linear', 'clf__nu': 0.2, 'clf__shrinking': False, 'clf__tol': 1e-05}
0.7319864864864865: {'clf__kernel': 'linear', 'clf__nu': 0.2, 'clf__shrinking': False, 'clf__tol': 1e-06}
0.7319864864864865: {'clf__kernel': 'linear', 'clf__nu': 0.2, 'clf__shrinking': False, 'clf__tol': 1e-07}
0.6864594594594594: {'clf__kernel': 'linear', 'clf__nu': 0.5, 'clf__shrinking': True, 'clf__tol': 1e-05}
0.6864594594594594: {'clf__kernel': 'linear', 'clf__nu': 0.5, 'clf__shrinking': True, 'clf__tol': 1e-06}
0.6864594594594594: {'clf__kernel': 'linear', 'clf__nu': 0.5, 'clf__shrinking': True, 'clf__tol': 1e-07}
0.6864594594594594: {'clf__kernel': 'linear', 'clf__nu': 0.5, 'clf__shrinking': False, 'clf__tol': 1e-05}
0.6864594594594594: {'clf__kernel': 'linear', 'clf__nu': 0.5, 'clf__shrinking': False, 'clf__tol': 1e-06}
0.6864594594594594: {'clf__kernel': 'linear', 'clf__nu': 0.5, 'clf__shrinking': False, 'clf__tol': 1e-07}
0.7339054054054055: {'clf__kernel': 'rbf', 'clf__nu': 0.1, 'clf__shrinking': True, 'clf__tol': 1e-05}
0.7339054054054055: {'clf__kernel': 'rbf', 'clf__nu': 0.1, 'clf__shrinking': True, 'clf__tol': 1e-06}
0.7339054054054055: {'clf__kernel': 'rbf', 'clf__nu': 0.1, 'clf__shrinking': True, 'clf__tol': 1e-07}
0.7339054054054055: {'clf__kernel': 'rbf', 'clf__nu': 0.1, 'clf__shrinking': False, 'clf__tol': 1e-05}
0.7339054054054055: {'clf__kernel': 'rbf', 'clf__nu': 0.1, 'clf__shrinking': False, 'clf__tol': 1e-06}
0.7339054054054055: {'clf__kernel': 'rbf', 'clf__nu': 0.1, 'clf__shrinking': False, 'clf__tol': 1e-07}
0.7324864864864866: {'clf__kernel': 'rbf', 'clf__nu': 0.2, 'clf__shrinking': True, 'clf__tol': 1e-05}
0.7324864864864866: {'clf__kernel': 'rbf', 'clf__nu': 0.2, 'clf__shrinking': True, 'clf__tol': 1e-06}
0.7324864864864866: {'clf__kernel': 'rbf', 'clf__nu': 0.2, 'clf__shrinking': True, 'clf__tol': 1e-07}
0.7324864864864866: {'clf__kernel': 'rbf', 'clf__nu': 0.2, 'clf__shrinking': False, 'clf__tol': 1e-05}
0.7324864864864866: {'clf__kernel': 'rbf', 'clf__nu': 0.2, 'clf__shrinking': False, 'clf__tol': 1e-06}
0.7324864864864866: {'clf__kernel': 'rbf', 'clf__nu': 0.2, 'clf__shrinking': False, 'clf__tol': 1e-07}
0.6864594594594594: {'clf__kernel': 'rbf', 'clf__nu': 0.5, 'clf__shrinking': True, 'clf__tol': 1e-05}
0.6864594594594594: {'clf__kernel': 'rbf', 'clf__nu': 0.5, 'clf__shrinking': True, 'clf__tol': 1e-06}
0.6864594594594594: {'clf__kernel': 'rbf', 'clf__nu': 0.5, 'clf__shrinking': True, 'clf__tol': 1e-07}
0.6864594594594594: {'clf__kernel': 'rbf', 'clf__nu': 0.5, 'clf__shrinking': False, 'clf__tol': 1e-05}
0.6864594594594594: {'clf__kernel': 'rbf', 'clf__nu': 0.5, 'clf__shrinking': False, 'clf__tol': 1e-06}
0.6864594594594594: {'clf__kernel': 'rbf', 'clf__nu': 0.5, 'clf__shrinking': False, 'clf__tol': 1e-07}
```

Рисунок 7.2 - Вичерпний пошук гіперпараметрів OneClassSVM

OneClassSVM з лінійним ядром показує найкращі результати, але з

радіальною базовою функцією результати дуже близькі. Менші значення верхньої межі помилок навчання та нижньої межі опорних векторів дають кращі результати. Крім того, вибір критерію зупинки не має значного впливу. Слід зауважити, що результати на рисунку 16 не є перехресно-перевіреними.

Останній експеримент намагається знайти, чи видалення імен CPE із наборів даних дає дещо кращі результати, як показано в таблиці 7.6. Результати вказують, що показник AUC вищий, якщо імена CPE включені як у OneClassSVM, так і в класифікатор на основі ключових слів.

Таблиця 7.6

Вплив видалення імен CPE на класифікацію AUC-бал

Назва CPE	Включено	Виключено
OneClassSVM	0,729	0,717
KeywordStemClassifie	0,732	0,718

7.2 CVSS та класифікація CWE

Було порівняно чотири різні алгоритми та техніки машинного навчання, класифікуючи оцінки CVSS та категорії CWE.

Алгоритми запропоновані Scikit-learn для класифікації тексту. У таблиці 7.7 класифікатори CVSS2 F1-бали представлені з різними векторизаторами. Відповідні результати для CVSS3 представлені в таблиці 7.8. У таблиці 7.9 відповідні результати представлені для класифікації CWE.

Таблиця 7.7

Класифікація балів CVSS2 за допомогою векторизаторів

	MultinomialNB	**Класифікатор SGD**	**LinearSVC**	**KN Сусіди**
CountVectorize	0,75	0,803	0,808	0,73
StemmedCount	0,748	0,803	0,807	0,724
LemmaCount	0,749	0,802	0,810	0,716
TFIDFVectorizer	0,677	0,804	**0,817**	0,718
StemmedTFIDF	0,675	0,803	0,815	0,715
LemmaTFIDF	0,678	0,805	**0,817**	0,716
Час w. TFIDF	16,5	15,4 с	17,5 с	30,5

Таблиця 7.8

Класифікація балів CVSS3 за допомогою векторизаторів

	MultinomialNB	**Класифікатор SGD**	**LinearSVC**	**KN Сусіди**
CountVectorize	0,702	0,782	0,792	0,672
StemmedCount	0,7	0,784	0,792	0,67
LemmaCount	0,704	0,784	0,794	0,671
TFIDFVectorizer	0,532	0,77	0,795	0,679
StemmedTFIDF	0,527	0,771	0,795	0,678
LemmaTFIDF	0,533	0,771	**0,796**	0,678
Час w. TFIDF	19-ті	19,6 с	21,8 с	40,9 с

Таблиця 7.9

Класифікація CWE з векторизаторами

	MultinomialNB	**Класифікатор SGD**	**LinearSVC**	**KN Сусіди**
CountVectorize	0,665	0,796	0,811	0,569
StemmedCount	0,653	0,791	0,808	0,556
LemmaCount	0,665	0,796	0,809	0,574
TFIDFVectorizer	0,541	0,804	**0,812**	0,642
StemmedTFIDF	0,529	0,802	0,811	0,631
LemmaTFIDF	0,544	0,805	0,811	0,643
Час w. TFIDF	48,4	74,9 с	137,8	56,3

Результати показують, що класифікатор LinearSVC з ваговим коефіцієнтом TFIDF дає найкращі показники, і жодних значних покращень не досягається шляхом стримування або лематизації в задачах класифікації CVSS2, CVSS3 та CWE. Scikit-learn забезпечує HashingVectorizer для створення відображення в пам'яті від рядкових маркерів до цілочисельних індексів об’єктів, але продуктивність була значно нижчою, ніж в CountVectorizers та TFIDFVectorizers. На рисунку 17 класифікатори SGDClassifier та LinearSVC порівнюються з різними розмірами набору даних.

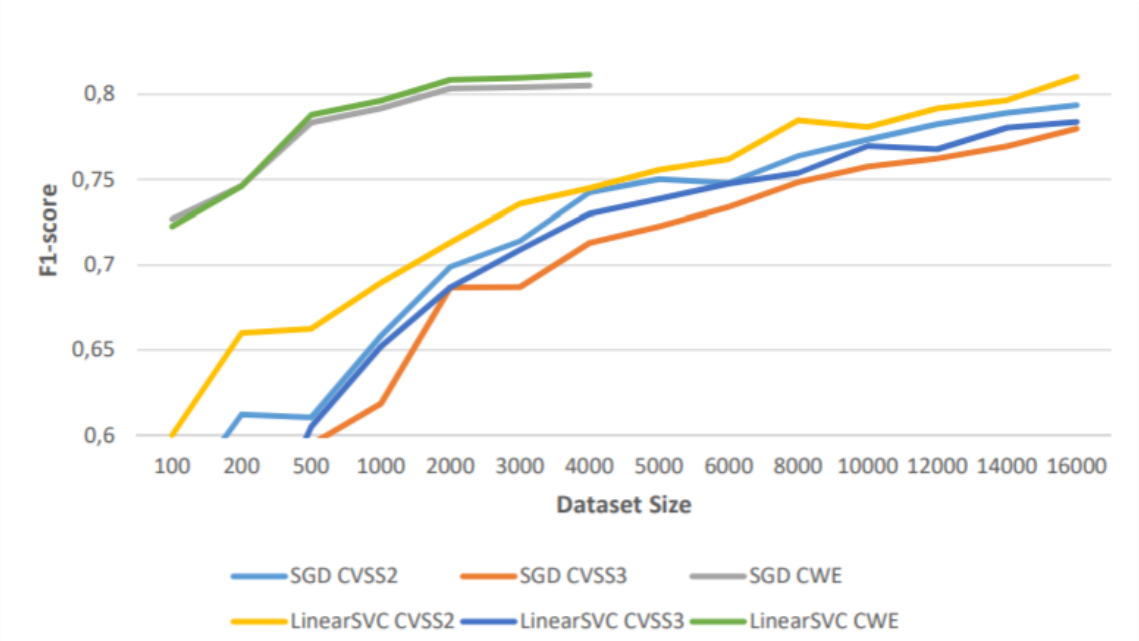

Рисунок 7.3 - Ефективність класифікатора з різними розмірами даних у класифікації CVSS2, CVSS3 та CWE

У класифікації CWE оцінка F1 досягає 0,8 при максимум 2000 зразків на визначений клас. У класифікації CVSS 0,8 F1-бал перевищується в класифікації CVSS2 лише з LinearSVC. На рисунку 17 розмір даних - це повний розмір набору даних в експериментах CVSS і передискретизований спеціально для кожного класу в класифікації CWE. Негативний ефект недодискретизації можна побачити на рисунку 7.3 щодо класифікації CWE та на рисунку 7.4 щодо класифікацій CVSS.

Як зазначено на рисунках 7.3 та 7.4, стратегія недовибірки не дає кращих результатів. Оцінка F1 варіюється від 0,87 до 0,975, але, поєднуючи всі показники з використанням категорій тяжкості, найвищий бал F1 можна отримати в CVSS2, маючи оцінку 0,81 та у CVSS3 0,783.

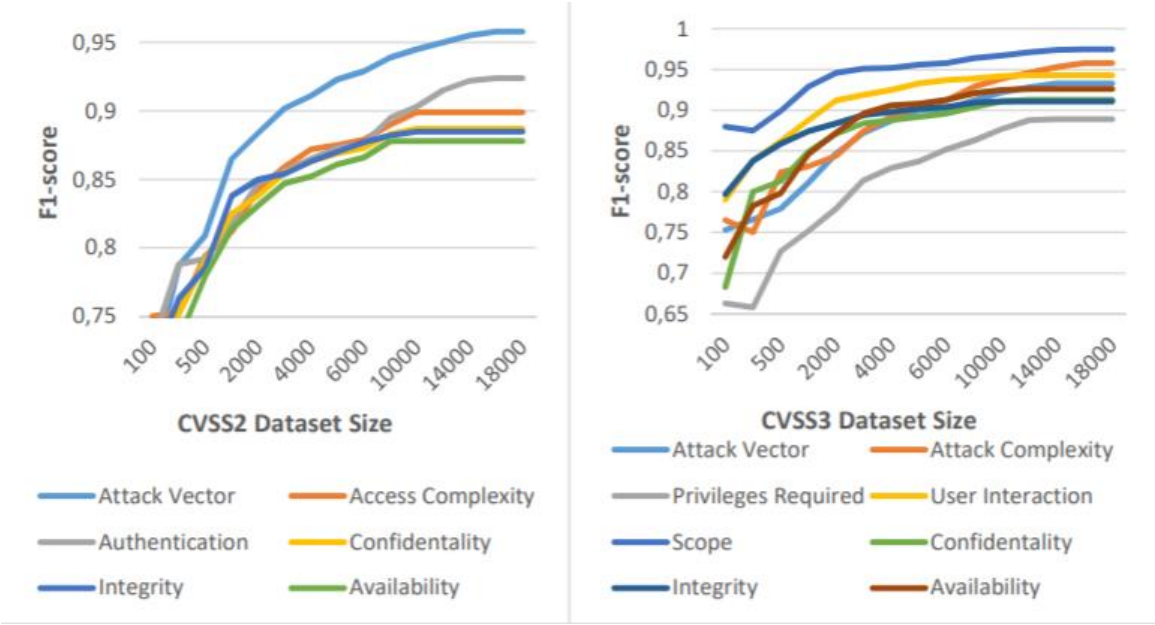

Рисунок 7.4 - Негативний ефект недодискретизації в метриках CVSS2 та CVSS3

Оскільки класифікатор LinearSVC із простим векторизатором TFIDF обраний для подальших експериментів, спостерігались різні діапазони N-грамів. У таблиці 22 представлені F1-бали класифікацій CVSS та CWE з різними представленими діапазонами N-грамів.

Таблиця 7.10

Ефективність класифікації CVSS та CWE з діапазонами N-грам

N-грамовий діапазон	CVSS2	CVSS3	CWE	**Кількість функцій**	**Час**
1-1	0,795	0,774	0,796	20059	30,8 с
1-2	0,814	0,792	**0,813**	135644	62,2 с
1-3	**0,817**	**0,795**	0,812	293891	105,1
2-2	0,805	0,772	0,787	114286	65,5 с
2-3	0,804	0,762	0,78	273832	129,2
3-3	0,787	0,723	0,71	159546	115,4

Дуже подібні результати в F1-балі можна отримати з 1-2 грамами, ніж 1-3 грами, але кількість функцій зростає, використовуючи більш широкі діапазони N-грамів. Щоб побачити вплив мінімальної частоти документів у багатокласовій класифікації, були проведені експерименти із значенням параметра df_min від 1 до 50. У таблиці 7.11 оцінки F1 із підрахунком особливостей та минулим часом показані в класифікації CVSS2, CVSS3 та CWE. Параметр df_min пояснюється в таблиці 8 серед інших параметрів векторизатора.

Таблиця 7.11

Вплив параметра min_df у багатокласовій класифікації

min_df	**CVSS2**	**CVSS3**	**CWE**	**Кількість функцій**	**Час**
1	**0,816**	**0,797**	**0,814**	135136	70,4 с
2	0,815	**0,797**	0,811	42383	52,3 с
3	0,811	0,791	0,81	24060	46,3 с
4	0,807	0,789	0,809	16299	44,4 с
5	0,804	0,786	0,806	12478	42,1 с
10	0,791	0,775	0,803	6277	40,9 с
20	0,781	0,775	0,796	3340	39,5 с
30	0,777	0,758	0,792	2376	38,6 с
40	0,772	0,75	0,786	1858 рік	38,2 с
50	0,768	0,748	0,784	1561 рік	38,4 с

Найкраща ефективність у F1-балі досягається при менших значеннях параметра min_df. Однак великих наслідків для продуктивності при вищих значеннях мінімальної частоти документів немає, але кількість функцій значно зменшується. Час обробки також свідчить про незначне, але не значне зменшення. У більших наборах даних за рахунок зменшення

кількості функцій він може споживати менше енергії процесора.

Останнім експериментом було перевірити, чи видалення загальних імен перерахування платформ дає кращі результати у багатокласовій класифікації. Результати F1-балів наведені в таблиці 24 для класифікації CVSS2, CVSS3 та CWE.

Таблиця 7.12.

Вплив видалення імен CPE на багатокласову класифікаціюікація F1-бал

Імена CPE	**У комплекті**	**Виключено**
CVSS2	**0,818**	0,812
CVSS3	**0,798**	0,791
CWE	**0,814**	0,813

Результати вказують на дещо краще, але незначне збільшення продуктивності F1-балу, якщо імена CPE не видаляються під час процесу видалення стоп-слів. Результати дуже схожі в одинарній та багатокласовій класифікації.

Налаштування гіперпараметрів було виконано для LinearSVC за допомогою GridSearchCV.

ВИСНОВКИ

Машинне навчання та прийоми природної мови були протестовані в одинарній та багатокласовій класифікації. В одинарній класифікації перевірялися однокласні алгоритми класифікації та виявлення аномалій. Алгоритми були обрані, за допомогою рекомендації Scikit-learn. Бібліотека пропонує векторизатори для перетворення лексем слів у числові матриці.

Комбіновані векторизатори та алгоритмові гіперпараметри мають безліч варіантів, і в цій роботі варіанти були ретельно вивчені. Порівняно нові дані про вразливість NVD використовувались як навчальні дані однокласних класифікаторів, а також як набори даних багатокласових класифікаційних експериментів.

Класифікатор, заснований на ключових словах, був застосований для порівняння рішення немашинного навчання з однокласними класифікаторами. Для перевірки класифікаторів використовувались бази даних про дефекти.

Метою одинарної класифікації було виявлення потенційних вразливостей за короткими текстовими описами. Багатокласові класифікаційні експерименти були розділені, передбачаючи класи CVSS2 та CVSS3, а також слабкі сторони CWE. Оцінки CVSS складаються з метрик використання та впливу, які утворюють відмінні класи для обчислення фактичного балу CVSS.

Для порівняння результатів остаточної класифікації оцінки були зіставлені в чотирьох рівнях важкості. Існують сотні різних класів CWE, які описують основні причини типів слабкості.

Для вимірювання одинарної класифікації було обрано AUC-бал, а для вимірювання ефективності мультикласифікації - мікрооцінку F1. Всі експерименти були перевірені за допомогою 10-кратного методу перехресної перевірки. Класифікатор OneClassSVM явно перевершує LocalOutlierFactor та IsolationForest. Функція радіального базису працює так само, як лінійне ядро і

зважування TFIDF, що працюють краще з OneClassSVM.

Вразливості можна виявити з тексту, використовуючи також рішення на основі ключових слів. Класифікатор на основі ключових слів найкраще працює з використанням 2-грамових ключових слів вразливості із стемінгом. Оцінка AUC в одинарній класифікації досягла найвищого значення 0,73. Деякі одинарні класифікаційні експерименти з різними наборами даних можна детальніше розглянути в Додатку 2.

Результати показують, що виявлення вразливості з тексту є складним і результати далеко не ідеальні. У багатьох експериментах генерація помилкових тривог виникала дуже часто. Класифікатори отримують стабільний рівень, використовуючи від 4000 до 6000 описів вразливості в якості навчальних даних. У багатокласовій класифікації класифікатор LinearSVC перевершує інші перевірені класифікатори: багаточленний класифікатор Байєса, Stochastic Gradient Descent та Nearest Neighbors. Класифікатор із зважуванням TFIDF дає трохи кращі результати, ніж звичайне представлення «bug-of-words».

Результати показують, що класифікатори працюють краще, використовуючи повний набір даних без застосування стратегії вибірки. F1-бал отримав 0,82 в класифікації CVSS2 та CWE, але в CVSS3 оцінка трохи нижче. Хан та ін. [2017] отримали F1-бал 0,816 ± 0,052 у своєму дослідженні, класифікуючи оцінки CVSS2, використовуючи вбудовування слова + 1-шарові методи CNN. У їх дослідженні негативний ефект недодискретизації схожий на цю роботу. Вони виявили, що збільшення навчальних даних покращує продуктивність моделі в цілому, але має більший вплив на менші набори навчальних даних, а покращення продуктивності стає меншим та повільнішим із збільшенням набору навчальних даних.

У класифікації CVSS викликом є кілька метрик для класифікації. У CVSS3 існує 8 показників для класифікації, які всі повинні мати високі показники, щоб отримати ефективні результати. Завдання класифікації CWE полягає у великій кількості відмінних класів. Методи полягають у зменшенні флективних

варіацій у словах, що зменшує кількість ознак. Зменшуючи кількість функцій, він може споживати менше енергії процесора.

Моделювання зламів та атак може зіграти вирішальну роль у захисті ключових організаційних активів шляхом моделювання ймовірних методів атак у всіх векторах атак, надаючи пріоритетні вказівки щодо виправлення.

Розглянуті методи та створений програмний код здатний значно розширити можливості системи симуляції зламів і атак, за рахунок прогнозування та знаходження ланцюга вразливостей, на основі отриманих даних з використаних датасетів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Arlot, S., Celisse, A. (2010). A Survey of cross-validation procedures for model selection. Statistics surveys 4: 40-79.

2. Arnold, J., Abbott, T., Elhage, N., Thomas, G. and Kaseorg, A. (2009). Security impact ratings considered harmful. 12th workshop on Hot Topics in Operating Systems. http://web.mit.edu/tabbott/www/papers/hotos.pdf

3. Bellinger, C., Sharma, S., Japkowicz, N. (2012). One-Class versus Binary classification: Which and When?. 11th International Conference on Machine Learning and Applications. IEEE Computer Society. https://doi.org/10.1109/ICMLA.2012.212

4. Bozorgi, M., Saul, L., Savage, S., Voelker, G. (2010). Beyond Heuristics: Learning to Classify Vulnerabilities and Predict Exploits. Department of Computer Science and Engineering, University of California, SanDiego. http://cseweb.ucsd.edu/~saul/papers/kdd10_exploit.pdf

5. Breunig, M., Kriegel, H., Ng, R., Sander, J. (2000). LOF: Identifying Density-Based Local Outliers. ACM SIGMOD International Conference on Management of Data. DOI: 10.1145/342009.335388.

6. Camacho-Collados, J., Pilehvar, M. (2018). On the Role of Text Preprocessing in Neural Network Architectures: An Evaluation Study on Text Categorization and Sentiment Analysis. https://arxiv.org/pdf/1707.01780.pdf

7. CVSS2. (2007). A Complete Guide to the Common Vulnerability Scoring System Version

8. Forum of Incident Response and Security Teams. https://www.first.org/cvss/cvss-v2-guide.pdf

9. CVSS3. (2015). Common Vulnerability Scoring System v3.0: Specification Document. Forum of Incident Response and Security Teams. https://www.first.org/cvss/cvss-v30-specification-v1.8.pdf

10. Han, Z., Li, X., Xing, Z., Liu, H., Feng, Z. (2017). Learning to Predict Severity of Software Vulnerability Using Only Vulnerability Description. 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME). https://doi.org/10.1109/ICSME.2017.52

11. Hand, D. (2009). Measuring classifier performance: a coherent alternative to the area under the ROC curve. Springer Science+Business Media, LLC

12. Joachims, T. (1996). A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization. Proceedings of the Fourteenth International Conference on Machine Learning (ICML 1997), Nashville, Tennessee, USA, July 8-12, 1997

13. Joulin, A., Grave, E., Bojanowski, P., Mikolov, T. (2016). Bag of Tricks for Efficient Text Classification, Facebook AI Research. arXiv:1607.01759. https://arxiv.org/pdf/1607.01759

14. Jurman, G., Riccadonna, S., Furlanello, C. (2012). A Comparison of MCC and CEN Error Measures in Multi-Class Prediction. PLoS ONE 7(8): e41882. doi:10.1371/journal.pone.0041882

15. Kantardzic, M. (2011). Data mining: Concepts, models, and algorithms. John Wiley & Sons. IEEE Press. ISBN: 978-1-118-02912-1.

16. Kohavi, R. (1995). A Study of cross-validation and bootstrap for accuracy estimation and model selection. Proceedings of the 14th international joint conference on artificial intelligence - Volume 2, 1137-1143. IJCAI'95. Montreal, Quebec, Kanada: Morgan Kaufmann Publishers Inc. ISBN: 1-55860-363-8.

17. Lamkanfi, A., Demeyer, S., Soetens, Q. D., and Verdonck, T. (2011). Comparing mining algorithms for predicting the severity of a reported bug. Software Maintenance and Reengineering (CSMR).

18. Li, Z., Zou, D., Xu, S., Ou, X. Jin, H., Wang, S., Deng, Z., Zhong, Y. (2018).

19. VulDeePecker: A Depp Learning-Based System for Vulnerability Detection. Network and Distributed Systems Security (NDSS) Symposium 2018. 18-21 February 2018, San Diego, CA, USA. ISBN 1-1891562-49-5.

20. Liu, F., Ting, K., Zhou, Z. (2012). Isolation-based Anomaly Detection. ACM Transactions on Knowledge Discovery from Data 6(1):1-39.

DOI: 10.1145/2133360.2133363.

21. Manning, C., Surdeanu, M., Bauer, J., Finkel, J. Bethard, S. McClosky, D. (2014). The Stanford CoreNLP Natural Language Processing Toolkit. Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations, pages 55–60. Association for Computational Linguistics

22. Miyamoto, D., Yamamoto, Y., Nakayama, M. (2015). Text-Mining Approach for Estimating Vulnerability Score. Conference: 2015 4th International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS).

23. Nothman, J., Hanmin, Q., Yurchak, R. (2018). Stop Word Lists in Frr Open-source Software Packages. Proceedings of Workshop for NLP Open Source Software, pages 7–12. Association for Computational Linguistics.

24. Peters, P., Tun, T., Yu, Y., Nuseibeh, B. (2017). Text Filtering and Ranking for Security Bug Report Prediction. IEEE Transactions on Software Engineering. https://doi.org/10.1109/TSE.2017.2787653

25. Perera, P., Patel, V.M. (2018). Learning Deep Features for One-Class Classification.

26. arXiv:1801.05365. https://arxiv.org/pdf/1801.05365

27. Porter, M. (1980). An algorithm for suffix stripping. Program 14.3 (1980): 130-137. Ruohonen, J., Leppänen, V. (2018). Toward Validation of Textual Information

28. Retrieval Techniques for Software Weaknesses. Proceedings of the 29th

29. International Conference on Database and Expert Systems Applications (DEXA 2018), Regensburg, Springer, pp.~265—277. https://arxiv.org/abs/1809.01360

30. Sanguino, L. A. B., Uetz, R. (2017). Software Vulnerability Analysis Using CPE and CVE. arXiv:1705.05347. https://arxiv.org/abs/1705.05347

31. Satapathy, R., Guerreiro, C., Chaturvedi, I., Cambria, E. (2017). Phonetic-Based Microtext Normalization for Twitter Sentiment Analysis. 2017 IEEE International Conference on Data Mining Workshops

32. Schölkopf, B., Platt, J., Shawe-Taylor, J., Smola, A., Williamson, R. (2001). Estimating the Support of a High-Dimensional Distribution. Neural Computation. Volume 13 Issue 7, Pages 1443 – 1471. MIT Press Cambridge, MA, USA

33. Scikit, (2019, May 27). Scikit-learn user guide. Release 0.21.2. Scikit-learn developers. https://scikit-learn.org/stable/user_guide.html

34. Gates, C., Taylor, C. (2007). Challenging the anomaly detection paradigm: A provocative discussion. Proceedings of the 2006 workshop on new security paradigms, 21-29. NSPW ‘06. Germany: ACM

35. Tyo, J. P. (2016). Empirical Analysis and Automated Classification of Security Bug Reports. Lane Department of Computer Science and Electrical Engineering. Morgantown, West Virginia.
https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20160014477.pdf

36. Wang, S., Liu, Q., Zhu, E., Porikli, F., Yin, J. (2017). Hyperparameter selection of one- class support vector machine by self-adaptive data shifting. Pattern Recognition Volume 74, Pages 198-211. Elsevier Ltd. http://dx.doi.org/10.1016/j.patcog.2017.09.012

37. Wijayasekara, D., Manic, M., Wright, J., McQueen, M. (2012). Mining Bug Databases for Unidentified Software Vulnerabilities. 5th International Conference on Human System Interactions. https://doi.org/10.1109/HSI.2012.22

38. Wijayasekara, D., Manic, M., McQueen, M. (2014). Vulnerability identification and classification via text mining bug databases. IECON 2014-40th Annual Conference of the IEEE Industrial Electronics Society pp. 3612-3618.

39. Witten, I. H., Frank, E. (2005). Data mining: Practical machine learning tools and techniques.Morgan Kaufmann Series in data management systems. Morgan Kaufmann. ISBN: 9780120884070.

40. Wright, J., Larsen, J., McQueen, M. (2013). Estimating Software Vulnerabilities: A Case Study Based on the Misclassification of Bugs in MySQL Server. Idaho National Laboratory.

41. Yang, Y. (1998). An Evaluation of Statistical Approaches to Text Categorization. INRT Journal

ДОДАТОК А

ПРОГРАМНИЙ КОД

```
#

# Class to find parent and root cwe#

# Example:

# finder = CweFinder()

# root_cwe = finder.find_root_cwe(123)#

import pandas as pd
class CweFinder():

    def___init__(self):

        # CSV file available at https://cwe.mitre.org/data/csv/1000.csv.zip
        self.cwes = pd.read_csv('data/1000.csv', index_col=False)
        self.cwes['CWE-ID'] = self.cwes['CWE-ID'].values.astype(str)


    def find_parent_cwe(self, cwe_id):# method returns the parent cwe of given cwecwe
        = self.cwes[self.cwes['CWE-ID'] == cwe_id]

        if(cwe.empty):
            return ''

        cwe = cwe.iloc[0]['Related Weaknesses']if
        type(cwe) != str:

            return ''

        s = cwe.find('ChildOf:CWE ID:')#15 characters longe
        = cwe.find(':', s+15)

        if s == -1 or e == -1:
            return ''

        return cwe[s+15:e]


    def find_root_cwe(self, cwe_id):# method return the root cwe of given cweparent
        = self.find_parent_cwe(cwe_id)

        if len(parent) <= 0:
            return cwe_id

        else:

            return self.find_root_cwe(parent)



#

# LinearSVC hyperparameter tuning#

from sklearn.model_selection import GridSearchCV
ngram_s = 1

ngram_e = 2

df = 1

t = r'(?u)\b\w*[a-zA-Z]{3,}\w*\b'
vectorizer =

    TfidfVectorizer(stop_words=swds, ngram_range=(ngram_s, ngram_e), min_df=df, token_pattern=t)


classifier = LinearSVC()

pipe = Pipeline([('vect', vectorizer), ('cls', classifier)])


#hyperparameters
```

```
parameters = {'cls__loss': ('hinge', 'squared_hinge'),#default: squared_hinge'cls
              dual': (True, False),#default: True

              'cls__multi_class': ('ovr', 'crammer_singer'),#default: ovr'cls
              max_iter': (1000, 2000),#default: 1000

             }

gs = GridSearchCV(pipe, parameters, scoring='f1_micro', cv=10, error_score=np.nan)


for i in range(6):# 6 classes on cvss2 metrics

    gs = gs.fit(cvss2_texts[i]['text'], cvss2_texts[i]['label'])
    print(str(i)+':'+str(gs.best_score_))
    print(str(i)+':'+str(gs.best_params_))


#

# Vulnerability detection classifiers and vectorizers#

def run_test(vzr, cls):

    print(str(cls)[0:str(cls).find('(')] + ' ' + str(vzr)[0:str(vzr).find('(')])


    #get a new testing set

    mixed = vulns_common.get_mixed_dataset(reports['report'], 1000)


    scores = []

    t = time.time()

    for i in range(nfold): #n-fold cross val score
        predicted = pipe.predict(mixed['report'])

        #score = f1_score(y_true=mixed['security'], y_pred=predicted, average='micro') score
        = roc_auc_score(y_true=mixed['security'], y_score=predicted, average='micro')
        scores.append(score)


    scores = np.array(scores)

    print(str(nfold)+'-fold cross-validated roc-auc-score:' + str(scores.mean()) + '\n')print('Time
    taken: ' + str(round(time.time() - t, 1)) + 's')


for c in  classifiers:for v
    in vectorizers:

        pipe = Pipeline([('vect', v), ('clf', c)]) #get
        a new learning data before the fit method
        vuln_data = shuffle(nvd_vulns, n_samples=6000)
        vuln_descs = []

        for d in vuln_data:

            if not d[1].startswith('** REJECT'):#some descriptions are rejected by NVD
                vuln_descs.append(d[1])

        print('Dataset to fit:'+str(len(vuln_descs)))
        pipe = pipe.fit(vuln_descs)

        run_test(v, c)


#

# Keyword-based Classifier with stemming#

class StemmedCountVectorizer(CountVectorizer):def
    build_analyzer(self):

        self.stemmer = SnowballStemmer("english")

        analyzer = super(CountVectorizer, self).build_analyzer()


        return lambda doc: (analyzer(' '.join([self.stemmer.stem(word) for word in doc.split(' ')])))

class KeywordStemClassifier(BaseEstimator, ClassifierMixin):
```

```
    def__init__(self, min_ngrams=2, max_ngrams=2):
        self.min_ngrams = min_ngrams self.max_ngrams
        = max_ngrams

        unwanted_words                                                                                    =
['issue','defect','bug','fault','flaw','mistake','error','version','system','because','before','dispu
ted']

        stop_words = text.ENGLISH_STOP_WORDS#.union(cpe_names)stop_words
        = stop_words.union(unwanted_words)

        self.vectorizer = StemmedCountVectorizer(stop_words=stop_words,

                                                 lowercase=True, ngram_range=(min_ngrams,
                                                 max_ngrams),min_df=1,

                                                 token_pattern=r'(?u)\b\w*[a-zA-Z]{3,}\w*\b')


    def fit(self, raw_documents, y=None):
        self.vectorizer.fit(raw_documents)


        return self


    def predict(self, raw_documents, y=None):

        assert (len(self.vectorizer.vocabulary_) > 0), "You must call fit() before predicting data!"
        scores = self.score(raw_documents)


        predictions = []

        for count in scores:

            if count >= 2:#at least two vulnerability n-grams classified as security related
                predictions.append(1)

            else:

                predictions.append(-1)

        return np.array(predictions)


    def word_grams(self, words, min, max):s
        = []

        for n in range(min, max+1):

            for ngram in ngrams(words, n):

                s.append(' '.join(str(i) for i in ngram))
        return s


    def _score_single(self, tokens):
        count= 0

        for index, token in enumerate(tokens):

            if token in self.vectorizer.vocabulary_:
                count = count + 1

        return count


    def score(self, raw_documents, y=None):

        assert (len(self.vectorizer.vocabulary_) > 0), "You must call fit() before scoring data!"
        stemmer = SnowballStemmer("english")

        scores = []

        for index, row in enumerate(raw_documents):
            stems = []

            for word in row.split():
                stems.append(stemmer.stem(word))

            tokens = self.word_grams(stems, self.min_ngrams, self.max_ngrams)count
            = self._score_single(tokens)
```

```
        scores.append(count)
    return np.array(scores)
```

ДОДАТОК Б

ЕКСПЕРИМЕНТИ НА ВИЗНАЧЕННЯ ВРАЗЛИВОСТІ

Keyword-based класифікатор

```
Manually labeled vulnerability dataset: 148
              precision    recall  f1-score   support

          -1       0.00      0.00      0.00         0
           1       1.00      0.66      0.80       148

   micro avg       0.66      0.66      0.66       148
   macro avg       0.50      0.33      0.40       148
weighted avg       1.00      0.66      0.80       148

TN=0, FP=0, FN=50, TP=98
Vulnerability learning dataset:3979
              precision    recall  f1-score   support

               precision    recall  f1-score   support

        False       0.98      0.93      0.95       971
         True       0.09      0.24      0.13        29

    micro avg       0.91      0.91      0.91      1000
    macro avg       0.53      0.59      0.54      1000
 weighted avg       0.95      0.91      0.93      1000

 TN=902, FP=69, FN=22, TP=7
 Test on data/Ambari.csv. Rows: 1000

               precision    recall  f1-score   support

        False       0.98      0.93      0.95       971
         True       0.09      0.24      0.13        29

    micro avg       0.91      0.91      0.91      1000
    macro avg       0.53      0.59      0.54      1000
 weighted avg       0.95      0.91      0.93      1000

 TN=902, FP=69, FN=22, TP=7
 Test on data/Ambari.csv. Rows: 1000

                precision    recall  f1-score   support

         False       0.97      0.79      0.87       967
          True       0.04      0.28      0.07        32

      micro avg       0.77      0.77      0.77       999
      macro avg       0.51      0.54      0.47       999
   weighted avg       0.94      0.77      0.84       999

   TN=763, FP=204, FN=23, TP=9
   Test on data/Wicket.csv. Rows: 1000
```

OneClassSVM класифікатор

```
Manually labeled vulnerability dataset: 148
              precision    recall  f1-score   support

          -1       0.00      0.00      0.00         0
           1       1.00      0.73      0.84       148

   micro avg       0.73      0.73      0.73       148
   macro avg       0.50      0.36      0.42       148
weighted avg       1.00      0.73      0.84       148

TN=0, FP=0, FN=40, TP=108
Vulnerability learning dataset:5967
```

```
              precision    recall  f1-score   support

          -1       0.00      0.00      0.00         0
           1       1.00      0.79      0.88      5967

   micro avg       0.79      0.79      0.79      5967
   macro avg       0.50      0.39      0.44      5967
weighted avg       1.00      0.79      0.88      5967

TN=0, FP=0, FN=1281, TP=4686
Mixed dataset: 1148

              precision    recall  f1-score   support

          -1       0.95      0.71      0.81      1000
           1       0.27      0.73      0.39       148

   micro avg       0.71      0.71      0.71      1148
   macro avg       0.61      0.72      0.60      1148
weighted avg       0.86      0.71      0.76      1148

TN=706, FP=294, FN=40, TP=108
Test on data/Ambari.csv. Rows: 1000
              precision    recall  f1-score   support

       False       0.97      0.94      0.95       971
        True       0.05      0.10      0.06        29

   micro avg       0.91      0.91      0.91      1000
   macro avg       0.51      0.52      0.51      1000
weighted avg       0.95      0.91      0.93      1000

TN=910, FP=61, FN=26, TP=3
Test on data/Camel.csv. Rows: 1000
```

ДОДАТОК В

ЕКСПЕРИМЕНТИ КЛАСИФІКАЦІЇ CSVV

CVSS 2

```
Training data:13593 Testing data:4531
              precision    recall  f1-score   support

    critical       0.72      0.72      0.72       325
        high       0.68      0.71      0.70       728
         low       0.76      0.66      0.71       533
      medium       0.87      0.89      0.88      2945

   micro avg       0.82      0.82      0.82      4531
   macro avg       0.76      0.74      0.75      4531
weighted avg       0.82      0.82      0.82      4531

Confusion matrix:
[[ 235   39    2   49]
 [  47  517   13  151]
 [   1    5  352  175]
 [  45  194   96 2610]]
```

CVSS 3

```
Training data:13446 Testing data:4482
              precision    recall  f1-score   support

    critical       0.60      0.83      0.70       645
        high       0.81      0.81      0.81      2061
         low       0.51      0.40      0.45        50
      medium       0.90      0.78      0.84      1726

   micro avg       0.80      0.80      0.80      4482
   macro avg       0.71      0.70      0.70      4482
weighted avg       0.81      0.80      0.80      4482

Confusion matrix:
[[ 537  101    1    6]
 [ 274 1663    8  116]
 [   3    6   20   21]
 [  81  289   10 1346]]
```