Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)
Кафедра Штучного інтелекту
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Розробка та дослідження використання нейронних мереж у задачах виявлення шахрайських транзакцій мережі Ethereum
(тема)

Виконав:
студент 2 курсу, групи СШІм-22-1
Кітов А.В.
(прізвище, ініціали)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)
Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)
Освітня програма Системи штучного інтелекту
(повна назва спеціалізації)
Керівник проф. Філатов В.О.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ В.О. Філатов
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)
Кафедра Штучного інтелекту
(повна назва)
Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)
Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)
Освітня програма Системи штучного інтелекту
(повна назва)

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

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Кітову Антону Вадимовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка та дослідження методів використання нейронних мереж у задачах виявлення шахрайських транзакцій мережі Ethereum
затверджена наказом університету від 1 квітня 20 24 р. № 260Ст
2. Термін подання студентом роботи до екзаменаційної комісії 4 червня 20 24 р.
3. Вихідні дані до роботи Теоретичне дослідження рішення і опис технологій, робота з потоком даних, математична основа рішення, задача в предметній галузі, практична частина роботи з вибраним алгоритмом, вибір алгоритму машинного навчання, обробка текстових даних, попередня обробка та візуалізація даних для роботи, застосовані технології, програмна реалізація застосунку, процес роботи з текстами та вибір ЄВМ, проектування та тестування

4. Перелік питань, що потрібно опрацювати в роботі
1) Огляд існуючих систем моніторингу та розпізнавання образів
2) Вивчення математичної основи рішення
3) Розробка системи аналізу транзакцій
4) Тестування ефективності розроблених рішень шляхом багаторазового повтору
5) Оцінка розробленої системи
6) Вивчення прийнятності

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	01.04.2024	виконано
2	Аналіз предметної галузі та постановка задачі	04.04.2024 – 15.04.2024	виконано
3	Аналіз методів щодо вирішення поставленої	15.04.2024 – 20.04.2024	виконано
	задачі		
4	Вибір методу для вирішення задачі	20.04.2024 – 25.04.2024	виконано
5	Розробка базового прототипу	01.05.2024 – 15.05.2024	виконано
6	Підготовка пояснювальної записки	16.05.2024 – 05.06.2024	виконано
7	Захист перед ЕК	04.06.2024	виконано

Дата видачі завдання 1 квітня 2024 р.

Студент _____
(підпис)

Керівник роботи _____ проф. Філатов В.О.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 84 с., 45 рис., 3 табл., 1 дод., 23 джерела.

БІНАРНА КЛАСИФІКАЦІЯ, ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ, МАШИННЕ НАВЧАННЯ, НЕЙРОННІ МЕРЕЖІ, ШАХРАЙСЬКІ ОПЕРАЦІЇ.

Предмет дослідження – алгоритми класифікації транзакцій на основі структурних моделей.

Об'єкт дослідження – відкритий набір даних про транзакції користувачів платіжної системи з веб-платформи Kaggle.

Мета роботи – дослідження та розробка методів виявлення шахрайських операцій.

Отримані результати – використовуючи теоретичні та емпіричні методи дослідження, було розроблено інформаційну технологію з виявлення шахрайських операцій.

Актуальність дослідження зумовлена стрімким розвитком процесів глобалізації та цифровізації фінансових послуг, зокрема – здійсненням грошових переказів та проведенням операцій з використанням Ethereum, інтегрованих у платіжні сервіси фінансового сектору. У зв'язку з цим виникає нагальна потреба в розробленні передових методів виявлення незаконних дій, особливо з використанням криптовалют, для запобігання неправомірним транзакціям. Для оперативного виявлення шахрайських операцій доцільно застосовувати технології машинного навчання.

ABSTRACT

Master's thesis contains: 84 pp., 45 fig., 3 tabl., 1 ann., 23 references.

BINARY CLASSIFICATION, FRAUDULENT TRANSACTIONS, INFORMATION TECHNOLOGY, MACHINE LEARNING, NEURAL NETWORK.

Subject of research – algorithms for classifying transactions based on structural models.

The object of research is an open data set of transactions of payment system users from the Kaggle web platform.

Purpose – to study and develop methods for detecting fraudulent transactions.

The results obtained – using theoretical and empirical research methods, an information technology for detecting fraudulent transactions was developed.

The relevance of the study is due to the rapid development of globalization and digitalization of financial services, in particular, money transfers and transactions using Ethereum, integrated into the payment services of the financial sector. In this regard, there is an urgent need to develop advanced methods for detecting illegal activities, especially with the use of cryptocurrencies, to prevent illegal transactions. Machine learning technologies can be used to detect fraudulent transactions in a timely manner.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ШІ – штучний інтелект;

DL – Deep Learning – глибинне навчання;

DNN – Deep Neural Network – глибока нейронна мережа;

FL – Feature Learning – навчання ознак;

MLP – Multi Layer Perseptrone – мультишаровий персептрон;

RNN – Recurent Neural Network – рекурентна нейронна мережа.

ВСТУП

З року в рік блокчейн-технології та криптовалюти стають дедалі популярнішими. Ефіріум, зокрема, пропонує унікальні можливості для створення і взаємодії зі смарт-контрактами, децентралізованими додатками і багатьом іншим. Але такі можливості також приносять ризики: складність гарантування безпеки, ризики, пов'язані з приватністю і, звісно, шахрайські транзакції.

У міру того як сектор криптовалют зростає, зростає і необхідність у надійних інструментах для забезпечення його безпеки. Зокрема, шахрайські транзакції можуть спричинити колосальні збитки для інвесторів, бірж, стартапів у сфері DeFi та багатьох інших учасників екосистеми. Для боротьби з цими та іншими проблемами було розроблено системи протидії відмиванню грошей (AML) і протидії фінансуванню тероризму (CFT). Ці системи спрямовані на виявлення та запобігання потенційно підозрілим або шахрайським транзакціям.

Дослідження ефективних методів для виявлення шахрайських дій у криптовалютній мережі, особливо в мережі Ethereum, стає вкрай важливим. Індустрія попередження шахрайства прагне до інновацій, застосовуючи методи машинного навчання для вдосконалення своїх систем.

Об'єкт цього дослідження – навчальні вибірки даних зі звичайними та шахрайськими криптовалютними транзакціями.

Мета роботи – дослідити різні підходи до опрацювання незбалансованих даних і розробки моделей машинного навчання для виявлення шахрайства в мережі Ethereum.

Предмет дослідження – математичні методи аугментації даних, статистичні моделі та методи машинного навчання, включно з глибоким навчанням.

У криптовалютному секторі, де прозорість і децентралізація стоять наріжним каменем, правильне застосування систем AML і CFT може

слугувати важливим інструментом для зміцнення довіри і безпеки. Зрештою, це допоможе створити більш надійний і безпечний цифровий простір для всіх учасників екосистеми. Шахрайство є серйозною проблемою для урядів і бізнесу, і потрібні спеціальні методи аналізу для виявлення шахрайства з їх використанням. Деякі з цих методів включають виявлення знань у базах даних, інтелектуальний аналіз даних, машинне навчання і статистику.

Враховуючи ще й той факт, що щосекунди через криптовалютну мережу Ethereum проходять десятки операцій, тобто обсяг даних надзвичайно великий, виявлення та запобігання шахрайству в секторі є класичним завданням для машинного навчання з учителем. Завдання роботи:

– проаналізувати різні методи семплінгу в умовах незбалансованих класів (адже шахрайських транзакцій суттєво менше, ніж звичайних, що сильно впливає на роботу алгоритмів машинного навчання), дослідити вплив переобробки даних на ефективність побудованих моделей;

– визначити метрики для оцінювання моделей, порівняти різні архітектури для вирішення завдання виявлення шахрайства, підібрати оптимальні параметри;

– зробити висновки про можливість використання отриманої моделі для комерційних завдань, описати сильні сторони та недоліки отриманого рішення.

У першому розділі роботи досліджено предметну галузь: розглянуто питання актуальності та проблематику задачі, її формалізацію. У другому розділі розглянуто наявні підходи до виявлення операційних ризиків на основі статистичних методів і методів штучного інтелекту. Третій розділ представляє практичні результати дослідження та візуалізацію роботи системи.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Актуальність проблеми

Ethereum, подібно до інших криптовалютних мереж, став об'єктом незаконної діяльності різних користувачів. Деякі з цих діячів займаються такими схемами, як «фінансові піраміди», фішинг, відмивання грошей, шахрайство і злочинна діяльність [3].

Незважаючи на те, що псевдоанонімність у мережі Ethereum вважається однією з основних характеристик, вона стає об'єктом зловживань з боку деяких користувачів, які здійснюють незаконні дії. Ця тенденція чинить негативний вплив на суспільне сприйняття криптовалют загалом, незважаючи на постійний розвиток і вдосконалення криптовалютних технологій у прагненні створити надійну децентралізовану систему.

У рамках мережі Ethereum, де рівень анонімності користувачів високий, було виявлено випадки використання псевдонімів для проведення незаконних операцій. Однак, у відповідь на такі виклики, було розроблено методики та технології виявлення подібних дій, спрямовані на підвищення безпеки мережі та запобігання зловживанням.

Цікаво зазначити, що наявні дослідження наголошують на необхідності постійного вдосконалення систем безпеки в контексті криптовалют, щоб підтримувати довіру та позитивне сприйняття суспільства щодо цих інноваційних технологій [14].

З огляду на дійсність, своєчасне виявлення та попередження шахрайства є вкрай важливим завданням для компаній-постачальників платіжних послуг. Адже прогалини в цій сфері призводять до значних репутаційних втрат, збільшуючи відтік клієнтів і бізнесів через неналежний рівень безпеки платіжних систем.

Методи машинного навчання є одним із багатообіцяючих видів технологічної зброї проти фінансових шахрайств. Алгоритми машинного

навчання здатні відрізняти шахрайські операції від законних, не викликаючи підозр у тих, хто виконує транзакції. Машинне навчання може боротися з фінансовим шахрайством, оперуючи великими масивами даних краще і швидше, ніж коли-небудь могли б люди. Воно також значною мірою виключає проблему людського фактора і імплементація його методів зазвичай добре піддається масштабуванню.

1.2 Огляд проблематики виявлення шахрайства

Під аналітикою виявлення шахрайства розуміють поєднання методів виявлення шахрайства та аналітики даних, які використовуються для виявлення та запобігання випадкам шахрайства. До числа методів аналізу даних, що використовуються для виявлення шахрайства, належать інтелектуальний аналіз даних, кластерний аналіз, попереднє опрацювання даних і зіставлення даних.

Також можна виявити дані, свідомо пов'язані з шахрайством. Можливо, шахрайські дії найчастіше відбуваються в певний час доби, у певних географічних точках, на певних типах рахунків або в певних обсягах.

Нижче перераховано причини, через які аналітика виявлення шахрайства важлива для організацій:

– зниження ризику шахрайства – завдяки аналітиці виявлення шахрайства система закриває всі лазівки для проведення шахрайських дій. Шахраї мають обмежені можливості для здійснення шахрайських дій, що знижує ймовірність шахрайства;

– надійне виявлення шахрайства, аналітичні системи виявлення шахрайства дають змогу надійно виявляти шахрайські дії ще до того, як було завдано збитків. Йдеться про системи раннього виявлення. Такі системи здатні виявити будь-яку спробу вчинення шахрайських дій. Це підвищує рівень контролю і безпеки в організації;

– підвищення довіри клієнтів – аналітика виявлення шахрайства гарантує, що система організації перебуває в безпеці. Коли клієнти практично не стикаються з проблемами безпеки, у них виникає довіра. Підвищення довіри клієнтів сприяє підвищенню їхньої лояльності, що важливо для зростання організації;

– використання неструктурованих даних, багато шахраїв здійснюють шахрайські дії, коли дані неструктуровані. Аналітичні системи виявлення шахрайства здатні аналізувати неструктуровані дані для виявлення та запобігання шахрайським діям;

– підтримує інтеграцію даних – ця система збирає дані з різних джерел і об'єднує їх, що сприяє інтеграції даних в організації;

– виявлення прихованих закономірностей, деякі з традиційних методів виявлення шахрайства не дають змоги виявити приховані закономірності. Аналітичні методи виявлення шахрайства перевершують ці методи, оскільки дають змогу виявляти приховані тенденції, сценарії та закономірності;

– підвищення ефективності роботи організації – аналітичні методи виявлення шахрайства дають змогу звести до мінімуму шахрайські дії, що значно скорочує втрати доходів унаслідок шахрайства. У результаті застосування аналітики шахрайства організації отримують величезні фінансові вигоди. Ці системи підвищують ефективність і покращують процеси проведення фінансових операцій в організаціях.

Старі методи аналізу даних були орієнтовані на витяг кількісних і статистичних характеристик даних. Ці методи сприяють корисній інтерпретації даних і можуть допомогти краще зрозуміти процеси, що лежать в основі інформації.

Хоча звичайні методи аналізу даних можуть побічно привести нас до знань, вони все одно створюються людьми-аналітиками.

Щоб вийти за ці рамки, система аналізу знань повинна володіти значним обсягом знань і бути готовою до виконання завдань міркування на основі цих знань і, відповідно, наданих даних. У спробі розв'язати це завдання дослідники звернулися до ідей зі сфери машинного навчання.

Рішення в галузі машинного навчання і штучного інтелекту також можна розділити на дві категорії: «контрольоване» і «неконтрольоване» навчання.

Ці методи шукають рахунки, клієнтів, постачальників тощо, які поводяться «незвично», щоб вивести підозрілі оцінки, правила або візуальні аномалії, залежно від тактики.

Незалежно від того, чи використовуються контрольовані або неконтрольовані методи, слід мати на увазі, що їхні результати дають нам лише уявлення про ймовірність шахрайства. Жоден окремий статистичний аналіз не може гарантувати, що конкретний об'єкт може бути шахрайським, але вони виявлятимуть їх з дуже високим ступенем точності.

Згідно із законодавством України, шахрайство – це заволодіння чужим майном або придбання права на майно шляхом обману чи зловживання довірою [2]. Розглянемо специфіку цих злочинних дій, щоб краще зрозуміти, які основні питання можуть виникнути при виявленні шахрайських операцій методами машинного навчання.

Існує багато проблем і труднощів, коли справа доходить до виявлення шахрайства з нерегульованими криптовалютними операціями. Цей тип виявлення шахрайства значною мірою залежить від вивчення даних, і більша частина цих даних може бути недоступна у фінансових установах через їхній чутливий та особистий характер.

Крім того, через велику кількість транзакцій щодня, аналіз створює значні проблеми з точки зору інформаційних технологій і для дослідників, які аналізують дані. Оскільки методи виявлення шахрайства розвиваються та вдосконалюються, шахраї з часом змінюють свої методи досягнення цілей.

Шахрайські дії, пов'язані з платіжними системами, та власне транзакціями в сучасних цифрових криптомережах, можна віднести до категорії identity theft – злочину, вчинення якого передбачає незаконне заволодіння персональними даними жертви шахрайства. Жертвою може бути власник картки, платіжного рахунку тощо, а способи шахрая видати себе за цю особу доволі різноманітні – від злому інтернет-акаунта особи до викрадення її платіжної інформації, тобто різноманітні способи підтвердження особи для здійснення операцій з рахунком.

Розглянувши класифікацію шахрайств, можна припустити, що ефективне запобігання їм повинне передбачати наявність низки перевірок як операції, так і особи користувача, щоб втрата особистих даних не могла призвести до повного заволодіння шахраєм особи власника рахунку. Використання методів машинного навчання може стати результативним кроком перевірки транзакції на потенційну шахрайськість.

Говорячи про наявні комерційні рішення цього питання, варто згадати AnChainAI – відома своїм спеціалізованим підходом до розроблення імунітету блокчейну від шахрайства та кіберзлочинності. Вони використовують технології штучного інтелекту (зокрема, нейромережі) для виявлення ненормальної активності та аномалій у блокчейн-системах. Запатентована програма здатна підлаштовуватися під постійно мінливі шахрайські схеми і сповіщає користувачів про знайдені шахрайські транзакції.

1.3 Огляд наявних досліджень

Численні джерела літератури, що стосуються виявлення аномалій або шахрайства з криптотранзакціями, вже опубліковані та доступні для загального користування. Наприклад, детальний огляд, проведений Кліфтоном Фуа і його соратниками, показав, що методи, які використовуються в цій галузі, включають інтелектуальний аналіз даних і

автоматизацію виявлення шахрайства [3]. З дослідження випливає, що іноді вчені використовують занадто складні моделі, як-от глибокі нейронні мережі, замість менш ресурсовитратних методів, як-от логістична регресія або байєсівська мережа. При цьому ці простіші методи демонстрували конкурентоспроможні результати на тих самих даних, що може бути важливим міркуванням при аналізі мережі Ethereum.

В іншій статті Berg A., науковий співробітник GJUS&T в Hisar HCE, розглянув застосування моделей машинного навчання з учителем і без для виявлення шахрайства в мережі Bitcoin. Ним було проведено порівняння роботи моделей дерев рішень, методу опорних векторів і нейронних мереж. Незважаючи на те, що ці методи та алгоритми продемонстрували певні успіхи у виявленні шахрайських операцій, вони не змогли забезпечити постійне та послідовне рішення для виявлення шахрайства [5].

Дослідження, результатом якого стало створення більш ефективних моделей класифікації транзакцій, представили Wen-Fang YU та Na Wang, де вони використовували алгоритми Outlier mining, Outlier detection та Distance Sum [23], щоб точно передбачити шахрайську транзакцію в експерименті емуляції набору даних транзакцій певної біржі. Outlier mining – це галузь аналізу даних, яка здебільшого використовується в монетарній та інтернет-сфері. Вона займається виявленням об'єктів, які відокремлені від основної системи, тобто транзакцій, які є підозрілими. Вони взяли атрибути поведінки клієнта і на основі значення цих атрибутів розрахували відстань між спостережуваним значенням цього атрибута і його заздалегідь визначеним значенням. Цей алгоритм був заснований на реконструкції мережі, який дає змогу створювати уявлення про відхилення одного екземпляра від контрольної групи. Як правило, цей метод виявлявся ефективним для онлайн транзакцій середнього розміру [5].

Усім згаданим вище дослідженням необхідно було працювати з проблемою незбалансованості класів.

У таких навчальних вибірках, як правило, шахрайських транзакцій було не більше 10%.

Варто зазначити, що алгоритми машинного навчання в даний час використовуються, як правило, для аналізу всіх авторизованих транзакцій, з подальшим звітом про підозрілі випадки. Такі звіти зазвичай досліджуються співробітниками платіжних систем або децентралізованих бірж, які також можуть зв'язатися з ініціаторами транзакції операції. Це зі свого боку дасть змогу дослідити зміни в поведінці шахраїв.

1.4 Виклики в розробленні ML моделей для запобігання шахрайству

Більшість проблем, з якими стикаються інженери, що займаються розробкою моделей для виявлення криптошахрайства, не унікальні, а типові для виявлення шахрайства загалом. Так, можна виділити такі головні виклики:

– дані вкрай незбалансовані – 99% транзакцій звичайні;

– шахраї починають вигадувати і впроваджувати нові схеми з фроду після того, як розуміють, що їхні транзакції блокуються. Таким чином, еволюція їхніх методів відбувається досить швидко;

– характер транзакцій звичайних користувачів і шахраїв може бути сезонним – покупки перед святами або на чорну п'ятницю відрізняються за багатьма факторами від звичайних. Тобто, потрібно розмічати і навчати модель на великих часових проміжках;

– необхідність регулярно проводити пост-аналіз (рисунок 1.1), для виявлення нових сценаріїв і поповнення ними датасету (випливає з пункту 2);

– затримка в кілька днів у поданні актуальних даних для моделі, адже скарги від клієнтів на шахрайство доходять не відразу.

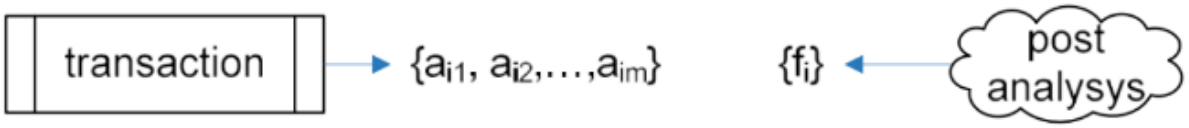

Рисунок 1.1 – Процес пост-аналізу

1.4.1 Складнощі навчання на незбалансованих даних

Дані вважаються незбалансованими, коли спостерігаються такі розподіли класів: 1:2, 1:10, 1:100, 1:1000 тощо. У датасеті Ethereum Fraud Detection Dataset [6], що був обраний для теоретичного дослідження в роботі, 20302 транзакцій, але лише 5675 з них шахрайські. Таким чином маємо 72.05% звичайних транзакцій (рисунок 1.2).

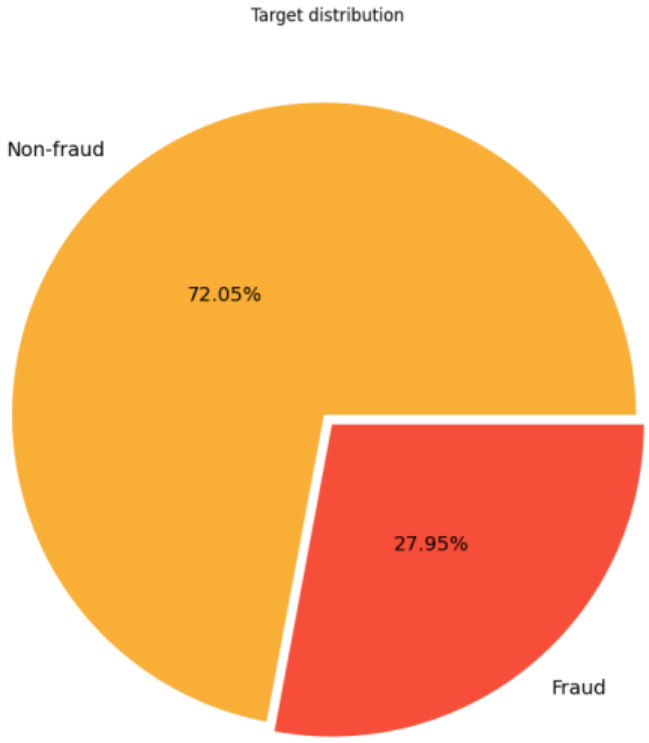

Рисунок 1.2 – Кількість звичайних і фродових транзакцій

Також виникає проблема в кількості даних, якщо ми до менш представленого класу застосовуємо різноманітні методи збільшення кількості спостережень, як от SMOTE або oversampling, і збільшуємо кількість прикладів до співвідношення 1:1, то отримуємо майже вдвічі більший датасет, що негативно позначається на швидкості навчання. Наприклад, для звичайної задачі класифікації зі збалансованим розподілом класів ми можемо необхідні тисячі або десятки тисяч прикладів для того, щоб розробити, оцінити і вибрати модель. вибравши ж для аналізу транзакції за кілька днів або місяців, отримаємо десятки мільйонів прикладів для тренування. Отже, працюючи із завданням виявлення шахрайства, необхідно дослідити методи не лише збільшення кількості прикладів мінорного класу для тренування моделі, а й методи зменшення кількості прикладів домінуючого класу, такі як TOMEK та random undersampling.

Дисбаланс класів є визнаним ускладнюючим фактором для класифікації. Хоча, деякі дослідження вказують на те, що висока диспропорція в класах не є єдиною причиною сильного падіння якості моделі під час навчання [7, с. 253]. Існує багато чинників, що погіршують, серед яких:

– шуми в характеристиках (label noise);
– розподіл даних;
– кількість прикладів із меншого класу [8].

Шуми в характеристиках є в прикладах, опис яких містить певні помилки або шум (рисунок 1.3). Виділяють два типи шумів: шум в атрибутах і шуму цільової змінної. Загалом вважається, що шум у цільовій змінній завдає набагато більше шкоди моделі, ніж шум атрибутів навіть для звичайних збалансованих моделей. вплив на незбалансовану модель, коли прикладів одного з класів і так мало, помилки в атрибутах або класі прикладу завдають руйнівної шкоди.

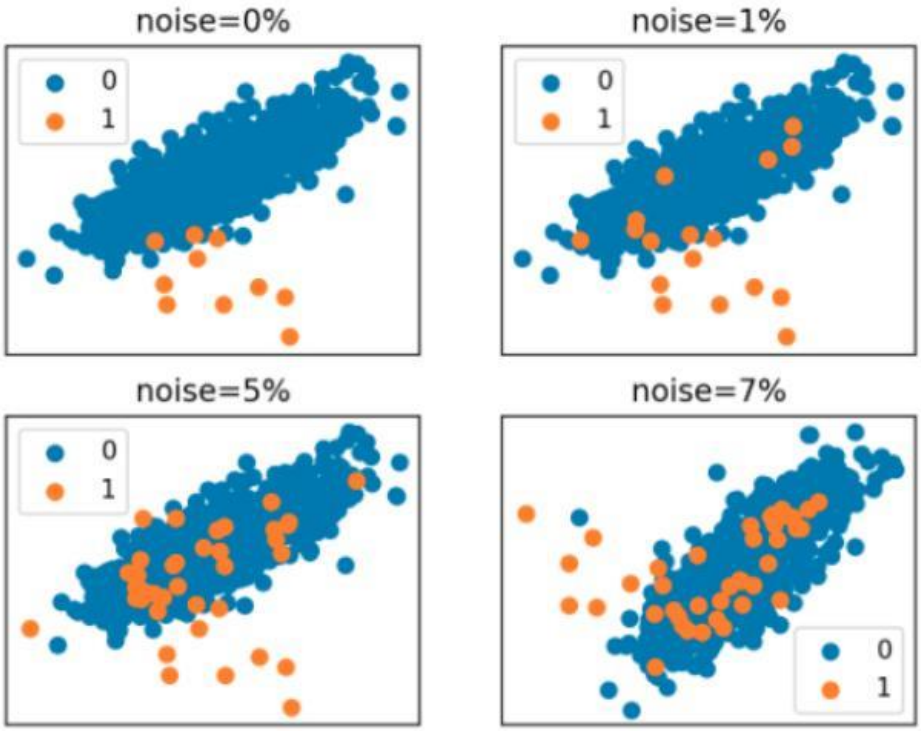

Рисунок 1.3 – Штучно зашумлені дані [8]

Другим важливим ускладнюючим фактором є розподіл прикладів у просторі ознак. Так, якщо зобразити візуально простір ознак (просто вивести на графік, якщо маємо 2–3 ознаки, або якщо ознак багато, то використати один з алгоритмів машинного навчання для візуалізації даних, як-от PCA чи t-SNE), то найкращим варіантом буде чіткий розподіл на кластери (рисунок 1.4). Тоді, одразу можемо сказати, що вдасться побудувати класифікатор, який досягає хороших результатів. Це можна сказати і про дані зі збалансованим, і про дані з незбалансованим розподілом. Однак, якщо простір ознак не має чіткої структури і приклади розташовані без чіткого розподілу на кластери, це може ускладнити завдання класифікації. У такому випадку, модель може мати складності з визначенням границь між класами і, відповідно, досягненням високої точності.

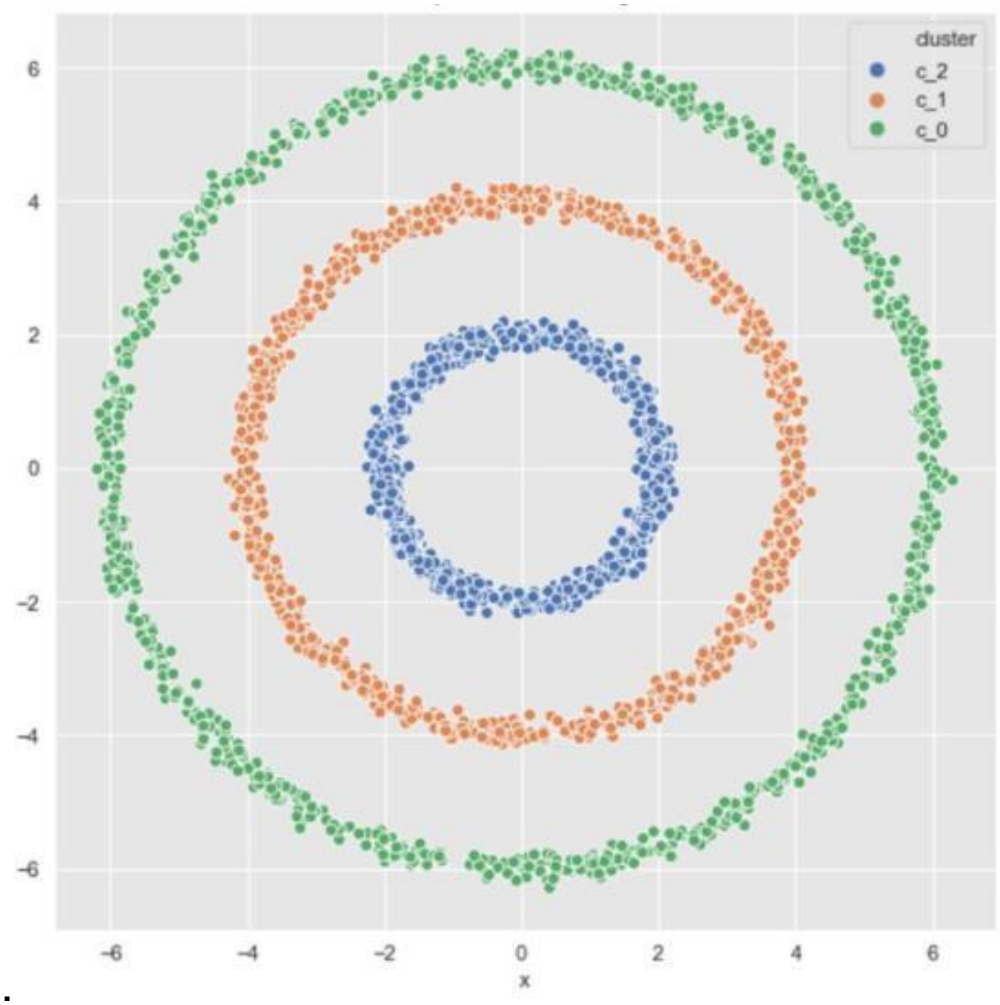

Рисунок 1.4 – Чіткий розподіл ознак x,y на кластери за класом [9]

У реальних, а не навчальних задачах це трапляється нечасто, тобто немає чіткого візуального розподілу на класи за візуалізацією ознак, і, скоріше за все, кожний клас можна додатково поділити на кілька підкласів, що призводить до створення кількох різних груп або кластерів прикладів за простором ознак. Наприклад, на рисунку 1.5 видно, що клас 0 сам розділений на 2 кластери, а також між класами 0 і 1 немає чіткого розподілу на просторі ознак.

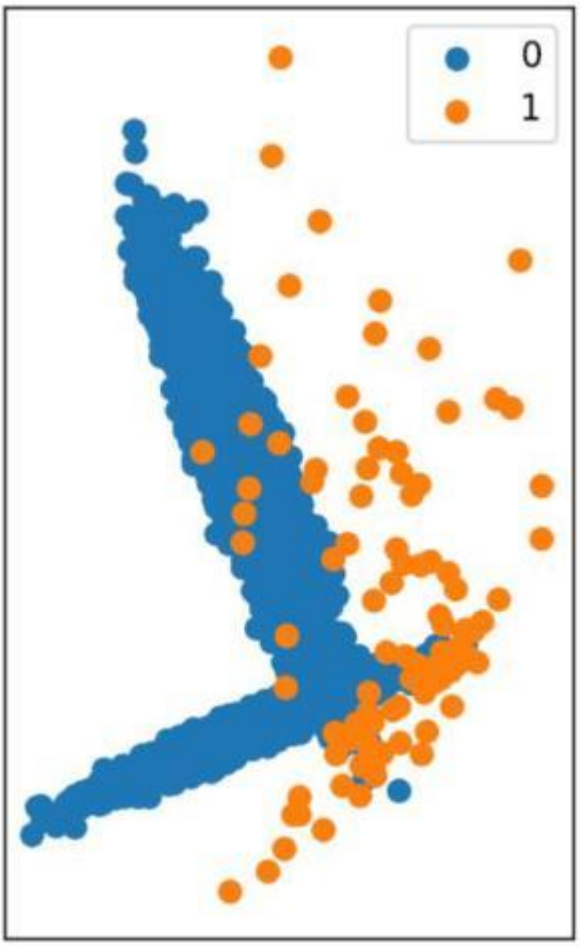

Рисунок 1.5 – Нечіткий розподіл ознак x,y на кластери за цільовою змінною [8].

Такі підгрупи формально називають диз'юнктами. диз'юнктати можуть бути різними за розміром. Невеликим диз'юнктом називають той диз'юнкт, який охоплює лише кілька прикладів із навчального набору даних [8]. Цей розподіл ускладнює відокремлюваність класів для моделі, адже необхідно ідентифікувати та включити кожну підгрупу чи кластер до визначення межі класу (рисунок 1.6). Статистичний аналіз даних для виявлення шахрайства виконує різні статистичні операції, такі як збір даних про шахрайство, виявлення шахрайства та підтвердження шахрайства шляхом проведення детальних розслідувань.

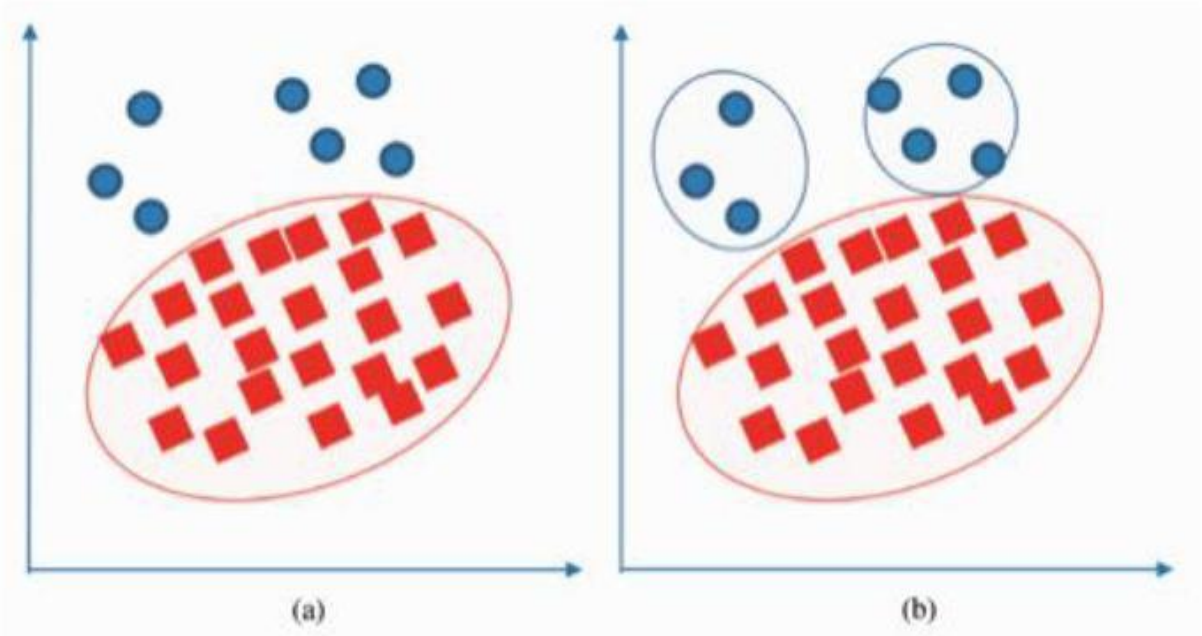

Рисунок 1.6 – Підгрупи в меншому класі [10]

Коли ми маємо дуже незбалансовані дані з невеликою кількістю прикладів меншого класу, поділ класів на підгрупи в межах одного класу є яскравою проблемою. Адже, маючи мало даних, маємо малу густину, в той час як кожен приклад дуже важливий, тому важко не перенавчити модель, і водночас розпізнати невеликі диз'юнкти навіть за малою кількістю прикладів. тут може знадобитися консультація експерта з предметної області.

Така недостатність щільності має особливий вплив в алгоритмах, що використовують підхід «розділяй і володарюй» (як-от дерева рішень) і в задачах покриття множин (як-от індукція правил), у яких наявність підгруп веде до створення невеликих диз'юнктів, адже в алгоритмах покриття множин і «розділяй і володарюй» моделі розділено на кілька частин, а концепцію класу представлено як диз'юнкт цих частин для кожного класу [10].

1.4.2 Необхідність регулярного донавчання моделі

Як було описано вище, нові сценарії шахрайства з'являються регулярно, як і поступово змінюється характер звичайних транзакцій, тож з'являється потреба дотренувати модель із частотою від кількох днів до місяця. така потреба також створює значні ускладнення, адже постає питання регулярної розмітки даних. У більшості блокчейн-мереж, включно з Ethereum, функції боротьби з шахрайством зазвичай реалізовані через розумні контракти та системи безпеки. Експерти з безпеки в цих мережах регулярно проводять моніторинг і аналізують дії потенційних шахраїв. Однак через велику кількість транзакцій у мережі Ethereum, часто доводиться фокусуватися на виявленні шахрайських операцій великих масштабів, тоді як втрати від невеликих сум можуть бути покриті за рахунок підвищеної комісії для проведення операцій.

Тож інформація, подана подібним відділом, була б неповною і недостатньою, враховуючи, що необхідно регулярно збирати інформацію і про звичайні транзакції, а не лише шахрайські.

На практиці для поповнення моделі актуальними даними часто використовують такий підхід:

– автоматичний збір скарг клієнтів на шахрайство (затримка 1–2 дні);

– автоматичне опрацювання звітів, складених відділом з протидії шахрайству, в яких міститься інформація про шахрайські операції, що призвели до значних збитків (затримка 2–7 днів);

– пост-аналіз (ручний або напівавтоматичний), спрямований на виявлення нових сценаріїв шахрайських дій (затримка в 30 днів), унаслідок якого датасет може бути поповнений не тільки шахрайськими прикладами, а й прикладами звичайних операцій, що також дає змогу коригувати модель на сезонність.

Процес дотренування моделі зображено на рисунку 1.7.

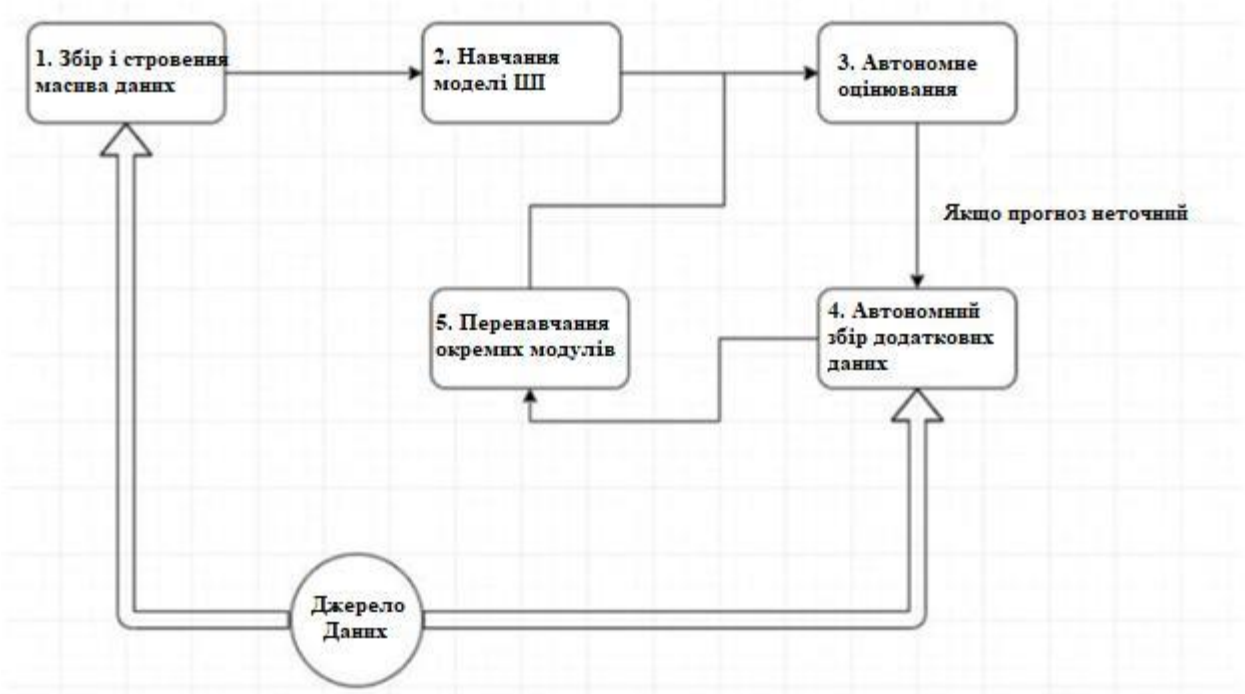

Рисунок 1.7 – Автоматичне дотренування моделі [11, с. 446]

Після того як було розроблено первинну модель і процес було автоматизовано, оновлення результатів відбувається наступним чином: до тестового набору датасету додають нові приклади та отримують передбачення за допомогою наявної моделі. Якщо показник обраної для оцінювання моделі метрики або метрик не змінюється суттєво, то для виявлення шахрайства залишаємо стару модель. В іншому випадку, вона має бути перетренована, для чого існують кілька зручних і швидких бібліотек мовою Python. У результаті, отримаємо нову модель для класифікації без втручання людини.

1.5 Постановка задачі

У рамках дослідження питання виявлення шахрайських операцій методами машинного навчання буде здійснено вивчення предметної області, а також розробку та застосування актуальних підходів до навчання математичних моделей з метою виявлення шахрайських транзакцій. Таким

чином, метою дослідження є створення інформаційної технології з виявлення шахрайства, що містить у собі результати виконання описаних вище кроків.

Варто зазначити, що завдання виявлення шахрайських дій у цьому контексті можна сформулювати як завдання бінарної класифікації. Беручи на вхід тестову вибірку з даними низки транзакцій, результатом роботи моделі є набір значень змінної-маркера шахрайства (назвемо її isFraud) для транзакцій з представленої вибірки.

Розглянувши наявні рішення у сфері виявлення шахрайських операцій, можна сформулювати низку вимог до майбутньої інформаційної технології. По-перше, вона повинна виконувати попередню обробку даних, що покращуватиме прогнозну силу майбутніх моделей. По-друге, вона має будувати моделі класифікації, які встановлюватимуть взаємозв'язки в навчальних даних із прийнятною точністю. Також доцільно вимагати можливість подання графіків залежностей у навчальній вибірці, які можуть бути корисними користувачеві.

При цьому, з погляду структури, розробка інформаційної технології передбачає створення окремих модулів для опрацювання та дослідження даних, що потребуватиме використання високорівневих мов програмування.

Цей розділ було присвячено дослідженню актуальності питання виявлення шахрайських операцій у середовищі криптовалютних транзакцій Ethereum, огляду існуючих на сьогодні рішень і підходів у цій сфері, а також постановці завдання дослідження й опису його етапів.

На основі наведеної інформації можна зробити висновок, що шахрайські операції є актуальною проблемою для сучасних платіжних систем, та існує потреба у створенні інформаційних технологій для виявлення шахрайських операцій.

Також варто згадати, що завдання виявлення шахрайських операцій було вирішено розглядати як випадок розв'язання задачі бінарної класифікації, яку можна розв'язати методами машинного навчання.

2 МАТЕМАТИЧНІ МОДЕЛІ ВИЯВЛЕННЯ ШАХРАЙСЬКИХ ТРАНЗАКЦІЙ

2.1 Стратегії семплінгу в умовах незбалансованості даних

Оверсемплінг (oversampling) і андерсемплінг (undersampling) – це важливі кроки під час роботи з незбалансованими даними, які дають змогу «вирівняти» розподіл кількості прикладів різних класів до 50:50.

Найпростіша стратегія оверсемплінгу – це копіювання наявних прикладів меншого класу, а андерсемплінгу – випадкове видалення елементів більшого класу (random undersampling). Однак, такі прості підходи не працюють для даних у децентралізованому секторі, адже у випадку з random undersampling доведеться видалити важливі дані про нешахрайські транзакції. Наприклад, під час аналізу даних щодо шахрайських транзакцій у мережі Ethereum можливе випадкове видалення більшості звичайних транзакцій із високим об'ємом, наприклад, вищим за встановлений поріг у 330 одиниць Ethereum – це зазначено в атрибуті maxValueSent [4], далі на малюнку наведено розподіл кількості транзакцій, що позначені як шахрайські FLAG = 1 (рисунок 2.1).

Ці звичайні транзакції можуть являти собою всього лише невелику частину, проте їх випадкове видалення може спотворити розподіл звичайних транзакцій. У той час як кількість виявлених шахрайських транзакцій може збільшитися, оскільки вони можуть мати різний розподіл обсягу.

Для підвищення точності системи виявлення шахрайства критично важливо оптимізувати обробку даних, враховуючи особливості розподілу обсягу транзакцій. Це може включати в себе використання методів, адаптованих до характеристик криптовалютних транзакцій, а також застосування більш точних і ефективних алгоритмів обробки даних, специфічних для аналізу шахрайських сценаріїв у мережі Ethereum.

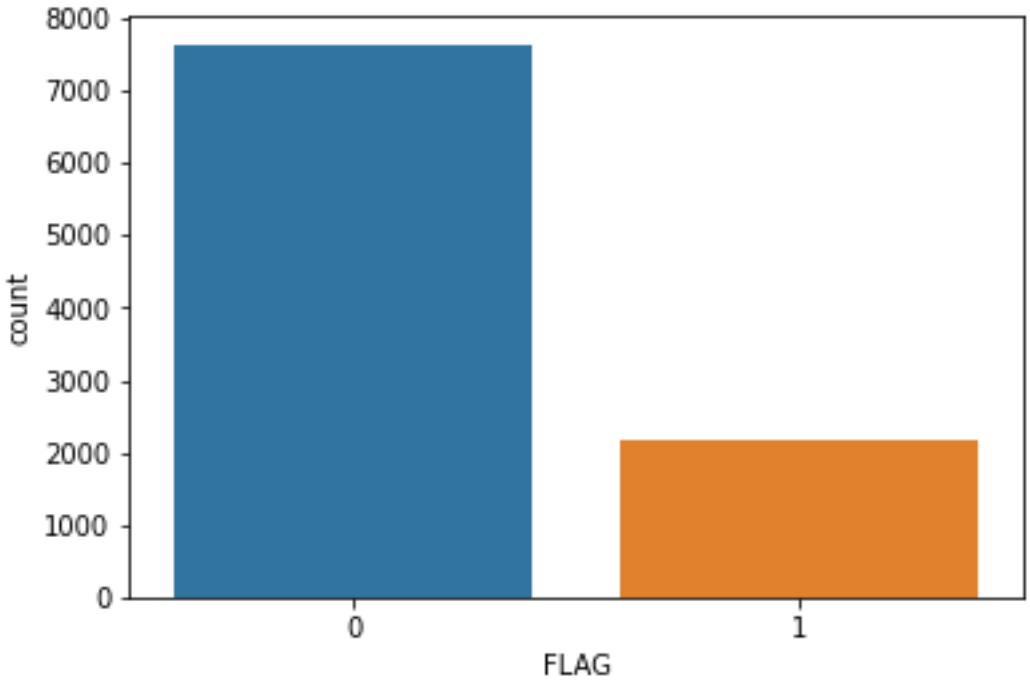

Рисунок 2.1 – Розподіл транзакцій за класами

У результаті, модель матиме високу точність, але при цьому не буде придатною до використання через упередженість до більших за сумою транзакцій, тому є сенс дослідити й використовувати в роботі математичні методи оверсамплінгу й андерсамплінгу, а не випадкові видалення та копіювання даних.

2.2 Синтетичне наповнення датасету за допомогою SMOTE

Алгоритм Synthetic Minority Oversampling Technique дає змогу замість копіювання наявних прикладів, на їхній основі створювати нові «синтетичні» приклади. На створення цього підходу вплинув успішний досвід аугментації даних, використаний для завдання класифікації датасету написаних від руки символів. Тоді додаткові дані для тренування створювалися за допомогою повороту, розтягування та інших перетворень фотографій, що позитивно вплинуло на точність класифікації. Однак, на

відміну від попереднього прикладу SMOTE синтезує нові приклади, оперуючи простором ознак, а не простором даних.

Більшу кількість меншого класу синтезують, використовуючи алгоритм K-NN (k найближчих сусідів), який застосовують до всіх прикладів класу, а потім створюють нові приклади, що лежать на сегментах ліній, які з'єднують найближчих сусідів класу. Залежно від того, яка кількість нових прикладів потрібна, сусіди з k найближчих сусідів вибираються випадковим чином. Більшість реалізацій SMOTE використовують 5 найближчих сусідів. Так, якщо потрібно збільшити вибірку на 400%, обирають 4 сусідів із 5-ти найближчих, і один новий генерується за кожним напрямком, псевдокод на рисунку 2.3.

Нові приклади створюються за таким алгоритмом:

– обчислюється різниця між вектором ознак (прикладом), який розглядається, і його найближчим вектором;

– отриману різницю множать на випадкове число від (0;1) і додають до початкового вектора ознак.

У результаті отримують випадкову точку на лінії між найближчим сусідом і розглянутим прикладом, на рисунку 2.2, [12, с. 328].

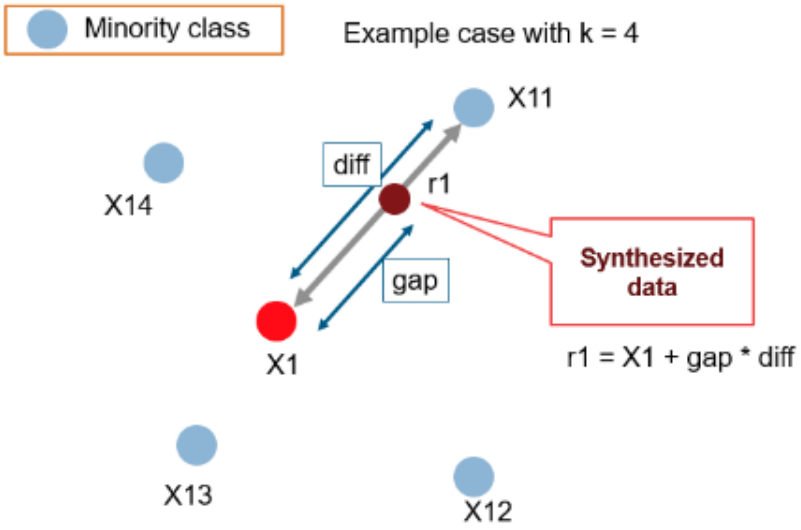

Рисунок 2.2 – Генерація нової точки в просторі ознак [13]

Псевдокод SMOTE.

Algorithm *SMOTE*(T, N, k)
Input: Number of minority class samples T; Amount of SMOTE N%; Number of nearest neighbors k
Output: $(N/100) * T$ synthetic minority class samples
1. (* *If N is less than 100%, randomize the minority class samples as only a random percent of them will be SMOTEd.* *)
2. **if** $N < 100$
3. **then** Randomize the T minority class samples
4. $T = (N/100) * T$
5. $N = 100$
6. **endif**
7. $N = (int)(N/100)$ (* *The amount of SMOTE is assumed to be in integral multiples of 100.* *)
8. k = Number of nearest neighbors
9. *numattrs* = Number of attributes
10. *Sample*[ ][ ]: array for original minority class samples
11. *newindex*: keeps a count of number of synthetic samples generated, initialized to 0
12. *Synthetic*[ ][ ]: array for synthetic samples
(* *Compute k nearest neighbors for each minority class sample only.* *)
13. **for** $i \leftarrow 1$ **to** T
14. Compute k nearest neighbors for i, and save the indices in the *nnarray*
15. Populate(N, i, *nnarray*)
16. **endfor**

Populate(N, i, *nnarray*) (* *Function to generate the synthetic samples.* *)
17. **while** $N \neq 0$
18. Choose a random number between 1 and k, call it nn. This step chooses one of the k nearest neighbors of i.
19. **for** $attr \leftarrow 1$ **to** *numattrs*
20. Compute: $dif = Sample[nnarray[nn]][attr] - Sample[i][attr]$
21. Compute: gap = random number between 0 and 1
22. $Synthetic[newindex][attr] = Sample[i][attr] + gap * dif$
23. **endfor**
24. $newindex{+}{+}$
25. $N = N - 1$
26. **endwhile**
27. **return** (* *End of Populate.* *)
End of Pseudo-Code.

Рисунок 2.3 – Алгоритм SMOTE [12, с. 329]

У результаті, завдяки згенерованим прикладам класифікатор створює більші та менш специфічні регіони рішень (рисунок 2.4). Точність коректного розпізнавання меншого класу теж зростає за оверсамплінгу за допомогою SMOTE > 200%, порівняно з простим копіюванням прикладів (рисунок 2.5).

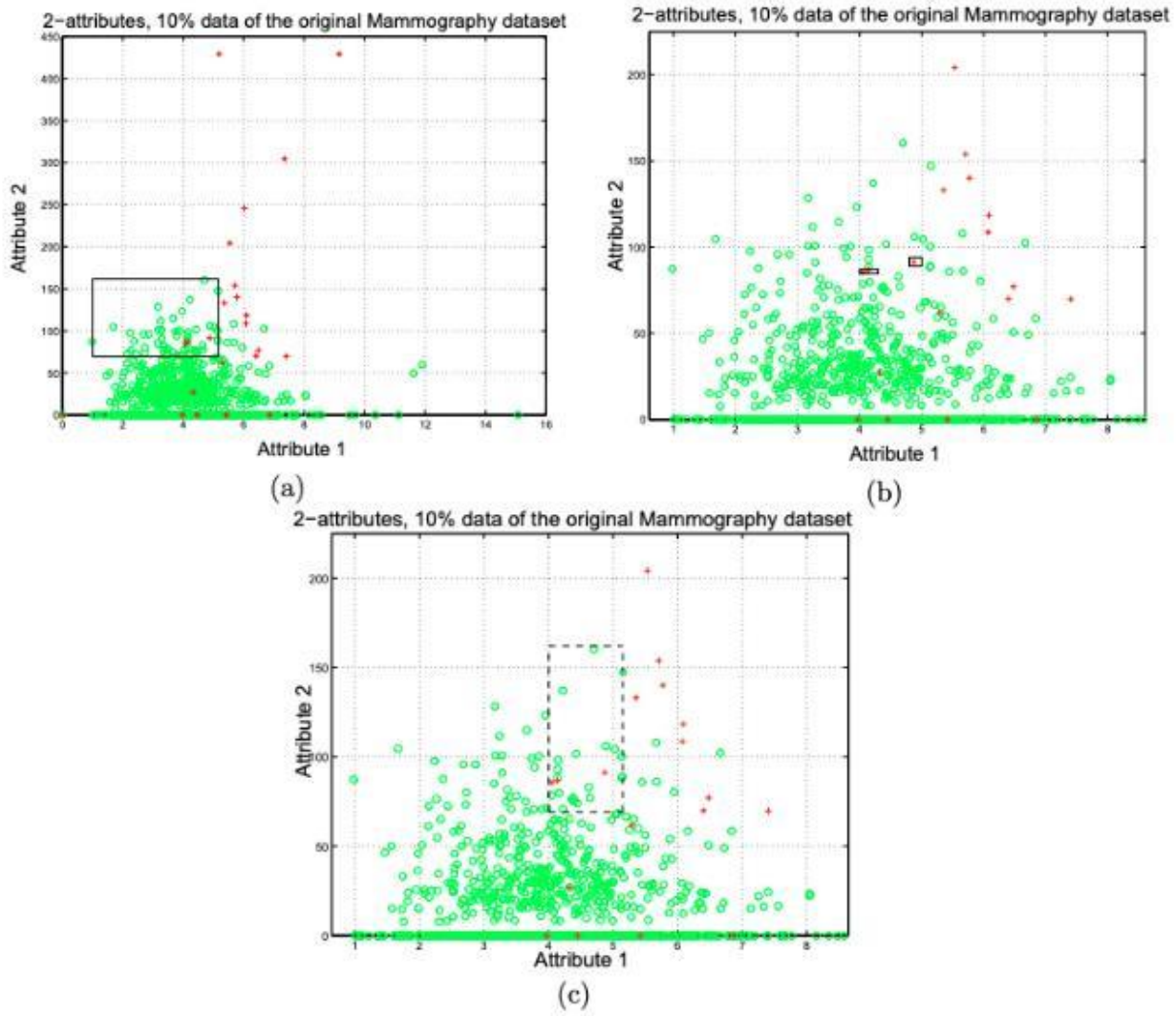

Рисунок 2.4 – Вплив згенерованих за допомогою SMOTE прикладів (позначені на c) [12, с. 327].

Крім того, експерименти показали, що якщо комбінувати такі методи оверсамплінгу, як SMOTE та андерсемплінг, то це позитивно впливає на якість вихідної моделі, бо в такий спосіб модель повністю звільняється від упередженості до більшого класу. Отже, перед використанням класифікатора важливо провести аналіз простору ознак та розподілу класів, щоб забезпечити оптимальні умови для досягнення високої ефективності моделі.

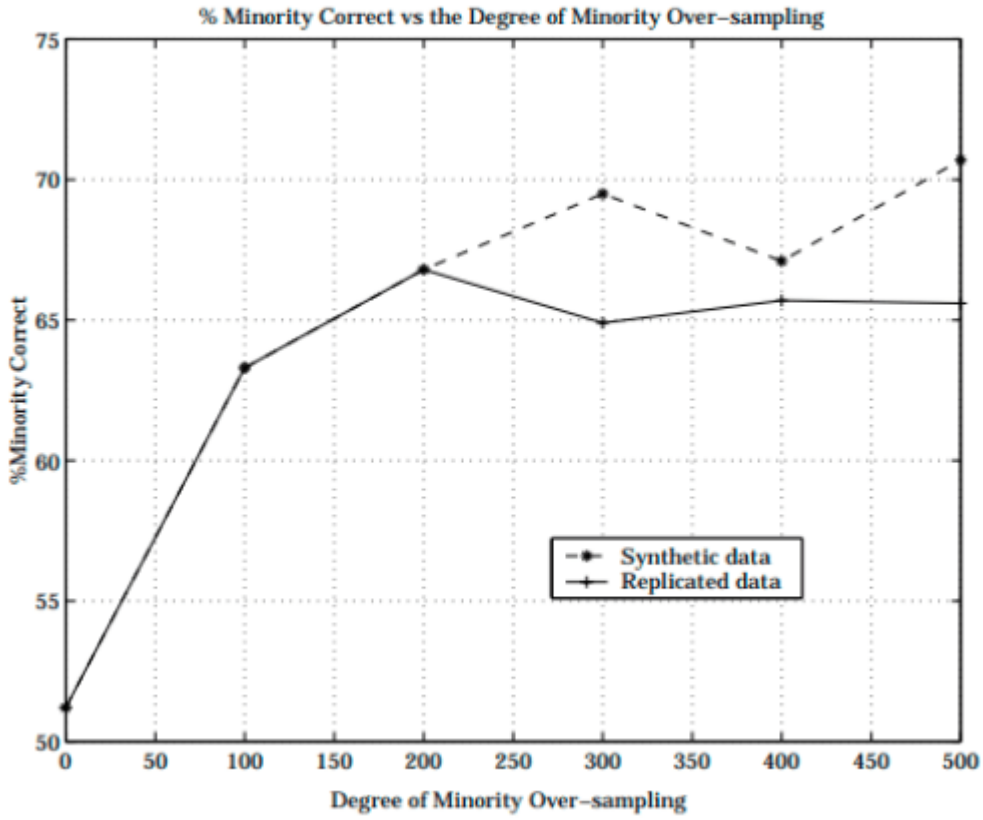

Рисунок 2.5 – Порівняння точності розпізнавання меншого класу з використанням дублювання та SMOTE [12, с.321].

2.2.1 Математичні методи видалення прикладів мажоритарного класу

Один із найпопулярніших підходів до андерсемплінгу – це використання зв'язків Томека (Tomek Links), що був презентований 1976 р., проте залишається актуальним і сьогодні через свою простоту та ефективність.

Нехай приклади E_i і E_j належать до різних класів, а d (E_i і E_j) – відстань між зазначеними прикладами. Пара (E_i і E_j) називається зв'язком Томека, якщо не знайдеться жодного прикладу E_m такого, що буде справедливою сукупність нерівностей (2.1):

$$d(E_i, E_m) < d(E_i, E_j). \quad (2.1)$$

Згідно з цим підходом усі мажоритарні записи, що входять у зв'язки Томека, мають бути видалені з набору даних. Цей спосіб добре видаляє записи, які можуть виявитися шумом (рисунок 2.6) [14].

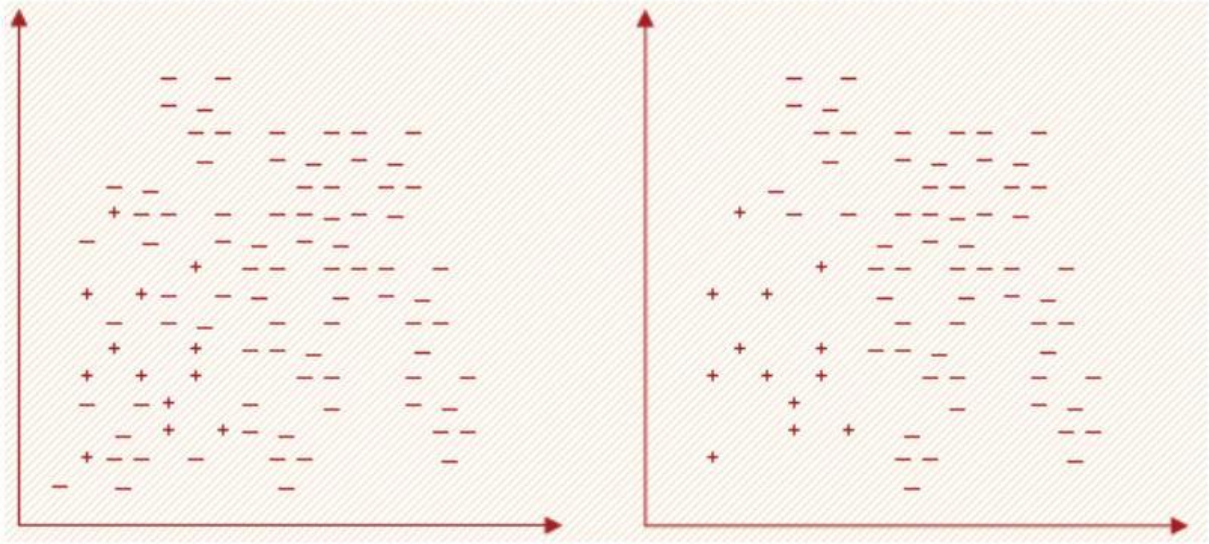

Рисунок 2.6 – Датасет до видалення зв'язків Томека та після [14]

Правило найближчого ущільненого сусіда (Condensed Nearest Neighbor Rule Undersampling, правило US-CNN) – правило, яке використовується для андерсемплінгу, і кінцева мета застосування якого – обрати такий набір прикладів, що не спричинятиме падіння в ефективності моделі. Таку вибірку називають мінімальним консистентним набором.

Такого результату досягають шляхом перебору всіх прикладів датасету та додаванням їх до мінімально консистентного набору, якщо їх не можна правильно класифікувати, використовуючи приклади, які вже є в наборі. Спочатку, такий підхід був запропонований і використаний, щоб зменшити кількість пам'яті комп'ютера, яку необхідно виділити для роботи алгоритму k найближчих сусідів (KNN).

Для використання цього підходу для задачі незбалансованої класифікації, мінімальний консистентний набір складається з усіх прикладів

меншого класу, і тільки приклади більшого класу, які не вдається коректно класифікувати, послідовно додають до набору. Під час роботи алгоритму використовується KNN, щоб класифікувати розглянуті приклади і вирішити, чи потрібно їх додавати в набір [15]. Таким чином, на відміну від методу зв'язків Томека, де обирають приклади, які необхідно видалити, правило найближчого ущільненого сусіда обирає, які приклади слід вибрати для подальшої класифікації.

Робота алгоритму на простому датасеті з двома ознаками (рисунок 2.7, 2.8).

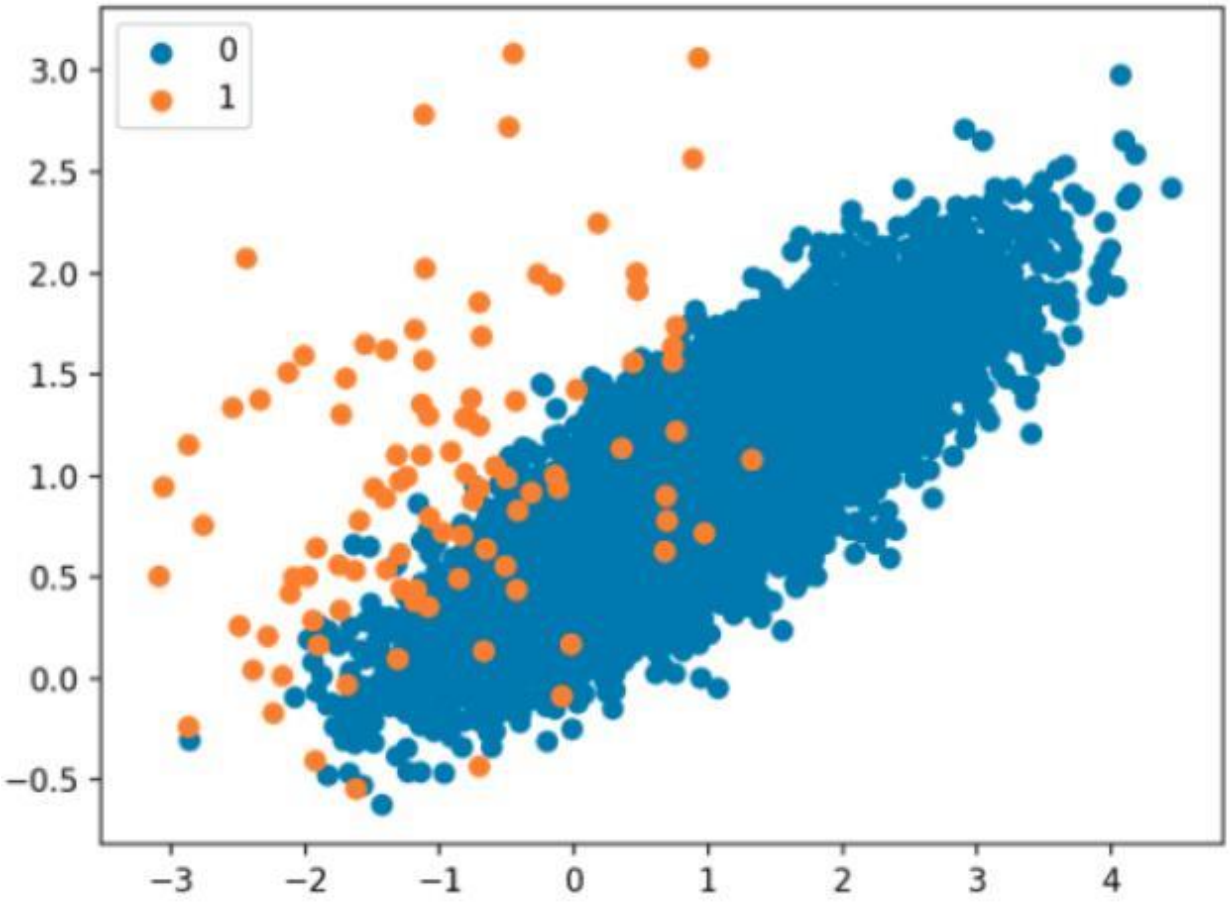

Рисунок 2.7 – Набір даних до андерсемплінгу US-CNN[15].

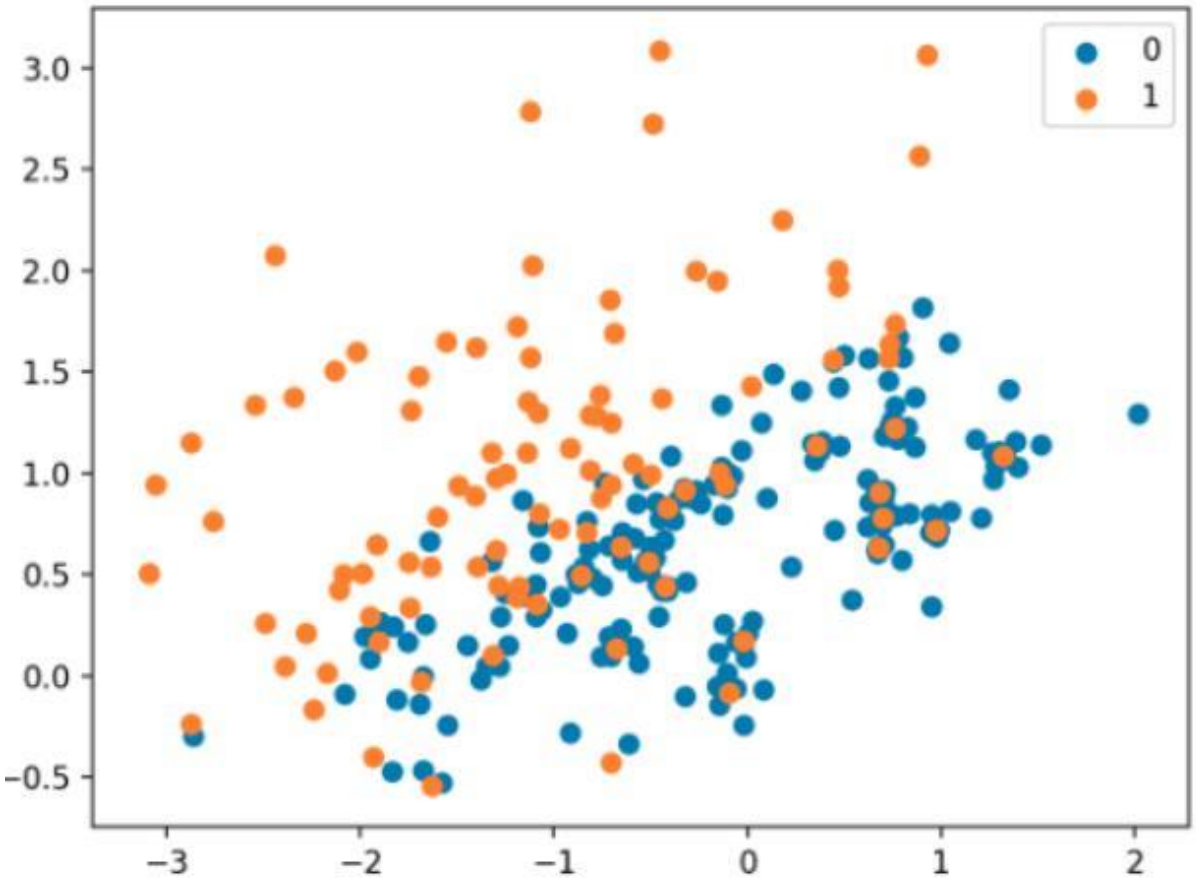

Рисунок 2.8 – Набір даних після андерсемплінгу US-CNN [15].

Дійсно, залишаються найважливіші приклади для класифікації на поточних даних. Однак слід звернути увагу на те, що всі точки далі за віссю X ($X \in [2.5, 4.5]$) класу 0 були видалені. Тобто залишаються лише приклади більшого класу «навколо» прикладів меншого.

Метод одностороннього вибору (One-sided Selection, OSS) поєднує два вищезгадані алгоритми – зв'язки Томека та US-CNN, а точніше полягає в послідовному застосуванні першого та другого алгоритмів.

Приклади одного класу, що лежать дуже близько до прикладів іншого опиняються за зв'язками Томека, і видаляються, адже розглядаються як шум і не погіршують якість класифікатора, який може досягнути хороших показників у метриках лише перенавчаючись, якщо таких прикладів багато. Після цього, метою US-CNN є прибрати приклади з більшого класу, що лежать далеко від межі рішень (decision border).

Приклад (рисунок 2.9) – що є результатом роботи алгоритму – усі приклади меншого класу і незашумлені та релевантні приклади більшого класу [7, с. 84].

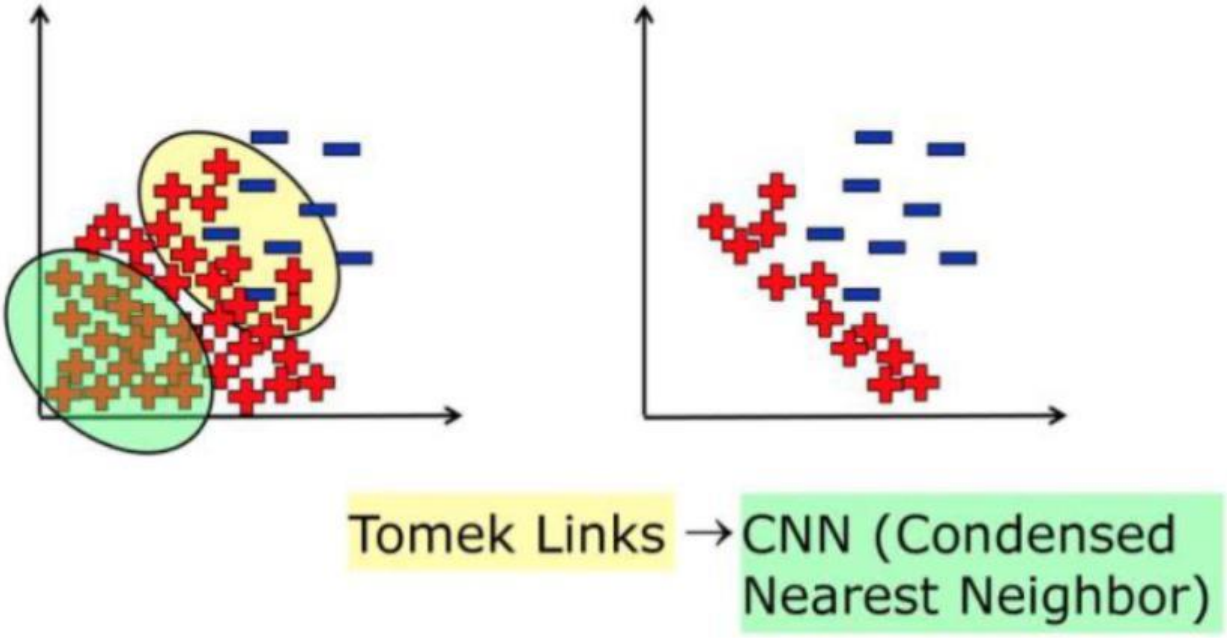

Рисунок 2.9 – Алгоритм OSS [16]

2.2.2 Плюси та мінуси використання under та over-семплінгу

Хоча описані вище підходи (SMOTE, зв'язки Томека, US-CNN, OSS) працюють із даними набагато «розумніші», ніж random oversampling і простий undersampling, все ж недоліки використання цих підходів не нівелюються повністю. Основні недоліки наведено в таблиці 2.1. У цій методиці відображаються моделі та розподіли ймовірностей різних видів шахрайської діяльності в бізнесі, як з точки зору різних параметрів, так і з точки зору розподілів ймовірностей.

Таблиця 2.1 – Недоліки використання оверсемпінгу та андерсемплінгу

Недодискретизація	Передискретизація
– можуть бути видалені потенційно корисні дані	– перенавчання; – значно збільшує час, що витрачається на обробку даних і навчання; – можливе копіювання або генерування нових прикладів на основі шуму.

Однак не використовуючи ці підходи, якість вихідної моделі буде значно нижчою, адже зазначені підходи мають низку переваг, список яких наведено в таблиці 2.2.

Таблиця 2.2 – Переваги Використання оверсемплінгу та андерсемплінгу

Недодискретизація	Передискретизація
– значно зменшує час, що витрачається на обробку даних і навчання; – знижує частковий збіг між класами на користь меншого класу, правильно визначити який важливіше, ніж приклади більшого класу; – знищує дані, які точно не будуть корисні під час класифікації.	– немає втрати інформації; – можливість наповнити датасет копіями, або навіть штучно згенерованими прикладами, що дуже важливо в умовах нестачі даних.

2.3 Методи прогнозування

У контексті розробки системи визначення шахрайських транзакцій у мережі Ethereum, важливим етапом є вибір ефективного алгоритму машинного навчання. Кожен з алгоритмів являє собою унікальний підхід до вирішення завдання і має свої переваги та недоліки.

Перед нами стояло завдання порівняти кілька ключових алгоритмів, щоб вибрати найкраще рішення для наших конкретних потреб. У цьому аналізі ми розглянемо чотири широко використовуваних алгоритми: Логістична регресія, Random Forest Classifier, XGBoost і Багатошаровий персептрон (MLP).

Розглянемо порівняння переваг і недоліків алгоритмів у таблиці 2.3.

Таблиця 2.3 – Порівняльна характеристика обраних алгоритмів

Алгоритм	Переваги	Недоліки
Логістична регресія	1. Простота інтерпретації. 2. Низькі вимоги до обчислювальних ресурсів.	1. Неефективний при складних нелінійних залежностях. 2. Потрібна лінійність.
Класифікатор Random Forest Classifier	1. хороше опрацювання великого обсягу даних. 2. Здатність працювати з різноманітними типами даних.	1. Споживання ресурсів за великої кількості дерев. 2. Чутливий до шуму.
XGBoost	1. Висока продуктивність. 2. Робота з різними типами даних.	1 Вимагає ретельного налаштування гіперпараметрів. 2. Можливість перенавчання.
Багатошаровий персептрон	1. Здатність моделювати складні нелінійні залежності. 2. Пристосованість до великих обсягів даних.	1 Потрібне ретельне налаштування гіперпараметрів. 2. Чутливий до вибірки.

Ми обрали Логістичну регресію, Random Forest Classifier, XGBoost і Багатошаровий персептрон (MLP) для вирішення задачі визначення шахрайських транзакцій з таких причин:

– різноманіття підходів: кожен з обраних алгоритмів надає унікальний підхід до вирішення завдання. Логістична регресія підходить для простих моделей з лінійними залежностями, тоді як Random Forest і XGBoost здатні обробляти складні нелінійні взаємозв'язки. MLP, своєю чергою, підходить для моделювання складних нелінійних залежностей у даних;

– стійкість до великих даних, Random Forest і XGBoost відомі своєю здатністю ефективно обробляти великі обсяги даних, що є важливим критерієм для завдання визначення шахрайських транзакцій у мережі Ethereum;

– висока продуктивність, XGBoost відомий своєю високою продуктивністю і широким використанням у змаганнях з машинного навчання. MLP також має здатність моделювати складні взаємозв'язки і підлаштовуватися під дані;

– комбінований підхід – використання декількох алгоритмів дає змогу створити комбіновану модель, яка може поліпшити узагальнення і точність прогнозів;

– гнучкість налаштування, у вибраних алгоритмів є гіперпараметри, які можна налаштовувати під конкретні вимоги завдання, забезпечуючи гнучкість у підході до оптимізації моделей.

Вибір цих чотирьох алгоритмів забезпечує нас безліччю інструментів для роботи з даними в мережі Ethereum і дає змогу задовольнити різноманітні вимоги завдання визначення шахрайських транзакцій.

2.3.1 Логістична регресія

Логістична регресія – це статистичний алгоритм бінарної класифікації, який використовує логістичну (сигмоїдну) функцію для формування передбачень. Логістична регресія, порівняно з іншими, є доволі простим алгоритмом, проте часто показує результати класифікації, а також є швидкою в реалізації та тренуванні.

У результаті передбачення логістичної регресії отримують імовірність p від 0 до 1. Що є результатом використання сигмоїдної функції для передбачення та значною перевагою порівняно зі схожим алгоритмом – лінійною регресією, результат передбачення якої може виходити за 0 або 1, рисунок 2.10.

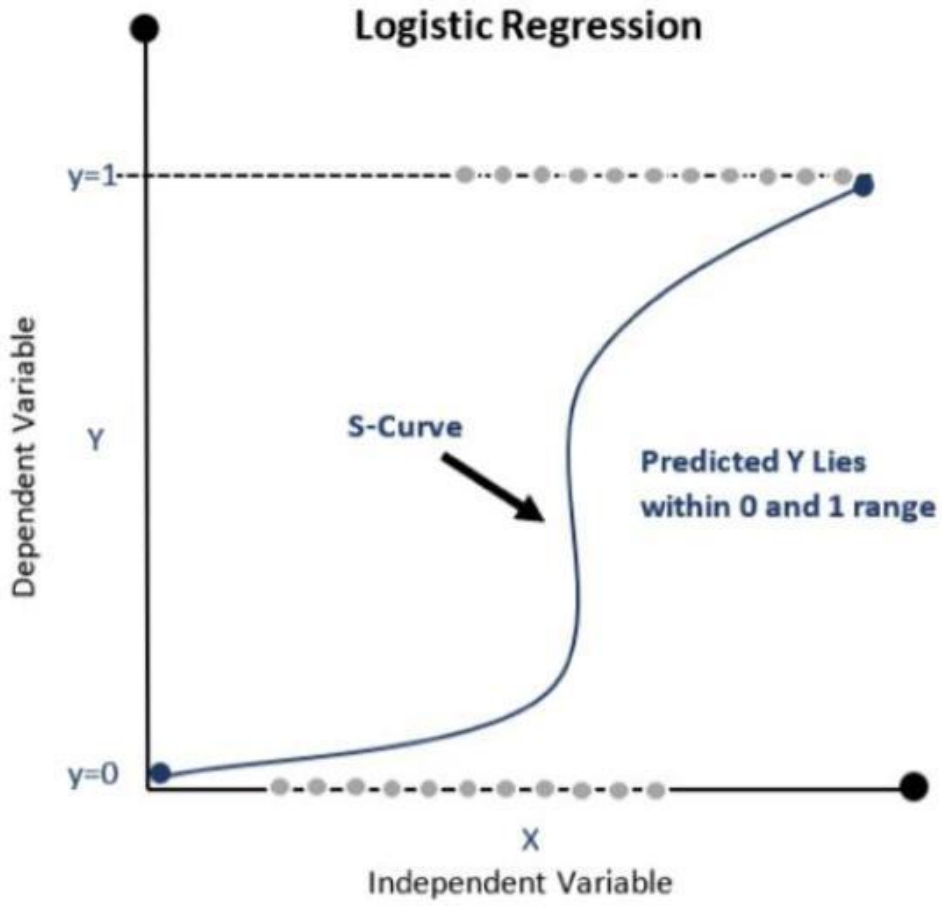

Рисунок 2.10 – Передбачення в алгоритмі логістичної регресії [17]

Належність же до класу визначається таким чином:

$$p \geq 0.5, \text{класс} = 1, \tag{2.2}$$

$$p < 0.5, \text{класс} = 0. \tag{2.3}$$

Формула сигмоїдної функції:

$$S(z) = \frac{1}{1 + e^{-z}}. \tag{2.4}$$

Під час навчання для знаходження оптимальних ваг використовується така функція втрат, значення якої ми власне хочемо оптимізувати за допомогою градієнтного спуску:

$$J(\theta) = -\sum_{i=1}^{n} y_i * \ln(z_i) + (1 - y_i) * \ln(1 - z_i), \tag{2.5}$$

де θ – ваги;

n – кількість прикладів;

z = $h\theta(x)$=$S(\theta_0 + \theta_1 x_1 + \cdots + \theta_b x_b)$;

x – вхідний вектор;

y – вихідний клас.

Функція 2.4 була спеціально виведена для використання із сигмоїдною функцією активації, і є інтуїтивно зрозумілою (наприклад, якщо клас y = 1, то найкращий варіант – щоб z_i (пророцтво для поточних ваг) було якомога ближче до 1, тоді ln (z_i) буде близьким до 0 і значення всієї функції буде вищим, тобто кращим; аналогічно для y = 0), що є ще одним плюсом використання логістичної регресії.

Як було зазначено раніше, оптимізація 2.3 відбувається за допомогою градієнтного спуску (рисунок 2.11).

$$\theta_j := \theta_j - a * \frac{\partial}{\partial - \theta_j} * J(\theta), \tag{2.6}$$

де α – крок навчання.

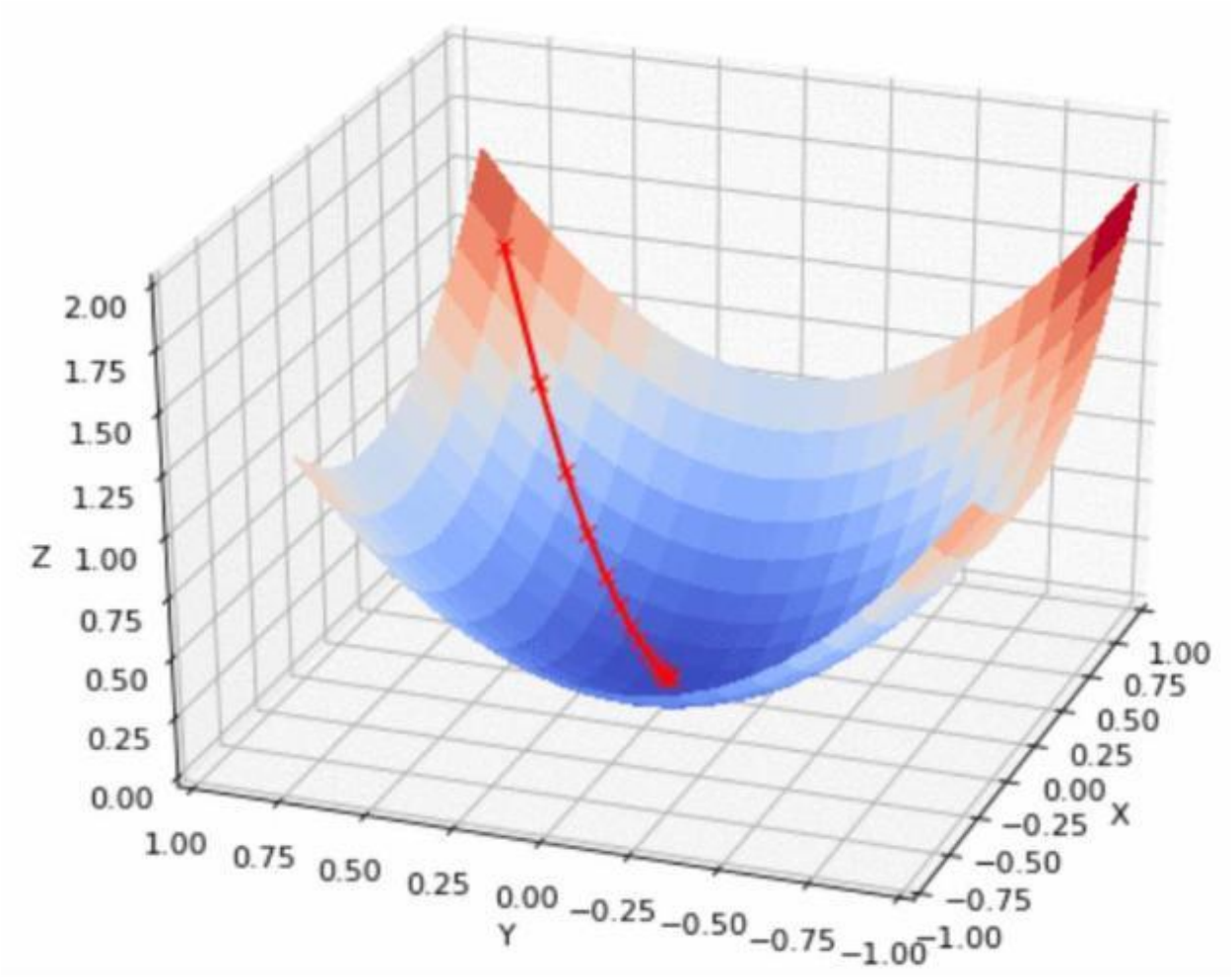

Рисунок 2.11 – Пошук оптимуму градієнтним спуском [18]

Впровадження штучного інтелекту для запобігання шахрайству допомогло компаніям посилити внутрішню безпеку та оптимізувати бізнес-процеси. Завдяки підвищенню ефективності штучний інтелект став важливою технологією для запобігання шахрайству у фінансових установах.

2.3.2 Класифікатор Random Forest Classifier

Метод Random Forest – це беггінг над вирішальними деревами. Крім того, цей метод використовує випадкові підпростори. Опишемо основну ідею.

Маємо навчальну вибірку X розміром $L * D$ (L об'єктів із D ознаками):

– обираємо кількість дерев у лісі: N;

– кожне з N дерев будуємо на вибірці X_n (n=$\overline{1,N}$), отриманій за допомогою бутстрепа, при цьому дерево будується не на всьому ознаковому просторі, а тільки з використанням d<D ознак. Ці d ознак вибирають випадковим чином. Дослідним шляхом отримано такі рекомендації: у задачах класифікації вибирати $d = \sqrt{D}$, у задачах регресії d=D/3;

– підсумковий класифікатор отримують шляхом агрегування відповідей N дерев (вибором мажоритарного класу для класифікації; медіани або середнього для регресії).

Перейдемо до основних гіперпараметрів випадкового лісу.

Випадкові ліси для задач класифікації та регресії в бібліотеці Scikit-learn представлені класами RandomForestClassifier і RandomForestRegressor. Основними гіперпараметрами моделей є:

– n_estimators (кількість дерев в ансамблі);

– criterion (критерій для розбиття вибірки у вершині);

– max_features (кількість ознак для побудови одного дерева; див. рекомендацію вище);

– min_samples_leaf (мінімальна кількість об'єктів у аркуші; рекомендується встановлювати рівним 1 для класифікації і рівним 5 для регресії);

– max_depth (максимальна глибина дерева).

Зауважимо, що для побудови випадкового лісу дерева навмисно роблять перенавченими. По-перше, це прискорює процес їхньої побудови, а

по-друге, зменшує корельованість дерев між собою. Від загального перенавчання випадковий ліс «страхує» випадковий вибір об'єктів для навчання бутстреп, випадковий вибір підпросторів ознак і усереднення відповідей великої кількості дерев [19].

Що більше дерев, то краща якість, але час налаштування і роботи RF також пропорційно збільшуються. Звернемо увагу, що часто при збільшенні n_estimators якість на навчальній вибірці підвищується (може навіть доходити до 100%), приклад на рисунку 2.12, 2.13.

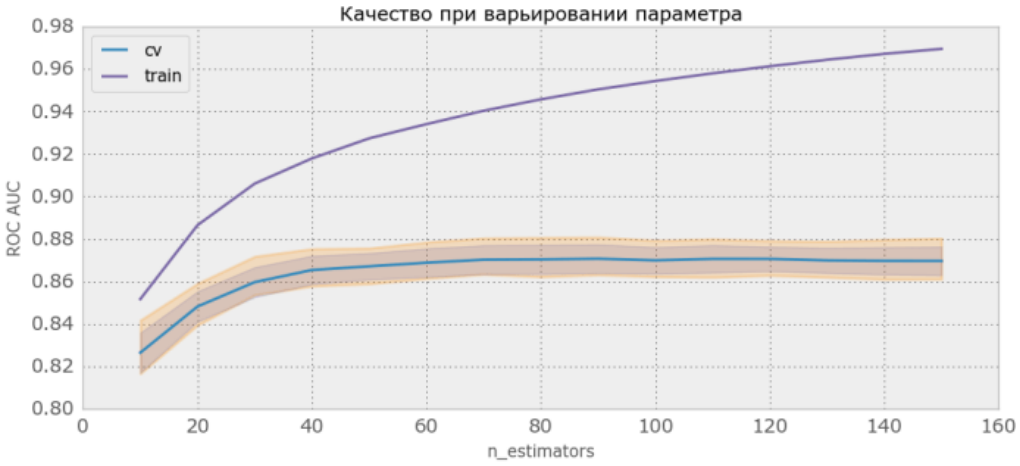

Рисунок 2.12 – Збільшення n_estimators на графіку

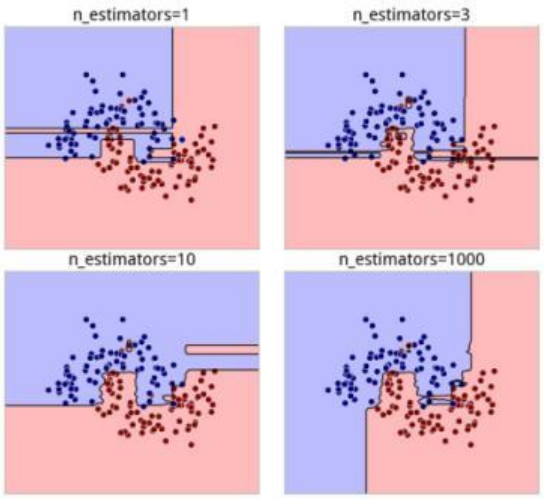

Рисунок 2.13 – Різниця кількості n_estimator

Характеристика числа ознак для вибору розщеплення – max_features. Графік якості на тесті від значення цього параметра унімодальний, на навчанні він строго зростає. При збільшенні max_features збільшується час побудови лісу, а дерева стають «більш одноманітними». За замовчуванням він дорівнює sqrt(n) у завданнях класифікації та n/3 у завданнях регресії (рисунок 2.14). Його налаштовують насамперед.

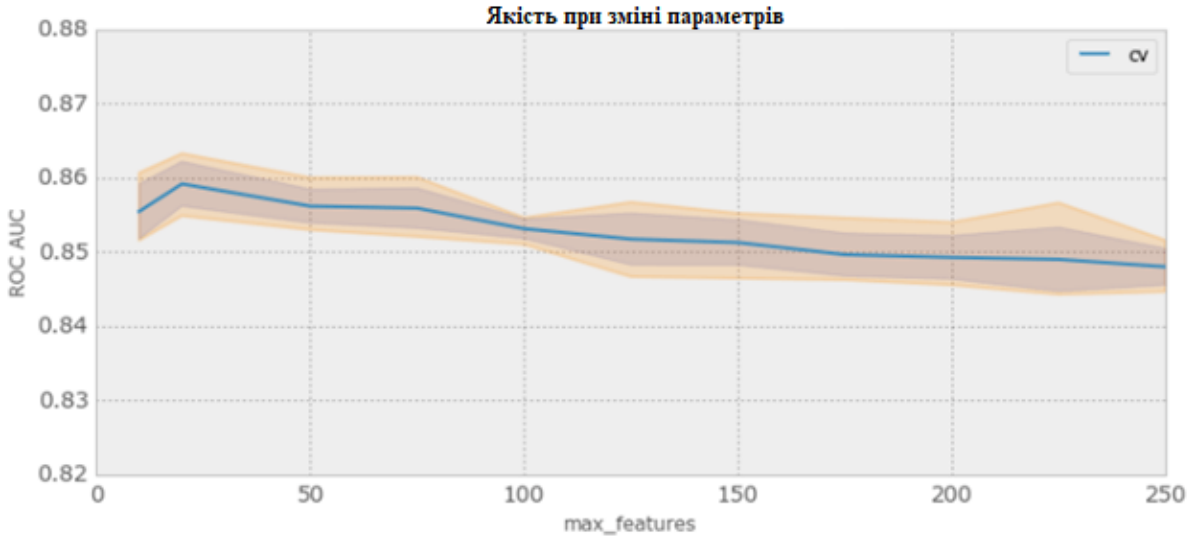

Рисунок 2.14 – Налаштування параметра max-features

2.3.3 Нейронна мережа

Існує безліч алгоритмів навчання заснованих на концепції нейронних мереж, адаптованих під різні завдання, як-от CNN для розпізнавання зображень, LSTM для роботи з часовими даними. Натомість для завдання створення класифікатора для запобігання шахрайству в секторі транзакцій у блокчейні достатньо класичного алгоритму, що застосовує штучні нейронні мережі – багатошарового перцептрона (MLP), (рисунок 2.15).

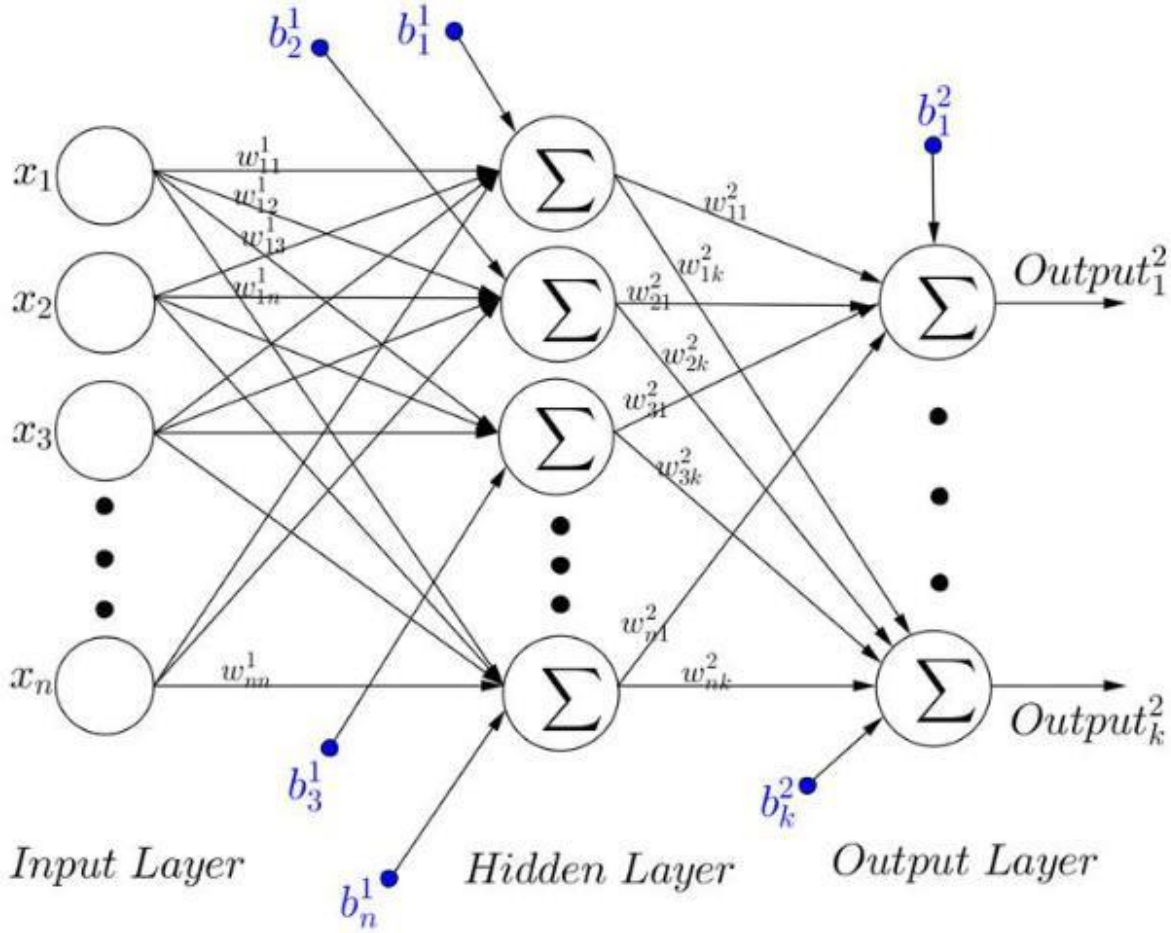

Рисунок 2.15 – Графічне представлення архітектури багатошарової нейронної мережі

MLP можна розглядати як алгоритм апроксимації функції. У теорії, якщо нейронна мережа має достатню кількість шарів, добре спроєктована та обмежень в обчислювальних здібностях немає, то будь-яка функція може бути приблизно змодельована за допомогою MLP.

Спрощено, процес навчання виглядає так:

– початкові ваги вибираються випадково;

– навчальні дані перший раз проходять через мережу й отримуємо перший результат;

– обчислюється помилка в обчисленнях, використовуючи певну функцію втрат;

– відштовхуючись від помилки і функції обчислюються часткові похідні, методом градієнтного спуску вирішується, в який бік рухатися, щоб мінімізувати помилку;

– весь процес повторюється доти, доки помилка не буде мінімальною.

Під час побудови алгоритму на основі нейронної мережі слід звертати увагу на підбір гіперпараметрів – змінних, що визначають структуру мережі та як алгоритм буде тренуватися. Найважливішими параметрами є:

– кількість прихованих шарів мережі – слід їх додавати доки помилка на тестових даних не перестане значно покращуватися;

– виняток (dropout) – деякий відсоток нейронної мережі можна вимкнути, щоб не перенавчитися на вхідних даних і побудувати в міру глибоку мережу;

– швидкість навчання – велика швидкість змушує модель навчатися швидше, однак можна «перескочити» оптимум;

– функція активації – додає нелінійний компонент до моделі, вибір залежить від типу завдання. Наприклад, для бінарної класифікації використовують сигмоїдну функцію активації.

2.3.4 XGBoost алгоритм

XGBoost – популярна бібліотека градієнтного бустингу для навчання на GPU, розподілених обчислень і розпаралелювання. Вона точна, добре адаптується до будь-яких типів даних і завдань, має чудову документацію і загалом дуже проста у використанні.

Нині це стандартний алгоритм для отримання точних результатів у предиктивному моделюванні за допомогою машинного навчання. Це найшвидша бібліотека градієнтного посилення для R, Python і C++ з дуже високою точністю.

Ансамблеве навчання об'єднує кілька моделей, що навчаються, для поліпшення загальної продуктивності, підвищення передбачуваності та точності в машинному навчанні та прогностичному моделюванні.

З технічного погляду сила ансамблевих моделей проста: вони можуть об'єднувати тисячі дрібніших навчальних моделей, навчених на підмножинах вихідних даних. Це може призвести до таких цікавих спостережень, як:

– дисперсія загальної моделі значно зменшується завдяки об'єднанню в ансамбль;

– зміщення також зменшується завдяки бустингу;

– загальна передбачувальна здатність покращується завдяки підсумовуванню.

Ансамблеві методи можна розділити на дві групи залежно від способу генерації учнів:

– послідовні ансамблеві методи – учні генеруються послідовно. У цих методах використовується залежність між базовими учнями. Кожен, кого навчають, впливає на наступного, таким чином, можна вивести загальну батьківську поведінку. Популярним прикладом послідовних ансамблевих алгоритмів є AdaBoost;

– паралельні ансамблеві методи – навчальні генеруються паралельно. Базові навчальні елементи створюють незалежно один від одного, що дає змогу вивчити й використовувати ефекти, пов'язані з їхньою незалежністю, і зменшити похибку завдяки усередненню результатів. Прикладом реалізації цього підходу є Random Forest.

Ансамблеві методи можуть використовувати однорідні навчальні (навчальні з однієї родини) або різнорідні навчальні (навчальні з кількох родин, якомога точніші й різноманітніші).

Перейдемо до важливих характеристик ансамблевих алгоритмів.

Пакування створено для зменшення загальної дисперсії за рахунок усереднення ефективності декількох оцінок. Агрегування декількох вибіркових підмножин вихідного набору даних для навчання різних учнів, які вибираються випадковим чином із заміною, що відповідає основній ідеї бутстреп-агрегування. Зазвичай у Bagging використовується механізм голосування для класифікації (Random Forest) і усереднення для регресії (рисунок 2.16).

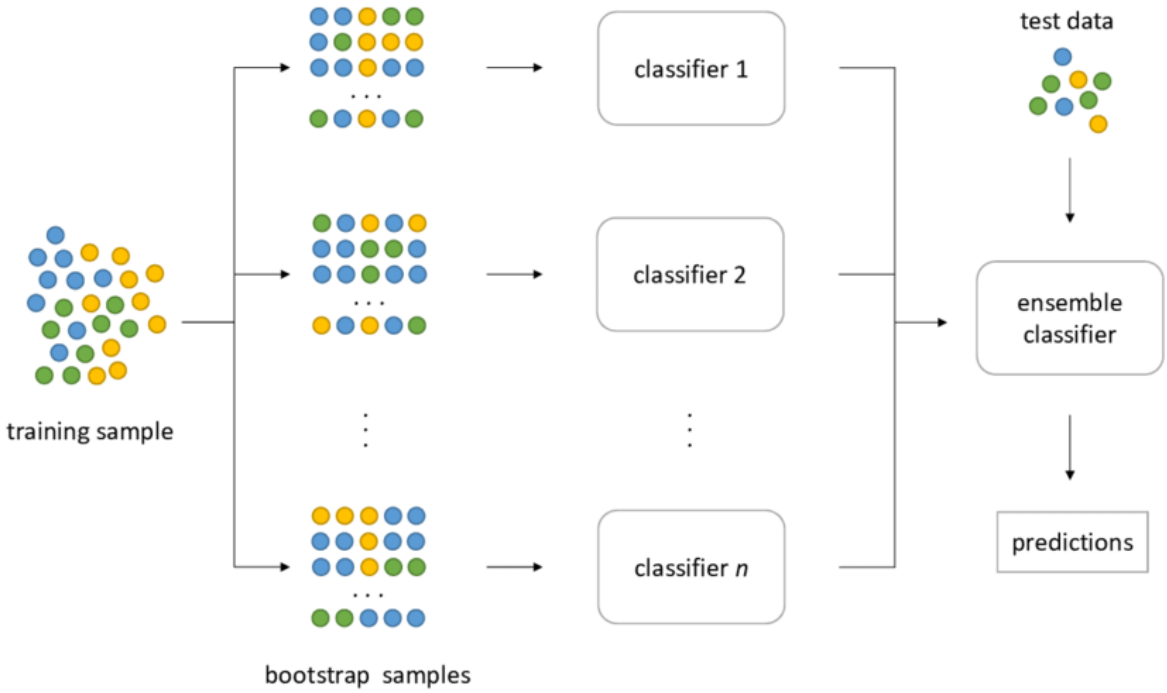

Рисунок 2.16 – Візуалізація характеристики ансамблевої моделі Bagging

Методика підсилення дає змогу підібрати слабкі навчані – навчані, які мають низьку передбачувальну здатність і працюють трохи краще, ніж випадкове вгадування – до певної зваженої підмножини вихідного набору даних. Вищу вагу мають підмножини, які раніше були неправильно класифіковані (рисунок 2.17).

Потім прогнози учнів об'єднуються за допомогою механізмів голосування в разі класифікації або зваженої суми для регресії.

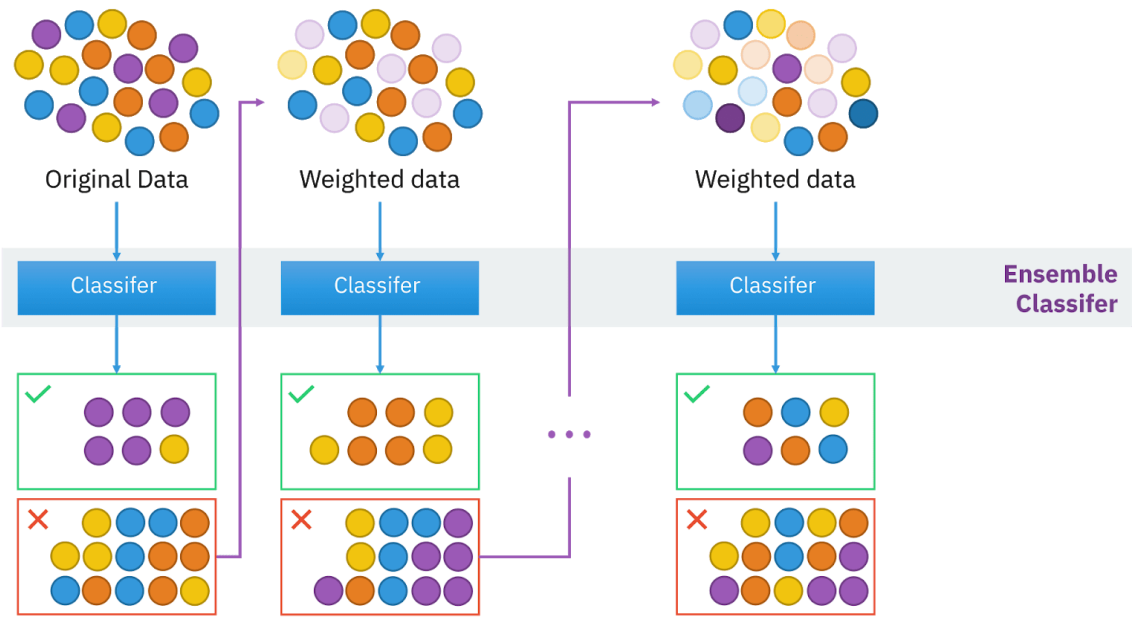

Рисунок 2.17 – Характеристика моделі Boosting

AdaBoost – Adaptive Boosting. Логіка, реалізована в алгоритмі, така:

– усі класифікатори (ті, що навчаються) першого раунду навчаються з використанням однакових вагових коефіцієнтів;

– у наступних раундах бустингу адаптивний процес збільшує вагу точок даних, які були неправильно класифіковані тими, кого навчають, у попередніх раундах, і зменшує вагу для правильно класифікованих.

Алгоритми бустінгу працюють за принципом спочатку побудови моделі на навчальному наборі даних, а потім побудови другої моделі для виправлення помилок у першій моделі. Ця техніка повторюється до тих пір, поки помилки не зменшаться і не буде досягнуто точного прогнозування набору даних. Алгоритми бустінгу функціонують аналогічно, поєднуючи численні моделі (слабкі навчальні моделі) для отримання кінцевого результату (сильні навчальні моделі).

На рисунку 2.18 наведено псевдокод операції.

Initialization:

1. Given training data from the instance space
$S = \{(x_1, y_1), ..., (x_m, y_m)\}$ where $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y} = \{-1, +1\}$.
2. Initialize the distribution $D_1(i) = \frac{1}{m}$.

Algorithm:

for $t = 1, ..., T$: **do**

Train a weak learner $h_t : \mathcal{X} \to \mathrm{R}$ using distribution D_t.
Determine weight α_t of h_t.
Update the distribution over the training set:

$$D_{t+1}(i) = \frac{D_t(i) e^{-\alpha_t y_i h_t(x_i)}}{Z_t}$$

where Z_t is a normalization factor chosen so that D_{t+1} will be a distribution.

end for

Final score:
$f(x) = \sum_{t=0}^{T} \alpha_t h_t(x)$ and $H(x) = sign(f(x))$

Рисунок 2.18 – Псевдокод алгоритму бустингу

XGBoost розшифровується як Extreme Gradient Boosting. Це розпаралелена і ретельно оптимізована версія алгоритму градієнтного бустингу. Розпаралелювання всього процесу бустингу значно збільшує час навчання.

Замість того щоб навчати найкращу з можливих моделей на даних (як у традиційних методах), ми навчаємо тисячі моделей на різних підмножинах навчального набору даних і потім голосуємо за модель, що показала найкращі результати.

У багатьох випадках XGBoost виявляється кращим, ніж звичайні алгоритми градієнтного посилення. Реалізація мовою Python дає доступ до

величезної кількості внутрішніх параметрів, які можна налаштовувати для підвищення точності та достовірності.

Важливими особливостями XGBoost є:

– розпаралелювання – модель реалізується для навчання на кількох ядрах процесора;

– регуляризація – XGBoost включає різні штрафи регуляризації для запобігання перебору. Штрафна регуляризація забезпечує успішне навчання, завдяки чому модель може адекватно узагальнювати;

– нелінійність XGBoost може виявляти і навчатися на основі нелінійних моделей даних;

– перехресна валідація вбудована і постачається з коробки;

– масштабованість XGBoost може працювати розподілено завдяки розподіленим серверам і кластерам, таким як Hadoop і Spark, що дає змогу обробляти величезні обсяги даних. Крім того, він доступний для багатьох мов програмування, таких як C++, JAVA, Python.

2.4 Оцінювання якості отриманого результату

Вибір правильних метрик оцінювання якості моделі – надзвичайно важливий крок, коли йдеться про класифікатор, що працює з незбалансованими даними, адже точність або ROC AUC, що часто використовуються для оцінювання класифікаторів, дадуть викривлене уявлення про результат, а точніше дадуть надто позитивну його оцінку. Помилкове розуміння метрик призведе до некоректно натренованої моделі.

Для високо незбалансованих даних використовують такі метрики:

– точність (precision);

– повнота (recall);

– F1-міра (F1-score);

– ВП крива (PR curve, precision recall curve).

Приймемо, що клас 0 -більшість, а клас 1 -меншість.

Точність намагається відповісти на запитання «який відсоток відмічених на тесті прикладів було відмічено правильно». Модель, яка під час передбачення не зробила жодної помилки першого роду, матиме точність 1.0.

Повнота намагається відповісти на запитання «який відсоток усіх тестових прикладів класу 1 було відзначено правильно». Модель, яка під час передбачення не зробила жодної помилки другого роду, матиме повноту 1.0.

$$Precision = \frac{TP}{TP + FP}, \tag{2.7}$$

$$Recall = \frac{TP}{TP + FN}, \tag{2.8}$$

де TP (true positives) – кількість правильно спрогнозованих прикладів класу 1;

FP (false positives) – кількість прикладів класу 0, які спрогнозовані як клас 1;

FN (false negatives) – кількість прикладів класу 1, які спрогнозовані як клас 0.

Щоб оцінити модель, необхідно проаналізувати і точність, і повноту. Модель із високою повнотою, але низькою точністю поверне багато результатів класу 1, але більшість із них будуть не вірні. Модель із високою точністю, але низькою повнотою поверне мало результатів класу 1, хоча більшість із них класифіковані правильно.

Усереднену оцінку між точністю і повнотою можна отримати за допомогою міри F1:

$$F1 = \frac{2 * Presicion * Recall}{Precision + Recall}. \tag{2.9}$$

ВП крива являє собою графічне зображення компромісу між точністю і повнотою для певної моделі на рисунку 2.19.

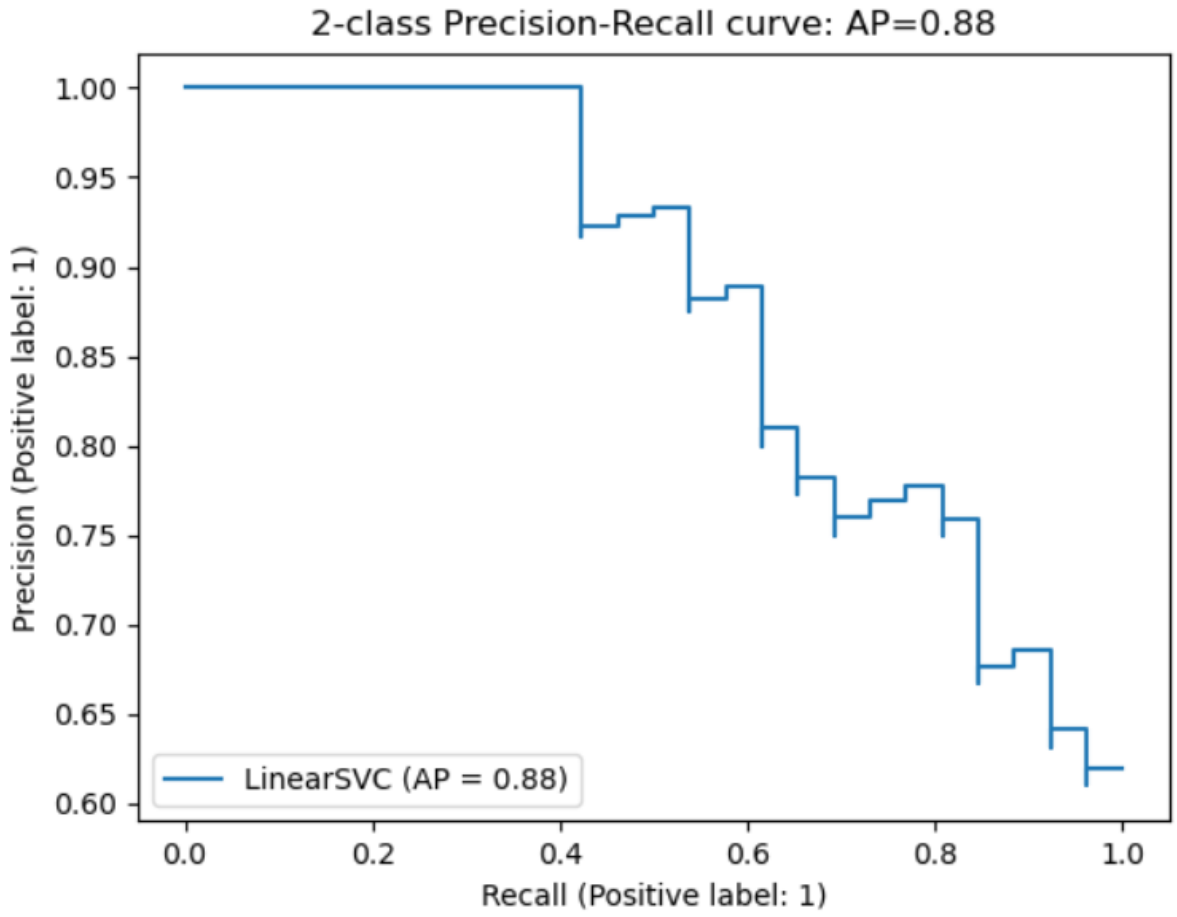

Рисунок 2.19 – ВП крива

Щоб досягти балансу між точністю і повнотою в задачах з незбалансованими даними, необхідно розуміти бізнес завдання. Що важливіше – ідентифікувати більше прикладів меншого класу або менше випадків, але при цьому і менше помилятися. Для завдання з боротьби з шахрайством вища повнота важливіша за вищу точність. Щоб зменшити втрати, мережа змінює свої зміщення та ваги. Алгоритм зворотного поширення робить це, обчислюючи градієнти втрат для кожного параметра мережі. За допомогою методів оптимізації, таких як градієнтний спуск, ці градієнти використовуються для оновлення ваг та зміщень.

3 АНАЛІЗ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

3.1 Навчальна вибірка даних і постановка задачі

Обраний для побудови моделі датасет містить транзакції, проведені в мережі Ethereum європейськими користувачами в період 2022–2023 року.

Ми сформували збалансований набір даних [4] за таким принципом:

– positive_sample = df[df[«Flag»] == 1]: створюється підвибірка (positive_sample) з вихідного DataFrame (df), яка містить тільки ті рядки, де значення стовпчика «Flag» дорівнює 1;

– negative_sample = df[df[«Flag»] == 0].sample(len(positive_sample), random_state=23): створюється підвибірка (negative_sample) з вихідного DataFrame (df), що містить тільки ті рядки, де значення стовпчика «Flag» дорівнює 0. При цьому використовується метод sample, щоб вибрати випадкову кількість рядків, що дорівнює кількості рядків у positive_sample. Параметр random_state використовується для забезпечення відтворюваності випадкової вибірки;

– data = pd.concat([negative_sample, positive_sample], axis=0): Об'єднання підвибірок negative_sample і positive_sample у новий DataFrame (data). Це відбувається шляхом конкатенації (об'єднання) даних уздовж осі 0 (за рядками);

– y = data[«Flag»]: створюється цільова змінна (y), що містить значення стовпця «Flag» з нового DataFrame data;

– X = data.iloc[:, 2:]: створюється матриця ознак (X), що містить усі стовпці, починаючи з третього (індекс 2), до кінця DataFrame data. Це зроблено для виключення перших двох стовпців, які вважаються індексами та цільовою змінною.

Таким чином, цей код виконує операції з формування збалансованого набору даних для навчання моделі машинного навчання, з огляду на позитивні (Flag == 1) і негативні (Flag == 0) класи.

Кожна транзакція має 34 ознаки, (рисунок 3.1, 3.2) і значення цільової змінної.

Detail Compact Column 34 of 34 columns

# flag	# minTimeBetween...	# maxTimeBetween...	# avgTimeBetween...
0 1	0 62m	0 1.66b	0 829m
1	0.000000	2387389.000000	58076.545455
1	0	0	0
1	37.000000	25112882.000000	1710279.133333
1	0.000000	642460.000000	15761.064220

Рисунок 3.1 – Зовнішній вигляд даних

```
print(df["Flag"].value_counts())

pie, ax = plt.subplots(figsize=[15,10])
labels = ['Non-fraud', 'Fraud']
colors = ['#f9ae35', '#f64e38']
plt.pie(x = df['Flag'].value_counts(), autopct='%.2f%%', explode=[0.02]*2, labels=labels, pctdistance=0.
plt.title('Target distribution')

plt.show()
```

```
0    14627
1     5675
Name: Flag, dtype: int64
```

Рисунок 3.2 – Розподіл транзакцій за кількістю

Ми аналізуватимемо один набір даних окремо, проте ми досліджували інші наявні набори даних і навели їхнє порівняння в таблиці 3.1.

Таблиця 3.1 – Порівняння наборів даних

	Кількість Ознак	Усього випадків	Шахрайські операції	Не шахрайські
Набір даних для виявлення шахрайства в Ethereum	37	9816	2179	7637
Набір даних про шахрайство в Ethereum	34	20302	5675	14627

Метою аналізу даних є виявлення потенційного шахрайства шляхом виявлення аномалій або відхилень від «нормальної» поведінки чи патернів. Для цього експерт встановлює базовий рівень нешахрайської діяльності, який порівнюється з набором підозрілих даних. Для розуміння концепції аналітики виявлення шахрайства необхідно знати визначення термінів «шахрайство» та «виявлення шахрайства». Шахрайство – це злочин або обманні дії, які вчиняє злочинець з метою одержання незаконної вигоди або незаконного доступу до інформації та активів.

Виявлення шахрайства – це процес виявлення такої форми обманних дій. Воно може здійснюватися до початку шахрайства, у процесі шахрайства або після його вчинення.

3.2 Обґрунтування вибору мови та платформи розробки

Для розробки програми, яка вирішує поставлене завдання, було обрано мову Python 3 і середовище Jupyter Notebook.

Мова Python хоча й поступається за швидкістю виконання коду таким популярним мовам, як Java, C++ тощо, але має низку важливих переваг – компактність і простота синтаксису, а отже, швидкість розроблення, підтримка багатьох парадигм програмування, наявність високої кількості бібліотек для машинного навчання та візуалізації даних.

Jupyter Notebook – це середовище розробки, що дає змогу виконувати код покроково, причому порядок кроків може змінюватись, а результати виконання кожного кроку записуються глобально, таким чином середовище є надзвичайно зручним для візуалізацій та машинного навчання, адже можна швидко переробляти моделі чи додавати нові, використовуючи попередній результат або попередньо оброблені дані, приклад на рисунку 3.3.

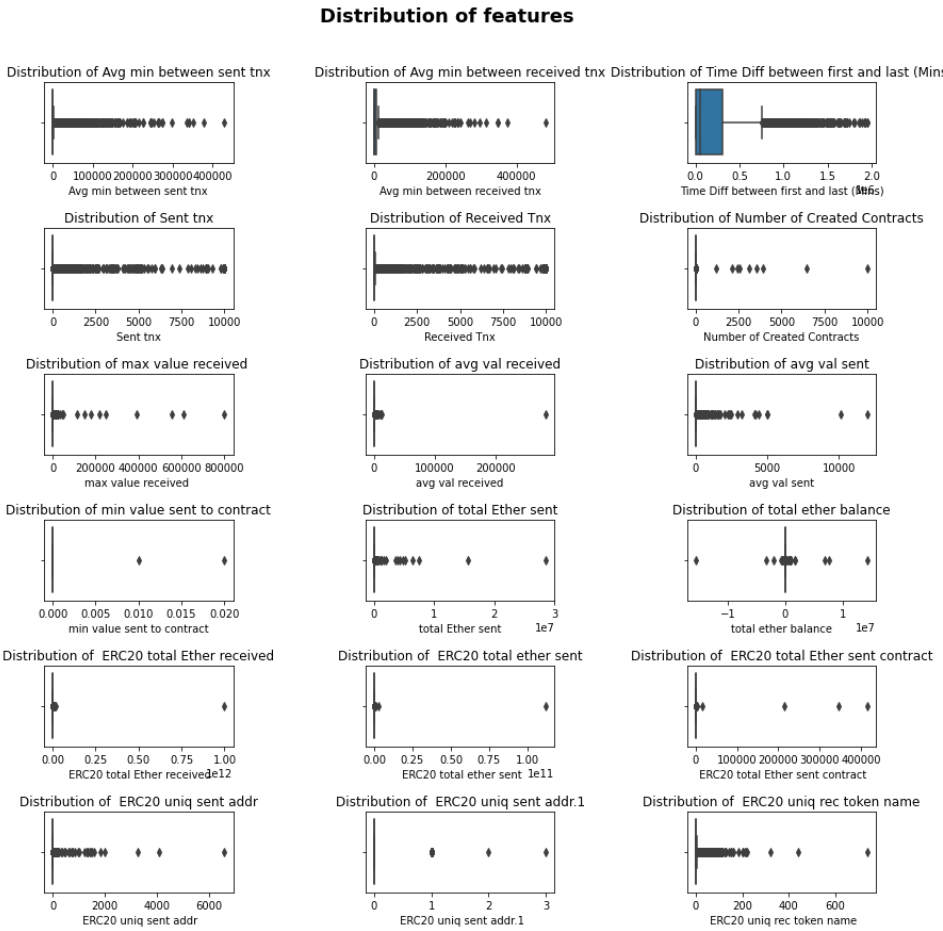

Рисунок 3.3 – Jupyter Notebook і Python 3

Для розробки програмного продукту було використано такі бібліотеки:

– Matplotlib і Seaborn для візуалізації результатів;

– Pandas для зручної роботи з таблицями та обробки csv файлів;

– Numpy для швидкої та ефективної роботи з числовими масивами;

– Imblearn для оверсамплінгу та андерсамплінгу, побудови пайплайну моделі;

– Sklearn для розрахунку метрик, пошуку оптимальних параметрів, побудови моделей;

– Tensorflow/Keras для побудови нейронної мережі. Keras – зручний інтерфейс для Tensorflow (однієї з найдосконаліших бібліотек розроблених Google для машинного навчання);

– Ligthgbm для побудови LightGBM моделі;

– Pycm – модуль для побудови персептрона.

3.3 Обробка V-змінних

З огляду на пропущені значення в датасеті виникає припущення про мультиколінеарність змінних. Загалом у наборі даних налічується понад 10 таких змінних, співвідношення кореляції яких дорівнює 0.8. Дослідимо, які з них сильно корелюють між собою, побудувавши кореляційні матриці та відображення пропущених значень після обробки, (рисунок 3.4, 3.5).

Шахрайство з обліковими записами поділяється на шахрайство з новими обліковими записами та шахрайство з захопленням облікових записів. При шахрайстві з новими акаунтами нові акаунти створюються з використанням фальшивих ідентифікаційних даних. Такі шахрайства можна виявити, використовуючи шаблони різних пристроїв та індикатори сеансів для виявлення фальшивих ідентифікаторів.

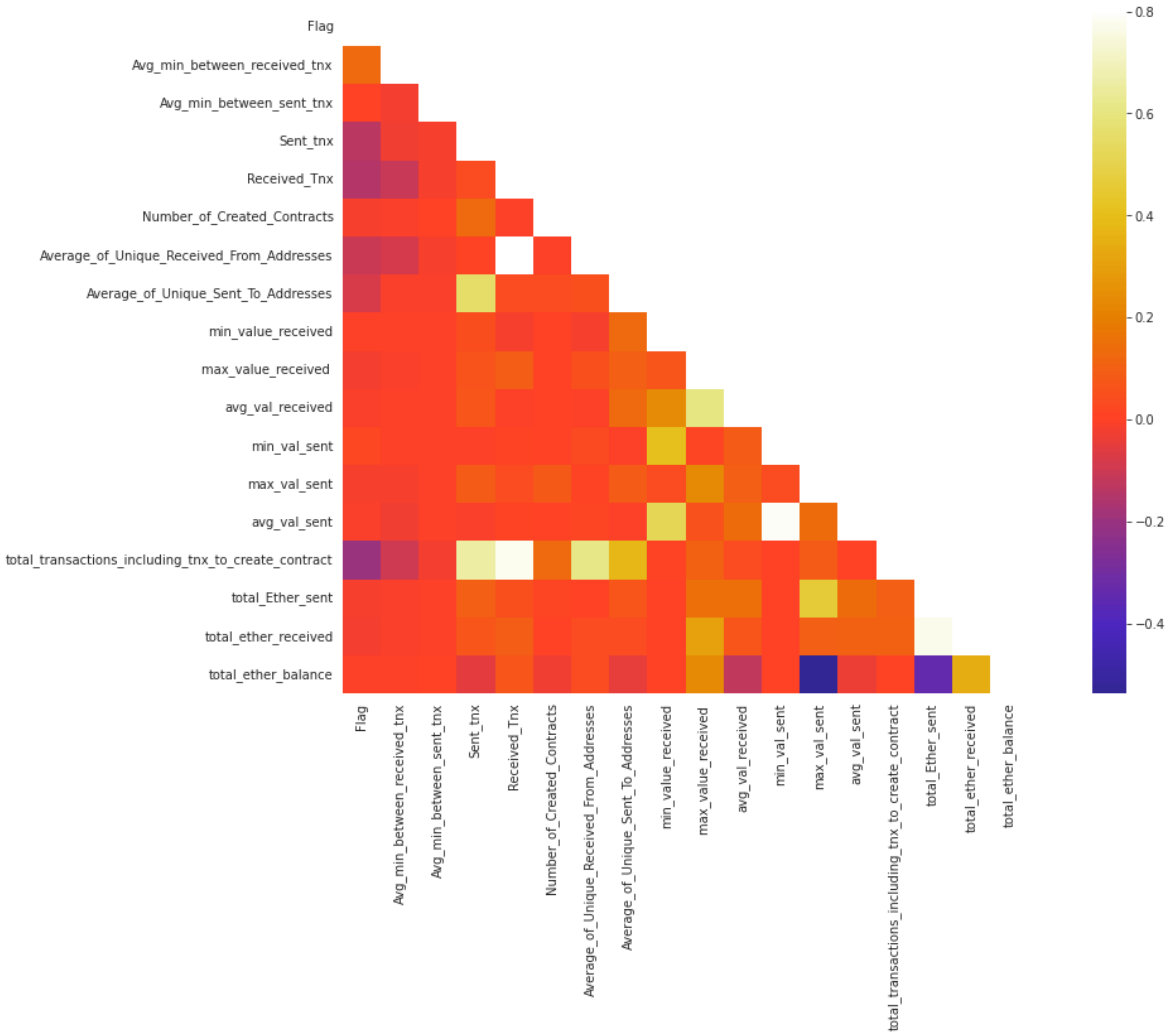

Рисунок 3.4 – Кореляційна матриця

Як бачимо, серед цих змінних значною мірою присутня мультиколінеарність. Виокремивши ці групи за патернами пропущених значень (за їхньою однаковою кількістю) і розглянувши їхні кореляційні матриці, удалося скоротити кількість змінних до 14, решта значень із високим значенням кореляції, такі як:

– max_val_sent;
– total_ether_received;
– min_value_received;
– min_value_sent.

Їхню кореляційну матрицю представлено нижче.

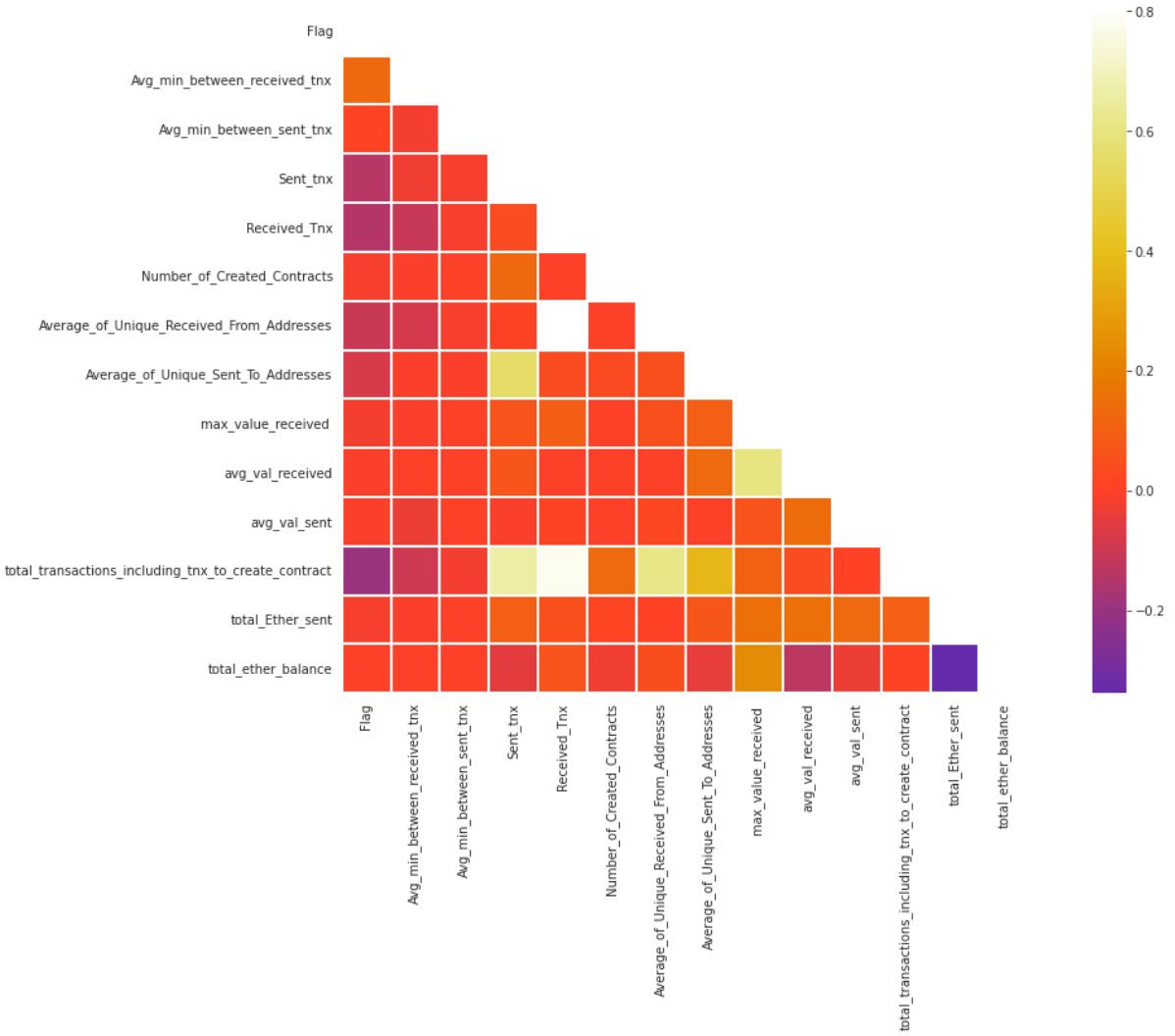

Рисунок 3.5 – Виділення оновленої матриці кореляції

3.4 Проектування програмної реалізації

Для побудови системи із запобігання шахрайству, як зазначено в розділі 2, було обрано кілька відомих архітектур моделей машинного навчання: логістична регресія, RandomForestClassifier, XGB classifier, нейронна мережа MLPC, тож ці архітектури були взяті за основу досліджень. Також проведено експерименти з різними методами оверсемплінгу та андерсемплінгу, які відіграли важливу роль у збільшенні заданих метрик моделей – точність, повнота, F1-міри.

З моделлю логістичної регресії на необроблених даних отримано погані показники, зображені на рисунку 3.6, та матриця на рисунку 3.7.

```
print(classification_report(y_test, preds))
print(confusion_matrix(y_test, preds))
plot_confusion_matrix(LR, norm_test_f, y_test)
```

```
              precision    recall  f1-score   support

           0       0.96      0.89      0.92      1547
           1       0.68      0.88      0.77       422

    accuracy                           0.89      1969
   macro avg       0.82      0.88      0.85      1969
weighted avg       0.90      0.89      0.89      1969

[[1373  174]
 [  51  371]]
```

Рисунок 3.6 – Результати тренування логістичної регресії на необроблених даних

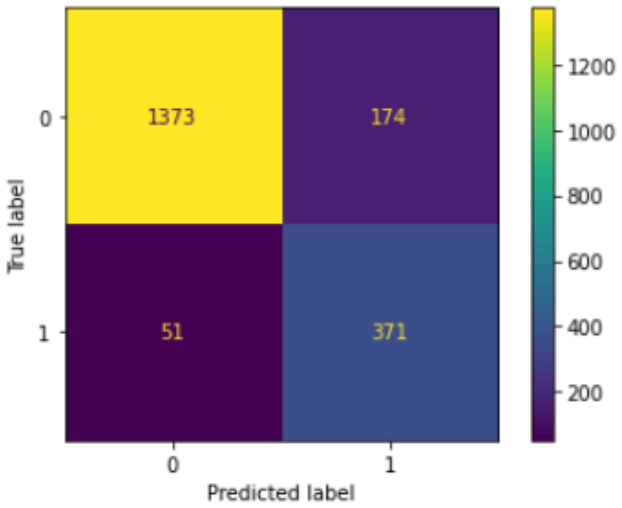

Рисунок 3.7 – Матриця змішування

Наочно видно, що незважаючи на високу точність метрики accuracy = 89%, ця модель непридатна для використання, адже точність (precision) становить лише 0.68, адже в контексті логістичної регресії під точністю розуміють частку правильно передбачених позитивних випадків з-поміж усіх випадків, які були передбачені як позитивні. Вона дає уявлення про здатність моделі правильно ідентифікувати позитивні випадки. Також одразу видно, що використання логістичної регресії, найімовірніше, не принесе найкращих результатів, у разі використання параметра class_weight = «balanced» (врахування нерівномірного розподілу класів під час знаходження ваг для моделі) є надто низьким, навіть з огляду на той факт, що дані не перероблені.

Для поліпшення результатів тренування побудовано систему з підбору оптимальних параметрів, що ґрунтується на GridSearchCV і комбінації андерсамплінгу та оверсамплінгу SMOTE + RandomUndersampling, схема якої міститься на рисунку 3.8.

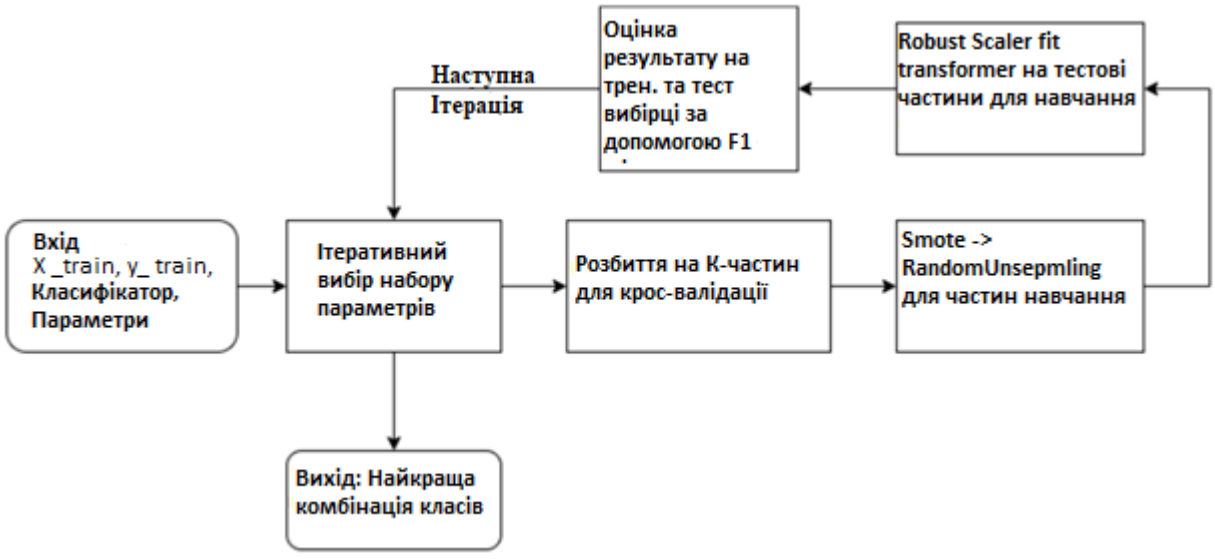

Рисунок 3.8 – Система підбору параметрів для моделі

На рисунку 3.9 зображено процес підбору параметрів для логістичної регресії.

```
oversample = SMOTE()
print(f'Shape of the training before SMOTE: {norm_train_f.shape, y_train.sha
pe}')

x_tr_resample, y_tr_resample = oversample.fit_resample(norm_train_f, y_trai
n)
print(f'Shape of the training after SMOTE: {x_tr_resample.shape, y_tr_resamp
le.shape}')
```

```
Shape of the training before SMOTE: ((7872, 16), (7872,))
Shape of the training after SMOTE: ((12230, 16), (12230,))
```

Рисунок 3.9 – Підбір параметрів на прикладі логістичної регресії

Як і показала попередня модель, очевидно, що логістична регресія, на жаль, не підходить як єдиний класифікатор для поставленого завдання. Хоч трохи адекватний результат можна отримати тільки змістивши ймовірність, необхідну для прийняття рішення, з 0.5 до 0.99 (рисунок 3.10).

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f50
12264110>
```

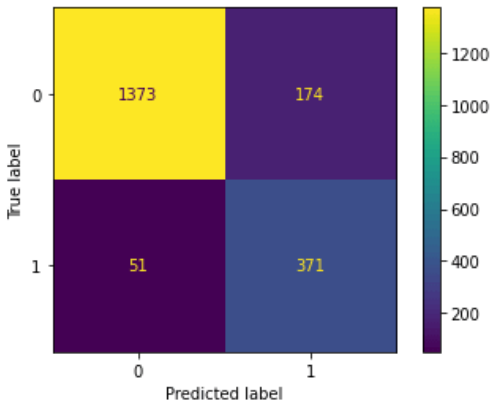

Рисунок 3.10 – Точність і повнота для логістичної регресії за необхідної ймовірності для прийняття рішення 0.99

З урахуванням матриці змішування:

– LR-модель правильно визначила 373 (TP) випадки FRAUD із 422 (P);

– LR-модель відмітила як шахрайство 171 випадок (FP) із 1547, коли ці випадки насправді були НЕ шахрайством.

Під час роботи зі сценарієм виявлення шахрайства нас більше хвилюють транзакції, які насправді були шахрайськими, але були розцінені нашою моделлю як НЕ шахрайські (FN – 49).

Таким чином, спробуємо збільшити відгук за допомогою інших методів. Перейдемо до RandomForestClassifier (рисунок 3.11, 3.12).

```
RF = RandomForestClassifier(random_state=42)
RF.fit(x_tr_resample, y_tr_resample)
preds_RF = RF.predict(norm_test_f)

print(classification_report(y_test, preds_RF))
print(confusion_matrix(y_test, preds_RF))
plot_confusion_matrix(RF, norm_test_f, y_test)
```

```
              precision    recall  f1-score   support

           0       0.99      0.98      0.98      1547
           1       0.93      0.95      0.94       422

    accuracy                           0.98      1969
   macro avg       0.96      0.97      0.96      1969
weighted avg       0.98      0.98      0.98      1969

[[1518   29]
 [  20  402]]
```

Рисунок 3.11 – Параметри моделі RFC

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f5003a57650>
```

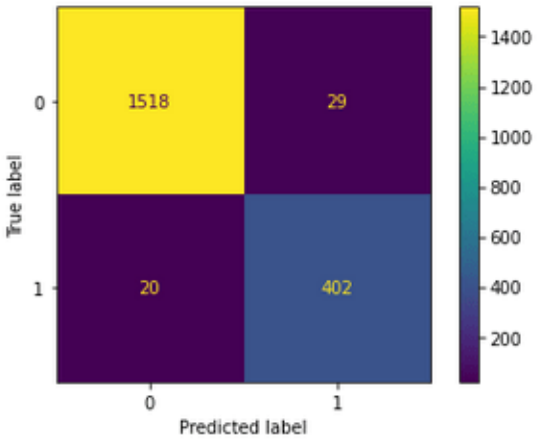

Рисунок 3.12 – Візуалізація класифікації RFC

Класифікатор RF, мабуть, дає ефективніші результати:

– значно знижуються показники FP і FN, збільшуються показники recall і precision;

– при використанні RF модель не виявляє 20 випадків FRAUD. Однак тепер показник precision зріс до 0.93.

Подивимося, чи зможемо ми збільшити ці результати. Перейдемо до XGB класифікатора, (рисунок 3.13).

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f5011adb9d0>
```

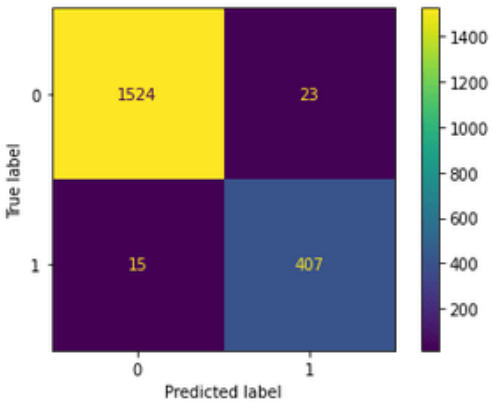

Рисунок 3.13 – Точність, для XGBClassifier

Результати XGBClassifier показують, що він працює трохи краще, ніж RF, коли йдеться про НЕ шахрайські транзакції, позначаючи 22 випадки як шахрайські, коли вони насправді такими не були.

Що стосується виявлення шахрайства, то XGBClassifier пропустив 16 транзакцій із 422, показавши найкращий результат за відгуками.

З огляду на це, XGBClassifier є тим вибором, який нам потрібен.

3.4.1 Проектування MLPC моделі

Багатошаровий персептрон (MLP) – це різновид штучних нейронних мереж, які використовуються в глибокому навчанні для вирішення завдань класифікації та регресії. Основна ідея полягає в тому, щоб мати кілька шарів нейронів (включно з вхідним і вихідним) і здатність навчатися складних функцій завдяки прихованим шарам.

MLP складається з трьох типів шарів:

– вхідний шар – нейрони в цьому шарі представляють вхідні ознаки або характеристики;

– приховані шари – містять нейрони, які використовуються для внутрішнього представлення і виявлення складних залежностей у даних. Кількість і розмір цих шарів може варіюватися;

– вихідний шар генерує вихід, який може бути відповіддю на завдання класифікації або прогнозом для завдання регресії.

Переваги MLP:

– універсальність MLP є універсальним апроксиматором, що означає, що вони можуть наближати будь-яку функцію з достатньою кількістю нейронів і шарів;

– здатність навчатися складних залежностей: Завдяки багатошаровій структурі MLP може виявляти і моделювати складні нелінійні залежності в даних;

– застосування в глибокому навчанні MLP є базовим будівельним блоком для багатьох глибоких нейронних мереж, де можна використовувати складніші архітектури для вирішення високорівневих завдань.

Ефективність MLP залежить від розмірів та архітектури мережі, які обираються для конкретного завдання. Велика кількість нейронів і шарів може призвести до перенавчання, тоді як мала кількість може не забезпечити достатньої гнучкості для розв'язання складних завдань.

У разі ефективності для виявлення шахрайських транзакцій у мережі Ethereum, важливо експериментувати з параметрами, такими як кількість нейронів у прихованих шарах, швидкість навчання і функції активації, щоб досягти оптимальних результатів.

MLP є потужним інструментом для вирішення різних завдань, включно з виявленням шахрайських транзакцій. Важливо налаштувати параметри та архітектуру з урахуванням конкретних вимог завдання для досягнення оптимальної ефективності.

Ця система розроблена з метою ефективного виявлення та класифікації шахрайських транзакцій у мережі криптовалюти Ethereum. Вона використовує нейронну мережу з архітектурою MLPClassifier для навчання і передбачення шахрайських транзакцій на основі набору факторів.

Підготовка даних:

– дані для тренування і тестування піддаються попередній обробці та масштабуванню за допомогою StandardScaler;

– використання методу KFold для поділу даних на навчальні та тестові набори в 10 ітерацій для оцінки результатів на різних підвибірках.

Для навчання моделі використовувався MLPClassifier з параметрами hidden_layer_sizes = (100, 100) і solver = «adam» для створення і тренування нейронної мережі.

Оцінка результатів:

– використання матриці невідповідностей (Confusion Matrix) для кожного тестового набору для отримання метрик, таких як точність, F1-міра, Kappa, Precision і Recall;

– визначення середніх значень метрик для оцінки загальної ефективності системи.

Виведення результатів:

– виведення результатів для кожної з 10 ітерацій, включно із середніми значеннями метрик;

– візуалізація матриці невідповідностей для найгірших і найкращих результатів.

Система на базі MLPClassifier показала високу точність і ефективність у виявленні шахрайських транзакцій у мережі Ethereum.

Середні значення метрик свідчать про хорошу відтворюваність і стабільність результатів на різних тестових наборах.

Визначення найгірших і найкращих результатів дає змогу визначити сценарії, у яких система може показати меншу ефективність або вищу надійність, (рисунок 3.15, 3.16).

Усі ці етапи є важливими для розуміння та оптимізації роботи системи в реальних умовах і вдосконалення її ефективності з плином часу, реалізація створення та налаштування моделі (рисунок 3.14).

Щоб зменшити втрати, мережа змінює свої зміщення та ваги. Алгоритм зворотного поширення робить це, обчислюючи градієнти втрат для кожного параметра мережі. За допомогою методів оптимізації, таких як градієнтний спуск, ці градієнти використовуються для оновлення ваг та зміщень.

```
        METRIC_LIST = ["Точність", "F1", "Каппа", "Точність", "Відклик"].
        nn_params = {"hidden_layer_sizes": (100, 100), "solver": "adam"}
        nn_model = MLPClassifier(**nn_params)
        nn_model.fit(X, y)
        kf = KFold(n_splits=10, random_state=23, shuffle=True)
        cm_metric_list = []
        nn_cm_worst = None
        nn_cm_best = None
        nn_cm_list = []
        for train_index, test_index in kf.split(X):
            X_train, X_test = X.iloc[train_index], X.iloc[test_index]
            y_train, y_test = y.iloc[train_index], y.iloc[test_index]
            scaler = StandardScaler().fit(X_train)
            X_train = scaler.transform(X_train)
            X_test = scaler.transform(X_test)
            nn_model.fit(X_train, y_train)
            cm = ConfusionMatrix(actual_vector = y_test.values, predict_vector =
nn_model.predict(X_test))
            cm_metric_list.append({"Точність":cm.Overall_ACC, "F1": cm.F1[1], "Каппа": cm.Kappa,
"Точність":cm.PPV[1]
                                   , "Recall": cm.TPR[1]})
            cm.relabel({1: "Шахрайство", 0: "Нешахрайство"})
            nn_cm_list.append(cm)
            якщо nn_cm_worst is None:
                nn_cm_worst = cm
            іще:
                if cm.Overall_ACC < nn_cm_worst.Overall_ACC:
                    nn_cm_worst = cm

            якщо nn_cm_best is None:
                nn_cm_best = cm
            іще:
                if cm.Overall_ACC > nn_cm_best.Overall_ACC:
                    nn_cm_best = cm

        print("Назва моделі: Нейромережевий класифікатор")
        print("10-Fold Metrics: \n")
        для метрики в METRIC_LIST:
            temp = []
            for item in cm_metric_list:
                temp.append(item[metric])
            print("{0} : {1}\n".format(metric, np.mean(temp).round(2))))
        nn_cm_worst.plot(title="Нейронна мережа найгірша", number_label=True)
        nn_cm_best.plot(title="Нейронна мережа найкраща", number_label=True)
        plt.show()
```

Рисунок 3.14 – Опис моделі MLPC з набором параметрів

```
Model Name: Neural Network Classifier
10-Fold Metrics:

Accuracy : 0.82

F1 : 0.82

Kappa : 0.65

Precision : 0.82

Recall : 0.85
```

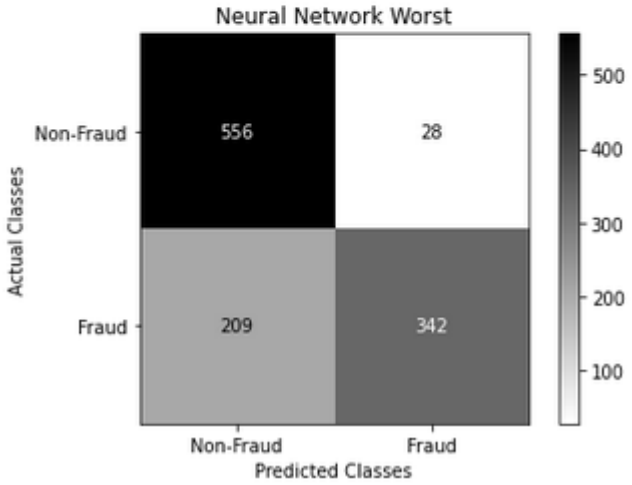

Рисунок 3.15 – Аналіз результату в найгіршому випадку класифікації

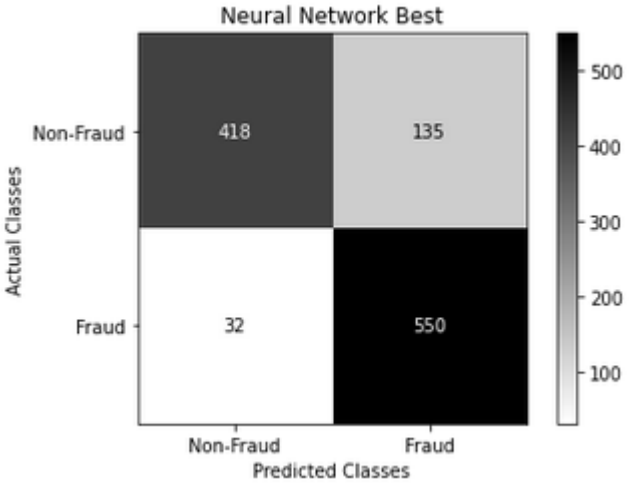

Рисунок 3.16 – Аналіз результату в гіршій кращій класифікації

corr = df.corr(): код обчислює матрицю кореляції для всіх числових змінних у вашому наборі даних df. Кореляція вимірює ступінь лінійної залежності між двома змінними. Значення кореляції може перебувати в діапазоні від -1 до 1, де -1 вказує на повну зворотну лінійну залежність, 1 – на повну лінійну залежність, а 0 – на відсутність кореляції.

Створення маски для верхнього трикутника:

– mask = np.zeros_like(corr): створюємо масив нулів такого ж розміру, як і матриця кореляції;

– mask[np.triu_indices_from(mask)]=True: встановлюємо всі елементи, вищі за головну діагональ матриці кореляції, у значення True. Це потрібно для того, щоб відобразити тільки верхній трикутник матриці кореляції, оскільки матриця кореляції симетрична щодо головної діагоналі, і нам нецікаві дубльовані значення.

Створення теплокарти (heatmap) кореляції:

– with sns.axes_style('white'): задаємо стиль для графіка (біле фонове полотно);

– fig, ax = plt.subplots(figsize=(18,10)): створюємо об'єкт фігури й осі для графіка теплокарти;

– sns.heatmap(corr, mask=mask, annot=False, cmap='CMRmap', center=0, linewidths=0.1, square=True): застосовуємо теплокарту для відображення матриці кореляції.

Параметри включають:

– mask=mask: використовує створену нами маску, щоб приховати нижній трикутник;

– annot = False вимикає відображення значень у комірках;

– cmap = «CMRmap» встановлює колірну карту (карту кольорів);

– center = 0: вказує середину колірної шкали (для відображення негативних і позитивних кореляцій);

– linewidths = 0.1, товщина ліній, що розділяють комірки;

– square = True вказує, що графік має бути квадратним.

Візуалізація матриці кореляції дає змогу швидко оцінити взаємозв'язок між числовими змінними у вашому датасеті. Колірна шкала вказує на силу і напрямок кореляції. Це може допомогти вам виявити взаємні залежності та визначити, які змінні можуть бути важливими під час розроблення моделі.

Цей код (рисунок 3.17) ілюструє використання бібліотек numpy і seaborn для створення і візуалізації матриці кореляції у вигляді теплової карти (рисунок 3.18).

```
corr = df.corr()
mask = np.zeros_like(corr)
mask[np.triu_indices_from(mask)]=True
with sns.axes_style('white'):
    fig, ax = plt.subplots(figsize=(18,10))
    sns.heatmap(corr, mask=mask, annot=False, cmap='CMRmap', center=0, linewidths=0.1,
square=True)
```

Рисунок 3.17 – Створення матриці кореляції

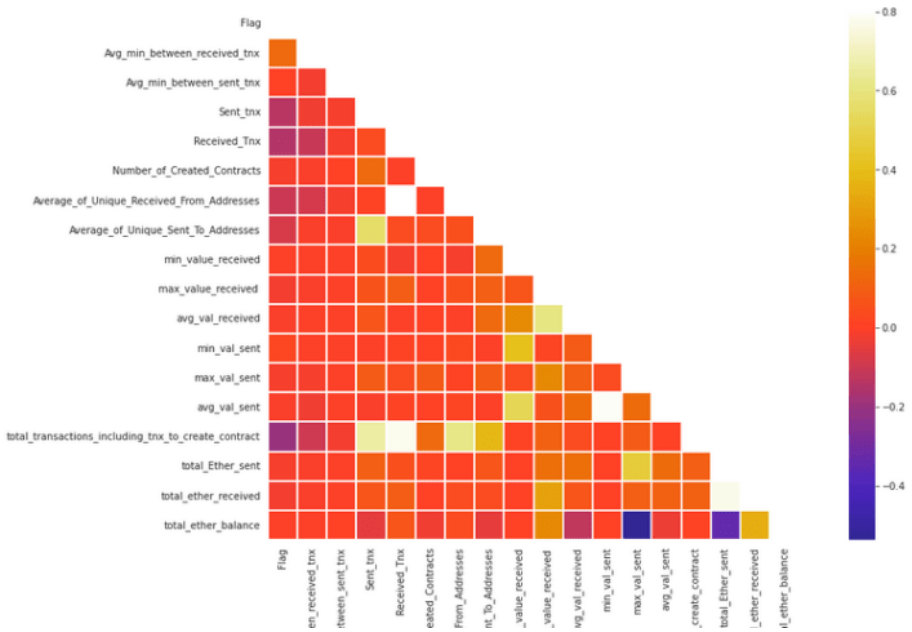

Рисунок 3.18 – Теплова карта за набором даних [4]

Опис процесу Оверсемплінгу та відображення результатів:

– oversample = SMOTE(): використовується метод оверсемплінгу SMOTE (Synthetic Minority Over-sampling Technique), який створює синтетичні екземпляри меншої кількості класу для балансування кількості прикладів в обох класах;

– x_tr_resample, y_tr_resample = oversample.fit_resample(norm_train_f, y_train): застосовує SMOTE до тренувального набору даних. Нові синтетичні приклади додаються до менш представленого класу, щоб збалансувати класи.

Порівняння розподілу класів до і після оверсемплінгу:

– non_fraud і fraud обчислюють кількість безвідмовних і шахрайських транзакцій у вихідному тренувальному наборі перед оверсемплінгом;

– no і yes визначають кількість безвідмовних і шахрайських транзакцій у тренувальному наборі після оверсемплінгу.

Оверсемплінг застосовується в ситуаціях, коли кількість прикладів менш представленого класу недостатня для навчання ефективної моделі, що може призвести до перенавчання.

Очікується, що після оверсемплінгу розподіл класів буде збалансованим, що поліпшить ефективність моделі під час навчання на виправленому тренувальному наборі. У даному випадку, ми очікуємо підвищення кількості шахрайських транзакцій у тренувальному наборі, реалізація на рисунку 3.19.

```
BEFORE OVERSAMPLING
        Non-frauds: 11670
        Fauds: 4571
AFTER OVERSAMPLING
        Non-frauds: 11670
        rFauds: 11671
```

Рисунок 3.19 – Отримане значення після примирення Oversample

Очікується, що цей код на рисунку 3.20, навчатиме MLPC модель на нормалізованих даних, використовуючи K-Fold Cross-Validation, і зберігатиме метрики якості моделі для кожного фолда. Також, виведе ConfusionMatrix для найгіршого і найкращого результату за всі фолди. Це дасть змогу оцінити ефективність моделі на різних частинах навчального набору (рисунок 3.21, 3.22).

```
nn_params = {"hidden_layer_sizes": (100, 100), "solver": "adam"}
# x_tr_resample, y_tr_resample
nn_model = MLPClassifier(nn_params)
nn_model.fit(X, y)
kf = KFold(n_splits=10, random_state=23, shuffle=True)
cm_metric_list = []
nn_cm_worst = None
nn_cm_best = None
nn_cm_list = []
for train_index, test_index in kf.split(X):
    # X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    # y_train, y_test = y.iloc[train_index], y.iloc[test_index]
    # scaler = StandardScaler().fit(X_train)
    # X_train = scaler.transform(X_train)
    # X_test = scaler.transform(X_test)
    t_df = norm.fit_transform(X_test)
    nn_model.fit(norm_df, y_train)
    cm = ConfusionMatrix(actual_vector = y_test.values, predict_vector = nn_model.predict(t_df))
    cm_metric_list.append({"Точність":cm.Overall_ACC, "F1": cm.F1[1], "Каппа": cm.Kappa,
"Точність":cm.PPV[1]
                           , "Recall": cm.TPR[1]})
    cm.relabel({1: "Шахрайство", 0: "Нешахрайство"})
    nn_cm_list.append(cm)
    якщо nn_cm_worst is None:
        nn_cm_worst = cm
    іще:
        if cm.Overall_ACC < nn_cm_worst.Overall_ACC:
            nn_cm_worst = cm

    якщо nn_cm_best is None:
        nn_cm_best = cm
    іще:
        if cm.Overall_ACC > nn_cm_best.Overall_ACC:
            nn_cm_best = cm
```

Рисунок 3.20 – Реалізація моделі MLPC і запуск візуалізації результатів класифікації

```
Model Name: Neural Network Classifier
10-Fold Metrics:

Accuracy : 0.9

F1 : 0.81

Kappa : 0.75

Precision : 0.85

Recall : 0.77
```

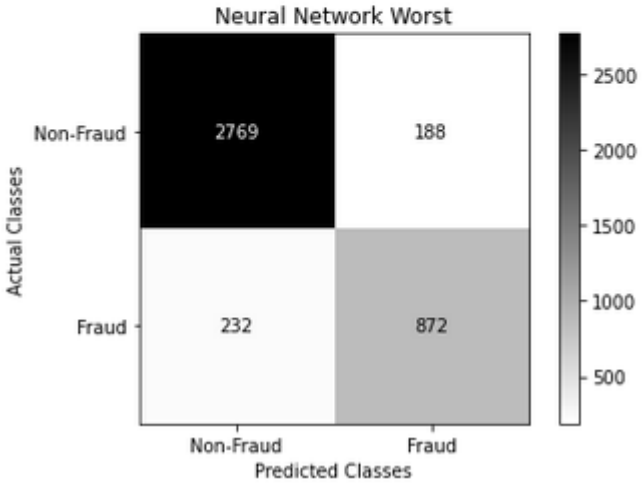

Рисунок 3.21 – Виведення результатів моделі MLPC у найгіршому випадку класифікації

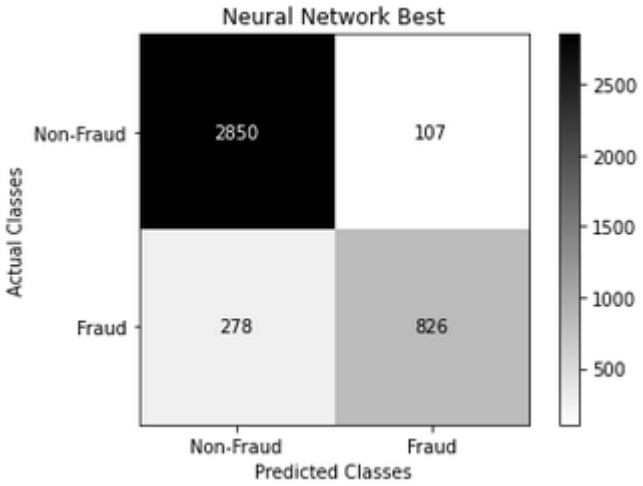

Рисунок 3.22 – Виведення результатів моделі MLPC у найгіршому випадку класифікації

3.5 Використання результатів моделювання

Провівши первинну підготовку даних і проаналізувавши роботу навчених на них моделей, можна імплементувати інформаційну технологію для виявлення шахрайських операцій.

Щоб зберегти модель після її навчання з використанням бібліотеки scikit-learn (як у вашому випадку з MLPClassifier), ми скористалися модулем joblib. Звернемо увагу, що joblib забезпечує більш ефективне збереження і завантаження об'єктів Python порівняно з pickle для великих масивів даних, що може бути важливим під час збереження великих моделей або великих обсягів даних, унаслідок чого можна звернутися до збереженої моделі та викликати її, передавши на вхід дані, результат у консольному виводі на рисунку 3.23.

```
# Выводим результаты предсказаний и точности
print("Predictions for test data:")
print(predictions)
print("Accuracy: {:.2f}%".format(accuracy * 100))
```

```
C:\Users\NASA\anaconda3\lib\site-packages\pandas\c
or newer of 'numexpr' (version '2.7.1' currently i
  from pandas.core.computation.check import NUMEXP
Predictions for test data:
[0 0 0 ... 1 1 1]
Accuracy: 83.96%
```

```
[ ]:
```

Рисунок 3.23 – Консольний вивід точності під час виклику моделі

Для додавання візуального інтерфейсу використовувалася бібліотека Tkinter, яка входить до стандартної бібліотеки Python і дає змогу створювати прості графічні інтерфейси. У цьому прикладі користувач може вибрати файл із даними для навчання моделі. Після вибору файлу і натискання кнопки, модель навчається на обраних даних і зберігається. Є можливість

додати функціональність для завантаження нових даних і виконання передбачень у відповідних місцях (рисунок 3.24).

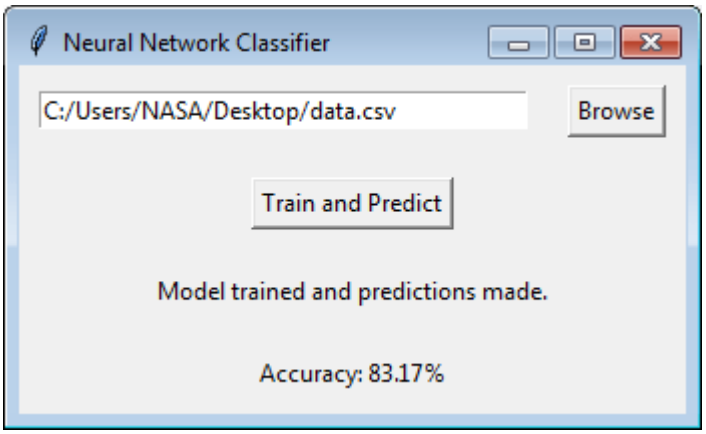

Рисунок 3.24 – Результат роботи системи у віконному режимі

Вибір найкращого алгоритму в контексті визначення шахрайських транзакцій у мережі Ethereum є важливим етапом, що вимагає уважного зважування переваг і недоліків кожного методу. На основі проведеного аналізу, ми доходимо висновку, що Багатошаровий персептрон (MLP) є найкращим вибором, незважаючи на те, що його точність трохи нижча, ніж у Логістичної регресії.

Важливо зазначити, що точність, хоча і є важливим показником, не завжди повністю відображає здатність моделі ефективно виявляти шахрайські транзакції в реальних умовах. Логістична регресія, з її високою точністю в 96%, може бути схильна до перенавчання, що означає, що вона може добре справлятися з відомими даними, але погано узагальнюватися на нові, невідомі дані.

На відміну від цього, MLP, хоча і має точність близько 84%, показує високий потенціал у моделюванні складних нелінійних залежностей у даних. Ця здатність особливо цінна в завданнях, де шахрайські транзакції можуть проявляти складні патерни, важкі для виявлення простими алгоритмами.

ВИСНОВКИ

У результаті виконання роботи було проведено теоретичні та практичні дослідження розділів магістерської дисертації. Дослідивши предметну область, вивчивши математичні методи виявлення шахрайських операцій, було обрано алгоритми навчання моделей у третій – практичній частині роботи. Було проведено аналіз даних щодо транзакцій розподіленої публічної блокчейн-мережі Ethereum.

На основі цих даних методами машинного навчання було побудовано моделі класифікації транзакцій для виявлення шахрайських операцій. Розробка проводилася за допомогою мови програмування Python, з редактором Jupyter як основним середовищем розробки. Ми запропонували метод налаштування гіперпараметрів для поліпшення продуктивності нашої моделі багатошарового персептрона (MLP). У цьому контексті ми використовуємо такі метрики для оцінювання якості моделі: Accuracy, F1, Kappa, Precision і Recall. Ці метрики входять до списку METRIC_LIST, який є основою для оцінювання ефективності алгоритму.

На основі виконаного моделювання було запропоновано інформаційну технологію для виявлення шахрайських транзакцій.

Дана інформаційна технологія може бути використана для імплементації системи підтримки прийняття рішень у криптобіржах і системах забезпечення онлайн платежів. Також у рамках роботи було запропоновано концепцію готового програмного забезпечення, що реалізує систему виявлення шахрайських операцій для подальшої монетизації отриманих напрацювань. У рамках подальших досліджень можна працювати над підвищенням точності класифікації створених моделей, а також залучення інших алгоритмів машинного навчання до розв'язання поставленої задачі – наприклад, нейронних мереж і створення варіантів ансамблів попередньо навчених алгоритмів.

У роботі було визначено актуальність і необхідність дослідження способів створення системи з виявлення та запобігання шахрайству в мережі Ethereum, було проаналізовано труднощі роботи з даними, які містять шахрайські та звичайні транзакції.

Зокрема, це висока незбалансованість даних і низька кількість прикладів шахрайства, що значно ускладнює побудову моделі машинного навчання.

Досліджено методи боротьби із зазначеними проблемами та різні архітектури моделей, які підходять для створення системи з виявлення шахрайства, визначено методи оцінювання для складених моделей машинного навчання. У практичній частині на прикладі обраного датасету побудовано такі моделі:

- логістична регресія;
- класифікатор Random Forest Classifier;
- нейронна мережа MLPC;
- класифікатор XGBoost.

Виявлення та запобігання шахрайству має бути головним пріоритетом для будь-якого бізнесу. Добре розроблена та впроваджена система виявлення шахрайства може значно зменшити ймовірність виникнення шахрайства в організації. Крім того, своєчасне виявлення шахрайства безпосередньо впливає на бізнес у позитивний бік, зменшуючи майбутні потенційні збитки.

Ефективні методи виявлення, такі як штучний інтелект та аналіз статистичних даних, слугують стримуючим фактором для потенційних шахраїв. Оскільки регуляторні вимоги та вимоги до комплаєнсу зростають, надзвичайно важливо впровадити надійну програму виявлення та запобігання шахрайству.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1) Pascari V., Gagarina L. G., Sliusar V. V. Methods and Software Complex Development for Remote Electronic Voting Based on Ethereum Blockchain Platform. *Proceedings of Universities. Electronics*. 2021. Vol. 26, no. 6. P. 565–579.

2) Ковтун В., Овсієнко О. КРИПТОВАЛЮТНІ ВІДНОСИНИ В УКРАЇНІ: ЕКОНОМІКО-ПРАВОВИЙ АНАЛІЗ. *Економіка та суспільство*. 2021. № 31.

3) Волосович С. В. Віртуальна валюта: глобалізаційні виклики і перспективи розвитку. *Економіка України*. 2016. № 4 (653). С. 68–78.

4) Ethereum Fraud Detection Dataset. *Kaggle: Your Machine Learning and Data Science Community*. URL: https://www.kaggle.com/datasets/gescobero/ethereum-fraud-dataset (Дата звернення: 13.04.2024).

5) Berg A. The identity, fungibility and anonymity of money //Economic Papers: A journal of applied economics and policy. 2020. T. 39. No. 2. S. 104-117.

6) Michalski R., Dziubałtowska D., Macek P. Revealing the character of nodes in a blockchain with supervised learning //Ieee Access. 2020. T. 8. S. 109639-109647.

7) Ковальчук О. В. Відмінності адміністративно-правового регулювання обігу електронних коштів та криптовалюти. *LEGAL SCIENCES: RESEARCH AND EUROPEAN INNOVATIONS*. 2021.

8) Phua C., Lee V., Smith K., Gayler R. A Comprehensive Survey of Data Miningbased Fraud Detection Research, URL: https://www.researchgate.net/publication/302557906_A_Comprehensive_Survey_of_Data_Mining-based_Fraud_Detection_Research (Дата звернення: 01.05.2024).

9) Suman, Hisar M. Survey Paper on Credit Card Fraud Detection, International Journal of Advanced Research in Computer Engineering and Technology. 2014. Vol. 3, No. 3.

10) Wen-Fang Y., Wang N. Research on Credit Card Fraud Detection Model Based on Distance Sum, in International Joint Conference on Artificial Intelligence. Hainan, China, 2009.

11) Mohri M., Rostamizadeh A., Talwalkar A. Foundations of Machine Learning. Second edition. Cambridge, MA. MIT Press, 2018. 488p.

12) Ямковий К. С., Любчик Л. М. Експертно-статистичне оцінювання інтегральних індикаторів методами машинного навчання : thesis. 2019.

13) PYTHON FOR DATA PROCESSING AND SIMULATION OF FINANCIAL AND ECONOMIC INDICATORS / T. A. Chupilko et al. *Information technology and computer engineering*. 2021. Vol. 51, no. 2. P. 68–77.

14) Рокач Л., Маймон О. Data mining with decision trees: theory and applications Machine Perception and Artificial Intelligence. Vol. 81, 2014. 305 p.

15) Friedman J.H. Greedy Function Approximation: A Gradient Boosting Machine", The Annals of Statistics. Vol. 29, pp.1189-1232, 2001.

16) Чому XGBoost перемагає на всіх змаганнях по машинному навчанню. URL: https://syncedreview.com/2017/10/22/tree-boosting-with-xgboost-why-doesxgboost-win-every-machine-learning-competition (Дата звернення: 02.05.2024).

17) Ke G., Finley T., Wang T., Chen W., Ma W., Ye Q., Lin T. LightGBM: A Highly Efficient Gradient Boosting Decision Tree, 31st Conference on Neural Information Processing Systems, Long Beach, CA, USA, 2017.

18) Provost F., Fawcett T. The case against accuracy estimation for comparing induction algorithms, In Proc. 15th Intl. Conf. On Machine Learning, pp. 445-453, Madison, WI, USA, 1998.

19) F. Provost., and T. Fawcett, "Robust Classification for Imprecise Environments", Machine Learning vol.42, pp. 203-231, 2001, DOI: 10.1023/A:1007601015854.

20) Nelder J.A., Mead R. A simplex method for function minimization The Computer Journal, Vol. 7, pp. 308-313, 1965.

21) Статистика покупок в США за даними компанії Google. URL: https://trends.google.com/trends/explore?cat=18&date=2017-04-01%202018-1231&geo=US (Дата звернення 24.04.2024).

22) A Survey of Pseudonymous Misuse and Abuses of BPMN. In Business Process Management (BPM) (Vol. 2, pp. 145–160). doi: 10.1007/978-3-319-58457-7_10.

23) Breunig, M. M., Kriegel, H.-P., Ng, R. T., & Sander, J. (2000). LOF: Identifying Density-Based Local Outliers. In Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data (стр. 93–104). doi: 10.1145/342009.335388.