

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА КІБЕРБЕЗПЕКИ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до випускної кваліфікаційної роботи магістра на тему:
«Детектор вторгнень для «розумного» будинку»

Виконав студент 2-го курсу, групи ЗКБм-22-1
спеціальності 125 «Кібербезпека»

 К.І. Пулеко

Керівник, доцент кафедри комп'ютерної інженерії та
кібербезпеки, кандидат технічних наук, доцент

 Ю.М. Россінський

Рецензент, професор кафедри комп'ютерної інжене-
рії та кібербезпеки, доктор технічних наук, доцент

 В.В. Воротніков

Житомир – 2023

Державний університет «Житомирська політехніка»
 Факультет інформаційно-комп'ютерних технологій
 Кафедра комп'ютерної інженерії та кібербезпеки
 Спеціальність 125 «Кібербезпека»
 Освітня програма «Кібербезпека»

ЗАТВЕРДЖУЮ

Завідувач кафедри комп'ютерної
інженерії та кібербезпеки

 А.Сфіменко

13 жовтня 2023 р.

ЗАВДАННЯ

на випускню кваліфікаційну роботу
студенту групи ЗКБМ-22-1

Пулеці Костянтину Ігоровичу

Тема роботи: Детектор вторгнень для «розумного» будинку
затверджена Наказом по університету від «13» жовтня 2023 р. № 572/с

Термін здачі студентом закінченої роботи – 11 грудня 2023 р.

Вихідні дані для роботи:

типова структура «розумного» будинку

Консультанти з випускної кваліфікаційної роботи із зазначенням розділів, що їх стосуються

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1			
2			
3			

Студент  К.І. Пулеко

Керівник  Ю.М. Россінський

Календарний план

№ з/п	Назва етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Постановка завдання	29.09.2023	Виконано
2	Пошук, огляд та аналіз літературних джерел	10.10.2023	Виконано
3	Обґрунтування актуальності теми роботи по розробці детектора вторгнень для «розумного» будинку	16.10.2023	Виконано
4	Аналіз проблем безпеки IoT-систем «розумних» будинків	17.10.2023	Виконано
5	Аналіз методів детектування вторгнень	23.10.2023	Виконано
6	Розробка методики побудови детектора вторгнень для «розумного» будинку	01.11.2023	Виконано
7	Розробка програми детектора вторгнень та його дослідження на моделях	13.11.2023	Виконано
8	Розробка програми детектора вторгнень та його дослідження на реальному наборі даних	28.11.2023	Виконано
9	Оформлення пояснювальної записки	28.11.2023	Виконано
10	Подача чернетки керівнику для перевірки	05.12.2023	Виконано
11	Доопрацювання роботи та перевірка керівником	06.12.2023	Виконано
12	Проведення попереднього захисту	09.12.2023	Виконано
13	Отримання відгуку керівника	11.12.2023	Виконано
14	Допуск завідувача кафедри	12.12.2023	Виконано
15	Подання роботи на рецензування	15.12.2023	Виконано
16	Захист		

Студент  К.І. Пулеко

Керівник  Ю.М. Россінський

РЕФЕРАТ

Випускна кваліфікаційна робота магістра присвячена вирішенню актуального завдання по розробці детектора вторгнень для «розумного» будинку.

Метою роботи є розробка методики побудови та дослідження програмного забезпечення детектора вторгнень здатного виявляти відхилення від нормальної роботи системи «розумний» будинок.

Об'єктом дослідження є технології виявлення вторгнень у «розумних» будинках. Предметом дослідження є методика побудови програмного забезпечення детектора вторгнень для «розумного» будинку з використанням методів машинного навчання.

У першому розділі проведено аналіз типових схем «розумних» будинків показав та загроз їх інформаційної безпеки. Показано, що для виявлення кіберзагроз доцільно розробити детектор вторгнень. Проведено аналіз підходів та методів які можуть застосовуватися для рішення задачі пошуку вторгнень.

У другому розділі на основі конвеєра машинного навчання розроблено методику побудови детектора вторгнень та обрано мову програмування та бібліотеку для її реалізації.

Третій розділ присвячено розробці та дослідженню варіантів детектора вторгнень.

Пояснювальна записка до випускної магістерської роботи містить 89 сторінок, 46 рисунків, 3 таблиці, 3 додатки та 34 літературних джерела.

КЛЮЧОВІ СЛОВА: «розумний» будинок, вторгнення в мережу, методи виявлення аномалій, модель класифікації, машинне навчання, нейронна мережа.

ABSTRACT

The master's degree thesis is dedicated to addressing the current challenge of developing an intrusion detector for a "smart" house.

The purpose of the work is to develop a methodology for building and researching the software of an intrusion detector capable of detecting anomalies/deviations from the normal operation of the "smart" house system.

The object of research is intrusion detection technologies in "smart" houses. The subject of the study is the method of building software for an intrusion detector for a "smart" house using machine learning methods.

In the first chapter, an analysis of typical schemes of "smart" houses was carried out and the threats to their information security were shown. It is demonstrated that developing an intrusion detector is essential for detecting cyber threats. An analysis of approaches and methods that can be used to solve the problem of finding intrusions has been carried out.

In the second chapter, based on the machine learning pipeline, the technique of building an intrusion detector is developed and the programming language and library for its implementation are selected.

The third section focuses on the development and exploration of intrusion detector variants. The explanatory note to the final master's thesis contains 89 pages, 46 figures, 3 tables, 3 appendices and 34 literary sources.

KEY WORDS: "smart" house, network intrusion, anomaly detection methods, classification model, machine learning, neural network.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ	6
ВСТУП	7
РОЗДІЛ 1. АНАЛІЗ ТЕОРЕТИЧНИХ АСПЕКТІВ ПОБУДОВИ ДЕТЕКТОРІВ ВТОРГНЕНЬ ДЛЯ «РОЗУМНОГО» БУДИНКУ	10
1.1. Обґрунтування актуальності теми роботи по розробці детектора вторгнень для «розумного» будинку	10
1.2. Аналіз типових проектів IoT-систем «розумних» будинків	13
1.3. Аналіз проблем безпеки IoT-систем «розумних» будинків	18
1.4. Аналіз методів детектування вторгнень	23
1.5. Постановка завдання	36
Висновки по 1 розділу	39
РОЗДІЛ 2. РОЗРОБКА МЕТОДИКИ ПОБУДОВИ ДЕТЕКТОРА ВТОРГНЕНЬ ДЛЯ «РОЗУМНОГО» БУДИНКУ	40
2.1. Розробка конвеєра машинного навчання для детектора вторгнень «розумного» будинку	40
2.2. Аналіз та вибір програмного забезпечення для реалізації детектора вторгнень	41
2.3. Методика побудови детектора вторгнень для «розумного» будинку	46
2.4. Критерії оцінки якості для детектора вторгнень	48
Висновки по 2 розділу	55
РОЗДІЛ 3. РОЗРОБКА ТА ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ ДЕТЕКТОРА ВТОРГНЕНЬ.....	56
3.1. Розробка програми детектора вторгнень та його дослідження на простих моделях	56
3.2. Дослідження детектора вторгнень на наборі даних	64
3.3. Рекомендації щодо вдосконалення детектора вторгнень	82
Висновки по 3 розділу	88
ВИСНОВКИ	89
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	90
ДОДАТКИ	93

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ABOD – Angle-based Outlier Detector;
AKNN – Average KNN;
AUC – Area Under The Curve
CBLOF – Cluster-based Local Outlier Factor;
CMS – Content Management System;
FB – Feature Bagging;
GMM – Outlier detection based on Gaussian Mixture Model;
HBOS – Histogram-base Outlier Detection;
IForest – Isolation Forest;
INNE – ансамбль найближчих сусідів для виділення аномалій;
IoT – Інтернет речей (Internet of Things);
KDE – Kernel Density Estimation;
KNN – K Nearest Neighbors;
LMDD – Linear Model Deviation-base outlier detection;
LOF – Local Outlier Factor;
LSCP – Locally Selective Combination;
MCD – Minimum Covariance Determinant;
OCSVM – One-class SVM;
PCA – Principal Component Analysis;
ROC – Receiver Operating Characteristics;
SQL – Structured Query Language;
МН – Машинне навчання;
МОВ – Метод опорних векторів;
ПЕОМ – персональна електронно-обчислювальна машина;
ПЗ – програмне забезпечення;
ШІ – Штучний інтелект.

ВСТУП

Актуальність теми. Незважаючи на значний прогрес в технологіях «розумного» будинку, проблеми кібербезпеки залишаються незмінними. І справа не тільки в питанні конфіденційності. Цифрова економіка поступово вторгається в повсякденне життя людей за допомогою розумних пристроїв, створюючи сприятливі умови для хакерів. Злом розумного будинку вважається серйозною проблемою, зокрема тому, що все більше людей працюють сьогодні віддалено. Забезпечення кібербезпеки має бути пріоритетом для виробників систем безпеки й відеоспостереження, оскільки навіть мінімальна вразливість може підірвати довіру клієнтів, яку компанії створюють за допомогою часу, грошей та зусиль [1].

Останнім часом ми переконалися, що навіть найнадійніші системи захисту не здатні захистити від атак комп'ютерні системи державних і комерційних установ. Одна з причин – у тому, що в більшості систем безпеки застосовують стандартні механізми захисту: ідентифікацію та аутентифікацію, механізми обмеження доступу до інформації згідно з правами суб'єкта і криптографічні механізми. Це традиційний підхід із своїми недоліками, як-то: незахищеність від власних користувачів-зловмисників, розмитість поділу суб'єктів системи на "своїх" і "чужих" через глобалізацію інформаційних ресурсів, порівняна легкість підбору паролів внаслідок використання їхнього змістового різновиду, зниження продуктивності і ускладнення інформаційних комунікацій внаслідок обмеження доступу до ресурсів організації. Звідси потреба в механізмах, які б, доповнюючи традиційні, уможливлювали виявлення спроб несанкціонованого доступу і інформували про це відповідальних за безпеку або реагували у відповідь. Важливо, щоб такі системи могли протистояти атакам, навіть якщо зловмисник уже був аутентифікований та авторизований і з формальної точки зору додержання прав доступу мав необхідні повноваження на свої дії [2].

Такі функції і виконують системи виявлення вторгнень (intrusion detection systems, IDS). Оскільки передбачити всі сценарії розгортання подій

в системі з активним "чужим" суб'єктом неможливо, слід або якомога детальноше описати можливі "зловмисні" сценарії або ж, навпаки описати "нормальні" і постулювати, що всяка активність, яка не підпадає під прийняте розуміння "нормальності", є небезпечною. Згідно з цим, IDS поділяються на системи, що реагують на відомі атаки – системи виявлення зловживань (misuse detection systems, MDS) і системи виявлення аномалій (anomaly detection systems, ADS), які реєструють відхилення еволюції системи від нормальної роботи.

Багато розумних пристроїв, таких як віртуальні помічники, розумні дверні дзвінки з IP камерами відеоспостереження, розумні термостати та всілякі охоронні датчики – НЕ розроблені з урахуванням найвищого рівня безпеки [3]. Через це вони, на жаль, стають цілями кіберзлочинців. Замість того щоб виконувати функцію, для якої були призначені, скомпрометовані пристрої служать примхам кіберзлочинців – дозволяючи тим красти такі дані, як банківські реквізити, історії переглядів та іншу конфіденційну інформацію. У деяких випадках ці дані можуть використовуватися для стимулювання загроз вимагачів. Тому **тема роботи**, що направлена на виявлення порушень у мережі шляхом розробки детектора вторгнень є **актуальною**.

Мета і завдання роботи. Метою роботи є розробка методики побудови та дослідження програмного забезпечення детектора вторгнень здатного виявляти відхилення від нормальної роботи системи «розумний» будинок.

Досягнення поставленої мети передбачає виконання таких **завдань**:

- проведення аналізу теоретичних аспектів побудови детекторів вторгнень для «розумного» будинку;
- проведення аналізу типових проектів IoT-систем «розумних» будинків та їх проблем безпеки;
- проведення аналізу відомих методів детектування вторгнень та шляхів їх реалізацій;
- розробки методики побудови програмного забезпечення детектора вторгнень для «розумного» будинку;

- реалізації програмного забезпечення детектора вторгнень;
- дослідження ефективності детектора вторгнень для «розумного» будинку;
- розробка рекомендацій щодо вдосконалення детектора вторгнень.

Об’єктом дослідження є процес виявлення вторгнень у «розумних» будинках.

Предметом дослідження є методика побудови програмного забезпечення детектора вторгнень для «розумного» будинку з використанням методів машинного навчання.

Методи дослідження. У кваліфікаційній роботі було використано такі методи: аналізу, виявлення аномалій, детектування вторгнень, машинного навчання, математичного моделювання.

Наукова новизна отриманих результатів полягає в такому:

вдосконалено методику побудови програмного забезпечення детектора вторгнень для «розумного» будинку на основі конвеєра машинного навчання для відомих методів класифікації та виявлення аномалій.

Практичне значення роботи полягає у такому:

- розроблена методика дозволяє реалізовувати програмне забезпечення детектора вторгнень для «розумного» будинку для багатьох відомих методів класифікації та виявлення аномалій, що надає можливість вибрати оптимальний для кожної конкретної задачі;
- надані пропозиції окреслюють перспективні напрямки вдосконалення детектора вторгнень.

Апробація результатів роботи. За матеріалами кваліфікаційної роботи зроблено дві доповіді на XIII Міжнародній науково-технічній конференції “Інформаційні комп’ютерні технології” (2023) та VI Всеукраїнській науково-технічній конференції: “Комп’ютерні технології: інновації, проблеми, рішення (2023) [33, 34].

РОЗДІЛ 1. АНАЛІЗ ТЕОРЕТИЧНИХ АСПЕКТІВ ПОБУДОВИ ДЕТЕКТОРІВ ВТОРГНЕНЬ ДЛЯ «РОЗУМНОГО» БУДИНКУ

1.1. Обґрунтування актуальності теми роботи по розробці детектора вторгнень для «розумного» будинку

Технологія «розумного» будинку вже давно перестала бути фантастикою. «Розумний» будинок (розумний дім/smart home, digital house) – система домашніх пристроїв, здатних виконувати дії і вирішувати певні повсякденні завдання без участі людини. Функціонально пов'язуються між собою усі електроприлади будівлі, якими можна керувати централізовано – з пульта дисплею. Прилади можуть бути під'єднані до комп'ютерної мережі, що дозволяє керувати ними за допомогою персонального комп'ютера (ПК) та надає віддалений доступ до них через Інтернет. Завдяки інтеграції інформаційних технологій у домашні умови, усі системи та прилади узгоджують виконання функцій між собою, порівнюючи задані програми та зовнішні показники (обстановку) [3].

Особливістю «розумного» будинку є керування за допомогою пульта, на якому людина може натиснути одну-єдину клавішу для створення певної обстановки. При цьому, сама система «розумного» будинку аналізує навколишню ситуацію та параметри усередині приміщення, та, керуючись власними висновками, виконує задані користувачем команди із відповідними налаштуваннями. Окрім того, електронні побутові прилади, встановлені у розумному будинку, можуть бути об'єднані у домашню мережу із виходом до Інтернету.

Основні функції «розумного» будинку (рис.1.1) [4]:

Надійна та проста у користуванні «розумна» система безпеки будинку, що включає систему охорони, відеонагляду та «розумних» замків.

Автоматична централізована корекція освітлення у залежності від часу доби та пересування людей по приміщенню (особливо важлива для тих, хто виховує дітей або доглядає за родичами похилого віку).

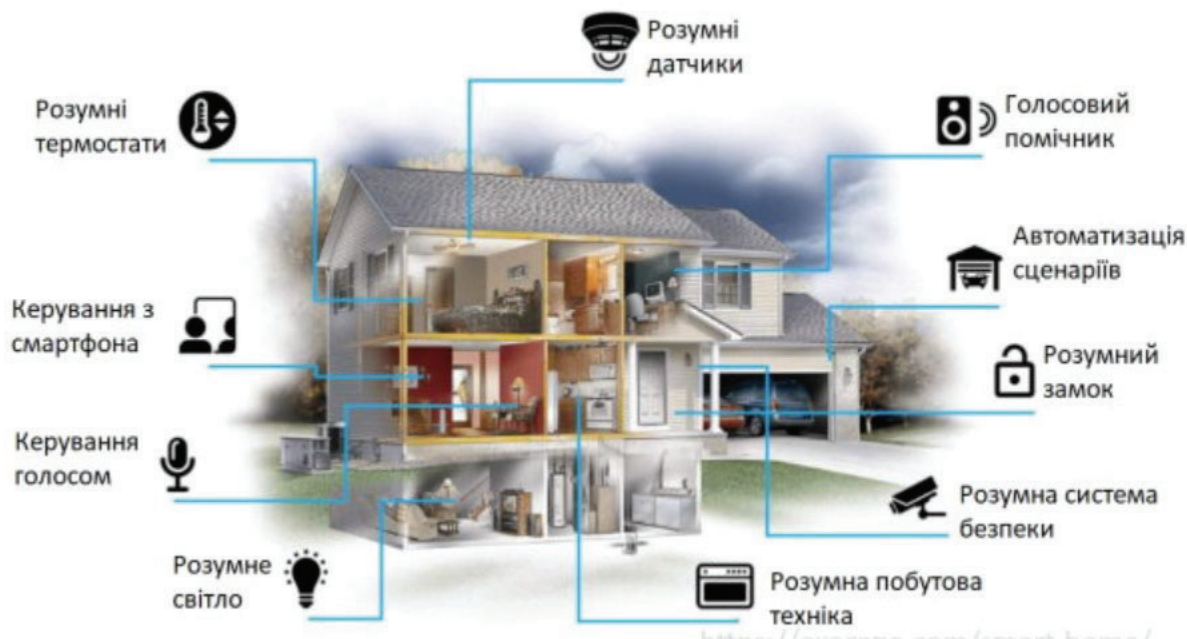


Рисунок 1.1 – Основні функції розумного будинку

Контроль за користуванням води та газу.

Орієнтація на енергозбереження. Розумний дім – це ще і енергоекономічна система. Домашня автоматика будинку дозволяє покращити умови життя та спростити побутові задачі для користування інвалідів та людей похилого віку.

Розумна побутова техніка має можливість керування через інтерфейс за допомогою мобільного зв'язку або Інтернет. Тобто, можливо робити якісь домашні справи через смартфон або веб-браузер, ще не діставшись самого будинку.

Усі функції управління виконуються за допомогою одного пульта-дисплея.

Зрозуміло, що така система «розумного» будинку має свої переваги та недоліки. Серед переваг слід відмітити такі:

Безпека. Система повністю контролює приміщення. Якщо був здійснений незапланований доступ, то вона надішле повідомлення про це. При виникненні надзвичайних подій «розумний» будинок спробує їх запобігти, в

тому числі виникнення пожеж. Він дозволяє спостерігати за тим, що відбувається в будинку з будь-якої точки світу [5].

Простота у використанні. Вся система управляється єдиним пристроєм. Найчастіше це смартфони.

Гнучкі налаштування. Система дозволяє підлаштовувати пристрої під себе, змінювати їх функції. Так само в неї можна додавати інші прилади в будь-який зручний час.

Економія. «Розумний» будинок знижує витрати за комунальні рахунки. Це пояснюється тим, що система відключає пристрої, які не використовуються в даний момент. Відповідно навантаження на мережу знижується, а разом з нею і споживання електроенергії. Економія на освітленні може досягати 40%, а на опаленні 30%.

Автоматизація. Велика частина побутових предметів може бути підключена до «розумного будинку». Відповідно це дає можливість управляти ними в автоматичному режимі. Це значно економить час.

Дизайн. Всі елементи системи (кнопки, термостати, датчики, розетки, вимикачі) виглядають сучасно і стильно вписуються в будь-який інтер'єр.

Можливість дистанційного включення техніки. Виробники розумної техніки створюють спеціальні програми, які потрібно встановити на смартфон для роботи зі смарт-приладом, де б ви не знаходилися.

Голосове управління. Вам навіть не доведеться вставати з дивана, щоб вимкнути світло, приготувати каву або відключити мультиварку.

Автоматичне оновлення. Смарт-пристрої мають модуль Wi-Fi, завдяки чому можуть оновлюватися автоматично, не вимагаючи втручання господарів.

Попередження надзвичайних ситуацій. У разі витoku газу, води або під час пожежі система швидко реагує і відправляє повідомлення на телефон.

До недоліків «розумного» будинку можна віднести такі [7]:

Ціна. Незважаючи на те, що побудова системи в основному опирається на примітивні датчики, камери, сенсори, її вартість дуже висока. Щоб окупи-ти «розумний» будинок потрібно мінімум 5 років.

Обслуговування. Як і будь-яка інша техніка, система може зламатися. У цьому випадку проблему можуть вирішити тільки досвідчені фахівці, яких поки що не дуже легко знайти. До того ж поломка однієї деталі може вивести з ладу і інші пристрої.

Установка. Система має бути дротова чи бездротова. Дротова вважається безпечнішою, але її потрібно встановлювати або відразу під час ремонту, або робити ремонт під неї.

Кібербезпека. Оскільки у переважній більшості системи розумного будинку є IoT системами, тобто мають канал виходу в Інтернет, то така система підлягає впливу усіх кіберзагроз що існують в мережі.

Сьогодні, коли управління «розумним» будинком здійснюється централізовано, а часто навіть віддалено, зловмисники можуть легко цим скористатися. Нове шкідливе програмне забезпечення, наприклад програм-вимагачів, вимагає у жертв викуп за повернення контролю за «розумним» будинком. Отримавши доступ до системи, зловмисники можуть керувати опаленням, кондиціонером, вентиляцією, пожежною сигналізацією, освітленням та навіть системою безпеки.

Перспективи розвитку таких загроз у майбутньому залежатимуть від дій правоохоронців та серйозності покарання зловмисників, кількості жертв, які готові платити кіберзлочинцям, та наявності потенційно цікавих для зловмисників цілей.

Тому тема роботи, що направлена на виявлення порушень у мережі шляхом розробки детектора вторгнень є актуальною.

1.2. Аналіз типових проектів IoT-систем «розумних» будинків

Система «розумний будинок» дозволяє узгодити роботу окремих елементів життєзабезпечення, налагодити моніторинг з використанням засобів Інтернету, за допомогою віддаленого відеоспостереження, встановити про-

грамну взаємодію компонентів з урахуванням часу дня, погодних умов, побажань господарів [8].

Типовий «розумний» будинок об'єднує і дозволяє здійснювати віддалене управління наступними підсистемами: опалення (радіатори та тепла підлога); клімат контроль; сигналізації систем охорони і захисту від пожежі; спостереження за аварійними ситуаціями; відеоспостереження; освітлювальні прилади всередині і зовні будівлі; відео і аудіопотік; енергоспоживання, дотримання лімітів по пікових навантажень; полив, каналізаційні, насосні станції; шлагбауми, ворота, ролокасети, жалюзі.

Для зручності управління будинком, IoT систему в ньому розділяють на декілька підсистем (рис. 1.2): управління освітленням, аудіо та відеоапаратурою, охоронною системою, захистом від надзвичайних ситуацій (протікання води, пожежа і т.д.), кліматичними системами (опаленням, кондиціонуванням) та іншими електроприладами.



Рисунок 1.2 – Основні підсистеми «розумного будинку»

Узагальнена структурна схема «розумного» будинку може бути подана у вигляді, як показано на рис. 1.3.

Система управління IoT - призначена для формування процесів управління підсистемами та формування і зберігання сценаріїв роботи системи у самому будинку.

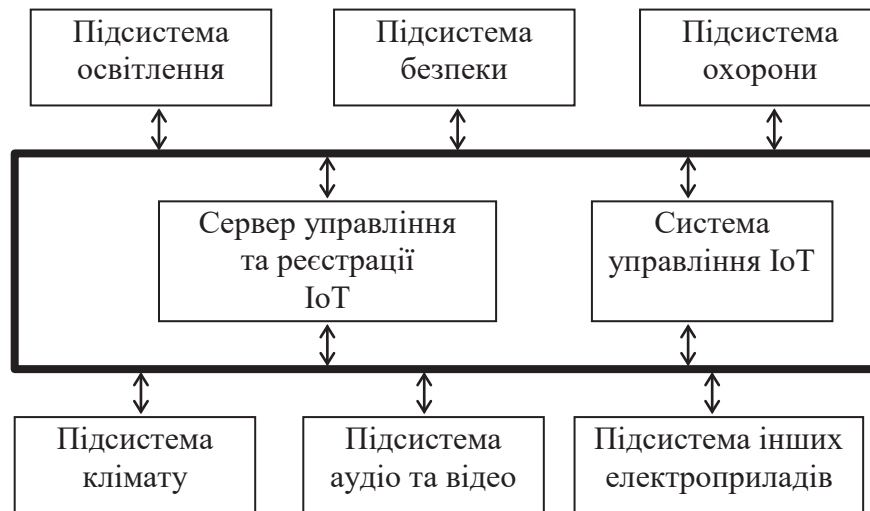


Рисунок 1.3 – Структурна схема IoT-системи для «розумного» будинку

Сервер управління та реєстрації IoT – призначений для віддаленого управління та реєстрації інформації від датчиків, фіксації подій. На сервері реєстрації IoT створюються облікові записи для контролю за всіма елементами в різних приміщеннях житлового будинку. Відповідно до рекомендацій безпеки такий сервер доцільно розгорнути у «хмарному» середовищі.

Підсистема освітлення. Управління світлом – це одна з найголовніших переваг системи «розумний будинок» і одна з самих затребуваних її функцій, бо саме світло ми використовуємо переважно у будинку і саме воно є невід’ємною частиною інтер’єру. Підсистема має сама управляти освітленням будинку: за заданим господарем сценарієм вмикаються і вимикаються лампочки та світильники, у визначений час запалюється вечірнє освітлення, вночі будинок сам вимикає світло у всіх кімнатах. Одним натисканням кнопки можна включити або виключити, відрегулювати будь-яке джерело світла в будинку. При цьому господарі можуть перебувати в будь-якій кімнаті будинку або навіть в іншому місті або країні, тому що управляти цією системою можна з мобільного телефону або за допомогою Інтернет [9].

Можна використати єдину панель управління світлом, що замінює всі вимикачі світла в будинку. Вона також забезпечує доступ до світла у всіх приміщеннях будинку (гараж, коридор, комора тощо), перебуваючи в одній

кімнаті можна керувати світлом у всіх інших приміщеннях будинку і в кожному з них окремо.

У разі відсутності господарів, освітлення працює таким чином, що створюється враження, що мешканці знаходяться вдома. Імітація присутності господарів – ви самі можете налаштувати систему так, щоб, коли вас немає вдома, вона управляла світлом, створюючи повну імітацію присутності господарів. Це особливо зручно, коли ви у відрядженні, відпустці або в тривалому від'їзді.

Управляти системою освітлення можна за допомогою заздалегідь заданих світлових сценаріїв, які зберігаються в пам'яті системи «розумного» будинку.

Підсистема безпеки призначена для забезпечення безпеки будинку та осіб що там проживають. Для захисту від пожежі передбачено датчики диму в кожній кімнаті. При спрацюванні вмикається сирена, відключається електроенергія, крім основних споживачів. Для захисту від газу передбачені датчики газу на кухні та в інших кімнатах. При їх спрацюванні вмикається сирена, перекривається клапан газу в квартирі, вмикається витяжна вентиляція [11].

Захист від протікання води можна реалізувати таким чином: у ванній та в кухні на підлозі розмістити датчики води. При потраплянні на них води, реле вмикає електромагнітний клапан і він перекриває центральну подачу води в квартирі. Відновлюється подача води в ручну з пульта, панелі, або натиснувши відповідну кнопку в системі управління.

Підсистема охорони. На дверях та вікнах розміщені геркони, які реагують на відкриття дверей. В залі багатофункціональний датчик стежить за рухом. При спрацюванні вмикається сирена, закривається електромагнітний замок на шафі з важливими речами та документами без можливості його відкриття поки будинок під охороною. При відкритті дверей можливе включення веб-камери для фіксації осіб, що входять.

Підсистема клімату призначена для управлінням кліматом в приміщеннях. Стабільність клімату досягається керуванням електричними теплими підлогами в приміщеннях, вентиляцією та кондиціонуванням. Система

управління дозволить встановлювати бажану температуру на будь-яку годину будь-якого дня тижня, а також керувати ними за будь-якими іншими зовнішніми ознаками (відключати при спрацюванні датчика диму, води). Якщо є обмеження на вхідну електричну потужність, то та ж система управління буде контролювати, щоб елементи теплої підлоги вмикались по чергово, а коли вмикається електрочайник – то вимкнуться нагрівальні кабелі підлоги.

Підсистема аудіо та відео призначена для управління мультимедією в квартирі. Наприклад, багатофункціональний датчик є також інфрачервоним передавачем, з пам'яттю для команд. Тобто він може керувати будь-яким приладом, який керується пультом. В даному випадку керуємо телевізором та музичним центром. Можна з однієї кнопки вмикати телевізор на улюбленому каналі на улюбленій гучності.

Підсистема інших електроприладів дозволяє включати техніку за сценарієм. Можна запрограмувати систему на включення або виключення будь-якого приладу в певний час.

Витяжка в ванній та на кухні, що може налаштовувати затримку часу, ввімкнення за певної умови (наприклад, ввімкнути витяжку на 2 хв, після закриття дверей в ванній).

Розетки. Керуючи розетками можна керувати будь-яким приладом, який в неї ввімкнений. Якщо ви ввімкнете в неї опалювальний прилад і відповідно запрограмуєте панель управління, то вона буде підтримувати задану вами температуру в приміщенні.

Електроконтактор в розподільчій коробці. Керуючи ним, ви керуєте всіма споживачами, які до нього під'єднані. Найчастіше до нього під'єднують всі розетки. Він може відключити всі розетки при спрацюванні датчика затоплення, щоб унеможливити потрапляння під дію електричного струму.

Електромагнітний замок – виконує функції безпеки.

Спеціальний електроклапан в ванній по команді, або по таймеру може набрати воду в ванній. Вимикатись пристрій може по датчику води, який розміщується зверху в ванній, або просто по таймеру.

Як бачимо з аналізу система «розумний» будинок цілком і повністю відноситься до IoT-систем, що у свою чергу призводить до наявності в ній усіх можливих проблем безпеки як і у будь-якої IoT-системи.

1.3. Аналіз проблем безпеки IoT-систем «розумних» будинків

Із попереднього аналізу слідує, що для управління «розумними будинками» надзвичайно критичною є вимога безпеки IoT-системи, адже успішні атаки означатимуть контроль зловмисника над приладами всередині будинку користувача, що є надзвичайно небезпечним та вкрай небажаним.

Безпека Інтернету речей – це сектор інформаційних технологій, який фокусується на захисті кінцевих пристроїв, мереж та даних, що стосуються Інтернету речей – підключених пристроїв, які не є комп'ютерами, смартфонами або планшетами. По суті, безпека Інтернету речей - це широкий термін, що охоплює стратегії, політики, процеси та технології безпеки, які компанії використовують для захисту своїх пристроїв Інтернету речей від інтелектуальних холодильників та камер відеоспостереження до моніторів на реактивних двигунах або автомобілях та пов'язаних з ними даних, додатків та мережі від злому або іншої шкоди.

Пристрої з підтримкою Інтернету створюють ряд проблем безпеки. Але хоча Інтернет речей дає можливість підключення до нових пристроїв, загальні проблеми кібербезпеки не є новими (рис. 1.4) [14].

Загрозами інформаційної безпеки «розумного будинку» є порушення конфіденційності, цілісності та доступності інформації.

Загрози конфіденційності - це загрози, які призводять до небажаного розкриття конфіденційної інформації. Під загрозами конфіденційної інформації прийнято розуміти потенційні чи реально можливі дії по відношенню до інформаційних ресурсів, що призводять до неправомірного оволодіння даними, які охороняються [19].

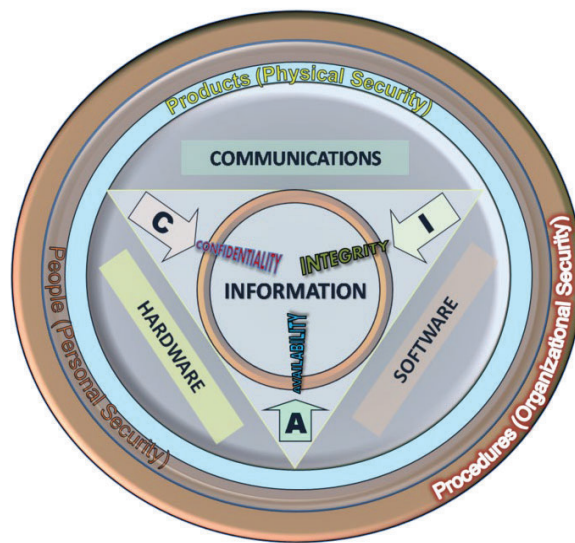


Рисунок 1.4 – Загальні види загроз IoT

Наприклад, порушення конфіденційності в системах домашнього моніторингу можуть привести до ненавмисного розкриття конфіденційних медичних даних. Навіть такі дані, як внутрішня температура будинку, можуть використовуватися для визначення того, зайнятий будинок чи ні. Втрата конфіденційності в таких речах, як ключі та паролі, призведе до несанкціонованих загроз доступу до системи.

Загрози доступу, ймовірно, є найбільшими загрозами. Несанкціонований доступ до системного контролеру, особливо на рівні адміністратора, робить всю систему небезпечною. Це може бути пов'язано з неправильним управлінням паролями і ключами або з неавторизованими пристроями, що підключаються до мережі. Навіть якщо контроль не може бути отриманий, несанкціоноване підключення до мережі може викрасти пропускну здатність мережі або привести до відмови в обслуговуванні законним користувачам. Оскільки багато пристроїв розумного будинку можуть працювати від акумуляторної батареї і мати бездротову мережу з низьким робочим циклом, переповнення мережі запитами може привести до атаки з виснаженням енергії - формі відмови в обслуговуванні.

До загроз доступу відноситься і слабка аутентифікація. Паролі є однією з перших ліній захисту від спроб злому. Але якщо ваш пароль недостатньо надійний, то і пристрій недостатньо захищений. Більшість паролів за замовчуванням є відносно слабкими, оскільки вони призначені для зміни, а в деяких випадках вони можуть бути загальнодоступними або зберігатися у вихідному коді програми (це надзвичайно ризиковано). Кінцеві користувачі також можуть встановити пароль, що легко запам'ятати. Але якщо його легко запам'ятати, то, ймовірно, легко проникнути.

Багато пристроїв IoT мають незначну аутентифікацію або взагалі не мають її. Навіть якщо на самому пристрої немає важливих даних, уразливий пристрій IoT може бути шлюзом до всієї мережі або його можна асимілювати в ботнет, де хакери можуть використовувати його потужність обробки для поширення шкідливого програмного забезпечення та розподіленої відмови в обслуговуванні (DDoS-атаки) [21].

Слабка автентифікація є серйозною проблемою безпеки IoT. Виробники можуть допомогти зробити автентифікацію більш безпечною, вимагаючи кількох кроків, використовуючи надійні паролі за замовчуванням та встановлюючи параметри, які ведуть до безпечних паролів, створених користувачами.

Суттєвою вразливістю є *доступність мережевої системи*. Оскільки сучасні системи «розумний будинок» підключені до Інтернету, атаки можуть проводитися дистанційно, або за допомогою прямого доступу до мережевих інтерфейсів управління, або шляхом завантаження шкідливих програм на пристрої.

Пристроєм Інтернету речей простіше використовувати ту саму мережу, що й інші пристрої кінцевого користувача, наприклад їхні Wi-Fi або LAN, але це також робить всю мережу більш вразливою. Хтось може зламати пристрій IoT, щоб отримати доступ до більш конфіденційних даних, що зберігаються в мережі чи інших підключених пристроях. Аналогічно, інший прист-

рій у мережі може бути використаний для злому пристрою IoT. У будь-якому з цих сценаріїв клієнти та виробники вказують один на одного пальцями. Тому в ідеалі кожна система IoT повинна використовувати окрему мережу та/або мати шлюз безпеки або брандмауер, тому, якщо на пристрої є порушення безпеки, він залишається ізольованим від пристрою. (Це одна з переваг стільникового Інтернету речей.) Віртуальна приватна мережа (VPN) допомагає захистити пристрої від зовнішньої мережі, але якщо ваша система має спільне з'єднання з іншими пристроями, вона все ще вразлива для атак з них, якщо вони пошкоджені.

Фізична доступність системи також є проблемою. Як для бездротових технологій, так і для операторів зв'язку по лініях електропередачі, до фізичних мереж можна отримати доступ зовні, навіть якщо сам будинок надійно заблокований [14].

Наступна вразливість - *обмежені системні ресурси*. Контролери пристроїв традиційно представляли собою невеликі 8-розрядні мікроконтролери з дуже обмеженими обчислювальними ресурсами і ресурсами збереження, що обмежувало їх здатність реалізовувати складні алгоритми безпеки.

Більшість додатків IoT використовують дуже мало даних. Це знижує витрати та подовжує термін служби батареї, але може ускладнити оновлення по повітрю і не дозволяє пристрою використовувати функції кібербезпеки, як-от брандмауери, антивірусні сканери та наскрізне шифрування. Це в кінцевому підсумку робить їх більш вразливими для злому. Саме тут важливо, щоб сама мережа мала вбудовані функції безпеки.

Фіксована прошивка - ще одна проблема. Існує дуже мало розумних побутових приладів, які надають будь-які регулярні послуги з оновлення програмного забезпечення для виправлення вразливостей.

Ще один з найбільших ризиків безпеки IoT – це якщо пристрої виходять на поле з помилкою, яка створює вразливості. Незалежно від того, чи надходять вони з вашого власного розробленого коду чи третьої сторони, ви-

робникам потрібна можливість випускати оновлення мікропрограми, щоб усунути ці ризики безпеки. В ідеалі це повинно відбуватися віддалено, але це не завжди можливо. Якщо швидкість передачі даних мережі занадто низька або вона має обмежені можливості обміну повідомленнями, можливо, вам доведеться фізично отримати доступ до пристрою, щоб запустити оновлення.

Системна неоднорідність також є вразливістю. Пристрої поставляються багатьма виробниками з різними мережевими стандартами і різними можливостями оновлення програмного забезпечення. Часто пристрої мають мало або взагалі не мають документації по своєму внутрішньому програмному забезпеченню, операційних системах і встановлених механізмах безпеки.

Застарілі пристрої в системі. Якщо система спочатку не була розроблена для підключення до хмари, вона, ймовірно, погано обладнана для боротьби з сучасними кібератаками. Наприклад, старі пристрої можуть бути не сумісними з новішими стандартами шифрування. Робити застарілі системи з підтримкою Інтернету, не вносячи суттєвих змін, ризиковано, але це не завжди можливо із застарілими пристроями. Вони з'єднувалися роками (можливо, навіть десятиліттями), що перетворює навіть невеликі покращення безпеки на монументальне починання [21].

Повільне впровадження стандартів – це також вразливість. Що стосується стандартів безпеки, то IoT є відкритий для всіх. Немає універсального стандарту для всієї галузі, а це означає, що всі компанії та ніші виробники повинні розробляти власні протоколи та рекомендації. Відсутність стандартизації ускладнює захист пристроїв Інтернету речей, а також ускладнює можливість міжмашинного зв'язку (M2M) без збільшення ризику.

У той час як деякі пропрієтарні системи, такі як підсистема моніторингу працездатності, можуть мати добре розроблену, відповідаючу стандартам безпеку, більшість сучасних пристроїв розумного будинку реалізують кілька, якщо взагалі застосовують, підходів до безпеки.

Відсутність шифрування. Однією з найбільших загроз для безпеки IoT є відсутність шифрування при регулярних передачах. Багато пристроїв IoT

(особливо старих) не шифрують дані, які вони надсилають, а це означає, що якщо хтось проникає в мережу, вони можуть перехопити облікові дані та іншу важливу інформацію, що передається на пристрій і з нього.

Суттєвою вразливістю є *брак професійних спеціалістів з безпеки*, які можуть керувати всіма складовими мережі «розумний будинок». Мало хто з домовласників може дозволити собі професійну допомогу в управлінні домашньої мережею [13]. Замість цього домовласники-любители повинні мати можливість самотійно, безпечно і надійно управляти своїми системами.

Таким чином, проблема виявлення атак та вторгнень є однією із найскладніших проблем кібербезпеки «розумного» будинку, що потребує свого вирішення.

Одним із можливих шляхів рішення цієї проблеми є розробка та впровадження детектора виявлення нестандартної поведінки мережі, тобто детектора вторгнень.

1.4. Аналіз методів детектування вторгнень

Виділення в даних нетипових, аномальних викидів є особливим завданням у машинному навчанні. Крім низки практичних застосувань (виявлення збоїв у показаннях датчиків, хакерських атак), це завдання є етапом побудови будь-якого алгоритму машинного навчання, у якому дані перевіряються на консистентність та очищаються від викидів і шуму.

У цій роботі будемо розглядати одну важливу і досить специфічну проблему навчання – завдання пошуку вторгнень (Anomaly Detection). А програмна реалізація такого пошуку вторгнень носить назву – детектора вторгнень.

Існує три широкі категорії методів виявлення вторгнень. *Методи контролюваного виявлення вторгнень* вимагають набору даних, позначеного як «нормальний» і «ненормальний», і включають навчання класифікатора. Однак цей підхід рідко використовується для виявлення вторгнень через загальну недоступність розмічених даних і притаманну класам незбалансованість.

Методи напівконтрольованого виявлення вторгнень припускають, що певна частина даних позначена. Це може бути будь-яка комбінація звичайних або аномальних даних, але найчастіше методи будують модель, що представляє нормальну поведінку, на основі заданого нормального набору даних навчання, а потім перевіряють ймовірність тестового екземпляра, який буде згенерований моделлю. *Методи неконтрольованого виявлення вторгнень* припускають, що дані не позначені (нерозмічені), і вони, безумовно, найчастіше використовуються через їх ширше та частіше застосування.

Строго кажучи, в аналізі даних є два напрямки, які займаються пошуком вторгнень: *детектування викидів (Outlier Detection)* та «новизни» (*Novelty Detection*). Як і викид "новий об'єкт" - це об'єкт, який відрізняється за своїми властивостями від об'єктів (навчальної) вибірки. Але на відміну від викиду, його в самій вибірці поки немає (він з'явиться через деякий час, і завдання якраз і полягає в тому, щоб виявити його з появою). Наприклад, якщо ви аналізуєте виміри температури та відкидаєте аномально великі чи маленькі, то Ви боретеся з викидами. А якщо Ви створюєте алгоритм, який для кожного нового виміру стану мережі оцінює, наскільки він схожий на минулі, і виділяє аномальні — Ви шукаєте новизну (по суті створюєте детектор вторгнень).

Викиди є наслідком:

- помилки у даних (неточності вимірювання, округлення, неправильного запису тощо);

- наявності шумових об'єктів (неправильно класифікованих об'єктів);

- присутності об'єктів «інших» вибірок (наприклад, показаннями датчика, що зламався).

Новизна, як правило, з'являється в результаті нової поведінки об'єкта. Скажімо, якщо наші об'єкти – опис роботи системи, то після проникнення в неї вірусу об'єкти стають «новизною». Тут важливо розуміти, що «новизна» називається новизною з тієї причини, що такі описи для нас абсолютно нові, а нові вони тому, що ми не можемо в навчальній вибірці мати інформацію про всілякі зараження вірусами або всілякі поломки. Формування такої нав-

чальної вибірки трудомістке і часто не має сенсу. Проте можна набрати досить велику вибірку прикладів нормальної (штатної) роботи системи чи механізму.

Варто зазначити, що можливих постановок завдань тут також багато. Наприклад, завдання Positive-Unlabeled Classification (PU learning) - це коли частина викидів позначена (клас 1), але в інших об'єктах навчання (клас 0) також можуть бути викиди. Наприклад, нам експерт сказав, що обладнання давало збій у такі моменти часу, але він міг помітити не всі збої.

Навіть коли завдання виявлення вторгнень схожі на звичайні завдання класифікації, є особливості, скажімо, дисбаланс класів (наприклад, поломки обладнання відносно рідкісні).

Розглянемо методи, які можуть бути застосовані для рішення цієї задачі.

1.4.1. Статистичні випробування (експерименти)

Як правило, застосовують для окремих ознак та відловлюють екстремальні значення (Extreme-Value Analysis). Для цього використовують, наприклад, Z-Value або Kurtosis measure [19].

$$\text{Z-value } Z_i = \frac{|x_i - \mu|}{\sigma} \quad \text{Kurtosis } \frac{1}{n} \sum_{i=1}^n Z_i^4 \quad (1.1)$$

Будь-який практик має якийсь свій перевірений спосіб знаходження екстремальних значень для певних типів даних. Багато методів візуалізації, наприклад ящик з вусами, мають вбудовані засоби для детектування та показу таких екстремальних значень (рис. 1.5).

Важливо розуміти, що екстремальне значення та аномалія – це різні поняття. Наприклад, у невеликій вибірці

[1, 39, 2, 1, 101, 2, 1, 100, 1, 3, 101, 1, 3, 100, 101, 100, 100]

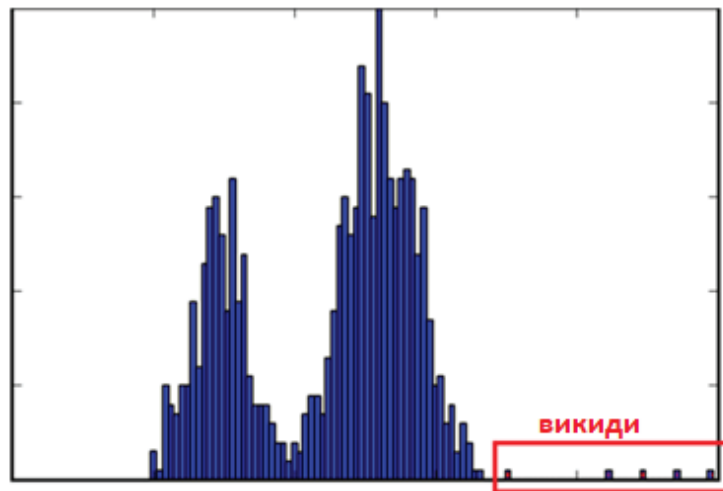


Рисунок 1.5 – Приклад візуалізації викидів

значення 39 можна вважати аномалією, хоча воно не є максимальним чи мінімальним. Також варто зазначити, що аномалія характеризується, як правило, не лише екстремальними значеннями окремих ознак (див. рис. 1.6).

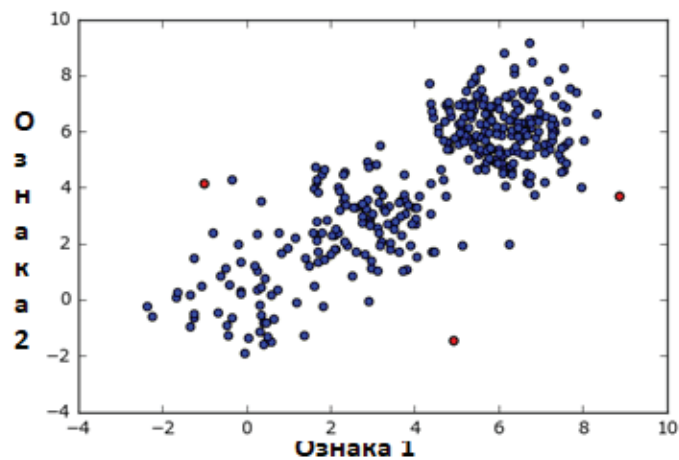


Рисунок 1.6 – Приклад викидів у задачі із двома ознаками

1.4.2. Модельні тести

Ідея дуже проста – ми будуємо модель, яка описує дані. Точки, які сильно відхиляються від моделі (на яких модель сильно помиляється) і є аномалії (див. рис. 1.7). При виборі моделі ми можемо зважити на природу завдання, функціонал якості тощо.

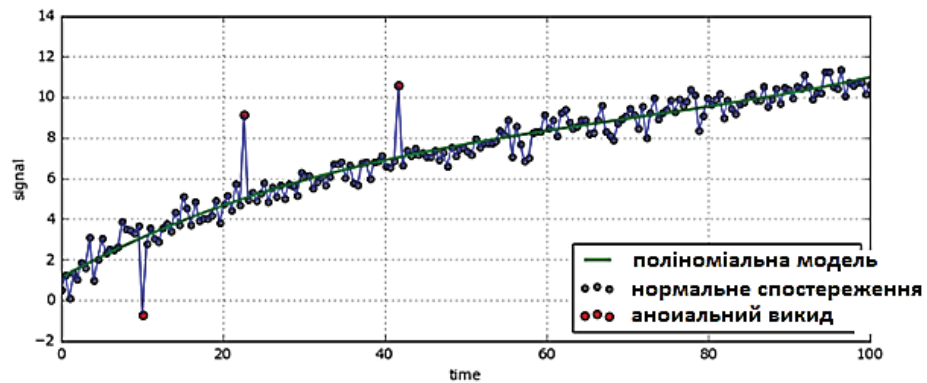


Рисунок 1.7 – Приклад викидів у часовому ряді

Такі методи хороші для визначення новизни, але гірше працюють у пошуку викидів. Дійсно, при налаштуванні моделі ми використовуємо дані, в яких є викиди (і вона під них «заточується») [25].

На рис. 1.8 показано застосування модельного підходу. Ми маємо матрицю і потрібно знайти в ній викиди. Ми використовуємо неповне сингулярне розкладання (SVD), щоб знайти матрицю невеликого рангу максимально схожу на нашу (для наочності всі числа округлені). Елементи, які сильно відрізняються від відповідних елементів матриці невеликого рангу, вважатимемо викидами.



Рисунок 1.8 – Застосування SVD для знаходження викидів у матриці

1.4.3. Ітераційні методи

Методи, які складаються з ітерацій, на кожній із яких видаляється група «особливо підозрілих об'єктів». Наприклад, у n-вимірному ознаковому просторі

можна виділяти опуклу оболонку наших точок-об'єктів, вважаючи її представників викидами (рис.1.9). Як правило, методи цієї групи досить трудомісткі [27].

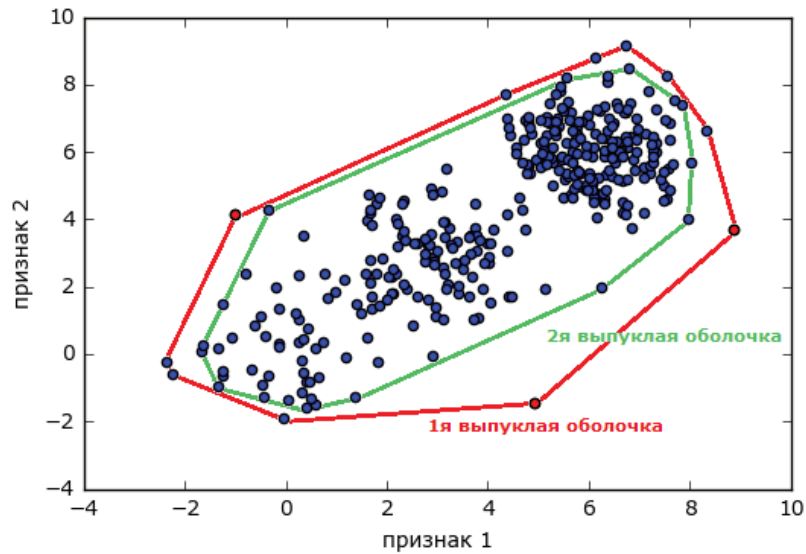


Рисунок 1.9 – Виуклі оболонки множини точок

1.4.4. Метричні методи

У них постулюється існування певної метрики у просторі об'єктів, що й допомагає знайти аномалії. Інтуїтивно зрозуміло, що у викида мало сусідів, а у типової точки багато. Тому хорошою мірою аномальності може бути, наприклад «відстань до k-го сусіда» (метод Local Outlier Factor (рис.1.10)).

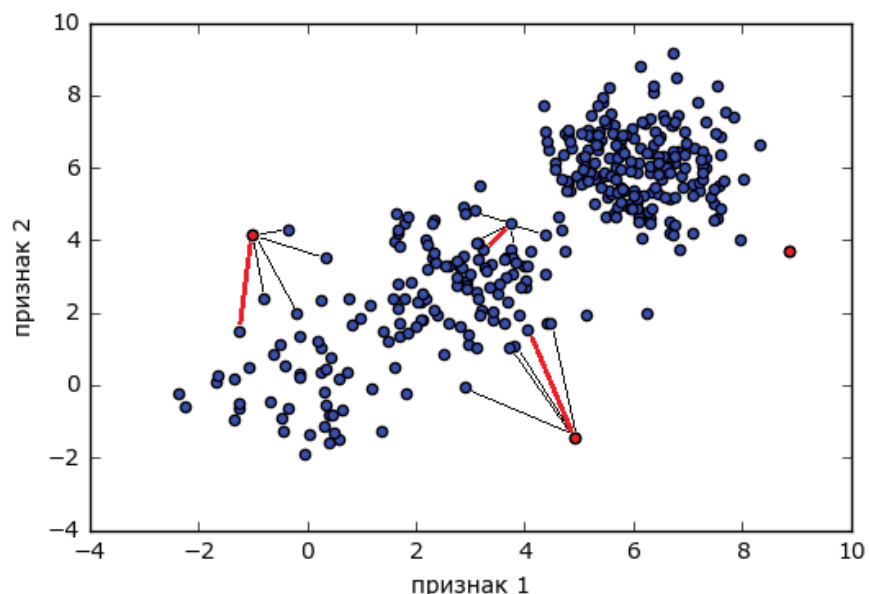


Рисунок 1.10 – Сусіди кількох елементів вибірки, зв'язок з 5-м показаний червоним

Метричні методи намагаються знайти у наборі даних такі точки, що у певному сенсі ізольовані від інших [7, 25]. Якщо у просторі задана деяка метрика $\rho(x1, x2)$, можна вводити такі подання аномальності:

- Аномалії – точки, які не потрапляють у жодний кластер. До даних застосовується один із алгоритмів кластеризації; Розмір кластера, в якому виявилася точка, оголошується її *anomaly_score*.

- Локальна щільність в аномальних точках низька. Для цієї точки *anomaly_score* визначається локальна густина, яка оцінюється деяким непараметричним способом (наприклад, ядерною оцінкою щільності Розенבלата - Парзена).

- Відстань від цієї точки до найближчих сусідів велика. Як *anomaly_score* може виступати:

- відстань до k-го найближчого сусіда;
- середня відстань до найближчих сусідів;
- медіана відстаней до найближчих сусідів;
- гармонійне середнє до k найближчих сусідів;
- частка з k найближчих сусідів, котрим дана точка не більше ніж k-им сусідом.

k-им сусідом.

Використання подібних алгоритмів з параметрами (наприклад, k) накладає вимогу наявності апріорної інформації про потенційні розміри кластерів вторгнень (рис. 1.11).

Такі кластери можуть залишитися не виявленими, якщо алгоритм розглядає невелику кількість найближчих сусідів. При великому числі сусідів мінімальне значення побудованої функції може бути далеко від підпростору нормальних даних (рис. 1.12).

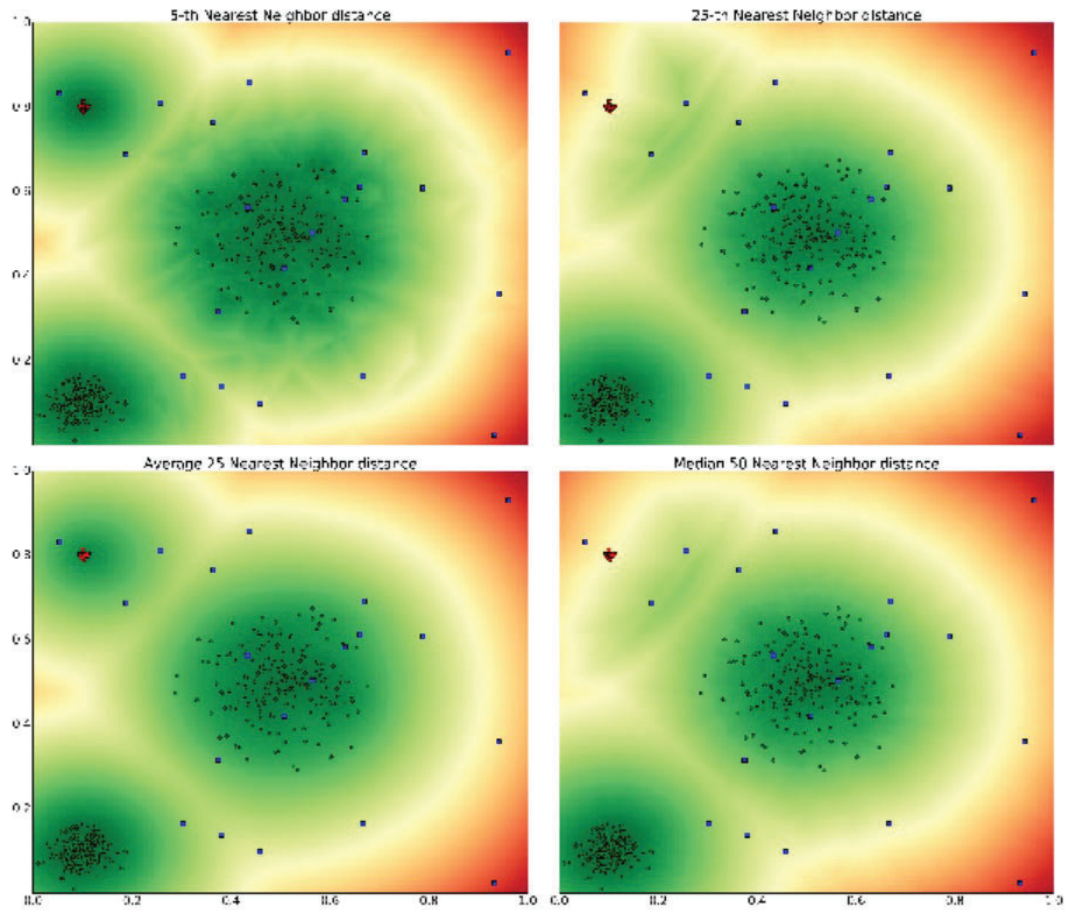


Рисунок 1.11 – Приклади роботи метричних алгоритмів

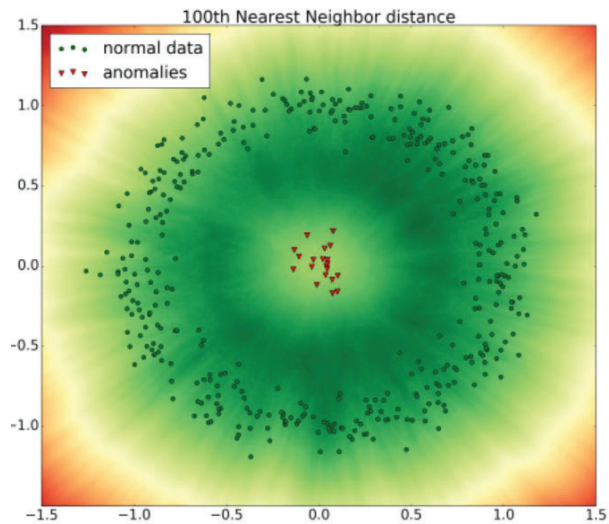


Рисунок 1.12 – Приклад завдання з нетривіальним підпростором нормальних даних, в якому аномалії розташовані в центрі мас

Головна перевага метричних методів – інтерпретованість. Можна сказати, що саме метричними методами завдання знаходження вторгнень вирі-

шує людина. Головним же недоліком є необхідність підбору метрики. На практиці в даних інтерпретована метрика відсутня; вона апроксимується стандартними метриками – наприклад, манхетенівською, евклідовою чи чебишевською. Вибір метрики без міток може виявитися сліпим вибором, а апроксимація навіть при найкращому підборі - грубою.

Типовим представником цього напрямку є *алгоритм LOF (Local Outlier Factor)* [29].

Більш тонкою проблемою метричних методів є той факт, що всі припущення, що лежать в їх основі, справедливі лише в доповненні один з одним: так, локальна щільність точки, що лежить в центрі невеликого кластера вторгнень, може виявитися вищою, ніж для будь-якої точки з великого кластера нормальних даних. Можливо і зворотне: ізольована точка-аномалія може розташовуватися, наприклад, у центрі мас кластера нормалій, і тоді середня відстань від неї до сусідів буде меншою, ніж для нормальних точок. Цю «властивість» метричних алгоритмів намагається врахувати алгоритм LOF (Local Outlier Factor) [16].

Нехай $D_k(y)$ – відстань від точки y до k найближчого сусіда. Тоді досяжністю (reachability distance) точки x щодо точки y будемо називати величину

$$R_k(x, y) = \max(\rho(x, y), D_k(y)) \quad (1.2)$$

Нехай $AR_k(x)$ – середня досяжність точки x щодо своїх найближчих сусідів, $N_k(x)$ – множина k найближчих сусідів x . Тоді:

$$LOF_k(x) = \frac{\text{mean}_{y \in N_k(x)} AR_k(y)}{AR_k(x)} \quad (1.3)$$

Інтуїція формули (1.3) полягає в тому, щоб порівняти середню досяжність точки та її найближчих сусідів. Для представників нормальних даних вірно не тільки те, що оцінка (1.2) локальної щільності мала, а й що вона значно відрізняється від такої ж оцінки для найближчих сусідів. Приклад роботи алгоритму наведено на рис. 1.13.

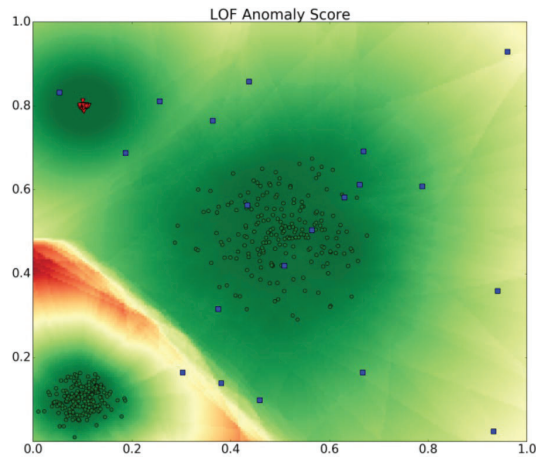


Рисунок 1.13 – Приклад роботи Local Outlier Factor

1.4.5. Методи підміни завдання

Коли виникає нове завдання, є велика спокуса вирішити її старими методами (орієнтованими на вже відомі завдання). Наприклад, можна зробити **кластеризацію**, тоді маленькі кластери, швидше за все, складаються з вторгнень. Якщо ми маємо часткову інформацію про аномалії (як у задачі PUC), можна вирішити її як завдання класифікації з класами 1 (розмічені аномалії) і 0 (всі інші об'єкти). Якби клас 0 складався лише з нормальних об'єктів, то таке рішення було б зовсім законним, інакше залишається сподіватися, що недетектованих вторгнень у ньому небагато [25].

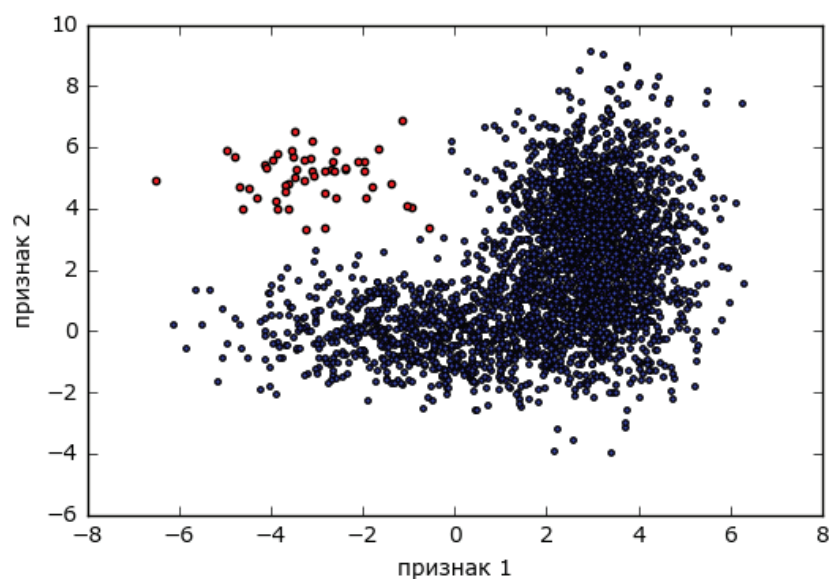


Рисунок 1.14 – Приклад кластеризації на малий (червоний)
та великий (синій) кластери

1.4.6. Специфічні методи машинного навчання

А що якщо сприйняти завдання знаходження вторгнень як нове завдання машинного навчання (відмінне від класифікації та кластеризації)?!

Найпопулярніші алгоритми (реалізовані навіть у `scikit-learn`) тут є:

- Метод опорних векторів для одного класу (`OneClassSVM`).
- Ізольюючий ліс (`IsolationForest`).
- Еліпсоїдна апроксимація даних (`EllipticEnvelope`).

Перший метод - це звичайний SVM, який відокремлює вибірку від початку координат. Ідея трохи сумнівна, але виявилася досить працездатною (див. рис. 1.15). Тут правда не так багато різноманітності у виборі параметрів, як при вирішенні завдань класифікації, оскільки як ядро підійде лише `rbf` (радіальні базисні функції), решта всіх ядер показують феноменально огидний результат. Цікаво, що багато років завдання детектування поломок складних механізмів вирішувалися саме за допомогою `OneClassSVM`, чомусь без розгляду альтернатив. Корисно пам'ятати, що `OneClassSVM` це швидше алгоритм пошуку новизни, а не викидів, так як «заточується» під навчальну вибірку.

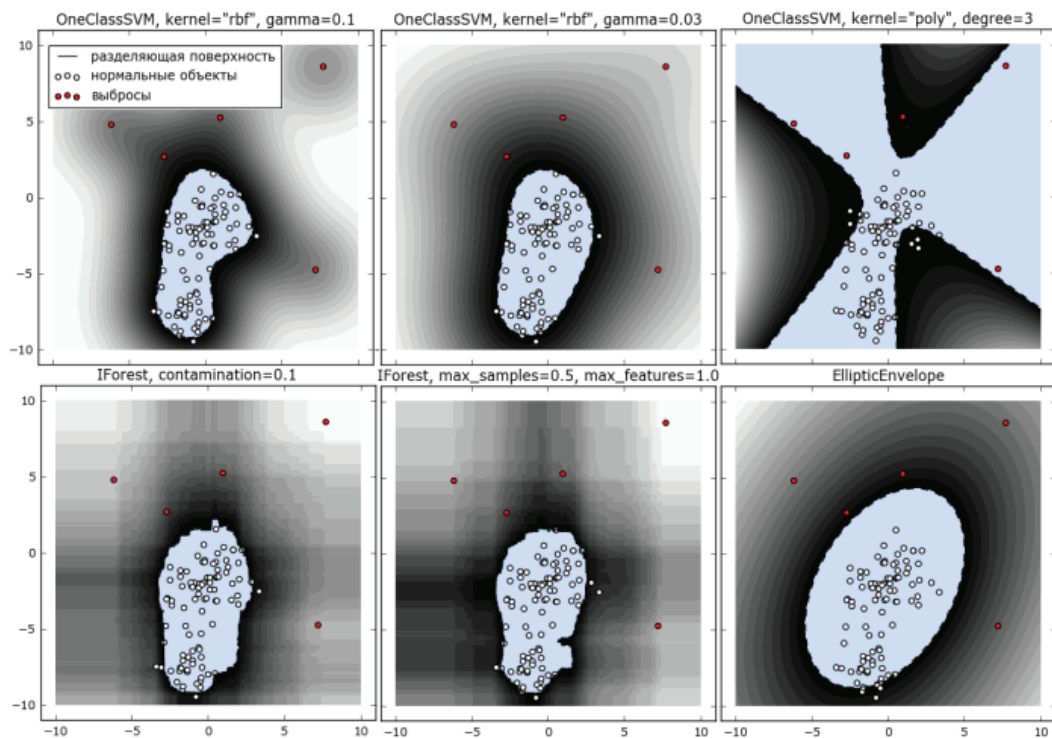


Рисунок 1.15 – Візуалізація роботи різних алгоритмів пошуку вторгнень

Другий метод *Isolation Forest*.

Ідея ізолюючого лісу (*Isolation Forest*) [12] заснована на принципі Монте-Карло: проводиться випадкове розбиття простору ознак, таке, що в середньому ізольовані точки відсікаються від нормальних, кластеризованих даних. Остаточний результат усереднюється з кількох запусків стохастичного алгоритму.

Алгоритм ізолюючого дерева (*Isolation Tree*) полягає у побудові випадкового бінарного вирішального дерева. Коренем дерева є весь простір ознак; у черговому вузлі вибирається випадкова ознака і випадковий поріг розбиття, семпльований з рівномірного розподілу на відрізку від мінімального до максимального значення обраної ознаки. Критерієм зупинки є тотожний збіг всіх об'єктів у вузлі, тобто вирішальне дерево будується повністю. Відповіддю в листку, яка також відповідає *anomaly_score* для алгоритма, оголошується глибина листка в побудованому дереві (рис.16)

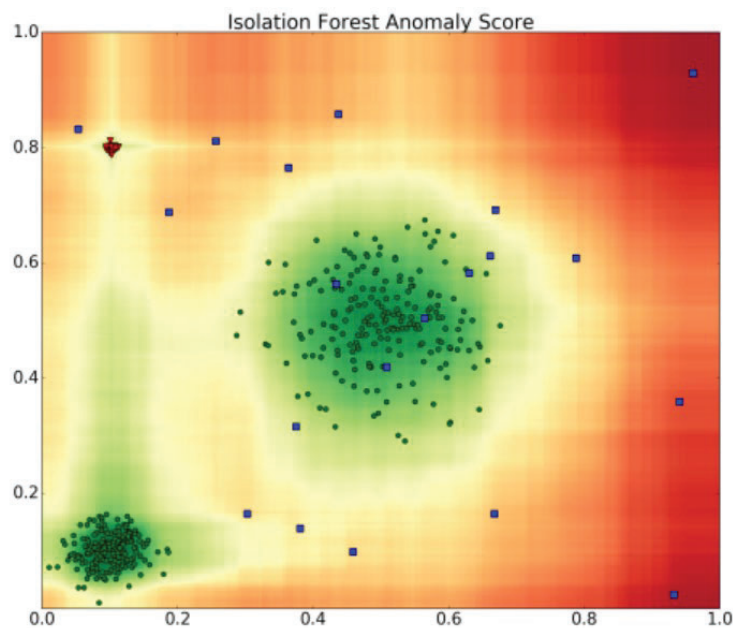


Рисунок 1.16 – Приклад роботи *Isolation Forest*

Стверджується, що аномальним точкам властиво опинятися у листі з низькою глибиною, тобто у листі, близьким до кореня, коли ж для розбиття гіперплощинами кластера нормальних даних дереву потрібно побудувати ще кілька рівнів. У цьому кількість таких рівнів пропорційно розміру кластера; отже, пропорційно і *anomaly_score* для точок, що лежать у ньому. Це означає,

що об'єкти з кластерів малих розмірів, які потенційно є аномаліями, будуть мати *anomaly_score* нижче, ніж із кластерів нормальних даних.

Одна із найпростіших версій цього алгоритму працює так:

- Кожне дерево будується до вичерпання вибірки.
- Для побудови розгалуження у дереві: вибирається випадкова ознака та випадкове розщеплення.
- Для кожного об'єкта обчислюється міра його нормальності – середнє арифметичне глибин листя, в яке він потрапив (ізолювався).

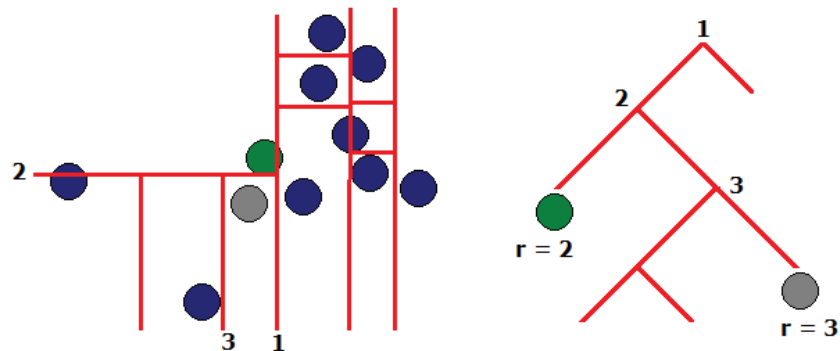


Рисунок 1.17 – Обчислення оцінки аномальності в ізолюючому лісі

Логіка алгоритму проста: при описаному «випадковому» способі побудови дерев викиди потраплятимуть у листя на ранніх етапах (на невеликій глибині дерева). Тобто, викиди простіше «ізолювати» (нагадаємо, що дерево будується доти, доки кожен об'єкт не опиниться в окремому листку). Алгоритм добре відловлює саме викиди (рис. 1.18).

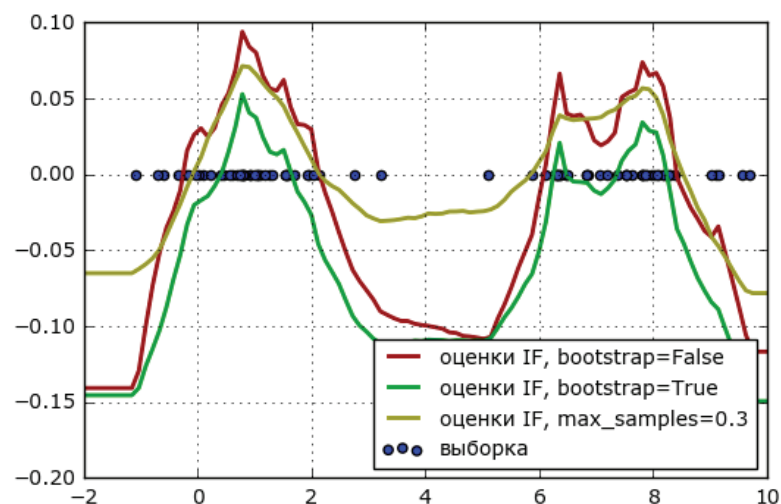


Рисунок 1.18 – Оцінки ізолюючого лісу для одновимірної вибірки

Даний алгоритм має ряд істотних переваг:

- Алгоритм розпізнає аномалії різних видів: як ізольовані точки з низькою локальною щільністю, так і кластери вторгнень малих розмірів.
- Складність ізолюючого дерева – $O(n \log n)$, що ефективніше за більшість інших алгоритмів.
- Не вимагає суттєвих витрат по пам'яті, на відміну від, наприклад, метричних методів, які часто потребують побудови матриці попарних відстаней.
- Відсутні параметри, які потребують вибору.
- Інваріантний до масштабування ознак; не вимагає завдання метрики або іншої апріорної інформації про будову даних.
- Стійкий до прокляття розмірності.

Існують і інші методи та підходи до виявлення вторгнень.

1.5. Постановка завдання

Завдання детектування вторгнень немає єдиного формулювання і найчастіше інтерпретуються по-різному залежно від характеру даних і поставленої мети [11, 13, 25]. На інтуїтивному рівні аномаліями називають те, що не вписується у загальні правила та закони, справедливі для поданих даних. Таке визначення потребує формального уточнення, перш ніж вирішувати завдання математичними методами.

Будь-які методи детектування вторгнень спираються на деяке суворе уявлення про те, що означає «відхилення від норми». У окремих випадках, вже це уявлення може відштовхуватися від контексту завдання та апріорної інформації. Наприклад, у найпростішому випадку, якщо дані є набором елементів деякої множини, будь-який елемент, що не належить цій множині, апріорі можна вважати аномалією. Проте така «характеристична» множина, за приналежністю до якої можна робити висновок про аномальність елемента, існує не завжди. Наприклад, якщо об'єктами є показання деякого датчика, а

аномаліями – показання зламаного датчика, може статися так, що і зламаний, і правильно працюючий датчики видадуть той самий результат.

Істотно різниться і природа появи вторгнень у даних. Це може бути як шум, тобто дані, що утворилися найчастіше випадковим чином, так і рідкісні явища, що становлять інтерес і потребують додаткового вивчення (наприклад, поява сторонніх об'єктів на знімках). В останньому випадку аномалії мають особливу природу і можуть навіть виділятися в відокремлені кластери (рис. 1.19).

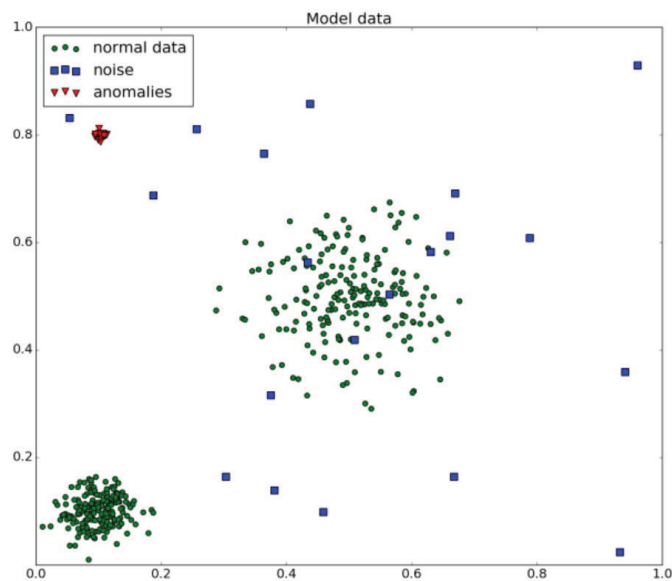


Рисунок 1.19 – Приклад модельної задачі.

Нормальні дані є двома кластерами, семпльованими з нормального розподілу. Аномалії утворюють окремий кластер малого розміру, крім того, є шум, семпльований з рівномірного розподілу

Підходи до виявлення вторгнень різних "видів" можуть суттєво відрізнятися.

Оскільки різні алгоритми розпізнавання вторгнень походять від різних визначень аномальності, відрізняється не тільки їх результат, а й підхід.

Можливе й протилежне: пропонований підхід визначає інтерпретацію аномальності. Наприклад, якщо як дані взяти набір чисел, то можна вважати, що видалення з нього вторгнень знизить дисперсію; потім можна знайти в

наборі підмножину чисел (наприклад, заданого розміру), видалення яких максимально знизить дисперсію, і оголосити ці точки аномаліями.

Математична постановка задачі

Надалі будемо вважати, що дані мають ознакове уявлення, тобто кожен об'єкт x заданий у вигляді деякого вектора $\mathbf{R}^d$. У класичній постановці завдання детектування вторгнень формулюється так: у заданій множині X для кожного елемента видати **0** якщо цей об'єкт відноситься до **класу нормальних даних**, та **1**, якщо **цей об'єкт аномальний**. Таке завдання відноситься до класу завдань *навчання без вчителя*, оскільки правильних відповідей на частини вхідних даних не надається.

В аналогічному завданні *навчання з учителем* на деякій частині X_{train} вхідних даних відома вірна відповідь, тобто для кожного об'єкта $x \in X_{\text{train}}$ відомі мітки $y(x) \in \{0, 1\}$ – чи є об'єкт аномалією. Завдання видачі міток для нових даних, X_{test} , формально є завданням бінарної класифікації, і, отже, може вирішуватися з допомогою будь-яких алгоритмів машинного навчання з учителем. Однак, можливий і «проміжний варіант», коли всі мітки $y(x)$, $x \in X_{\text{train}}$ дорівнюють 0, тобто задані приклади лише нормальних («перевічених», «хороших») даних. У такому випадку алгоритми розв'язання задачі бінарної класифікації видаватимуть нерелевантний константний прогноз.

Варто зазначити, що ця проблема, нехай меншою мірою, присутня і у випадку, коли в навчальній вибірці є приклади вторгнень: алгоритми навчання з учителем на даних, що не мають аналогів у навчанні, видають у випадку випадкову відповідь (причому часто, одиниця видається з ймовірністю $P(y(x) = 1)$, оціненої за навчальною вибіркою). З цієї причини є сенс розглядати завдання саме як завдання навчання без вчителя, а наявність навчальної вибірки (міток) – як можливість додатково налаштовувати параметри алгоритму.

Практично всі алгоритми детектування вторгнень зводяться до побудови деякої функції $anomaly_score(x)$, яка по кожному об'єкту видає певний «рейтинг» аномальності.

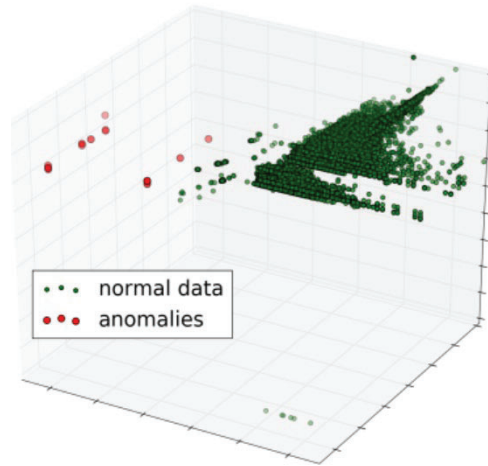


Рисунок 1.20 – Дані датасета Smtpr із репозиторія [12]

Після цього поділ на клас вторгнень і клас нормальних даних проводиться бінаризацією по деякому порозу, вибір якого є особливим етапом розв'язання задачі. В умовах відсутності міток або апіорної інформації, єдиною наявною інформацією є *одномірний розподіл* значень *anomaly_score* на наявних даних, чого для обґрунтованого вибору недостатньо. Найчастіше відома приблизна частка вторгнень даних, у таких випадках як поріг вибирається відповідний квантиль одномірного розподілу.

Функціонали якості в задачах детектування вторгнень використовують приблизно такі самі, як і в задачах класифікації: PR AUC, AUROC, і визначаються контекстом завдання (або замовником).

Висновки по 1 розділу

Таким чином, проблема забезпечення кібербезпеки «розумного» будинку є актуальною.

Задача виявлення проникнення злоумисників у систему може розглядатися як задача виявлення вторгнень і для її рішення необхідно розробити детектор вторгнень.

РОЗДІЛ 2. РОЗРОБКА МЕТОДИКИ ПОБУДОВИ ДЕТЕКТОРА ВТОРГНЕНЬ ДЛЯ «РОЗУМНОГО» БУДИНКУ

2.1. Розробка конвеєра машинного навчання для детектора вторгнень «розумного» будинку

Типовий проект машинного навчання починається з розуміння бізнес-цілі. Зазвичай бізнес-аналітик працює з клієнтом і аналітиком даних, щоб перетворити бізнес-проблему на інженерний проект. Інженерний проект може мати або не мати частину машинного навчання. У цій роботі ми розглядаємо проект, в якому задіяно машинне навчання для виявлення вторгнень у мережу «розумного» будинку.

Після того, як інженерний проект визначено, з цього починається сфера інженерії машинного навчання. У рамках більш широкого інженерного проекту машинне навчання має спочатку мати чітко визначену мету. Метою проекту машинного навчання є специфікація того, що статистична модель отримує як вхідні дані, що вона генерує як вихідні дані, а також критерії прийнятної (або неприйнятної) поведінки моделі.

Мета машинного навчання не обов'язково збігається з бізнес-метою. Бізнес-мета – це те, чого хоче досягти організація.

У більш простому вигляді, коли вирішується конкретна задача, як наша, створюється конвеєр машинного навчання (machine learning pipeline).

Основними кроками в конвеєрі машинного навчання є:

1. Визначення проблеми: визначення бізнес-проблеми.
2. Збирання даних: ідентифікація та збір набору даних.
3. Підготовка даних: здійснюється обробка та підготовка даних, використовуючи такі методи, як:
 - ° Введення відсутніх значень.
 - ° Видалення повторюваних записів.
 - ° Нормалізація значень (зміна числових значень в наборі даних для використання загальної шкали).
 - ° Очищення або зіставлення даних.

- ° Повне визначення ознак для навчання.
 - ° Виключення корельованих ознак.
4. Розділення даних на навчальний набір та набір для тестування.
 5. Навчання моделі на основі навчального набору даних. Це ядро науки про дані.
 6. Оцінка якості моделі кандидата: оцінюється якість моделей за обраними показниками, використовуючи тестові та перевіірочні підмножини даних.
 7. Розгортання моделі. Після того, як модель вибрано, розгорніть її для роботи.
 8. Моніторинг якості роботи моделі. Постійно відстежується продуктивність моделі, при суттєвому падінні показників якості, необхідно здійснювати перенавчання та відповідним чином калібрування моделі. Сюди ж відноситься і збирання нових даних, щоб продовжувати вдосконалювати модель і запобігати її старінню.

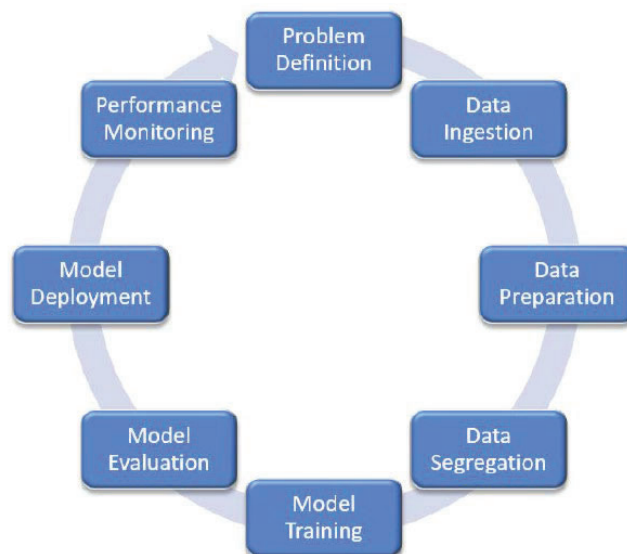


Рисунок 2.1 – Типовий конвеєр машинного навчання

2.2. Аналіз та вибір програмного забезпечення для реалізації детектора вторгнень

Одним з факторів, що суттєво впливає на час реагування на кібератаки є виявлення і прогнозування нових загроз. Так затримка при реагуванні виникає навіть при загрозах відомих типів. Нові ж види атак, моделі поведінки та ін-

струменти можуть збити фахівців з пантелику, в результаті чого вони будуть реагувати ще повільніше. Гірше того, такі менш помітні загрози, як крадіжка даних, іноді можуть залишитися і зовсім невиявленими. Тому, будуючи захист мереж, доводиться завжди враховувати постійний розвиток технологій, що застосовують зловмисники. На щастя, методи кібератак зазвичай не винаходяться з нуля, а основою для них часто служать тактики, платформи і вихідні коди минулих атак. Саме це і дозволяє застосовувати технології машинного навчання, бо їм є на чому базуватися при накопиченні знань.

Програмне забезпечення (ПЗ) на основі машинного навчання допомагає розпізнати атаку, виявивши спільні риси у новій загрозі та виявлених раніше. Машина, на відміну від людини, проведе таке порівняння швидко - що ще раз підкреслює необхідність застосування адаптивних моделей безпеки. Машинне навчання може полегшити прогнозування нових загроз і скоротити час реагування за рахунок більш ефективної роботи з базою існуючих загроз.

У комп'ютерній безпеці зловмисні проникнення можуть бути виявлені за рахунок незвичайного мережного трафіка або нетипових дій користувача. Такі вторгнення можуть порушити як приватну конфіденційність, так і організаційну. Їхнє виявлення зводиться до аналізу аномалій.

Підходи з використанням методів машинного навчання до проблеми виявлення вторгнень можна поділити на два класи:

1. Виявлення неправильного використання (Misuse Detection) – побудова прогностичної моделі на основі розмічених даних (екземпляри помічені як «нормальні» або «проникнення»). Вони демонструють високу точність виявлення великої кількості відомих атак, але не можуть виявити невідомі та нові атаки.

2. Виявлення аномалій (Anomaly Detection), коли система може виявляти нові атаки як відхилення від «нормальної» поведінки, однак при цьому можливий високий рівень хибних «спрацьовувань», так як виявлені відхилення не завжди являють собою реальне вторгнення.

У доповіді розглядаються саме програмні системи виявлення аномалій (Anomaly Detection Systems, ADS). Основним припущенням ADS є те, що дії зломисника (події в атакованій системі) обов'язково чимсь відрізняються від поведінки звичайного користувача (від подій в нормальному стані), тобто є аномаліями. Тому такі системи здатні реєструвати і невідомі атаки.

Серед платного ПЗ для рішення задачі виявлення аномалій доцільно було б провести тести та порівняти такі: Numenta, AVORA, Splunk Enterprise, Loom Systems, Elastic X-Pack, Anodot, CrunchMetrics. Однак із-за значної його вартості вдалося провести лише порівняльний огляд по описам та відгукам з Інтернету. Як краще безкоштовне ПЗ для виявлення аномалій розглядалися: Weka Data Mining, Shogun, RapidMiner Starter Edition, Dataiku DSS Community, ELKI, Scikit-learn. Оцінювалися як відгуки користувачів так і експериментальна якість виявлення аномалій.

Порівняння можливостей та функціоналу ПЗ проводилося за такими показниками:

Якість виявлення аномалій: це найважливіша функція програмного забезпечення виявлення аномалій, оскільки основна мета програмного забезпечення — виявляти аномалії. Програмне забезпечення повинно дозволяти бізнес-користувачам виявляти будь-які незвичайні моделі, поведінку чи події. Тут застосовувалися відомі показники якості виявлення аномалій на основі матриці помилок.

Кількість алгоритмів машинного навчання, що реалізовано в ПЗ. Можливість вибору та налаштування різних алгоритмів машинного навчання дозволяє обирати найкращі для рішення конкретної задачі.

Оповіщення в режимі реального часу: функція виявлення аномалій була б марною без механізму оповіщення користувачів, коли система виявляє аномалію. ПЗ повинно повідомляти користувачів у режимі реального часу, надсилаючи попередження електронною поштою, додатковими програмами та текстовими повідомленнями.

Можливості моніторингу та відображення: програмне забезпечення для виявлення аномалій повинно включати панелі моніторингу та відображення, що налаштовуються, які дозволяють користувачам відображати показники різними способами.

Інтеграція: Найкраще програмне забезпечення для виявлення аномалій повинно легко інтегрується з існуючими системами та поширеними мовами програмування.

На думку автора слід використовувати мову програмування Python та спеціалізовану бібліотеку PyOD.

PyOD — це масштабований набір інструментів Python для виявлення викидів у багатовимірних даних. Він надає доступ до приблизно 20 алгоритмів виявлення викидів у рамках єдиного добре задокументованого API.

Основними перевагами PyOD є наступні:

- Відкритий код із детальною документацією та прикладами різних алгоритмів.
- Підтримує розширені моделі, включаючи нейронні мережі, глибоке навчання та ансамблі методів.
- Оптимізована продуктивність за допомогою JIT (точно вчасно) і розпаралелювання за допомогою numba і joblib.
- Бібліотека сумісна як з Python 2, так і з 3.

PyOD також містить деякі моделі на основі нейронних мереж, які реалізовані в Keras. PyOD. Тому потрібно буде встановити Keras та інші бібліотеки вручну, якщо необхідно використовувати моделі на основі нейронної мережі.

Розглянемо основні алгоритми виявлення викидів, що використовуються в PyOD.

Будемо використовувати термін «Оцінка за межами». Це означає, що кожна модель певним чином оцінює точку даних, а потім використовує порогове значення, щоб визначити, чи є ця точка викидом чи ні.

Виявлення викидів на основі кута (ABOD)

Цей метод розглядає зв'язок між кожною точкою даних та її сусідами. Він не враховує відносини між цими сусідами. Дисперсію його зважених ко-

синусних оцінок для всіх сусідів можна розглядати як віддалену оцінку. ABOD добре працює з багатовимірними даними.

PyOD пропонує дві різні версії ABOD:

швидкий ABOD: використовує k -найближчих сусідів для наближення

оригінальний ABOD: Враховує всі навчальні моменти з високою часо-вою складністю.

Детектор найближчих сусідів (kNN)

Для будь-якої точки даних відстань до k -го найближчого сусіда можна розглядати як віддалений результат. PyOD підтримує три детектори kNN :

Найбільший: використовує відстань k -го сусіда, як оцінку викиду.

Середній: Використовує середнє значення всіх k сусідів, як оцінку викиду.

Медіана: використовує медіану відстані до k сусідів, як оцінку викиду.

Ізолюючий ліс.

Він використовує внутрішню бібліотеку scikit-learn. У цьому методі розділення даних здійснюється за допомогою набору дерев. Ізоляційний ліс надає оцінку аномалії, дивлячись на те, наскільки ізольована точка в структурі. Оцінка аномалії потім використовується для виявлення відхилень від нормальних спостережень. Ізолюючий ліс добре працює з багатовимірними даними.

Виявлення викидів на основі гістограм.

Це ефективний метод неконтрольованого навчання, який припускає незалежність ознак і обчислює викид, будуючи гістограми. Це набагато швидше, ніж багатоваріантний підхід, але ціною меншої точності.

Локальний кореляційний інтеграл (LOCI)

LOCI дуже ефективний для виявлення викидів і груп викидів. Він надає графік LOCI для кожної точки, який узагальнює багато інформації про дані в області навколо точки, визначає кластери, мікрокластери, їхні діаметри та відстані між кластерами. Жоден із існуючих інших методів виявлення викидів не може відповідати цій функції, оскільки вони виводять лише одне число для кожної точки.

Функція Bagging

Детектор Bagging за властивостями підходить до кількох базових детекторів на різних підвибірках набору даних. Він використовує усереднення

або інші комбіновані методи для підвищення точності передбачення. За замовчуванням, як базовий оцінювач використовується локальний коефіцієнт викиду (LOF). Однак будь-який оцінювач можна використовувати як базовий, наприклад kNN і ABOD. Розміщення ознак спочатку створює n підзразків шляхом випадкового вибору підмножини ознак. Це виявляє різноманітність базових оцінок. Нарешті, оцінка передбачення генерується шляхом усереднення або взяття максимуму всіх базових детекторів.

Фактор локального викиду на основі кластеризації

Він класифікує дані на малі та великі кластери. Потім оцінка аномалії обчислюється на основі розміру кластера, до якого належить точка, а також відстані до найближчого великого кластера.

Додаткові утиліти, надані PyOD

- Функцію `generate_data` можна використовувати для генерації випадкових даних із викидами. Дані генеруються за допомогою багатовимірного розподілу Гауса, а викиди – за допомогою рівномірного розподілу. Можемо надати власні значення частки викидів і загальну кількість зразків, які нам потрібні в нашому наборі даних. Будемо використовувати цю службову функцію для створення даних у частині моделювання реалізації.

2.3. Методика побудови детектора вторгнень для «розумного» будинку

Грунтуючись на конвеєрі машинного навчання та спеціалізованій бібліотеці PyOD Python сформуємо покрокову методику побудови детектора вторгнень для «розумного» будинку.

Крок 1. Визначаємо перелік параметрів мережі розумного будинку за якими будемо визначати вторгнення. Вони будуть виконувати роль ознак для задач класифікації.

Крок 2. Формування масиву вхідних даних (набір статистики та формування dataset). Набір даних для навчання повинний задовольняти декільком критеріям:

- Репрезентативність — дані повинні ілюструвати дійсне положення речей у предметній області.
- Несуперечність — суперечливі дані в навчальній вибірці призведуть до поганої якості навчання мережі.
- Обсяг — як правило, число записів у вибірці повинне на кілька порядків перевершувати кількість можливих комбінацій. У протилежному випадку детектор просто «запам'ятає» усю навчальну вибірку і не зможе виконати узагальнення.

Крок 3. Формування набору методів виявлення аномалій (із можливостей бібліотеки PyOD).

Крок 4. Вибір показників та критеріїв якості виявлення вторгнень. Цей крок детально розглядається у наступному підрозділі.

Крок 5. Розробка програмного забезпечення детектора вторгнень.

Крок 6. Розділення масиву вхідних даних на навчальну та контрольну вибірки (співвідношення 70% і 30% відповідно).

Навчальні дані подаються мережі для навчання, а контрольні використовуються для розрахунку похибки мережі (контрольні дані ніколи для навчання мережі не застосовуються). Таким чином, якщо на контрольних даних похибка зменшується, то мережа дійсно виконує узагальнення.

Якщо похибка на навчальних даних продовжує зменшуватися, а похибка на тестових даних збільшується, виходить, детектор перестав виконувати узагальнення і просто «запам'ятовує» навчальні дані. Це явище називається перенавчанням або оверфітінгом. У таких випадках навчання зазвичай припиняють. У процесі навчання можуть проявитися інші проблеми, такі як параліч або влучення детектора в локальний мінімум поверхні похибок. Неможливо заздалегідь передбачити прояв тієї або іншої проблеми, так само як і дати однозначні рекомендації до їх розв'язання.

Крок 7. Навчання детектора вторгнень.

Перед використанням, детектор вторгнень з контрольованим навчанням повинен бути навченим. Навчання займає певний час. Навчання вважається закінченим при досягненні детектором визначеного користувачем рівня ефективності і бажаної статистичної точності.

Крок 8. Оцінка якості роботи методів виявлення аномалій у детекторі вторгнень та вибір оптимального.

Крок 9. Оптимізація програмного коду детектора вторгнень для оптимального метода виявлення.

Крок 10. Введення в експлуатацію детектора вторгнень та моніторинг якості його роботи.

2.4. Критерії оцінки якості для детектора вторгнень

Для оцінки якості виявлення вторгнень на основі класифікації застосовують такі показники (метрики):

Матриця помилок (або матриця неточностей, англ. Confusion matrix)

Перед переходом до самих метрик необхідно ввести важливу концепцію для опису цих метрик в термінах помилок класифікації - confusion matrix (матриця помилок). Нехай система виявлення вторгнень використовує систему класифікації для виявлення атаки: нормальна робота мережі чи аномальна (наявність атаки). При цьому у першому випадку система і далі нормально працює, а у другому – видається сигнал аномальної роботи. Таким чином, виявлення неадекватної (аномальної) роботи мережі ($y = 1$) можна розглядати як "сигнал тривоги", що повідомляє про виявлення вторгнення.

Будь-який реальний класифікатор робить помилки. У нашому випадку таких помилок може бути дві:

Нормальна ситуація у мережі за даними трафіка розпізнається моделлю як аномальна. Даний випадок можна трактувати як "помилкову тривогу".

Аномальна ситуація розпізнається як нормальна і ніяких дій по захисту від атаки не відбувається. Даний випадок можна розглядати як "пропуск вторгнення".

Неважко помітити, що ці помилки нерівноцінні по зв'язаних з ними наслідками. У разі "помилкової тривоги" втрати складуть тільки марно потра-

чений час та ресурси на протидію неіснуючій загрозі. У разі "пропуску вторгнення" можна втратити набагато більше (інформацію, роботу мережі та інше, це залежить від виду атаки). Тому системі захисту важливіше не допустити "пропуску вторгнення", ніж "помилкову тривогу".

Оскільки з точки зору логіки завдання виявлення вторгнень нам важливіше правильно розпізнати аномалію (атаку) з міткою $y = 1$, ніж помилитися в розпізнаванні нормальної роботи мережі, будемо називати відповідний результат класифікації позитивним (аномалія чи атака виявлені вірно), а протилежний - негативним (аномалії чи атаки немає $y = 0$). Тоді можливі наступні чотири результати класифікації:

True Positive (TP) – наявність атаки класифікована як наявна атака, тобто позитивний клас розпізнано як позитивний. Спостереження, для яких це має місце називаються істинно-позитивними.

True Negative (TN) – нормальна робота мережі класифікована як нормальна робота без вторгнень, тобто негативний клас розпізнано як негативний. Спостереження, яких це має місце, називаються істинно негативними.

False Positive (FP) – нормальна робота мережі класифікована як аномальна, тобто мала місце помилка, в результаті якої негативний клас був розпізнаний як позитивний. Спостереження, для яких було отримано такий результат класифікації, називаються помилково-позитивними, а помилка класифікації називається помилкою I роду.

False Negative (FN) – атака чи аномальна робота мережі розпізнана як нормальна, тобто мала місце помилка, в результаті якої позитивний клас був розпізнаний як негативний. Спостереження, для яких було отримано такий результат класифікації, називаються помилково-негативними, а помилка класифікації називається помилкою II роду.

Таким чином, помилка I роду, або хибно-позитивний результат класифікації, має місце, коли негативне спостереження розпізнано моделлю як позитивне. Помилкою II роду, або хибно-негативних результатом класифікації,

називають випадок, коли позитивне спостереження розпізнано як негативне.

Пояснимо це за допомогою матриці помилок класифікації:

	$y = 1$	$y = 0$
$a(x) = 1$	Істинно-позитивний (True Positive - TP)	Помилково-позитивний (False Positive - FP)
$a(x) = 0$	Помилково-негативний (False Negative - FN)	Істинно-негативний (True Negative – TN)

Тут $a(x)$ - це відповідь алгоритму при конкретній ситуації, а y - справжня мітка класу для цієї ситуації. Таким чином, помилки класифікації бувають двох видів: False Negative (FN) і False Positive (FP). P означає що класифікатор визначає клас об'єкта як позитивний (N - негативний). T означає що клас передбачений правильно (відповідно F - неправильно). Кожен рядок в матриці помилок представляє прогнозований клас, а кожен стовпець - фактичний клас.

Тобто у загальному випадку, матриця неточностей - це матриця розміром N на N , де N – кількість класів, яка представляє собою табличне представлення прогнозованих і фактичних значень для кожного можливого класу.

Матриця помилок візуалізує кількість фактичних екземплярів класу в порівнянні з прогнозованими екземплярами класу. Таке подання дозволяє нам швидко побачити кількість правильних і неправильних прогнозів для кожної категорії.

На основі цієї матриці будується ряд інших характеристик. Розглянемо кожну з них більш детально.

Акуратність (англ. Accuracy).

Акуратністю називається пропорція точних прогнозів по відношенню до загальної кількості прогнозів, тобто це ймовірність того, що клас буде передбачений правильно (2.1). Тобто, accuracy - частка правильних відповідей алгоритму:

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

Хоча акуратність є швидким та інформативним індикатором продуктивності моделі, ми не можемо покладатися виключно на неї. Це пов'язано з тим,

що вона приховує наявність зсуву в моделі, що є звичайним явищем, якщо набір даних незбалансований, тобто негативних моментів значно більше, ніж позитивних, або навпаки. Тобто, ця метрика марна в задачах з нерівними класами, що як варіант можна виправити за допомогою алгоритмів семпліювання. Семпліювання (англ. Data sampling) - метод коригування навчальної вибірки з метою балансування розподілу класів у вихідному наборі даних.

Точність (англ. Precision).

Точністю називається частка правильних відповідей моделі в межах класу - це частка об'єктів, що дійсно належать даному класу, щодо всіх об'єктів які система віднесла до цього класу.

$$Precision = \frac{TP}{TP + FP} \quad (2.2)$$

Саме введення precision не дозволяє нам записувати всі об'єкти в один клас, так як в цьому випадку ми отримуємо зростання рівня False Positive.

Повнота (англ. Recall).

Повнота - це частка істинно позитивних класифікацій. Повнота показує, яку частку об'єктів, що реально належать до позитивного класу, ми передбачили вірно. Або ж іншими словами: це частка варіантів, класифікованих як позитивні, які насправді виявилися позитивними.

$$Recall = \frac{TP}{TP + FN} \quad (2.3)$$

Повнота (recall) демонструє здатність алгоритму виявляти даний клас взагалі.

Маючи матрицю помилок, дуже просто можна обчислити точність і повноту для кожного класу. Точність (precision) дорівнює відношенню відповідного діагонального елемента матриці і суми всього рядка класу. Повнота (recall) - відношенню діагонального елемента матриці і суми всього стовпчика класу. Оскільки класів може бути багато (не обов'язково два), то формально:

$$Precision_c = \frac{A_{c,c}}{\sum_{i=1}^n A_{c,i}}$$

$$Recall_c = \frac{A_{c,c}}{\sum_{i=1}^n A_{i,c}}$$

Тобто, результуюча точність класифікатора розраховується як середнє арифметичне його точності по всіх класах. Те ж саме з повнотою.

F-міра (англ. F-score).

Precision і recall не залежить від співвідношення класів (на відміну від accuracy) і тому можуть бути застосовні в умовах незбалансованих вибірок. Часто в реальній практиці стоїть завдання знайти оптимальний (для замовника) баланс між цими двома метриками. Зрозуміло що чим вище точність і повнота, тим краще. Але в реальному житті максимальна точність і повнота не досяжні одночасно і доводиться шукати якийсь баланс. Тому, хотілося б мати якусь метрику яка об'єднувала б у собі інформацію про точність та повноту нашого алгоритму. У цьому випадку нам буде простіше приймати рішення про те, яку реалізацію запускати у виробництво (у кого більше той і крутіше). Саме такою метрикою є F-міра.

F-міра є гармонійним середнім між точністю і повнотою. Вона прагне до нуля, якщо точність або повнота прагне до нуля.

$$F = \frac{2 \times precision \times recall}{precision + recall} \quad (2.4)$$

Дана формула надає однакову вагу точності і повноти, тому F-міра буде падати однаково при зменшенні і точності і повноти (рис. 2.2). Можливо розрахувати F-міру надавши різну вагу точності і повноті, якщо ви свідомо віддаєте пріоритет одній з цих метрик при розробці алгоритму:

$$F_{\beta} = \frac{(1 + \beta^2) \times precision \times recall}{(\beta^2 \times precision) + recall} \quad (2.5)$$

де β приймає значення в діапазоні $0 < \beta < 1$ якщо ви хочете віддати пріоритет точності, а при $\beta > 1$ пріоритет віддається повноті. При $\beta = 1$ формула зводиться до попередньої і ви отримуєте збалансовану F-міру (також її називають F_1).

F-міра досягає максимуму при максимальній повноті і точності, і близька до нуля, якщо один з аргументів близький до нуля.

F-міра є хорошим кандидатом на формальну метрику оцінки якості класифікатора. Вона зводить до одного числа дві інші основоположні метрики: точність і повноту. Маючи "F-міру" набагато простіше відповісти на питання: "змінився алгоритм в кращу сторону чи ні?"

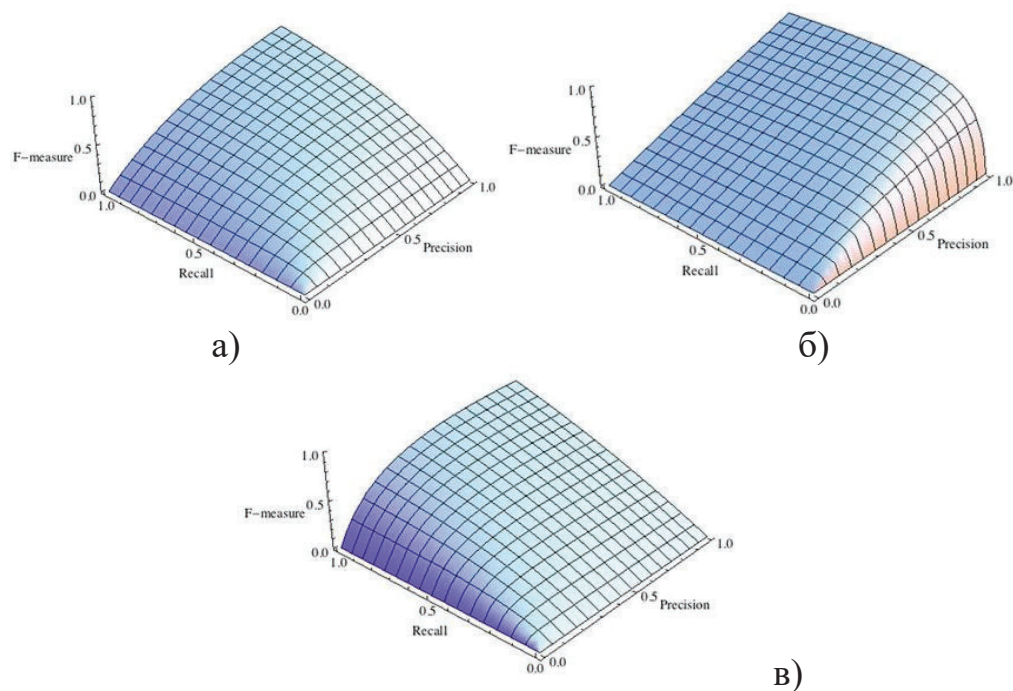


Рисунок 2.2 – Збалансована F-міра, $\beta=1$ (а); F-міра з пріоритетом точності, $\beta^2=1/4$ (б); F-міра с пріоритетом повноти, $\beta^2=2$ (в)

ROC-крива

Крива робочих характеристик (англ. Receiver Operating Characteristics curve). Використовується для аналізу поведінки класифікаторів при різних порогових значеннях. Дозволяє розглянути всі порогові значення для даного класифікатора. Показує частку хибно позитивних прикладів (англ. False positive rate, FPR) в порівнянні з часткою істинно позитивних прикладів (англ. True positive rate, TPR).

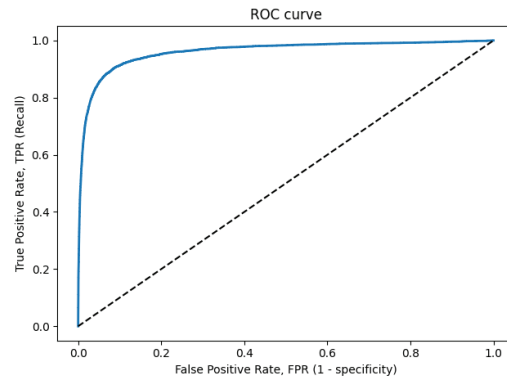


Рисунок 2.3 – ROC-крива

$$TPR = \frac{TP}{TP + FN} = Recall \quad (2.6)$$

$$FPR = \frac{FP}{FP + TN} \quad (2.7)$$

Частка FPR - це пропорція негативних зразків, які були некоректно класифіковані як позитивні.

$$FPR = 1 - TNR$$

де TNR - частка істинно негативних класифікацій (англ. True Negative Rate), що представляє собою пропорцію негативних зразків, які були коректно класифіковані як негативні.

Частка TNR також називається специфічністю (англ. Specificity). Отже, ROC-крива зображає чутливість (англ. Sensitivity), тобто повноту, в порівнянні з різницею $1 - specificity$.

Пряма лінія по діагоналі представляє ROC-криву чисто випадкового класифікатора. Хороший класифікатор тримається від зазначеної лінії настільки далеко, наскільки це можливо (прагнучи до лівого верхнього кута).

Один із способів порівняння класифікаторів передбачає вимір площі під кривою (англ. Area Under the Curve - AUC). Бездоганний класифікатор матиме площу під ROC-кривою (ROC-AUC), що дорівнює 1, тоді як чисто випадковий класифікатор - площу 0,5.

Графік ROC допомагає прийняти рішення про те, де встановити поріг класифікації, щоб максимізувати істинно позитивний рівень або мінімізувати псевдопозитивний показник, що в кінцевому підсумку є бізнес-рішенням.

Висновки по 2 розділу

Розглянуто типовий контейнер машинного навчання. На основі конвеєра машинного навчання розроблено методику реалізації детектора вторгнень.

Усі основні показники якості роботи детектора вторгнень будуються на основі матриці плутанини та дозволяють обирати оптимальним метод рішення задачі.

РОЗДІЛ 3. ДОСЛІДЖЕННЯ ПРАЦЕЗДАТНОСТІ ТА ЕФЕКТИВНОСТІ ДЕТЕКТОРА ВТОРГНЕНЬ

3.1. Розробка програми детектора вторгнень та його дослідження на моделях

Спочатку реалізуємо найпростіші методи виявлення аномалій Angle-based Outlier Detector (ABOD) та 'K Nearest Neighbors (KNN), щоб зрозуміти як вони працюють в двомірних випадках (лише дві ознаки) та оцінимо їх роботу на модельних прикладах.

Завантажуємо необхідні бібліотеки:

```
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
import matplotlib.font_manager

from pyod.models.abod import ABOD
from pyod.models.knn import KNN

import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
import matplotlib.font_manager
from pyod.utils.data import generate_data, get_outliers_inliers
```

Згенеруємо випадкові двохвимірні дані на яких будемо навчати та перевіряти алгоритми.

```
#generate random data with two features
X_train, Y_train = generate_data(n_train=200, train_only=True, n_features=2)
```

Для початку визначимо, що частка викидів дорівнює 0,1 від кількості згенерованих даних

```
outlier_fraction = 0.1

# Будемо зберігати викиди та внутрішні значення в різних масивах numpy
x_outliers, x_inliers = get_outliers_inliers(X_train, Y_train)

n_inliers = len(x_inliers)
n_outliers = len(x_outliers)
```

Розділяємо данні на дві функції

```
F1 = X_train[:, [0]].reshape(-1, 1)
F2 = X_train[:, [1]].reshape(-1, 1)

# Створюємо сітку з даних
xx, yy = np.meshgrid(np.linspace(-10, 10, 200), np.linspace(-10, 10, 200))
```

Будуємо діаграму розкиду, що має вигляд подібний до рис.3.1.

```
plt.scatter(F1,F2)
plt.xlabel('F1')
plt.ylabel('F2')
plt.show()
```

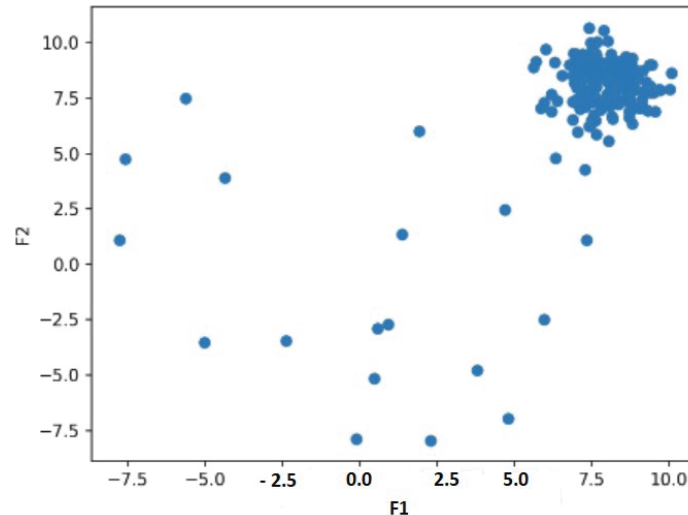


Рисунок 3.1 – Приклад модельованих вхідних даних

```
classifiers = {
    'Angle-based Outlier Detector (ABOD) ':
    ABOD(contamination=outlier_fraction),
    'K Nearest Neighbors (KNN) ':KNN(contamination=outlier_fraction)
}

# встановити розмір рисунку
plt.figure(figsize=(10, 10))

for i, (clf_name,clf) in enumerate(classifiers.items()) :
    # fit the dataset to the model
    clf.fit(X_train)
```

Бередбачаємо значення аномальне воно чи ні

```
# передбачити значення аномалії
scores_pred = clf.decision_function(X_train)*-1
```

Оцінюємо чи відповідає прогноз\з дійсності

```
y_pred = clf.predict(X_train)
```

Підраховуємо кількість помилок у передбаченні

```
n_errors = (y_pred != Y_train).sum()
print('No of Errors : ',clf_name, n_errors)
```

Далі будемо візуалізувати результати у вигляді графіків.

Встановлюємо порогове значення, щоб вважати точку даних звичайною або аномальною

```
threshold = stats.scoreatpercentile(scores_pred,100 *outlier_fraction)
```

Розраховуємо функцію прийняття рішення, що обчислює необроблену оцінку аномалії для кожної точки

```
Z = clf.decision_function(np.c_[xx.ravel(), yy.ravel()]) * -1
Z = Z.reshape(xx.shape)

subplot = plt.subplot(1, 2, i + 1)
```

Заповнюємо синьою кольоровою картою від мінімального балу аномальності до порогового значення. (чим синіше тим більша ймовірність аномальності).

```
subplot.contourf(xx, yy, Z, levels = np.linspace(Z.min(), threshold,
10), cmap=plt.cm.Blues_r)
```

Червоною контурною лінію позначаємо, де показник аномалії дорівнює пороговому значенню.

```
a = subplot.contour(xx, yy, Z, levels=[threshold], linewidths=2,
colors='red')
```

Заповнюємо помаранчевим діапазон оцінки аномальності, що знаходиться від порогового значення до максимальної оцінки аномальності.

```
subplot.contourf(xx, yy, Z, levels=[threshold, Z.max()], colors='orange')
```

Точно визначені нормальні значення позначаємо білими крапками

```
b = subplot.scatter(X_train[:-n_outliers, 0], X_train[:-n_outliers, 1],
c='white', s=20, edgecolor='k')
```

Точно визначені викиди позначаємо чорними крапками

```
c = subplot.scatter(X_train[-n_outliers:, 0], X_train[-n_outliers:, 1],
c='black', s=20, edgecolor='k')
subplot.axis('tight')
```

```
subplot.legend(
[a.collections[0], b, c],
['learned decision function', 'true inliers', 'true outliers'],
prop=matplotlib.font_manager.FontProperties(size=10),
loc='lower right')
```

```
subplot.set_title(clf_name)
subplot.set_xlim((-10, 10))
subplot.set_ylim((-10, 10))
plt.show()
```

У результаті роботи програми отримаємо графіки, такі як зображені на рис. 3.2. Повний код програми наведений у додатку А.

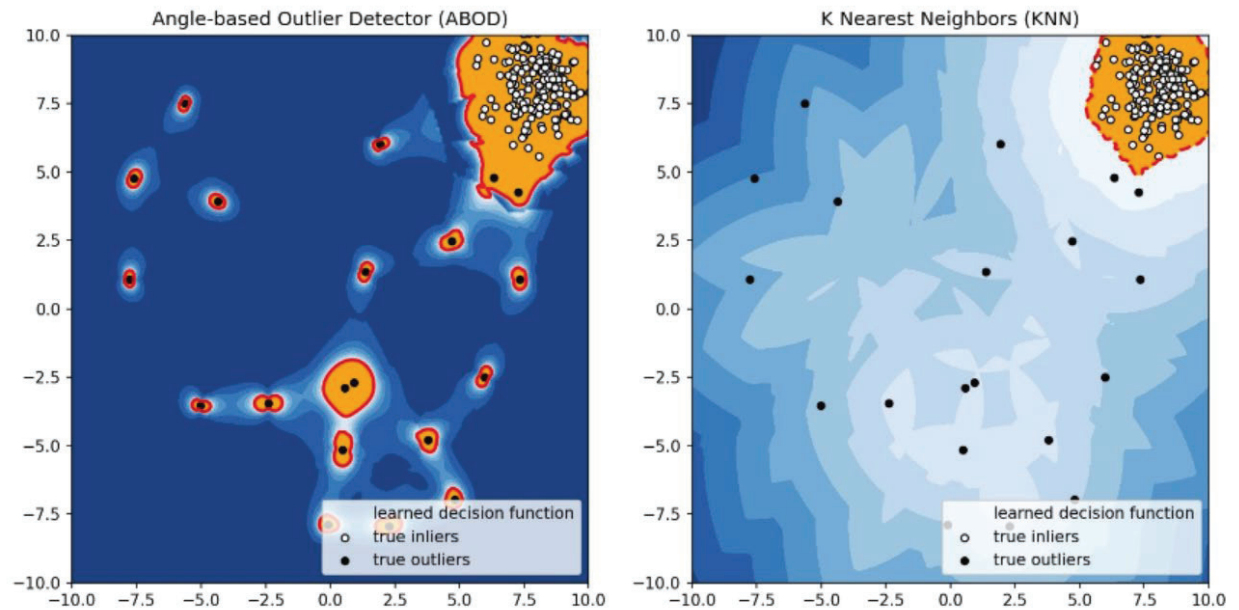


Рисунок 3.2 – Приклад результатів роботи методів ABOD та KNN

Із проведеного дослідження видно, що навіть для простих змодельованих вхідних даних результати роботи алгоритмів можуть дуже суттєво відрізнятися. Тобто різні методи дають оцінки різної якості, а тому має сенс порівняти різні методи PyOD між собою.

Розробимо програмне забезпечення для порівняння методів виявлення викидів PyOD.

Завантажуємо бібліотеки:

```
from __future__ import division
from __future__ import print_function

import os
import sys
from time import time

import numpy as np
from numpy import percentile
import matplotlib.pyplot as plt
import matplotlib.font_manager

# Імпорт усіх моделей
from pyod.models.abod import ABOD
from pyod.models.cblof import CBLOF
from pyod.models.feature_bagging import FeatureBagging
from pyod.models.hbos import HBOS
from pyod.models.iforest import IForest
from pyod.models.knn import KNN
from pyod.models.lof import LOF
from pyod.models.mcd import MCD
from pyod.models.ocsvm import OCSVM
from pyod.models.pca import PCA
from pyod.models.lscp import LSCP
```

```

from pyod.models.inne import INNE
from pyod.models.gmm import GMM
from pyod.models.kde import KDE
from pyod.models.lmdd import LMDD

```

Задаємо кількість вхідних значень та процентне співвідношення викидів.

```

n_samples = 200
outliers_fraction = 0.25
clusters_separation = [0]

```

Програма для порівняння результатів роботи детекторів за заданих налаштувань.

Здійснюємо ініціалізацію даних

```

xx, yy = np.meshgrid(np.linspace(-7, 7, 100), np.linspace(-7, 7, 100))
n_inliers = int((1. - outliers_fraction) * n_samples)
n_outliers = int(outliers_fraction * n_samples)
ground_truth = np.zeros(n_samples, dtype=int)
ground_truth[-n_outliers:] = 1

```

Здійснюємо ініціалізація набору детекторів

```

detector_list = [LOF(n_neighbors=5), LOF(n_neighbors=10), LOF(n_neighbors=15),
                 LOF(n_neighbors=20), LOF(n_neighbors=25), LOF(n_neighbors=30),
                 LOF(n_neighbors=35), LOF(n_neighbors=40), LOF(n_neighbors=45),
                 LOF(n_neighbors=50)]

```

Відображаємо статистику даних

```

print('Number of inliers: %i' % n_inliers)
print('Number of outliers: %i' % n_outliers)
print('Ground truth shape is {shape}. Outlier are 1 and inliers are 0.\n'.format(shape=ground_truth.shape))
print(ground_truth)

random_state = np.random.RandomState(42)

```

Визначення дев'яти інструментів виявлення викидів для порівняння

```

classifiers = {
    'Angle-based Outlier Detector (ABOD)':
        ABOD(contamination=outliers_fraction),
    'Cluster-based Local Outlier Factor (CBLOF)':
        CBLOF(contamination=outliers_fraction,
               check_estimator=False, random_state=random_state),
    'Feature Bagging':
        FeatureBagging(LOF(n_neighbors=35),
                       contamination=outliers_fraction,
                       random_state=random_state),
    'Histogram-base Outlier Detection (HBOS)': HBOS(
        contamination=outliers_fraction),
    'Isolation Forest': IForest(contamination=outliers_fraction,
                                random_state=random_state),
    'K Nearest Neighbors (KNN)': KNN(
        contamination=outliers_fraction),
}

```

```

'Average KNN': KNN(method='mean',
                    contamination=outliers_fraction),
'Local Outlier Factor (LOF)':
    LOF(n_neighbors=35, contamination=outliers_fraction),
'Minimum Covariance Determinant (MCD)': MCD(
    contamination=outliers_fraction, random_state=random_state),
'One-class SVM (OCSVM)': OCSVM(contamination=outliers_fraction),
'Principal Component Analysis (PCA)': PCA(
    contamination=outliers_fraction, random_state=random_state),
'Locally Selective Combination (LSCP)': LSCP(
    detector_list, contamination=outliers_fraction,
    random_state=random_state),
'INNE': INNE(contamination=outliers_fraction),
'GMM': GMM(contamination=outliers_fraction),
'KDE': KDE(contamination=outliers_fraction),
'LMDD': LMDD(contamination=outliers_fraction),
}

```

Формуємо цикл для відображення результатів роботи всіх детекторів.

```

for i, clf in enumerate(classifiers.keys()):
    print('Model', i + 1, clf)

```

У циклі здійснюємо навчання моделей на згенерованих даних та видаємо результати для порівняння характеристик моделі.

```

for i, offset in enumerate(clusters_separation):
    np.random.seed(42)
    # Генерація даних
    X1 = 0.3 * np.random.randn(n_inliers // 2, 2) - offset
    X2 = 0.3 * np.random.randn(n_inliers // 2, 2) + offset
    X = np.r_[X1, X2]
    # Додавання викидів
    X = np.r_[X, np.random.uniform(low=-6, high=6, size=(n_outliers, 2))]

    # Навчання моделі
    plt.figure(figsize=(15, 16))
    for i, (clf_name, clf) in enumerate(classifiers.items()):
        print(i + 1, 'fitting', clf_name)
        # fit the data and tag outliers
        clf.fit(X)
        scores_pred = clf.decision_function(X) * -1
        y_pred = clf.predict(X)
        threshold = percentile(scores_pred, 100 * outliers_fraction)
        n_errors = (y_pred != ground_truth).sum()
        # побудова лінії рівнів і точок

        Z = clf.decision_function(np.c_[xx.ravel(), yy.ravel()]) * -1
        Z = Z.reshape(xx.shape)
        subplot = plt.subplot(4, 4, i + 1)
        subplot.contourf(xx, yy, Z, levels=np.linspace(Z.min(), threshold, 7),
                        cmap=plt.cm.Blues_r)
        # a = subplot.contour(xx, yy, Z, levels=[threshold],
        #                      linewidths=2, colors='red')
        subplot.contourf(xx, yy, Z, levels=[threshold, Z.max()],
                        colors='orange')
        b = subplot.scatter(X[:-n_outliers, 0], X[:-n_outliers, 1], c='white',
                           s=20, edgecolor='k')
        c = subplot.scatter(X[-n_outliers:, 0], X[-n_outliers:, 1], c='black',
                           s=20, edgecolor='k')
        subplot.axis('tight')

```

```

subplot.legend(
[
# a.collections[0],
b, c],
[
# 'навчена функція прийняття рішень',
'true inliers', 'true outliers'],
prop=matplotlib.font_manager.FontProperties(size=10),
loc='lower right')
subplot.set_xlabel("%d. %s (errors: %d)" % (i + 1, clf_name, n_errors))
subplot.set_xlim((-7, 7))
subplot.set_ylim((-7, 7))
plt.subplots_adjust(0.04, 0.1, 0.96, 0.94, 0.1, 0.26)
plt.suptitle("Outlier detection")
plt.show()

```

У результаті виконання програми отримаємо набір графіків з виділеними аномаліями

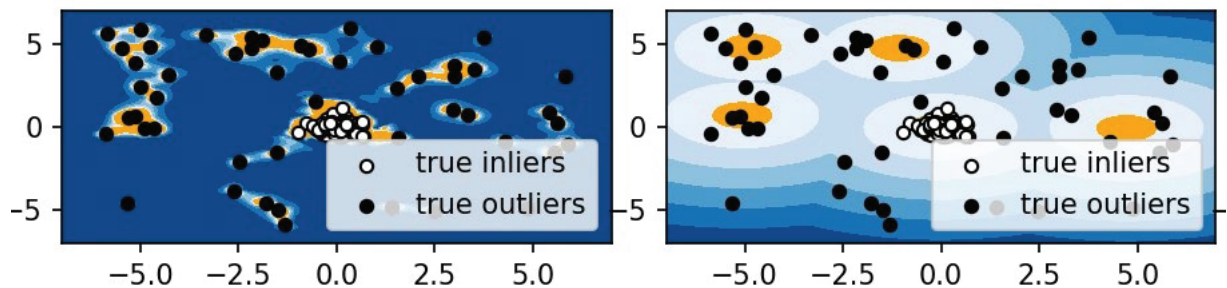


Рисунок 3.3 – Приклад результатів роботи методів Angle-based Outlier Detector (ABOD), Cluster-based Local Outlier Factor (CBLOF)

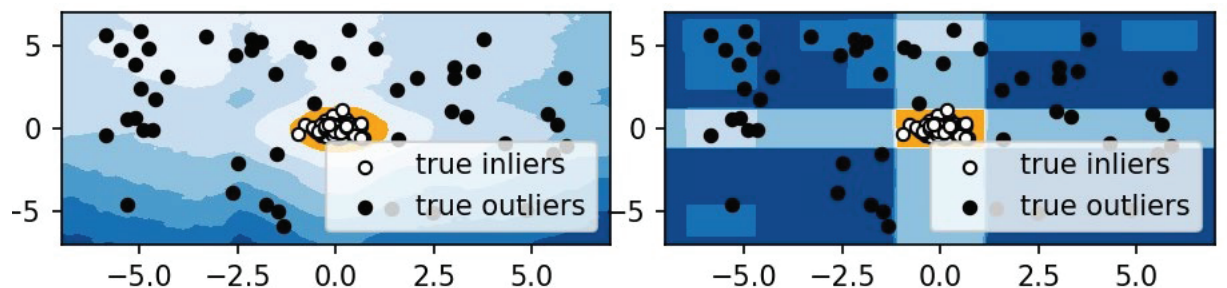


Рисунок 3.4 – Приклад результатів роботи методів Feature Bagging, Histogram-based Outlier Detection (HBOS)

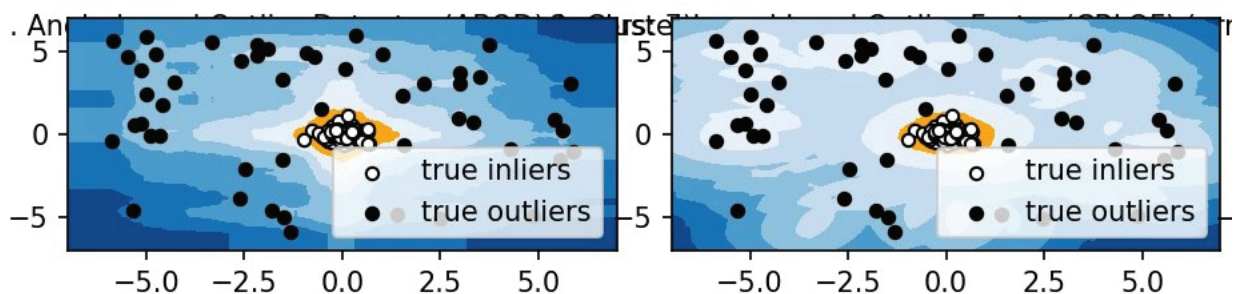


Рисунок 3.5 – Приклад результатів роботи методів Isolation Forest (IF), K Nearest Neighbors (KNN)

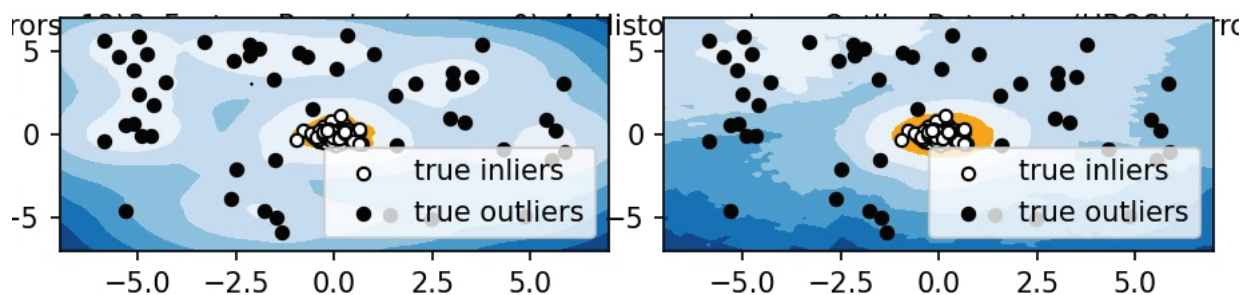


Рисунок 3.6 – Приклад результатів роботи методів Average KNN, Local Outlier Factor (LOF)

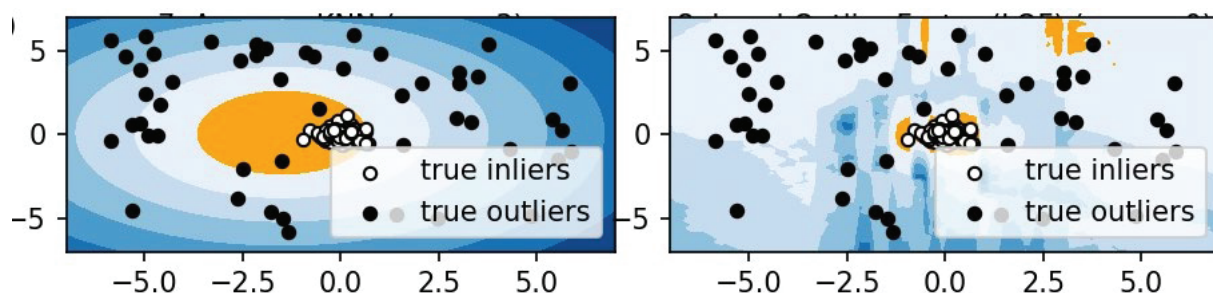


Рисунок 3.7 – Приклад результатів роботи методів Minimum Covariance Determinant (MCD), One-class SVM (OCSVM)

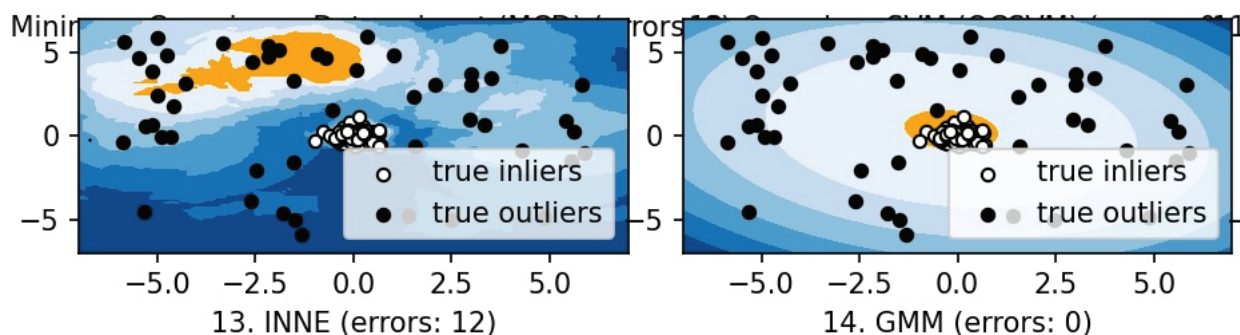


Рисунок 3.8 – Приклад результатів роботи методів Principal Component Analysis (PCA), Locally Selective Combination (LSCP)

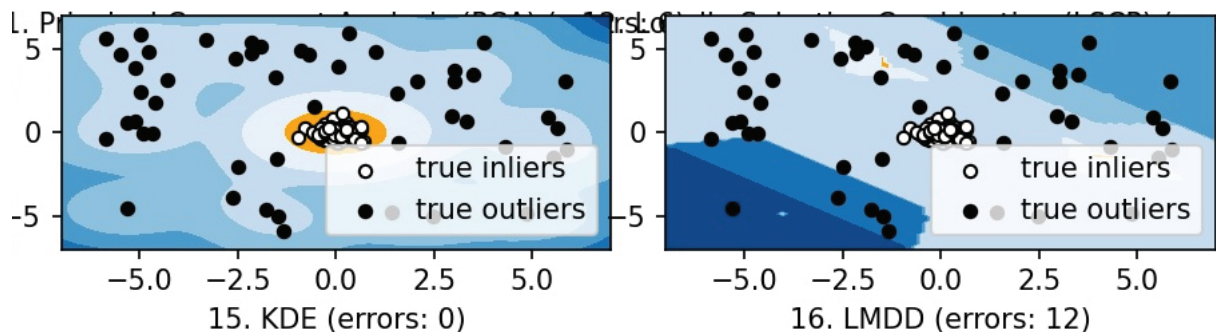


Рисунок 3.9 – Приклад результатів роботи методів , INNE, GMM

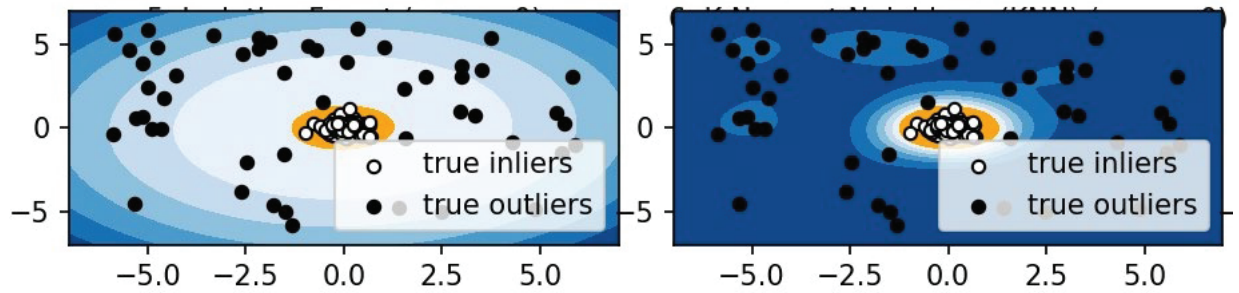


Рисунок 3.10 – Приклад результатів роботи методів KDE, LMDD

Як видно з рисунків кожен з цих методів з тою чи іншою похбкою може виявляти прості вторгнення. Однак ситуація значно ускладнюється коли маємо справу з реальними наборами даних, коли застосовується велика кількість ознак та велика кількість прикладів для навчання. І тут вже не обійтися без машинного навчання.

3.2. Дослідження детектора вторгнень на наборі даних

Для навчання моделей будемо використовувати набір даних KDD Cup.

Детектор аномалій(вторгнень) KDD - програмне забезпечення для виявлення вторгнень у мережу захищає комп'ютерну мережу від несанкціонованих користувачів, включаючи, можливо, інсайдерів. Навчальним завданням детектора вторгнень є побудова прогнозової моделі (тобто класифікатора), здатної розрізняти ``погані" з'єднання, які називаються вторгненнями або атаками, і ``хороші" звичайні з'єднання.

Вихідні навчальні дані становили близько чотирьох гігабайт стиснених двійкових даних дампа TCP за сім тижнів мережевого трафіку. Це було оброблено приблизно в п'ять мільйонів записів про з'єднання. Подібним чином, два тижні тестових даних дали близько двох мільйонів записів про підключення.

З'єднання - це послідовність TCP-пакетів, що починаються і закінчуються в певний час, між якими дані надходять до вихідної IP-адреси та до цільової IP-адреси за деяким чітко визначеним протоколом. Кожне з'єднання

позначене як звичайне або як атака з точно одним конкретним типом атаки. Кожен запис з'єднання складається приблизно із 100 байт.

Вторгнення діляться на чотири основні категорії:

1. DOS: відмова в обслуговуванні, напр. синхронний флуд;
2. R2L: несанкціонований доступ з віддаленої машини, наприклад вгадування пароля;
3. U2R: несанкціонований доступ до локальних привілеїв суперкористувача (кореня), наприклад, різні атаки “переповнення буфера”;
4. Probing: спостереження та інші зондування, наприклад, сканування портів.

Важливо зазначити, що дані тесту не походять з того самого розподілу ймовірностей, що й дані навчання, і вони включають конкретні типи атак, що не містяться в даних тренувань. Це робить завдання більш реалістичним. Деякі експерти про втручання вважають, що більшість нових атак є варіантами відомих атак, і "підпису" відомих атак може бути достатнім для виявлення нових варіантів. Набори даних містять загалом 24 типи навчальних атак, з додатковими 14 типами лише в тестових даних.

Stolfo та ін. визначені функції вищого рівня, які допомагають відрізнити звичайні зв'язки від атак. Існує кілька категорій похідних ознак.

Функції «того самого хоста» перевіряють лише з'єднання за останні дві секунди, які мають той самий хост призначення, що і поточне з'єднання, і обчислюють статистику, пов'язану з поведінкою протоколу, службою тощо.

Подібні функції «тієї ж служби» перевіряють лише ті з'єднання за останні дві секунди, які мають ту саму послугу, що і поточне з'єднання.

Функції «того самого хоста» і «тієї ж служби» разом називаються функціями трафіку, що базуються на часі, у записах з'єднання.

Деякі зондуючі атаки сканують хости (або порти) із значно більшим інтервалом часу, ніж дві секунди, наприклад раз на хвилину. Тому записи з'єднання також були відсортовані за хостом призначення, а функції побудовані

за допомогою вікна із 100 підключень до того самого хоста замість часового вікна. Це дає набір так званих особливостей трафіку на основі хоста.

На відміну від більшості атак DOS та зондування, схоже, немає послідовних шаблонів, які часто зустрічаються у записах атак R2L та U2R. Це пов'язано з тим, що атаки DOS та зондування передбачають багато з'єднань з деякими хостами за дуже короткий проміжок часу, але атаки R2L та U2R вбудовані в частини даних пакетів і зазвичай включають лише одне з'єднання.

Корисні алгоритми автоматичного видобування неструктурованих частин даних пакетів - це відкрите питання дослідження. Stolfo та ін. використовували знання домену для додавання функцій, які шукають підозрілу поведінку в частинах даних, наприклад кількість невдалих спроб входу. Ці функції називаються функціями «вмісту»[13].

Для кращого розуміння, повний перелік набору функцій приведений в табл. 3.1 - 3.3.

Таблиця 3.1

Основні особливості окремих з'єднань TCP

Назва функції	Опис	Тип
duration	тривалість (кількість секунд) з'єднання	безперервна
protocol_type	тип протоколу, напр. tcp, udp тощо	дискретна
service	послуга мережі за призначенням, наприклад, http, telnet тощо.	дискретна
src_bytes	кількість байтів даних від джерела до пункту призначення	безперервна
dst_bytes	кількість байтів даних від пункту призначення до джерела	безперервна
flag	нормальний або стан помилки з'єднання	дискретна
land	1 якщо з'єднання здійснюється з / до одного хосту / порту; 0 інакше	дискретна
wrong_fragment	кількість ``неправильних'' фрагментів	безперервна
urgent	number of urgent packets	безперервна

Таблиця 3.2

Функції вмісту в межах з'єднання, запропонованого доменом

Назва функції	Опис	Тип
hot	кількість «гарячих» показників	безперервна
num_failed_logins	кількість невдалих спроб входу	безперервна

logged_in	1 при успішному вході в систему; 0 інакше	дискретна
num_compromised	кількість «скомпрометованих» умов	безперервна
root_shell	1 якщо отримана коренева оболонка; 0 інакше	дискретна
su_attempted	1 при спробі команди «su root»; 0 інакше	дискретна
num_root	кількість «корневих» доступів	безперервна
num_file_creations	кількість операцій створення файлів	безперервна
num_shells	кількість підказок оболонки	безперервна
num_access_files	кількість операцій з файлами контролю доступу	безперервна
num_outbound_cmds	кількість вихідних команд у сеансі ftp	безперервна
is_hot_login	1, якщо логін належить до «гарячого» списку; 0 інакше	дискретна
is_guest_login	1, якщо логін є «гостьовим» логіном; 0 інакше	дискретна

Таблиця 3.3

Характеристики трафіку, обчислені за допомогою двосекундного часового вікна

Назва функції	Опис	Тип
count	кількість підключень до того самого хосту, що і поточне підключення за останні дві секунди	безперервна
serror_rate	% з'єднань, які мають помилки «SYN»	безперервна
rerror_rate	% з'єднань, які мають помилки «REJ»	безперервна
same_srv_rate	% підключень до тієї самої послуги	безперервна
diff_srv_rate	% підключень до різних служб	безперервна
srv_count	кількість підключень до тієї самої служби, що і поточне з'єднання за останні дві секунди	безперервна
srv_diff_host_rate	% of connections to different hosts	безперервна

Завантажуємо цей набір даних та проводимо його обробку.

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list
# all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# Any results you write to the current directory are saved as output.

import pandas as pd
from tensorflow.keras.utils import get_file
```

```

try:
    path = get_file('kddcup.data_10_percent.gz', origin='http://kdd.ics.uci.edu/databases/kddcup99/kddcup.data_10_percent.gz')
except:
    print('Error downloading')
    raise

print(path)

# This file is a CSV, just no CSV extension or headers
# Download from: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html
df = pd.read_csv(path, header=None)

# The CSV file has no column heads, so add them
df.columns = [
    'duration',
    'protocol_type',
    'service',
    'flag',
    'src_bytes',
    'dst_bytes',
    'land',
    'wrong_fragment',
    'urgent',
    'hot',
    'num_failed_logins',
    'logged_in',
    'num_compromised',
    'root_shell',
    'su_attempted',
    'num_root',
    'num_file_creations',
    'num_shells',
    'num_access_files',
    'num_outbound_cmds',
    'is_host_login',
    'is_guest_login',
    'count',
    'srv_count',
    'serror_rate',
    'srv_serror_rate',
    'rerror_rate',
    'srv_rerror_rate',
    'same_srv_rate',
    'diff_srv_rate',
    'srv_diff_host_rate',
    'dst_host_count',
    'dst_host_srv_count',
    'dst_host_same_srv_rate',
    'dst_host_diff_srv_rate',
    'dst_host_same_src_port_rate',
    'dst_host_srv_diff_host_rate',
    'dst_host_serror_rate',
    'dst_host_srv_serror_rate',
    'dst_host_rerror_rate',
    'dst_host_srv_rerror_rate',
    'outcome'
]

```

Для ознайомлення з даними сетом виведемо у таблицю частину даних

```
df[0:19289]
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_srv_c
0	0	tcp	http	SF	181	5450	0	0	0	0	...	9
1	0	tcp	http	SF	239	486	0	0	0	0	...	19
2	0	tcp	http	SF	235	1337	0	0	0	0	...	29
3	0	tcp	http	SF	219	1337	0	0	0	0	...	39
4	0	tcp	http	SF	217	2032	0	0	0	0	...	49
...	...	...	...	...	...	...	...	...	...	...	...	...
19284	0	tcp	smtp	SF	1089	492	0	0	0	0	...	242
19285	0	tcp	smtp	SF	1204	499	0	0	0	0	...	242
19286	0	udp	private	SF	28	0	0	1	0	0	...	1
19287	0	udp	private	SF	28	0	0	3	0	0	...	2
19288	0	udp	private	SF	28	0	0	3	0	0	...	3

Рисунок 3.11 – Загальний вигляд таблиці даних

Проаналізуємо набір даних

```
print("Read {} rows.".format(len(df)))
print('='*40)
print('The number of data points are:', df.shape[0])
print('='*40)
print('The number of features are:', df.shape[1])
print('='*40)
output = df['outcome'].values
labels = set(output)
print('The different type of output labels are:', labels)
print('='*125)
print('The number of different output labels are:', len(labels))
```

Результати

Read 494021 rows.

=====

The number of data points are: 494021

=====

The number of features are: 42

=====

The different type of output labels are: {'loadmodule.', 'smurf.', 'rootkit.', 'guess_passwd.', 'pod.', 'teardrop.', 'normal.', 'perl.', 'nmap.', 'spy.', 'buffer_overflow.', 'land.', 'back.', 'warezclient.', 'phf.', 'satan.', 'imap.', 'ipsweep.', 'portsweep.', 'neptune.', 'ftp_write.', 'multihop.', 'warezmaster.'}

=====

=====

The number of different output labels are: 23

```
print('Null values in dataset are', len(df[df.isnull().any(1)]))
print('='*40)
```

```
# Checkng for DUPLICATE values
```

```
df.drop_duplicates(keep='first', inplace = True)
```

```
# For now, just drop NA's (rows with missing values)
```

```
df.dropna(inplace=True,axis=1)
```

```
# stored the data into a pickle file so we can load through
```

```
# df.to_pickle('df.pkl')
```

```
print("Read {} rows.".format(len(df)))
```

```
# Exploratory data analysis
```

```
import matplotlib.pyplot as plt
from matplotlib.pyplot import *

plt.figure(figsize=(15,7))
class_distribution = df['outcome'].value_counts()
class_distribution.plot(kind='bar')
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in train data')
plt.grid()
plt.show()
```

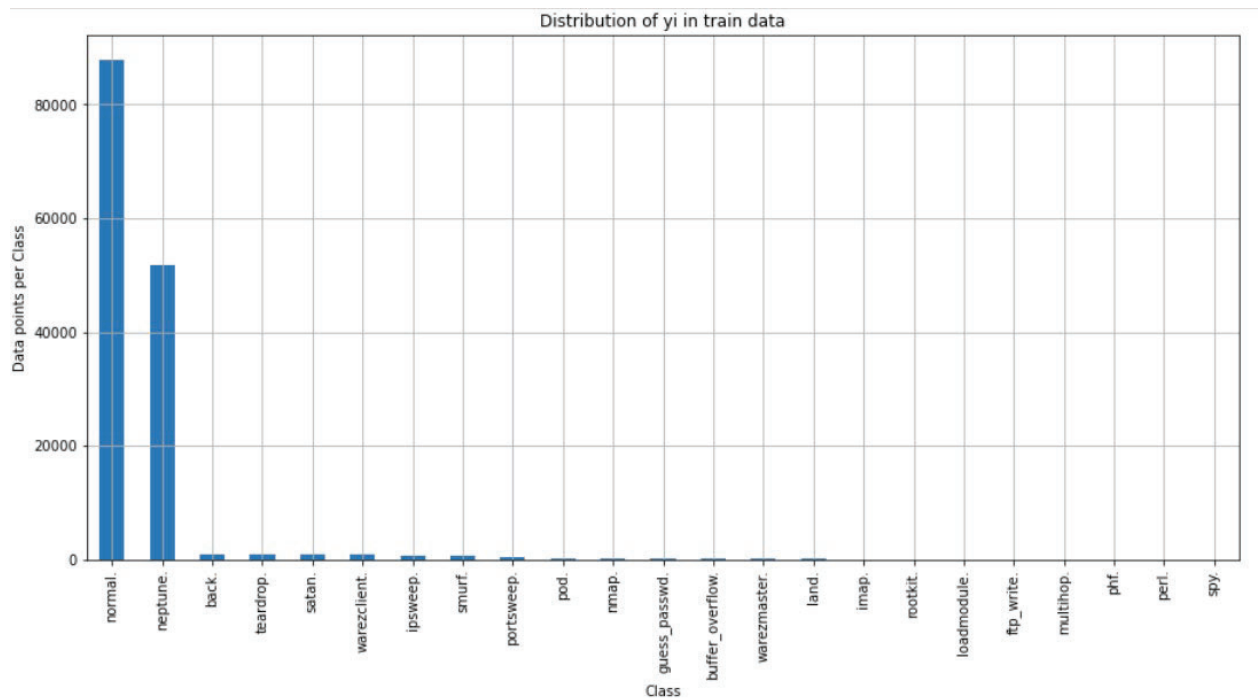


Рисунок 3.12 – Розподіл даних в наборі

```
sorted_yi = np.argsort(-class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', class_distribution.index[i],':',
          class_distribution.values[i],
          '(', np.round((class_distribution.values[i]/df.shape[0]*100), 3), '%)')

```

```
#df.groupby('outcome')['outcome'].count() #this could also be used if you want no-formatted for above.
```

```
ENCODING = 'utf-8'
```

```
def expand_categories(values):
    result = []
    s = values.value_counts()
    t = float(len(values))
    for v in s.index:
        result.append("{}:{}".format(v, round(100*(s[v]/t), 2)))
    return "[{}]" .format(",".join(result))
```

```
def analyze(df):
    print()
    cols = df.columns.values
    total = float(len(df))
```

```

print("{} rows".format(int(total)))
for col in cols:
    uniques = df[col].unique()
    unique_count = len(uniques)
    if unique_count>100:
        print("*** {}:{} ({}%)".format(col,unique_count,int(((unique_count
)/total)*100)))
    else:
        print("*** {}:{}".format(col,expand_categories(df[col])))
        expand_categories(df[col])

# Analyze KDD-99

import pandas as pd
import os
import numpy as np
from sklearn import metrics
from scipy.stats import zscore

analyze(df)

145586 rows
** duration:2495 (1%)
** protocol_type:[tcp:89.92%,udp:8.43%,icmp:1.65%]

```

Декодування вектора ознак

```

# Encode a numeric column as zscores
def encode_numeric_zscore(df, name, mean=None, sd=None):
    if mean is None:
        mean = df[name].mean()

    if sd is None:
        sd = df[name].std()

    df[name] = (df[name] - mean) / sd

# Encode text values to dummy variables(i.e. [1,0,0],[0,1,0],[0,0,1] for red,
green,blue)
def encode_text_dummy(df, name):
    dummies = pd.get_dummies(df[name])
    for x in dummies.columns:
        dummy_name = f"{name}-{x}"
        df[dummy_name] = dummies[x]
    df.drop(name, axis=1, inplace=True)

encode_numeric_zscore(df, 'duration')
encode_text_dummy(df, 'protocol_type')
encode_text_dummy(df, 'service')
encode_text_dummy(df, 'flag')
encode_numeric_zscore(df, 'src_bytes')
encode_numeric_zscore(df, 'dst_bytes')
encode_text_dummy(df, 'land')
encode_numeric_zscore(df, 'wrong_fragment')
encode_numeric_zscore(df, 'urgent')
encode_numeric_zscore(df, 'hot')
encode_numeric_zscore(df, 'num_failed_logins')
encode_text_dummy(df, 'logged_in')
encode_numeric_zscore(df, 'num_compromised')
encode_numeric_zscore(df, 'root_shell')
encode_numeric_zscore(df, 'su_attempted')
encode_numeric_zscore(df, 'num_root')
encode_numeric_zscore(df, 'num_file_creations')

```

```

encode_numeric_zscore(df, 'num_shells')
encode_numeric_zscore(df, 'num_access_files')
encode_numeric_zscore(df, 'num_outbound_cmds')
encode_text_dummy(df, 'is_host_login')
encode_text_dummy(df, 'is_guest_login')
encode_numeric_zscore(df, 'count')
encode_numeric_zscore(df, 'srv_count')
encode_numeric_zscore(df, 'serror_rate')
encode_numeric_zscore(df, 'srv_serror_rate')
encode_numeric_zscore(df, 'rerror_rate')
encode_numeric_zscore(df, 'srv_rerror_rate')
encode_numeric_zscore(df, 'same_srv_rate')
encode_numeric_zscore(df, 'diff_srv_rate')
encode_numeric_zscore(df, 'srv_diff_host_rate')
encode_numeric_zscore(df, 'dst_host_count')
encode_numeric_zscore(df, 'dst_host_srv_count')
encode_numeric_zscore(df, 'dst_host_same_srv_rate')
encode_numeric_zscore(df, 'dst_host_diff_srv_rate')
encode_numeric_zscore(df, 'dst_host_same_src_port_rate')
encode_numeric_zscore(df, 'dst_host_srv_diff_host_rate')
encode_numeric_zscore(df, 'dst_host_serror_rate')
encode_numeric_zscore(df, 'dst_host_srv_serror_rate')
encode_numeric_zscore(df, 'dst_host_rerror_rate')
encode_numeric_zscore(df, 'dst_host_srv_rerror_rate')

# display 5 rows

df.dropna(inplace=True,axis=1)
df[0:5]
# This is the numeric feature vector, as it goes to the neural net

```

	duration	src_bytes	dst_bytes	wrong_fragment	urgent	hot	num_failed_logins	num_compromised	root_shell	su_attempted	...	flag-S3	flag-SF	flag-SH	lai
0	-0.10785	-0.004293	0.042595	-0.084394	-0.004737	-0.070209	-0.018022	-0.007905	-0.01944	-0.008613	...	0	1	0	1
1	-0.10785	-0.004261	-0.039036	-0.084394	-0.004737	-0.070209	-0.018022	-0.007905	-0.01944	-0.008613	...	0	1	0	1
2	-0.10785	-0.004263	-0.025042	-0.084394	-0.004737	-0.070209	-0.018022	-0.007905	-0.01944	-0.008613	...	0	1	0	1
3	-0.10785	-0.004272	-0.025042	-0.084394	-0.004737	-0.070209	-0.018022	-0.007905	-0.01944	-0.008613	...	0	1	0	1
4	-0.10785	-0.004273	-0.013613	-0.084394	-0.004737	-0.070209	-0.018022	-0.007905	-0.01944	-0.008613	...	0	1	0	1

Рисунок 3.13 – Нормалізація даних в наборі

```

# Convert to numpy - Classification
x_columns = df.columns.drop('outcome')
x = df[x_columns].values
dummies = pd.get_dummies(df['outcome']) # Classification
outcomes = dummies.columns
num_classes = len(outcomes)
y = dummies.values

df.groupby('outcome')['outcome'].count()

```

Виконаємо два види класифікації: двійкову та мультикласифікацію.

```

# create our target classifications
binary_y = df['attack_flag']
multi_y = df['attack_map']

test_binary_y = test_df['attack_flag']
test_multi_y = test_df['attack_map']

# build the training sets
binary_train_X, binary_val_X, binary_train_y, binary_val_y = train_test_split(
    to_fit, binary_y, test_size=0.6)

```

```
multi_train_X, multi_val_X, multi_train_y, multi_val_y = train_test_split(to_fit, multi_y, test_size = 0.6)
```

Навчання моделі

Виходячи з характеру даних, які ми бачили вище, дерева рішень є хорошою відправною точкою для створення прогнозних моделей. У цьому випадку ми будемо використовувати випадковий ліс для створення та об'єднання кількох дерев. Ми почнемо з того, що просто візьмемо значення за замовчуванням.

```
# model for the binary classification
binary_model = RandomForestClassifier()
binary_model.fit(binary_train_X, binary_train_y)
binary_predictions = binary_model.predict(binary_val_X)

# calculate and display our base accuracy
base_rf_score = accuracy_score(binary_predictions, binary_val_y)
base_rf_score
```

```
0.9936097586790855
```

Отримали 99% точності з першої спроби.

Проаналізуємо, як різні моделі порівнюють із таким набором даних. Це досить легко зробити за допомогою `cross_val_score`.

```
# define the list of models that we want to test
models = [
    RandomForestClassifier(),
    LogisticRegression(max_iter=250),
    KNeighborsClassifier(),
]

# an empty list to capture the performance of each model
model_comps = []
# walk through the models and populate our list
for model in models:
    model_name = model.__class__.__name__
    accuracies = cross_val_score(model, binary_train_X, binary_train_y, scoring='accuracy')
    for count, accuracy in enumerate(accuracies):
        model_comps.append((model_name, count, accuracy))

# a box plot will do well to show us overall performance and the variation in the models.
result_df = pd.DataFrame(model_comps, columns=['model_name', 'count', 'accuracy'])
result_df.pivot(index='count', columns='model_name', values='accuracy').boxplot(
    rot=45)
```

З рис. 3.14 видно певну невідповідність між моделями. Випадковий ліс і К-найближчі сусіди є тісними групами з хорошою продуктивністю. Логістична регресія не спрацювала. Частково це може бути тому, що ми не провели достатню попередню обробку наших даних, щоб надати їм форму, оптимізовану для цієї моделі.

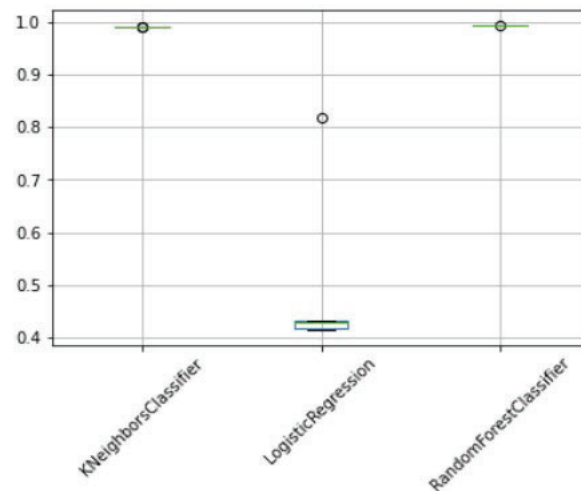


Рисунок 3.14 – Порівняння трьох методів

Оцінимо, як справдилися наші прогнози. Розробимо допоміжну функцію, щоб отримати деякі відповідні показники якості наших результатів.

```
# a helper function for getting some analytical data about our predictions
def add_predictions(data_set, predictions, y):
    prediction_series = pd.Series(predictions, index=y.index)

    # we need to add the predicted and actual outcomes to the data
    predicted_vs_actual = data_set.assign(predicted=prediction_series)
    original_data = predicted_vs_actual.assign(actual=y).dropna()
    conf_matrix = confusion_matrix(original_data['actual'],
                                   original_data['predicted'])

    base_errors = original_data[original_data['actual'] != original_data['predicted']]

    non_zeros = base_errors.loc[:, (base_errors != 0).any(axis=0)]

    false_positives = non_zeros.loc[non_zeros.actual==0]
    false_negatives = non_zeros.loc[non_zeros.actual==1]

    # put everything into an object
    prediction_data = {'data': original_data,
                       'confusion_matrix': conf_matrix,
                       'errors': base_errors,
                       'non_zeros': non_zeros,
                       'false_positives': false_positives,
                       'false_negatives': false_negatives}

    return prediction_data
```

Тепер ми можемо ближче подивитися на наші результати. Перше, що ми можемо зробити, це поглянути на матрицю плутанини, яка в цьому випадку відобразить прогнозовану класифікацію на фактичну класифікацію.

```
# capture our prediction data
binary_prediction_data = add_predictions(df,
                                         binary_predictions,
                                         binary_val_y)

# create a heatmap of the confusion matrix
```

```
sns.heatmap(data=binary_prediction_data['confusion_matrix'],
            xticklabels = ['Predicted Normal','Predicted Attack'],
            yticklabels = ['Actual Normal','Actual Attack'],
            cmap="YlGnBu",
            fmt='d',
            annot=True)
```

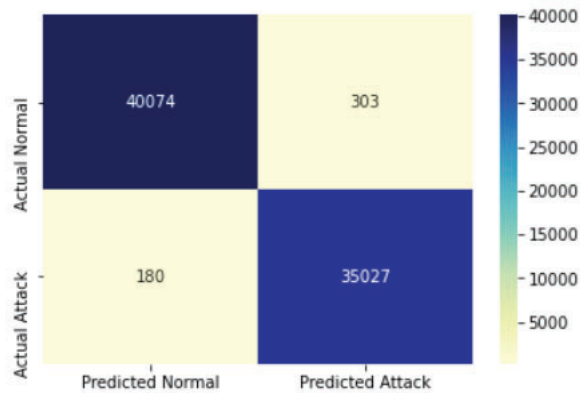


Рисунок 3.15 – Матриця плутанини для двохкласової класифікації

На рис 3.15 ми бачимо багато хибних спрацьовувань (звичайний трафік, який було позначено як атаку) і хибно-негативних (трафік атаки, який було позначено як нормальний).

Отже, необхідно трохи більше вивчити помилки передбачення та подивитися, чи можна отримати більше інформації та зменшити помилки (рис.3.16).

```
# dataframe to store incorrect classification
binary_prediction_data['errors'].describe()
```

	duration	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	num_failed_logins	logged_in	num_compromised	...
count	483.000000	4.830000e+02	4.830000e+02	483.000000	483.0	483.0	483.000000	483.000000	483.000000	483.000000	...
mean	57.188406	1.492597e+06	7.310814e+03	0.006211	0.0	0.0	0.115942	0.010352	0.084886	0.047619	...
std	673.241958	3.155596e+07	7.814234e+04	0.078647	0.0	0.0	0.912690	0.227508	0.279001	0.427054	...
min	0.000000	0.000000e+00	0.000000e+00	0.000000	0.0	0.0	0.000000	0.000000	0.000000	0.000000	...
25%	0.000000	0.000000e+00	0.000000e+00	0.000000	0.0	0.0	0.000000	0.000000	0.000000	0.000000	...
50%	0.000000	0.000000e+00	0.000000e+00	0.000000	0.0	0.0	0.000000	0.000000	0.000000	0.000000	...
75%	0.000000	0.000000e+00	0.000000e+00	0.000000	0.0	0.0	0.000000	0.000000	0.000000	0.000000	...
max	12743.000000	6.933756e+08	1.159100e+06	1.000000	0.0	0.0	18.000000	5.000000	1.000000	6.000000	...

Рисунок 3.16 – Оцінка статистики для параметрів

Зауважимо, що є кілька стовпців зі стандартним відхиленням 0. Це говорить нам про те, що з цих стовпців немає додаткової інформації. Тому можемо відкинути їх.

```
# data minus the rows with no variance
binary_prediction_data['non_zeros'].describe()
```

Виділимо основні відмінності в наших функціях помилок. Для цього розглянемо окремо хибно-позитивні та хибно-негативні результати та виділимо їх особливості.

```
# see the standard deviation of the false positives
binary_prediction_data['false_positives'].std()
```

```
# see the standard deviation of the false negatives
binary_prediction_data['false_negatives'].std()
```

У хибно негативних результатах усі стовпці мають певну дисперсію. Це свідчить про те, що в цих стовпцях може міститися корисна інформація, оскільки існує різниця між спостереженнями в одній класифікації та в іншій.

Оцінимо також помилково-негативні результати та побачимо, які типи атак ми пропустили.

```
# distribution of false negatives--what attacks did we miss?
binary_prediction_data['false_negatives'].attack.value_counts().plot.bar()
```

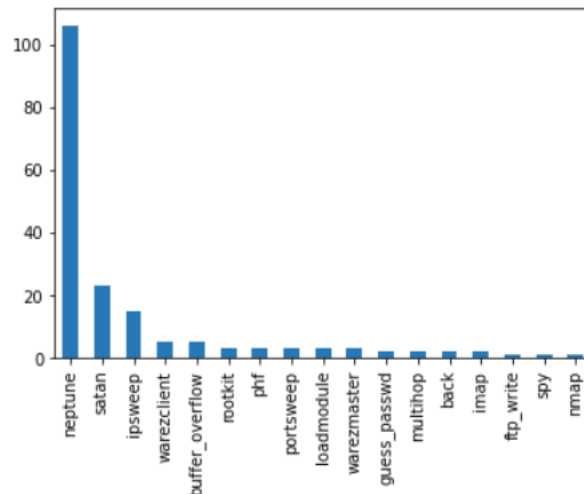


Рисунок 3.17 – Типи атак, які ми пропустили

Із рис. 3.17 можна побачити, що найбільш часто пропускались атаки Нептун і Сатана. Подивимося, чи зможемо ми це виправити.

Останній набір даних, з яким ми працювали, був `to_fit`. Тож ми з цим попрацюємо. Оскільки решта значень є числовими ознаками, ми можемо їх легко додати.

```
# отримаємо нові ознаки із набору даних
outcomes = ['attack_flag', 'attack_map', 'actual']

# отримати нові ознаки, які нас цікавлять,
# і відмовитися від попередніх результатів
new_features = (binary_prediction_data['false_positives']==0).all(axis=0)
feature_cols = binary_prediction_data['false_positives'].loc[:,new_features]
feature_cols = feature_cols.drop(outcomes,axis=1)

# складемо нові ознаки у список
new_feature_columns = list(feature_cols.columns)
new_feature_columns
```

Отримаємо:

```
['num_failed_logins',
 'su_attempted',
 'num_root',
 'num_access_files',
 'is_guest_login']
```

```
# Додаємо нові ознаки та проводимо донавчання
to_fit_new_features = to_fit.join(df[new_feature_columns])

# формуємо навчальну вибірку
new_feature_train_X, new_feature_val_X, new_feature_train_y, new_feature_val_y = train_test_split(to_fit_new_features, binary_y)
```

Оцінимо, як ми натренували.

```
# Модель бінарної класифікації
new_feature_model = RandomForestClassifier()
new_feature_model.fit(new_feature_train_X, new_feature_train_y)
new_feature_predictions = new_feature_model.predict(new_feature_val_X)

# get the score for the model
new_feature_score = accuracy_score(new_feature_predictions, new_feature_val_y)

new_feature_score
```

Результат: 0.9928873082907312

Побудуємо ще одну матрицю плутанини, щоб побачити пропущені передбачення.

```
# capture the prediction data
new_prediction_data = add_predictions(df,
                                     new_feature_predictions,
                                     new_feature_val_y)

# create a heatmap of the confusion matrix
sns.heatmap(data=new_prediction_data['confusion_matrix'],
            xticklabels = ['Predicted Normal', 'Predicted Attack'],
            yticklabels = ['Actual Normal', 'Actual Attack'],
            cmap="YlGnBu",
            fmt='d',
            annot=True)
```

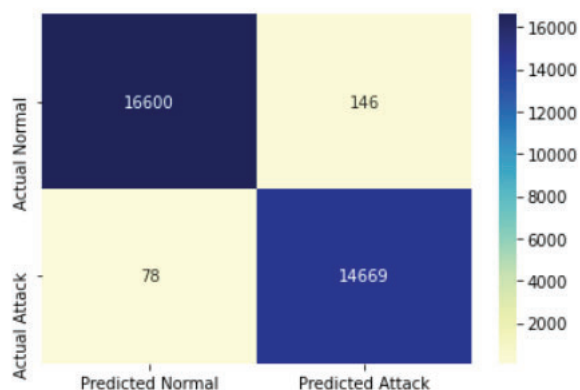


Рисунок 3.18 – Перерахована матриця плутанини для двохкласової класифікації

Із рис. 3.18 видно, що кількість помилок значно зменшилася.

Оцінимо помилки на конкретних атаках.

```
# distribution of the false negatives--what attacks did we miss?
new_prediction_data['false_negatives'].attack.value_counts().plot.bar()
```

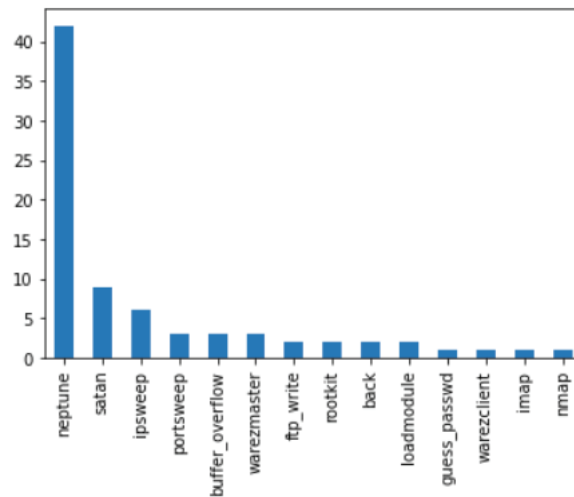


Рисунок 3.19 – Типи атак, які ми пропустили після донавчання

Розробимо модель виявлення вторгнень на основі нейронної мережі.

Ми налаштуємо нашу модель на повний набір даних, а потім перевіримо її на тестовому наборі, щоб побачити, як нам це вдалося.

Формуємо та тренуємо нейронну мережу

```
import pandas as pd
import io
import requests
import numpy as np
import os
from sklearn.model_selection import train_test_split
from sklearn import metrics
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation
from tensorflow.keras.callbacks import EarlyStopping

# Create a test/train split. 25% test
# Split into train/test
x_train, x_test, y_train, y_test = train_test_split(
    x, y, test_size=0.25, random_state=42)

# Create neural net
model = Sequential()
model.add(Dense(10, input_dim=x.shape[1], kernel_initializer='normal', activation='relu'))
model.add(Dense(50, input_dim=x.shape[1], kernel_initializer='normal', activation='relu'))
model.add(Dense(10, input_dim=x.shape[1], kernel_initializer='normal', activation='relu'))
model.add(Dense(1, kernel_initializer='normal'))
model.add(Dense(y.shape[1], activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam')
monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3, patience=5, verbose=1, mode='auto')
model.fit(x_train, y_train, validation_data=(x_test, y_test), callbacks=[monitor], verbose=2, epochs=19)
```

Під час роботи відображаються результати навчання по епохах

```
Epoch 1/19
3413/3413 - 6s - loss: 0.2155 - val_loss: 0.1031
Epoch 2/19
```

3413/3413 - 6s - loss: 0.0953 - val_loss: 0.0858

.....

3413/3413 - 6s - loss: 0.0513 - val_loss: 0.0530

Epoch 17/19

3413/3413 - 6s - loss: 0.0500 - val_loss: 0.0531

Epoch 18/19

3413/3413 - 7s - loss: 0.0485 - val_loss: 0.0505

Epoch 19/19

3413/3413 - 6s - loss: 0.0462 - val_loss: 0.0508

```
import tensorflow.keras.backend as K
print('Learning Rate - ')
print(K.eval(model.optimizer.lr))
print('='*50)
model.summary()
```

Learning Rate -
0.001

=====

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 10)	1210
dense_1 (Dense)	(None, 50)	550
dense_2 (Dense)	(None, 10)	510
dense_3 (Dense)	(None, 1)	11
dense_4 (Dense)	(None, 23)	46

=====

Total params: 2,327
Trainable params: 2,327
Non-trainable params: 0

```
import seaborn as sns
import datetime as dt
```

```
def confusion_matrix_func(y_test, y_test_pred):
```

```
    '''
```

Ця функція обчислює матрицю плутанини за допомогою прогнозованих і фактичних значень і будує теплову карту матриці плутанини

```
    '''
```

```
    C = confusion_matrix(y_test, y_test_pred)
    cm_df = pd.DataFrame(C)
    labels = ['back', 'butter_overflow', 'loadmodule', 'guess_passwd', 'imap',
, 'ipsweep', 'warezmaster', 'rootkit',
'multihop', 'neptune', 'nmap', 'normal', 'phf', 'perl', 'pod', 'portsweep', '
ftp_write', 'satan', 'smurf', 'teardrop', 'warezclient', 'land']
    plt.figure(figsize=(20,15))
    sns.set(font_scale=1.4)
    sns.heatmap(cm_df, annot=True, annot_kws={"size":12}, fmt='g', xticklabel
s=labels, yticklabels=labels)
    plt.ylabel('Actual Class')
    plt.xlabel('Predicted Class')

    plt.show()
```

```

# calculate roc curve
from sklearn.metrics import *
#fpr_RF, tpr_RF, thresholds_RF = roc_curve(y_test, pred)
from sklearn import preprocessing
def multiclass_roc_auc_score(y_test, pred, average="macro"):
    lb = preprocessing.LabelBinarizer()
    lb.fit(y_test)
    y_test = lb.transform(y_test)
    pred = lb.transform(pred)
    return roc_auc_score(y_test, pred, average=average)

print('Train data')
print(x_train.shape)
print(y_train.shape)
print('='*20)
print('Test data')
print(x_test.shape)
print(y_test.shape)
print('='*20)

from sklearn.metrics import accuracy_score, confusion_matrix, precision_score
, recall_score, f1_score, roc_auc_score
print('Predicting on the test data:')
start = dt.datetime.now()
escore = model.evaluate(x_test, y_test, batch_size=32)
pred = model.predict(x_test)
pred = np.argmax(pred,axis=1)
y_eval = np.argmax(y_test,axis=1)

vscore = metrics.accuracy_score(y_eval, pred)

rscore = recall_score(y_eval, pred, average='weighted')

ascore = precision_score(y_eval, pred, average='weighted')

f1score= f1_score(y_eval, pred, average='weighted') #F1 = 2 * (precision * re
call) / (precision + recall) for manual

roc_auc_socre = multiclass_roc_auc_score(y_eval, pred)

print('Completed')
print('Time taken:',dt.datetime.now()-start)
print('='*50)
print("Validation score: {}".format(vscore))
print('='*50)
print("Evaluation score: {}".format(escore))
print('='*50)
print("Recall score: {}".format(rscore))
print('='*50)
print("Precision score: {}".format(ascore))
print('='*50)
print("F1 score: {}".format(f1score))
print('='*50)
print("ROC-AUC score: {}".format(roc_auc_socre))

Train data
(109189, 120)
(109189, 23)
=====
Test data
(36397, 120)
(36397, 23)
=====

```

Predicting on the test data:

1138/1138 [=====] - 1s 1ms/step - loss: 0.0508

Completed

Time taken: 0:00:02.516503

Validation score: 0.9908234195125972

Evaluation score: 0.05082421377301216

Recall score: 0.9908234195125972

Precision score: 0.9885758843840302

F1 score: 0.988816489428654

ROC-AUC score: 0.6913971252584903

/opt/conda/lib/python3.7/site-packages/sklearn/metrics/_classification.py:1221: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero\_division` parameter to control this behavior.

_warn_prf(average, modifier, msg_start, len(result))

confusion_matrix_func(y_eval, pred)

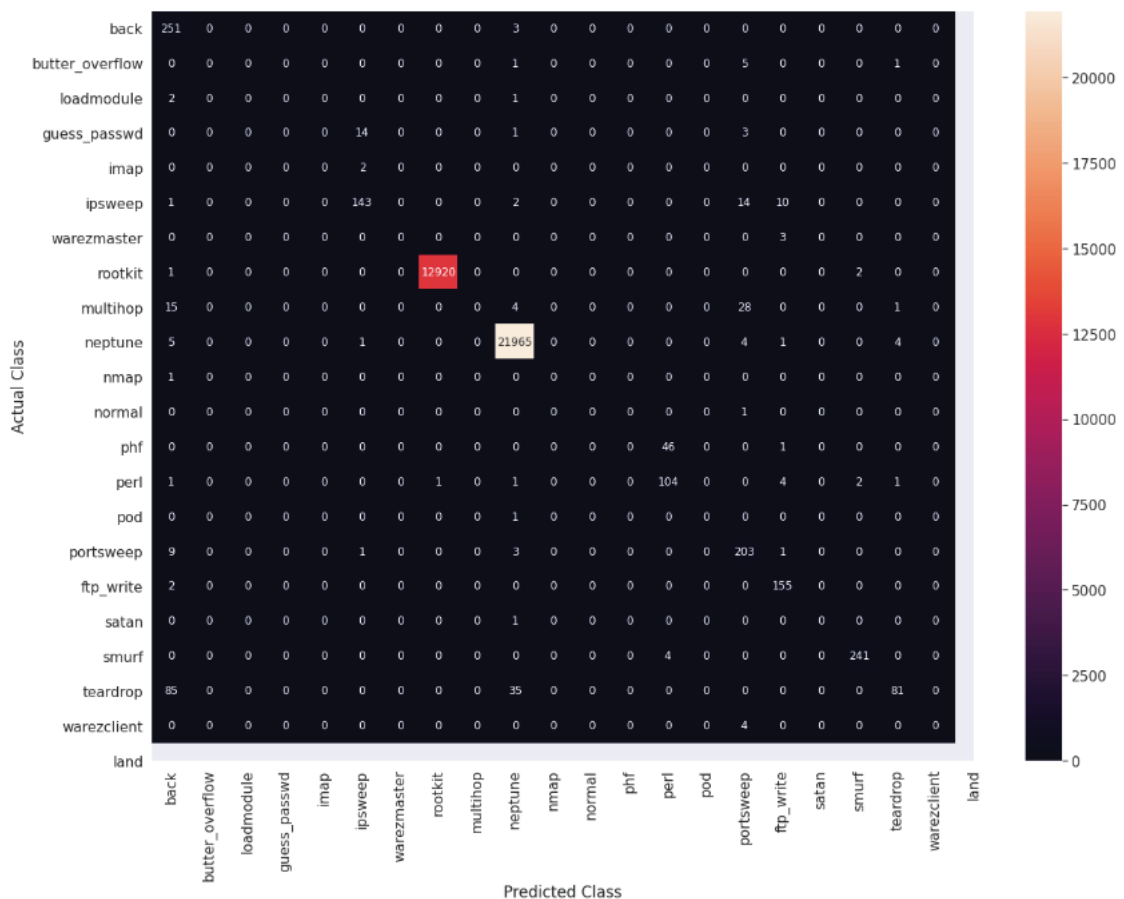


Рисунок 3.20 – Матриця плутанини для багатокласової класифікації

Таким чином, модель на основі навченої нейронної мережі досить гарно виявляє вторгнення у систему.

3.3. Рекомендації щодо вдосконалення детектора вторгнень

Для покращення роботи алгоритмів можна запропонувати цілий ряд рекомендацій які потребують також і подальших досліджень.

Семплювання.

Більшість алгоритмів розпізнавання вторгнень успішно працюють на вибірках малих розмірів. Ідея семплювання полягає в тому, щоб запустити алгоритм на кількох випадкових підвибірках та усереднити результат.

Розмір випадкових підвибірок може бути як фіксований, так і випадковий з деякого діапазону, але найчастіше він на порядки менше розміру вихідної вибірки. Інтуїція такого вибору полягає в тому, що шумові об'єкти потраплять у вибірку з низькою ймовірністю; кластери нормальних даних будуть представлені кількома представниками, а кластери вторгнень виродяться в ізольовані точки (рис. 3.21). Зазвичай на таких підвибірках алгоритми будують функцію $anomaly_score(x)$, що не сильно поступається результату, отриманому за всіма вихідними даними, навіть незважаючи на те, що у підвибірках може загубитися глобальна структура простору ознак. Усереднення по невеликій кількості запусків, зазвичай близько 100, на практиці вистачає для вирішення цієї проблеми.

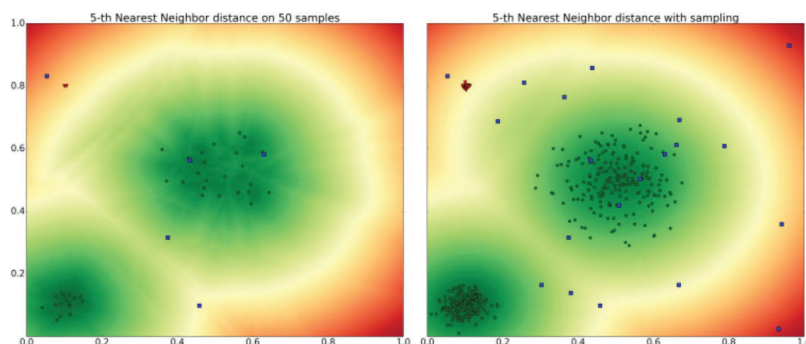


Рисунок 3.21 – Відстань до 5-го сусіда на випадковій підвибірці з 50 елементів (ліворуч) та результат усереднення по 100 випадковим підвибіркам (праворуч)

Крім значної оптимізації обчислювальної складності, семплювання зменшує ймовірність «підгону» результату алгоритму під конкретні наявні

дані. З огляду на особливості завдання, необхідна умова відсутності параметризації алгоритмів часто означає їх детермінованість (без стохастичності *anomaly_score* однозначно визначається за заданою вибіркою). Це важливо у тому випадку, якщо передбачається використовувати отриманий *anomaly_score* на нових даних. У випадку, з появою таких нових даних, будь-який алгоритм навчання без вчителя можна запустити наново на об'єднанні всієї наявної інформації; у разі, якщо використовується семплювання, можна не перенавчати алгоритм повністю, а додати запуски в ансамбль (т. зв. *warm start*).

Для ізолюючого лісу семплювання безпосередньо «вбудовано» в сам алгоритм: оскільки сам алгоритм є ансамблем з кількох ізолюючих дерев, кожне дерево будується за підвибором з *max_samples* елементів. На практиці *max_samples* вважають рівним 250, оскільки стохастична природа алгоритму робить налаштування цього параметра безглуздим. Таким чином, для великих вибірок алгоритм не використовує всі наявні дані, що збільшує його узагальнюючу здатність. Хороша працездатність алгоритму на малих вибірках обумовлена тим, що у кожному вузлі вирішального дерева використовується лише інформація про мінімальне і максимальне значення ознаки; а ось використання великої кількості точок для побудови одного дерева може призвести до занадто об'ємного дерева, що несе надмірну і, часто, марну інформацію.

Випадкове масштабування

Алгоритми детектування вторгнень часто не інваріантні щодо лінійних перетворень простору. Це призводить до залежності результату від ознакового уявлення об'єктів.

Так, однією з важливих переваг ізолюючого лісу є інваріантність до масштабування ознак. Наприклад, для метричних алгоритмів суттєвим є нормалізація ознак (приведення значень до діапазону $[-1, 1]$ або центрування та поділ на дисперсію). Однак таке стандартне нормування призводить до втрати потенційної інформації про значущість ознак. Часто погані результати роботи метричного алгоритму пов'язані саме з вибором стандартної метрики на

нормованому просторі, що не відповідає фізичному змісту ознак (яка, у свою чергу, зазвичай прихована) [1].

За наявності міток цю проблему можна вирішувати параметризацією метрики та подальшим підбором параметрів; проте, за їх відсутності це неможливо. Потенційним рішенням виступає таке міркування: якщо використання стандартно нормалізованих ознак необґрунтовано, можна взяти кілька різних необґрунтованих нормалізацій і усереднити їх. Іншими словами, для j -го алгоритму в ансамблі використовується перетворена вибірка:

$$\begin{aligned} w(j) &\sim \text{Uniform}[0, 1]^d \\ \forall j : \tilde{X}(j)_i &= w(j)X_i, \end{aligned} \tag{3.1}$$

де X_i - i -а компонента вихідної, попередньо нормалізованої стандартним чином, вибірки.

Це досить загальний спосіб боротьби з проблемою невідомої метрики, проте гарантій того, що результат виявиться кращим, не надає. Більше того, немає способу порівняти результат, отриманий із випадковими масштабуваннями, і без.

Rotated Bagging

На відміну від масштабування, ізолюючий ліс не інваріантний щодо поворотів простору ознак. Так, при побудові ліній рівня *anomaly_score* ізолюючого лісу на модельних даних ($d = 2$) можна виявити, що відходячи від кластерів нормальних даних уздовж осей координат, значення *anomaly_score* змінюється незначно. Потенційні аномалії, що потрапляють в цю область, можуть бути сплутані з граничними об'єктами кластера, незважаючи на те, що вони виражено ізольовані. Причина полягає в тому, що всі поділи простору ізолюючий ліс проводить гіперплощинами, паралельними осям координат.

Загальна ідея Rotated Bagging полягає у виборі випадкових підпросторів, на які проектується вибірка чергового алгоритму в ансамблі. Таким чином, для j -го алгоритму вибирається

$Q(j) \in R^{d' \times d}$ – довільна матриця з но-

рмованих попарно-ортогональних векторів, після чого перетворена вибірка виходить за формулою

$$\forall j : \tilde{X}(j) = XQ(j)^T \quad (3.2)$$

Зокрема, вважаючи $d' = d$, отримуємо довільне ортогональне перетворення простору ознак. Для ізолюючого лісу це рівносильно використанню гіперплощин, паралельних осям випадково обраного ортогонального базису; усереднення дозволяє отримати згладжені лінії рівня (рис. 3.22).

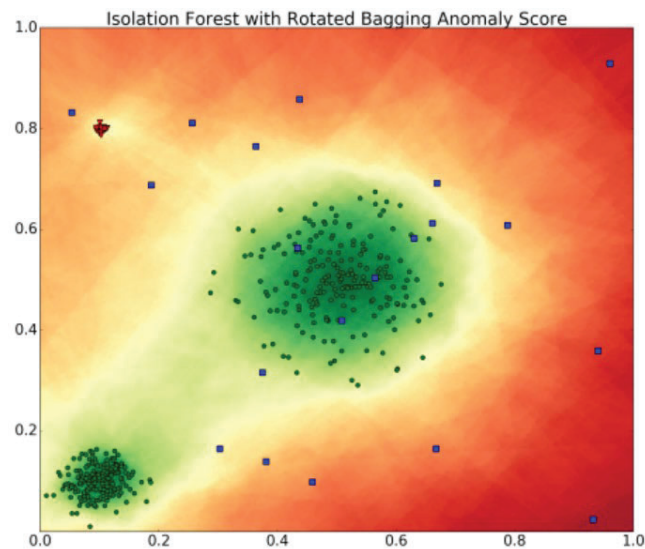


Рисунок 3.22 – Результат роботи ізолюючого лісу з використанням Rotated Bagging

Недоліком такої модифікації є втрата інваріантності масштабування ознак. Оскільки в модифікованому просторі кожна ознака є лінійною комбінацією вихідних, домінування значень одного з них за модулем призведе до підвищеної значущості цієї ознаки.

Тому використання цього методу для ізолюючого лісу вимагає попереднього нормування і потенційно може призвести до тих же проблем, що й використання метричних алгоритмів.

Теоретично Rotated Bagging, у тому числі і у випадку $d' = d$, може бути застосований і для інших алгоритмів, не інваріантних щодо поворотів про-

сторю, у тому числі й для метричних методів (при використанні неевклідової метрики).

Ансамблювання

У більш загальному вигляді, ансамблювання в задачі вторгнень може бути задано як використання кількох різних алгоритмів з подальшим усередненням їх *anomaly_score*. Однак, для різних алгоритмів видається ними *anomaly_score* має різні шкали та масштаби. Тому традиційне приведення значень функцій різних алгоритмів одного діапазону (наприклад, до $[0, 1]$) з наступним усередненням позбавлено своїх звичайних переваг.

На практиці більшого успіху можна досягти ансамблюванням одного і того ж алгоритму на модифікованих вибірках, наприклад, за допомогою семплювання, традиційного бегінга, випадкового масштабування або Rotated Bagging. Поєднання ж результатів різнорідних алгоритмів, заснованих на несхожих принципах і підходах, краще проводити не усереднення їх *anomaly_score*, а голосуванням за кінцевою відповіддю, чи є точка аномальною чи ні. Для цього для кожного з видів алгоритмів слід спочатку індивідуально визначити поріг бінаризації.

Ітеративний відбір

Ітеративний відбір в задачі детектування вторгнень є особливим методом ансамблювання кількох алгоритмів [1].

Припустимо, побудовано деяку модель, яка описує нормальні дані. Ця модель побудована на всіх наявних даних, що містять аномалії, і тому її точність може бути невеликою. Однак її наближення достатньо, щоб виявити явні аномалії. Відсортувавши всі точки по *anomaly_score*, можна вибрати найаномальніші об'єкти в даних і виключити їх з даних. Потім модель будуватиметься заново, причому точність її наближення буде вищою. Цей процес можна повторити кілька разів або ввести будь-які критерії зупинки (рис. 3.23).

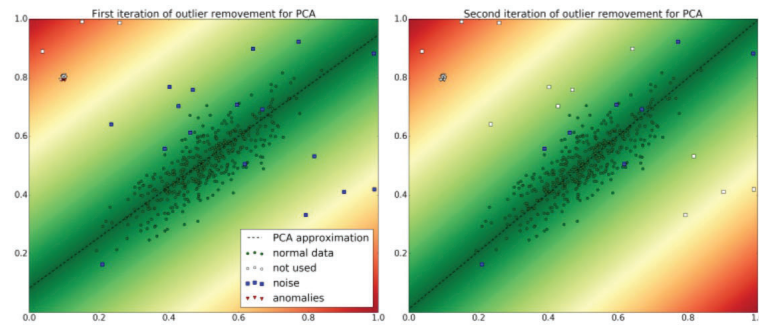


Рисунок 3.23 – Ітеративна побудова апроксимуючого одновимірного підпростору. На кожному кроці видаляється 15 аномальних об'єктів, після чого модель перебудовується

Ідея ітеративного відбору може бути узагальнена у різний спосіб. Результат роботи одного алгоритму може бути використаний для відсіювання явних вторгнень і налаштування нового алгоритму, що не обов'язково збігається з попереднім, на даних, що залишилися. Можлива і протилежна механіка: за результатами роботи одного алгоритму відбираються явні, гарантовані представники нормальних даних, і виключно на них будується модель, що їх описує.

Важливо, що одні алгоритми працюють тим краще, ніж менше вторгнень у дані (побудова моделей), коли інші припускають наявність у даних вторгнень (наприклад, ізолюючий ліс). Якщо останнім подати на вхід лише нормальні дані, побудована ними функція *anomaly_score* зазвичай не зможе розрізняти граничні нормальні дані з аномаліями, у тому числі явними.

Тому ітеративний відбір слід організовувати так, щоб перші алгоритми відсіювали явні аномалії та/або відбирали представників нормальних (Isolation Forest, LOF), а фінальні алгоритми були модельними, тобто будують функцію, виходячи з припущення про нормальність вхідної інформації (наприклад, лінійні моделі, ЕМ -Алгоритм, поліноміальна функція).

Більш специфічним інструментом є можливість вбудовування результатів роботи одного алгоритму "всередину" інших. Наприклад, можна прово-

дити побудову моделі за всіма наявними даними з вагами, обернено пропорційними їх рейтингу аномальності, виданим деяким іншим алгоритмом.

Висновки по 3 розділу

Серед багатьох відомих методів виявлення вторгнень будь-який може виявляти прості аномалії, що і видно по роботі з простими моделями. Однак коли виішуються складні задачі, на кшталт виявлення вторгнень у мережу, є необхідність збирати і працювати з великими наборами даних, то прості методи не забезпечують належної якості. У цих випадках необхідно використовувати машинне навчання та нейронні мережі.

ВИСНОВКИ

Випускна кваліфікаційна робота магістра присвячена вирішенню актуального завдання по розробці детектора вторгнень для «розумного» будинку.

Аналіз типових схем «розумних» будинків показав, що в їх основу складають IoT-системи, що призводить до появи загроз типових для IoT-систем.

Загрозами інформаційної безпеки «розумного» будинку є порушення конфіденційності, цілісності та доступності інформації.

Для виявлення таких кіберзагроз доцільно розробити детектор вторгнень, що зможе своєчасно їх виявляти.

Проведений аналіз показав, що існує досить багато підходів та методів, які можуть застосовуватися для рішення задачі пошуку вторгнень. Серед них великою популярністю користуються методи засновані на машинному навчанні.

Реалізація методів машинного навчання здійснюється шляхом побудови конвеєра машинного навчання. Тому, на основі конвеєра машинного навчання розроблено методику побудови детектора вторгнень.

Третій розділ присвячено розробці та дослідженню варіантів детектора вторгнень. У ході досліджень виявилось, що прості методи гарно працюють лише на простих даних. Роботу з великими наборами даних краще проводити методами машинного навчання, коли виявлення вторгнень здійснюється шляхом застосування алгоритмів класифікації.

Двохкласова класифікація реалізована за допомогою методів К-найближчих сусідів та випадкового лісу показали гарні результати, у той час як метод логістичної регресії – погані.

Багатокласову класифікацію реалізовано за допомогою нейронної мережі.

Таким чином, завдання на кваліфікаційну роботу виконано у повному обсязі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Internet of things : challenges, advances, and applications. — Boca Raton, 2018. — 1 online resource (xvii, 418 pages) c. — ISBN 978-1-315-15500-5.
2. Комплекти для розумного будинку. [Електронний ресурс] – Режим доступу: <https://worldvision.com.ua/komplekty-dlya-umnogo-doma/>
3. Ed Burns. Machine learning. URL: <https://www.techtarget.com/searchenterpriseai/definition/machine-learning-ML>
4. Al Perlman. The Growing Role of Machine Learning in Cybersecurity, 2022. URL: <https://www.securityroundtable.org/the-growing-role-of-machine-learning-in-cybersecurity/>
5. Stephen Cooper. 10 AI Cybersecurity Firms to watch in 2022, 2021. URL: <https://www.comparitech.com/blog/information-security/leading-ai-cybersecurity-companies/>
6. Abi Tyas Tunggal. What is a Cyber Attack? Common Attack Techniques and Targets, 2022. URL: <https://www.upguard.com/blog/cyber-attack>
7. Machine Learning Enabled Attacks: What You Need to Know. URL: <https://prowritersins.com/cyber-insurance-blog/machine-learning-enabled-attacks/>
8. Iqbal H. Sarker, A. S. M. Kayes, Shahriar Badsha, Hamed Alqahtani, Paul Watters and Alex Ng. Cybersecurity data science: an overview from machine learning perspective, 2020. URL: <https://journalofbigdata.springeropen.com/articles/10.1186/s40537-020-00318-5>
9. Arne Wolfewicz. Deep Learning vs. Machine Learning – What’s The Difference? 2022. URL: <https://levity.ai/blog/difference-machine-learning-deep-learning>
10. Katrina Wakefield. A guide to the types of machine learning algorithms and their applications. URL: https://www.sas.com/en_gb/insights/articles/analytics/machine-learning-algorithms.html
11. Pethuru Raj, Preetha Evangeline. Advances in Computers. Machine learning and deep learning algorithms on the Industrial Internet of Things (IIoT), 2020, p. 321-338.
12. Документація по PyOD, 2023. URL: <https://pyod.readthedocs.io/en/latest/pyod.models.html#pyod.models.inne.INNE>

13. James Shah. Six Elements Of Machine Learning - A Beginner's Guide, 2020. URL: <https://dev.to/jamesshah/six-elements-of-machine-learning-a-beginner-s-guide-4i35>
14. Anshul Saini. Decision Tree Algorithm – A Complete Guide, 2021. URL: <https://www.analyticsvidhya.com/blog/2021/08/decision-tree-algorithm/>
15. Decision Tree Classification Algorithm. URL: <https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm>
16. A Distance-Based Attribute Selection Measure for Decision Tree Induction. URL: <https://link.springer.com/article/10.1023/A:1022694001379>
17. Nagesh Singh Chauhan. Decision Tree Algorithm, Explained. URL: <https://www.kdnuggets.com/2020/01/decision-tree-algorithm-explained.html>
18. Monomita Chakraborty. What are the best programming languages for artificial intelligence, 2021. URL: <https://www.analyticsinsight.net/what-are-the-best-programming-languages-for-artificial-intelligence/>
19. John Selawsky. Top five programming languages for AI and machine learning you should learn this year, 2022. URL: <https://www.itproportal.com/features/top-five-programming-languages-for-ai-and-machine-learning-you-should-learn-this-year/>
20. Angela Beklemysheva. Why Use Python for AI and Machine Learning? URL: <https://steelkiwi.com/blog/python-for-ai-and-machine-learning/>
21. Amanda Iglesias Moreno. Data normalization with Pandas and Scikit-Learn, 2020. URL: <https://towardsdatascience.com/data-normalization-with-pandas-and-scikit-learn-7c1cc6ed6475>
22. Philip Wilkinson. Introduction to decision tree classifiers from scikit-learn, 2020. URL: <https://towardsdatascience.com/introduction-to-decision-tree-classifiers-from-scikit-learn-32cd5d23f4d>
23. Samarth Agrawal. Understanding the Confusion Matrix from Scikit learn, 2021. URL: <https://towardsdatascience.com/understanding-the-confusion-matrix-from-scikit-learn-c51d88929c79>
24. Jason Brownlee. How to Configure k-Fold Cross-Validation, 2020. URL: <https://machinelearningmastery.com/how-to-configure-k-fold-cross-validation/>
25. Jason Brownlee. Nested Cross-Validation for Machine Learning with Python, 2020.
26. Aggarwal, Charu C. Outlier Analysis // - 2017. -C. 1-247.

27. Liu F. T., Ting K. M., Zhou Z. H. Isolation Forest // Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on. – IEEE, 2008. – С. 413-422.
28. He Z. et al. Fp-outlier: Frequent pattern based outlier detection // Computer Science and Information Systems. – 2005. – Т. 2. –№. 1. – С. 103-118.
29. Hodge V., Austin J. A survey of outlier detection methodologies // Artificial intelligence review. – 2004. – Т. 22. –№. 2. – С. 85-126.
30. Pedregosa F. et al. Scikit-learn: Machine learning in Python // Journal of Machine Learning Research. – 2011. – Т. 12. –№. Oct. – С. 2825-2830.
31. URL: <http://scikit-learn.org/stable/index.html> (2017).
32. Machine Learning Metrics: How to Measure the Performance of a Machine Learning Mode. URL: <https://www.altexsoft.com/blog/machine-learning-metrics/>
33. Пулеко І.В., Пулеко К.І., Іщенко І.А. Огляд програм, що використовують машинне навчання при виявленні аномалій у кібербезпеці. Тези XIII Міжнародної науково-технічної конференції «Інформаційні комп'ютерні технології», м. Житомир, 30–31 березня 2023 р. – Житомир: Житомирська політехніка, 2023. – с. 38-39.
34. Пулеко І.В., Пулеко К.І., Іщенко І.А. Детектор вторгнень для «розумного» будинку. Тези VI Всеукраїнської науково-технічної конференції «Комп'ютерні технології: інновації, проблеми, рішення», м. Житомир, 29–30 листопада 2023 р. – Житомир: Житомирська політехніка, 2023. – с. 32-33.

ДОДАТКИ

```

import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
import matplotlib.font_manager

from pyod.models.abod import ABOD
from pyod.models.knn import KNN

import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
import matplotlib.font_manager
from pyod.utils.data import generate_data, get_outliers_inliers

X_train, Y_train = generate_data(n_train=200,train_only=True, n_features=2)

outlier_fraction = 0.1

# store outliers and inliers in different numpy arrays
x_outliers, x_inliers = get_outliers_inliers(X_train,Y_train)

n_inliers = len(x_inliers)
n_outliers = len(x_outliers)

F1 = X_train[:,[0]].reshape(-1,1)
F2 = X_train[:,[1]].reshape(-1,1)

xx , yy = np.meshgrid(np.linspace(-10, 10, 200), np.linspace(-10, 10, 200))

plt.scatter(F1,F2)
plt.xlabel('F1')
plt.ylabel('F2')
plt.show()

classifiers = {
    'Angle-based Outlier Detector
(ABOD)':ABOD(contamination=outlier_fraction),
    'K Nearest Neighbors (KNN)':KNN(contamination=outlier_fraction)
}

#set the figure size
plt.figure(figsize=(10, 10))

for i, (clf_name,clf) in enumerate(classifiers.items()) :
    # fit the dataset to the model
    clf.fit(X_train)

    scores_pred = clf.decision_function(X_train)*-1

    y_pred = clf.predict(X_train)

    n_errors = (y_pred != Y_train).sum()
    print('No of Errors : ',clf_name, n_errors)

# threshold value to consider a datapoint inlier or outlier
threshold = stats.scoreatpercentile(scores_pred,100 *outlier_fraction)

Z = clf.decision_function(np.c_[xx.ravel(), yy.ravel()]) * -1

```

```

Z = Z.reshape(xx.shape)

subplot = plt.subplot(1, 2, i + 1)

subplot.contourf(xx, yy, Z, levels = np.linspace(Z.min(), threshold,
10), cmap=plt.cm.Blues_r)

a = subplot.contour(xx, yy, Z, levels=[threshold], linewidths=2,
colors='red')

subplot.contourf(xx, yy, Z, levels=[threshold, Z.max()], colors='orange')

b = subplot.scatter(X_train[:-n_outliers, 0], X_train[:-n_outliers, 1],
c='white', s=20, edgecolor='k')
# scatter plot of outliers with black dots
c = subplot.scatter(X_train[-n_outliers:, 0], X_train[-n_outliers:, 1],
c='black', s=20, edgecolor='k')
subplot.axis('tight')

subplot.legend(
[a.collections[0], b, c],
['learned decision function', 'true inliers', 'true outliers'],
prop=matplotlib.font_manager.FontProperties(size=10),
loc='lower right')

subplot.set_title(clf_name)
subplot.set_xlim((-10, 10))
subplot.set_ylim((-10, 10))
plt.show()

```

Додаток Б

```

from __future__ import division
from __future__ import print_function

import os
import sys
from time import time

import numpy as np
from numpy import percentile
import matplotlib.pyplot as plt
import matplotlib.font_manager

# Імпорт усіх моделей
from pyod.models.abod import ABOD
from pyod.models.cblof import CBLOF
from pyod.models.feature_bagging import FeatureBagging
from pyod.models.hbos import HBOS
from pyod.models.iforest import IForest
from pyod.models.knn import KNN
from pyod.models.lof import LOF
from pyod.models.mcd import MCD
from pyod.models.ocsvm import OCSVM
from pyod.models.pca import PCA
from pyod.models.lscp import LSCP
from pyod.models.inne import INNE
from pyod.models.gmm import GMM
from pyod.models.kde import KDE
from pyod.models.lmdd import LMDD

# Визначте кількість вхідних і вихідних значень
n_samples = 200
outliers_fraction = 0.25
clusters_separation = [0]

# Порівняння результатів роботи детекторів за заданих налаштувань
# Ініціалізація даних
xx, yy = np.meshgrid(np.linspace(-7, 7, 100), np.linspace(-7, 7, 100))
n_inliers = int((1. - outliers_fraction) * n_samples)
n_outliers = int(outliers_fraction * n_samples)
ground_truth = np.zeros(n_samples, dtype=int)
ground_truth[-n_outliers:] = 1

# ініціалізація набору детекторів для LSCP
detector_list = [LOF(n_neighbors=5), LOF(n_neighbors=10),
LOF(n_neighbors=15),
LOF(n_neighbors=20), LOF(n_neighbors=25), LOF(n_neighbors=30),
LOF(n_neighbors=35), LOF(n_neighbors=40), LOF(n_neighbors=45),
LOF(n_neighbors=50)]

# Відображення статистики даних
print('Number of inliers: %i' % n_inliers)
print('Number of outliers: %i' % n_outliers)
print('Ground truth shape is {shape}. Outlier are 1 and inliers are 0.\n'.format(shape=ground_truth.shape))
print(ground_truth)

random_state = np.random.RandomState(42)
# Визначення дев'яти інструментів виявлення викидів для порівняння
classifiers = {

```

```

'Angle-based Outlier Detector (ABOD)':
    ABOD(contamination=outliers_fraction),
'Cluster-based Local Outlier Factor (CBLOF)':
    CBLOF(contamination=outliers_fraction,
           check_estimator=False, random_state=random_state),
'Feature Bagging':
    FeatureBagging(LOF(n_neighbors=35),
                   contamination=outliers_fraction,
                   random_state=random_state),
'Histogram-base Outlier Detection (HBOS)': HBOS(
    contamination=outliers_fraction),
'Isolation Forest': IForest(contamination=outliers_fraction,
                             random_state=random_state),
'K Nearest Neighbors (KNN)': KNN(
    contamination=outliers_fraction),
'Average KNN': KNN(method='mean',
                   contamination=outliers_fraction),
'Local Outlier Factor (LOF)':
    LOF(n_neighbors=35, contamination=outliers_fraction),
'Minimum Covariance Determinant (MCD)': MCD(
    contamination=outliers_fraction, random_state=random_state),
'One-class SVM (OCSVM)': OCSVM(contamination=outliers_fraction),
'Principal Component Analysis (PCA)': PCA(
    contamination=outliers_fraction, random_state=random_state),
'Locally Selective Combination (LSCP)': LSCP(
    detector_list, contamination=outliers_fraction,
    random_state=random_state),
'INNE': INNE(contamination=outliers_fraction),
'GMM': GMM(contamination=outliers_fraction),
'KDE': KDE(contamination=outliers_fraction),
'LMDD': LMDD(contamination=outliers_fraction),
}

# Відображення всіх детекторів
for i, clf in enumerate(classifiers.keys()):
    print('Model', i + 1, clf)

# Навчання моделей на згенерованих даних
# і порівняння характеристик моделі
for i, offset in enumerate(clusters_separation):
    np.random.seed(42)
    # Генерація даних
    X1 = 0.3 * np.random.randn(n_inliers // 2, 2) - offset
    X2 = 0.3 * np.random.randn(n_inliers // 2, 2) + offset
    X = np.r_[X1, X2]
    # Додавання викидів
    X = np.r_[X, np.random.uniform(low=-6, high=6, size=(n_outliers, 2))]

# Навчання моделі
plt.figure(figsize=(15, 16))
for i, (clf_name, clf) in enumerate(classifiers.items()):
    print(i + 1, 'fitting', clf_name)
    # fit the data and tag outliers
    clf.fit(X)
    scores_pred = clf.decision_function(X) * -1
    y_pred = clf.predict(X)
    threshold = percentile(scores_pred, 100 * outliers_fraction)
    n_errors = (y_pred != ground_truth).sum()
    # побудова лінії рівнів і точок

    Z = clf.decision_function(np.c_[xx.ravel(), yy.ravel()]) * -1
    Z = Z.reshape(xx.shape)
    subplot = plt.subplot(4, 4, i + 1)

```

```

subplot.contourf(xx, yy, Z, levels=np.linspace(Z.min(), threshold, 7),
                 cmap=plt.cm.Blues_r)
# a = subplot.contour(xx, yy, Z, levels=[threshold],
#                     linewidths=2, colors='red')
subplot.contourf(xx, yy, Z, levels=[threshold, Z.max()],
                 colors='orange')
b = subplot.scatter(X[:-n_outliers, 0], X[:-n_outliers, 1], c='white',
                   s=20, edgecolor='k')
c = subplot.scatter(X[-n_outliers:, 0], X[-n_outliers:, 1], c='black',
                   s=20, edgecolor='k')
subplot.axis('tight')
subplot.legend(
[
    # a.collections[0],
    b, c],
[
    # 'навчена функція прийняття рішень',
    'true inliers', 'true outliers'],
prop=matplotlib.font_manager.FontProperties(size=10),
loc='lower right')
subplot.set_xlabel("%d. %s (errors: %d)" % (i + 1, clf_name, n_errors))
subplot.set_xlim((-7, 7))
subplot.set_ylim((-7, 7))
plt.subplots_adjust(0.04, 0.1, 0.96, 0.94, 0.1, 0.26)
plt.suptitle("Outlier detection")
plt.show()

```

Додаток В

```

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list
# all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# Any results you write to the current directory are saved as output.

import pandas as pd
from tensorflow.keras.utils import get_file

try:
    path = get_file('kddcup.data_10_percent.gz', origin='http://kdd.ics.uci.edu/databases/kddcup99/kddcup.data_10_percent.gz')
except:
    print('Error downloading')
    raise

print(path)

# This file is a CSV, just no CSV extension or headers
# Download from: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html
df = pd.read_csv(path, header=None)

# The CSV file has no column heads, so add them
df.columns = [
    'duration',
    'protocol_type',
    'service',
    'flag',
    'src_bytes',
    'dst_bytes',
    'land',
    'wrong_fragment',
    'urgent',
    'hot',
    'num_failed_logins',
    'logged_in',
    'num_compromised',
    'root_shell',
    'su_attempted',
    'num_root',
    'num_file_creations',
    'num_shells',
    'num_access_files',
    'num_outbound_cmds',
    'is_host_login',
    'is_guest_login',
    'count',
    'srv_count',
    'serror_rate',
    'srv_serror_rate',

```

```

    'error_rate',
    'srv_error_rate',
    'same_srv_rate',
    'diff_srv_rate',
    'srv_diff_host_rate',
    'dst_host_count',
    'dst_host_srv_count',
    'dst_host_same_srv_rate',
    'dst_host_diff_srv_rate',
    'dst_host_same_src_port_rate',
    'dst_host_srv_diff_host_rate',
    'dst_host_serror_rate',
    'dst_host_srv_serror_rate',
    'dst_host_rerror_rate',
    'dst_host_srv_rerror_rate',
    'outcome'
]
# Analyzing the Dataset

print("Read {} rows.".format(len(df)))
print('='*40)
print('The number of data points are:', df.shape[0])
print('='*40)
print('The number of features are:', df.shape[1])
print('='*40)
output = df['outcome'].values
labels = set(output)
print('The different type of output labels are:', labels)
print('='*125)
print('The number of different output labels are:', len(labels))

# Data Cleaning

# Checking for NULL values
print('Null values in dataset are',len(df[df.isnull().any(1)]))
print('='*40)

# Checkng for DUPLICATE values
df.drop_duplicates(keep='first', inplace = True)

# For now, just drop NA's (rows with missing values)
df.dropna(inplace=True,axis=1)

# stored the data into a pickle file so we can load through
# df.to_pickle('df.pkl')

print("Read {} rows.".format(len(df)))

# Exploratory data analysis
import matplotlib.pyplot as plt
from matplotlib.pyplot import *

plt.figure(figsize=(15,7))
class_distribution = df['outcome'].value_counts()
class_distribution.plot(kind='bar')
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in train data')
plt.grid()
plt.show()

sorted_yi = np.argsort(-class_distribution.values)

```

```

for i in sorted_yi:
    print('Number of data points in class', class_distribution.index[i],':',
          class_distribution.values[i],
          '(', np.round((class_distribution.values[i]/df.shape[0]*100), 3), '
          %)'')

#df.groupby('outcome')['outcome'].count() #this could also be used if you want
#no-formatted for above.

ENCODING = 'utf-8'

def expand_categories(values):
    result = []
    s = values.value_counts()
    t = float(len(values))
    for v in s.index:
        result.append("{}:{}".format(v, round(100*(s[v]/t), 2)))
    return "[{}]" .format(", ".join(result))

def analyze(df):
    print()
    cols = df.columns.values
    total = float(len(df))

    print("{} rows".format(int(total)))
    for col in cols:
        uniques = df[col].unique()
        unique_count = len(uniques)
        if unique_count>100:
            print("*** {}: {} ({}%)".format(col, unique_count, int((unique_count
            )/total)*100))
        else:
            print("*** {}: {}".format(col, expand_categories(df[col])))
            expand_categories(df[col])

# Analyze KDD-99

import pandas as pd
import os
import numpy as np
from sklearn import metrics
from scipy.stats import zscore

analyze(df)

# Encode a numeric column as zscores
def encode_numeric_zscore(df, name, mean=None, sd=None):
    if mean is None:
        mean = df[name].mean()

    if sd is None:
        sd = df[name].std()

    df[name] = (df[name] - mean) / sd

# Encode text values to dummy variables(i.e. [1,0,0],[0,1,0],[0,0,1] for red,
green,blue)
def encode_text_dummy(df, name):
    dummies = pd.get_dummies(df[name])
    for x in dummies.columns:
        dummy_name = f"{name}-{x}"
        df[dummy_name] = dummies[x]
    df.drop(name, axis=1, inplace=True)

```

```

# Now encode the feature vector

encode_numeric_zscore(df, 'duration')
encode_text_dummy(df, 'protocol_type')
encode_text_dummy(df, 'service')
encode_text_dummy(df, 'flag')
encode_numeric_zscore(df, 'src_bytes')
encode_numeric_zscore(df, 'dst_bytes')
encode_text_dummy(df, 'land')
encode_numeric_zscore(df, 'wrong_fragment')
encode_numeric_zscore(df, 'urgent')
encode_numeric_zscore(df, 'hot')
encode_numeric_zscore(df, 'num_failed_logins')
encode_text_dummy(df, 'logged_in')
encode_numeric_zscore(df, 'num_compromised')
encode_numeric_zscore(df, 'root_shell')
encode_numeric_zscore(df, 'su_attempted')
encode_numeric_zscore(df, 'num_root')
encode_numeric_zscore(df, 'num_file_creations')
encode_numeric_zscore(df, 'num_shells')
encode_numeric_zscore(df, 'num_access_files')
encode_numeric_zscore(df, 'num_outbound_cmds')
encode_text_dummy(df, 'is_host_login')
encode_text_dummy(df, 'is_guest_login')
encode_numeric_zscore(df, 'count')
encode_numeric_zscore(df, 'srv_count')
encode_numeric_zscore(df, 'serror_rate')
encode_numeric_zscore(df, 'srv_serror_rate')
encode_numeric_zscore(df, 'rerror_rate')
encode_numeric_zscore(df, 'srv_rerror_rate')
encode_numeric_zscore(df, 'same_srv_rate')
encode_numeric_zscore(df, 'diff_srv_rate')
encode_numeric_zscore(df, 'srv_diff_host_rate')
encode_numeric_zscore(df, 'dst_host_count')
encode_numeric_zscore(df, 'dst_host_srv_count')
encode_numeric_zscore(df, 'dst_host_same_srv_rate')
encode_numeric_zscore(df, 'dst_host_diff_srv_rate')
encode_numeric_zscore(df, 'dst_host_same_src_port_rate')
encode_numeric_zscore(df, 'dst_host_srv_diff_host_rate')
encode_numeric_zscore(df, 'dst_host_serror_rate')
encode_numeric_zscore(df, 'dst_host_srv_serror_rate')
encode_numeric_zscore(df, 'dst_host_rerror_rate')
encode_numeric_zscore(df, 'dst_host_srv_rerror_rate')

# display 5 rows

df.dropna(inplace=True,axis=1)
df[0:5]
# This is the numeric feature vector, as it goes to the neural net

# Convert to numpy - Classification
x_columns = df.columns.drop('outcome')
x = df[x_columns].values
dummies = pd.get_dummies(df['outcome']) # Classification
outcomes = dummies.columns
num_classes = len(outcomes)
y = dummies.values

df.groupby('outcome')['outcome'].count()

# create our target classifications
binary_y = df['attack_flag']

```

```

multi_y = df['attack_map']

test_binary_y = test_df['attack_flag']
test_multi_y = test_df['attack_map']

# build the training sets
binary_train_X, binary_val_X, binary_train_y, binary_val_y = train_test_split(
    to_fit, binary_y, test_size=0.6)
multi_train_X, multi_val_X, multi_train_y, multi_val_y = train_test_split(to_fit,
    multi_y, test_size = 0.6)

# model for the binary classification
binary_model = RandomForestClassifier()
binary_model.fit(binary_train_X, binary_train_y)
binary_predictions = binary_model.predict(binary_val_X)

# calculate and display our base accuracty
base_rf_score = accuracy_score(binary_predictions,binary_val_y)
base_rf_score

# define the list of models that we want to test
models = [
    RandomForestClassifier(),
    LogisticRegression(max_iter=250),
    KNeighborsClassifier(),
]

# an empty list to capture the performance of each model
model_comps = []

# walk through the models and populate our list
for model in models:
    model_name = model.__class__.__name__
    accuracies = cross_val_score(model, binary_train_X, binary_train_y, scoring='accuracy')
    for count, accuracy in enumerate(accuracies):
        model_comps.append((model_name, count, accuracy))

# a box plot will do well to show us overall performance and the variation in
the models.
result_df = pd.DataFrame(model_comps, columns=['model_name', 'count', 'accuracy'])
result_df.pivot(index='count',columns='model_name',values='accuracy').boxplot(
    rot=45)

<matplotlib.axes._subplots.AxesSubplot at 0x7f73068501d0>

# a helper function for getting some analytical data about our predictions
def add_predictions(data_set,predictions,y):
    prediction_series = pd.Series(predictions, index=y.index)

    # we need to add the predicted and actual outcomes to the data
    predicted_vs_actual = data_set.assign(predicted=prediction_series)
    original_data = predicted_vs_actual.assign(actual=y).dropna()
    conf_matrix = confusion_matrix(original_data['actual'],
                                   original_data['predicted'])

    # capture rows with failed predictions
    base_errors = original_data[original_data['actual'] != original_data['predicted']]

    # drop columns with no value

```

```

non_zeros = base_errors.loc[:, (base_errors != 0).any(axis=0)]

# identify the type of error
false_positives = non_zeros.loc[non_zeros.actual==0]
false_negatives = non_zeros.loc[non_zeros.actual==1]

# put everything into an object
prediction_data = {'data': original_data,
                   'confusion_matrix': conf_matrix,
                   'errors': base_errors,
                   'non_zeros': non_zeros,
                   'false_positives': false_positives,
                   'false_negatives': false_negatives}

return prediction_data

# capture our prediction data
binary_prediction_data = add_predictions(df,
                                       binary_predictions,
                                       binary_val_y)

# create a heatmap of the confusion matrix
sns.heatmap(data=binary_prediction_data['confusion_matrix'],
            xticklabels = ['Predicted Normal', 'Predicted Attack'],
            yticklabels = ['Actual Normal', 'Actual Attack'],
            cmap="YlGnBu",
            fmt='d',
            annot=True)

# data minus the rows with no variance
binary_prediction_data['non_zeros'].describe()

# see the standard deviation of the false positives
binary_prediction_data['false_positives'].std()

# see the standard deviation of the false negatives
binary_prediction_data['false_negatives'].std()

# distribution of false negatives--what attacks did we miss?
binary_prediction_data['false_negatives'].attack.value_counts().plot.bar()

import pandas as pd
import io
import requests
import numpy as np
import os
from sklearn.model_selection import train_test_split
from sklearn import metrics
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation
from tensorflow.keras.callbacks import EarlyStopping

# Create a test/train split. 25% test
# Split into train/test
x_train, x_test, y_train, y_test = train_test_split(
    x, y, test_size=0.25, random_state=42)

# Create neural net
model = Sequential()
model.add(Dense(10, input_dim=x.shape[1], kernel_initializer='normal', activation='relu'))
model.add(Dense(50, input_dim=x.shape[1], kernel_initializer='normal', activation='relu'))

```

```

model.add(Dense(10, input_dim=x.shape[1], kernel_initializer='normal', activation='relu'))
model.add(Dense(1, kernel_initializer='normal'))
model.add(Dense(y.shape[1], activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam')
monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3, patience=5, verbose=1, mode='auto')
model.fit(x_train,y_train,validation_data=(x_test,y_test),callbacks=[monitor],verbose=2,epochs=19)

import tensorflow.keras.backend as K
print('Learning Rate - ')
print(K.eval(model.optimizer.lr))
print('='*50)
model.summary()

import seaborn as sns
import datetime as dt

def confusion_matrix_func(y_test, y_test_pred):
    '''
    This function computes the confusion matrix using Predicted and Actual values and plots a confusion matrix heatmap
    '''
    C = confusion_matrix(y_test, y_test_pred)
    cm_df = pd.DataFrame(C)
    labels = ['back', 'butter_overflow', 'loadmodule', 'guess_passwd', 'imap', 'ipsweep', 'warezmaster', 'rootkit', 'multihop', 'neptune', 'nmap', 'normal', 'phf', 'perl', 'pod', 'portsweep', 'ftp_write', 'satan', 'smurf', 'teardrop', 'warezclient', 'land']
    plt.figure(figsize=(20,15))
    sns.set(font_scale=1.4)
    sns.heatmap(cm_df, annot=True, annot_kws={"size":12}, fmt='g', xticklabels=labels, yticklabels=labels)
    plt.ylabel('Actual Class')
    plt.xlabel('Predicted Class')

    plt.show()

# calculate roc curve
from sklearn.metrics import *
#fpr_RF, tpr_RF, thresholds_RF = roc_curve(y_test, pred)
from sklearn import preprocessing
def multiclass_roc_auc_score(y_test, pred, average="macro"):
    lb = preprocessing.LabelBinarizer()
    lb.fit(y_test)
    y_test = lb.transform(y_test)
    pred = lb.transform(pred)
    return roc_auc_score(y_test, pred, average=average)

print('Train data')
print(x_train.shape)
print(y_train.shape)
print('='*20)
print('Test data')
print(x_test.shape)
print(y_test.shape)
print('='*20)

# Measure accuracy
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score, f1_score, roc_auc_score

```

```

print('Predicting on the test data:')
start = dt.datetime.now()
escore = model.evaluate(x_test, y_test, batch_size=32)
pred = model.predict(x_test)
pred = np.argmax(pred,axis=1)
y_eval = np.argmax(y_test,axis=1)

vscore = metrics.accuracy_score(y_eval, pred)

rscore = recall_score(y_eval, pred, average='weighted')

ascore = precision_score(y_eval, pred, average='weighted')

f1score= f1_score(y_eval, pred, average='weighted') #F1 = 2 * (precision * re
call) / (precision + recall) for manual

roc_auc_socre = multiclass_roc_auc_score(y_eval, pred)

print('Completed')
print('Time taken:',dt.datetime.now()-start)
print('='*50)
print("Validation score: {}".format(vscore))
print('='*50)
print("Evaluation score: {}".format(escore))
print('='*50)
print("Recall score: {}".format(rscore))
print('='*50)
print("Precision score: {}".format(ascore))
print('='*50)
print("F1 score: {}".format(f1score))
print('='*50)
print("ROC-AUC score: {}".format(roc_auc_socre))

confusion_matrix_func(y_eval, pred)

```