МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет
імені Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ
Завідувач кафедри інтелектуальних інформаційних систем, д-р техн. наук, проф.
_____Ю. П. Кондратенко
«____»_____2024 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

СИСТЕМА МОНІТОРИНГУ ФІНАНСОВИХ ОПЕРАЦІЙ ДЛЯ ВИЯВЛЕННЯ ШАХРАЙСТВА

Спеціальність 122 «Комп'ютерні науки»

122 – КРБ – 401.23401001

Виконав студент 4-го курсу, групи 401з
_____ Р. Д. Белл
«19» червня 2024 р.

Керівник: професор (б. в. з.) кафедри ІІС, д-р техн. наук, доцент
_____О. В. Козлов
«19» червня 2024 р.

Миколаїв – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет ім. Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

Рівень вищої освіти **бакалавр**
Спеціальність **122 «Комп'ютерні науки»**
(шифр і назва)
Галузь знань **12 «Інформаційні технології»**
(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри інтелектуальних інформаційних систем, д-р техн. наук, проф.
_____ Ю. П. Кондратенко
«____» _____2024 р.

З А В Д А Н Н Я
на виконання кваліфікаційної роботи

Видано студенту групи 401з факультету комп'ютерних наук Беллу Річарду Джозефовичу.

1. Тема кваліфікаційної роботи «Система моніторингу фінансових операцій для виявлення шахрайства. Створення інформаційної системи, яка використовує аналітику даних та машинне навчання для виявлення незвичайних та потенційно шахрайських операцій».

Керівник роботи Козлов Олексій Валерійович, професор (б. в. з.) кафедри інтелектуальних інформаційних систем, доктор технічних наук Чорноморського національного університету імені Петра Могили, доцент.

Затв. наказом Ректора ЧНУ ім. Петра Могили від «26» січня 2024 р. № 13/1

2. Строк представлення кваліфікаційної роботи студентом «19» червня 2024 р.

3. Вхідні (початкові) дані до роботи:

- дані про фінансові транзакції з різних джерел;
- історичні дані про шахрайські операції;

– експертні оцінки різних методів виявлення шахрайства;

– критерії для оцінювання ефективності методів виявлення шахрайства.

4. Перелік питань, що підлягають розробці (зміст пояснювальної записки):

– аналіз сучасного стану задачі виявлення шахрайства у фінансовому секторі;

– огляд існуючих методів машинного навчання для виявлення шахрайства;

– експертне оцінювання методів виявлення шахрайства за визначеними критеріями;

– порівняльний аналіз результатів застосування обраних методів машинного навчання для розв'язання поставленої задачі.

5. Перелік графічного матеріалу: презентація.

6. Завдання до спеціальної частини: «Охорона Праці»

7. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис
Спеціальна частина з охорони праці	Григор'єва Л. І., д-р біолог. наук, професор кафедри екології	

Керівник роботи __д-р техн. наук, проф. Козов О. В._____

(наук. ступінь, вчене звання, прізвище та ініціали)

_____

(підпис)

Завдання прийнято до виконання ___Белл Р. Д._____

(прізвище та ініціали)

_____

(підпис)

Дата видачі завдання «_22_» ____лютого____ 2024 р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

Тема: Система моніторингу фінансових операцій для виявлення шахрайства. Створення інформаційної системи, яка використовує аналітику даних та машинне навчання для виявлення незвичайних та потенційно шахрайських операцій

№	Найменування роботи	Початок	Закінчення	Примітки
1	Подання заяви на затвердження теми та керівників КРБ	22.01.2024	26.01.2024	Виконано
2	Отримання завдання на виконання КРБ	26.01.2024	15.02.2024	Виконано
3	Складання календарного плану роботи на весь період виконання КРБ	01.03.2024	10.03.2024	Виконано
4	Отримання завдання на переддипломну практику	15.04.2024	29.04.2024	Виконано
5	Проходження переддипломної практики, збір та аналіз матеріалів до КРБ	29.04.2024	11.05.2024	Виконано
6	Розробка звіту з переддипломної практики	12.05.2024	15.05.2024	Виконано
7	Виконання КРБ: аналіз існуючих система моніторингу фінансових операцій для виявлення шахрайства	13.05.2024	22.06.2024	Виконано
8	Перший попередній захист КРБ на засіданні комісії кафедри	27.05.2024	27.05.2024	Виконано
9	Доробка та остаточне оформлення КРБ	28.05.2024	09.06.2024	Виконано
10	Другий попередній захист КРБ на засіданні комісії кафедри	10.06.2024	10.06.2024	Виконано
11	Подання КРБ рецензенту	18.06.2024	19.06.2024	Виконано
11	Подання КРБ, її електронної копії та інших документів (відгуку, рецензії) до захисту	19.06.2024	21.06.2024	Виконано
12	Захист КРБ перед екзаменаційною комісією (ЕК)	25.06.2024	25.06.2024	Виконано

Розробив студент __Белл Р. Д._____ _____
(прізвище, ім'я, по батькові студента) (*підпис*)

Керівник роботи __ д-р техн. наук, проф. Козлов О. В. _____
(*посада, прізвище, ім'я, по батькові)* *(підпис)*

«_10_» _____03_____ 2024 р.

АНОТАЦІЯ

кваліфікаційної роботи студента групи 401з ЧНУ ім. Петра Могили
Бела Річарда Джозефовича
Тема: «Система моніторингу фінансових операцій для виявлення шахрайства»

Актуальність кваліфікаційної роботи визначається зростанням популярності та значущості електротранспорту для сталого розвитку міст. Оптимізація маршрутів електротранспорту стає ключовим завданням для забезпечення ефективного та екологічного транспорту. Ця робота спрямована на розвиток і застосування інтелектуальних технологій у сфері планування маршрутів електротранспорту з метою підвищення ефективності та зручності пересування користувачів.

Об'єкт роботи – фінансові операції в банківських та інших фінансових установах.

Предмет роботи – методи аналітики даних та алгоритми машинного навчання для виявлення шахрайських операцій.

Метою кваліфікаційної роботи є розробка інформаційної системи для моніторингу фінансових операцій, що використовує методи аналітики даних та машинного навчання для виявлення незвичайних і потенційно шахрайських операцій.

Пояснювальна записка складається зі вступу, чотирьох розділів, висновків та додатку.

У першому розділі розглядається основні типи банківського шахрайства та підходам до його виявлення.

У другому розділі досліджено математичні основи роботи.

У третьому розділі описані програмна реалізації і експериментальні результати.

У четвертому розділі наведено функціонально-вартісний аналіз програмного продукту.

В результаті проведено аналіз предметної області та наявних аналогів систем моніторингу фінансових операцій, визначено вимоги до системи, розроблено архітектуру та функціональні компоненти системи, описано алгоритми машинного навчання для виявлення шахрайства.

Кваліфікаційна робота містить 66 сторінок, 43 рисунки, 5 таблиць, 26 використаних джерел та 1 додаток.

Ключові слова: шахрайство, фінансові операції, моніторинг, аналітика даних, машинне навчання, виявлення аномалій.

ABSTRACT

of student of group 401з of ChNU named after Petro Mohyla

Bell Richard

on the topic: " Financial Transaction Monitoring System for Fraud Detection"

The relevance of this qualification work is determined by the increasing popularity and significance of financial transaction monitoring for the security of financial institutions. Developing systems to detect and prevent fraud is essential for reducing financial losses and maintaining trust in the banking system. This work focuses on creating an effective monitoring system to identify fraudulent activities..

Object of the work – financial transactions in banks and other financial institutions..

Subject of the work – data analytics methods and machine learning algorithms for detecting fraudulent transactions.

The objective of this qualification work is to develop an information system for monitoring financial transactions that uses data analytics methods and machine learning to detect unusual and potentially fraudulent transactions.

The explanatory note consists of the introduction, four chapters, conclusions, and an appendix.

The first chapter reviews the main types of banking fraud and approaches to its detection.

The second chapter investigates the mathematical foundations of the work.

The third chapter describes the software implementation and experimental results.

The fourth chapter presents a functional and cost analysis of the software product.

An analysis of the subject area and existing analogs of financial transaction monitoring systems was conducted, the system requirements were defined, the architecture and functional components of the system were developed, and machine learning algorithms for fraud detection were described.

The qualification work contains 66 pages, 43 figures, 5 tables, 26 references, and 1 appendix.

Keywords: fraud, financial transactions, monitoring, data analytics, machine learning, anomaly detection.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

ML – Machine Learning (машинне навчання)

ATM – Automatic Teller Machine (банкомат)

POS – Point of Sale (торговий термінал)

CNP – Card Not Present (операції без фізичної наявності картки)

CP – Card Present (операції з фізичною наявністю картки)

ATO – Account Takeover (викрадення або злом акаунту)

SMOTE – Synthetic Minority Over-sampling Technique(синтетична техніка надзразкового збільшення меншості)

ADASYN – Adaptive Synthetic Sampling (адаптивна синтетична вибірка)

US-CNN – Condensed Nearest Neighbor Rule Undersampling (правило конденсованого найближчого сусіда для зменшення вибірки)

OSS – One-sided Selection (односторонній відбір)

GBDT – Gradient Boosting Decision Tree (градієнтний бустинг на рішеннях дерев)

KNN – k Nearest Neighbors (k найближчих сусідів)

PCA – Principal Component Analysis (метод головних компонент)

ВСТУП

Шахрайські дії в банківському секторі завдають значних збитків банкам та викликають значні незручності для клієнтів, які змушені витрачати час на повернення своїх коштів. Лише 25% від втрат через шахрайство можна повернути [1]. У 2020 році збитки банківських установ становили 32,39 мільярда доларів. Прогнози експертів показують, що до 2027 року банки можуть втратити 40,62 мільярда доларів, тоді як у 2011 році зафіксовано лише 9,84 мільярда доларів прямих збитків, що втричі менше, ніж у 2020 році [2].

Отже, найефективнішою стратегією боротьби з шахрайством є своєчасне виявлення та блокування шахрайських транзакцій. Проблема полягає в тому, що це потрібно робити в режимі реального часу, адже часто шахрайські дії характеризуються багатьма факторами, які людині важко обробити та створити ефективну систему правил для їх виявлення. Додатково, банки мають доступ лише до обмеженої інформації про клієнтів, що ускладнює аналіз усієї історії транзакцій клієнтів інших банків.

З огляду на те, що щосекунди через кожен банк проходять десятки операцій, а обсяг даних надзвичайно великий, виявлення та запобігання шахрайству є традиційною задачею для навчання з учителем в машинному навчанні.

Задачі роботи:

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

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

– зробити висновки щодо можливості використання отриманої моделі для комерційних задач, описати сильні сторони та недоліки отриманого рішення.

РОЗДІЛ 1 ОСНОВНІ ТИПИ БАНКІВСЬКОГО ШАХРАЙСТВА ТА ПІДХОДИ ДО ЙОГО ДО ЙОГО ВИЯВЛЕННЯ

1.1 Банківське шахрайство: Основні поняття та актуальність проблеми

Проблема виявлення шахрайських транзакцій у банківській сфері залишається актуальною з року в рік, оскільки кількість таких операцій та їх частка постійно зростають (рис. 1.1). Станом на 2018 рік, найпоширенішими типами шахрайства були операції без фізичної наявності картки (79,5%) та шахрайство в ATM або POS терміналах з використанням викрадених або втрачених (11,2%) та підроблених (5,39%) карток [3].

Рисунок 1.1 – Динаміка кількості банківського шахрайства в Європейському Союзі за категоріями [3]

Операції без фізичної наявності картки (CNP – card-not-present) - це банківські операції, які відбуваються без присутності особи, що здійснює оплату, на місці проведення операції. Прикладами таких операцій є оплата в інтернет-магазині або

через телефон. У цих випадках продавець не може перевірити, чи дійсно покупець є власником картки, а лише перевіряються номер картки, дата завершення її дії, PIN та код безпеки на звороті.

Через значний ризик шахрайських операцій (рис. 1.2) банк-емітент встановлює підвищену комісію за такі транзакції з організацій, які регулярно їх проводять. На відміну від транзакцій з фізичною присутністю клієнта, банк-еквайер, що обслуговує торговий рахунок, зобов'язаний відшкодувати втрати власнику картки у випадку шахрайської транзакції [4].

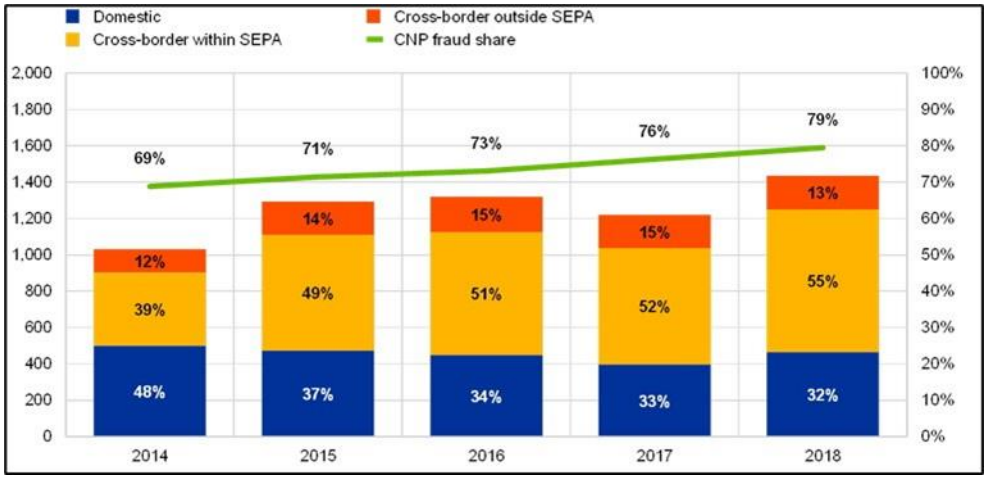


Рисунок 1.2 – Збитки та динаміка CNP шахрайства в Європейському Союзі у млн євро [3]

Другим типом шахрайства є фрод в операціях із фізичною наявністю картки (CP – card-present). У разі CP фрода, збитки клієнта відшкодовує банк, який обслуговує відповідний рахунок [5].

Використання вкрадених або загублених карток для транзакцій через ATM або POS є найпоширенішим типом CP шахрайства, хоча за останні роки частка таких махінацій зменшилася з 25% у 2014 році до 20% у 2018 році [3].

Використання підроблених карток залишається проблемою, проте частка такого шахрайства продовжує зменшуватись завдяки покращенню систем безпеки

банківських терміналів, на які зловмисники встановлюють пристрої для зчитування даних з карток.

До інших типів CP шахрайства належать викрадення або злом акаунту, отримання доступу до логіна і пароля для входу в банківську систему (ATO – account takeover), а також злом самої банківської системи. Як видно з рис. 1.3 (категорія "інші"), такі сценарії шахрайства стають дедалі популярнішими. Головною причиною цього є часті витоки даних та звичка користувачів інтернету використовувати однакові або недостатньо видозмінені паролі для різних сервісів, а також розвиток даркнету.

Збитки від CP шахрайства знижуються (рис. 1.3), тоді як збитки від CNP шахрайства лише зростають.

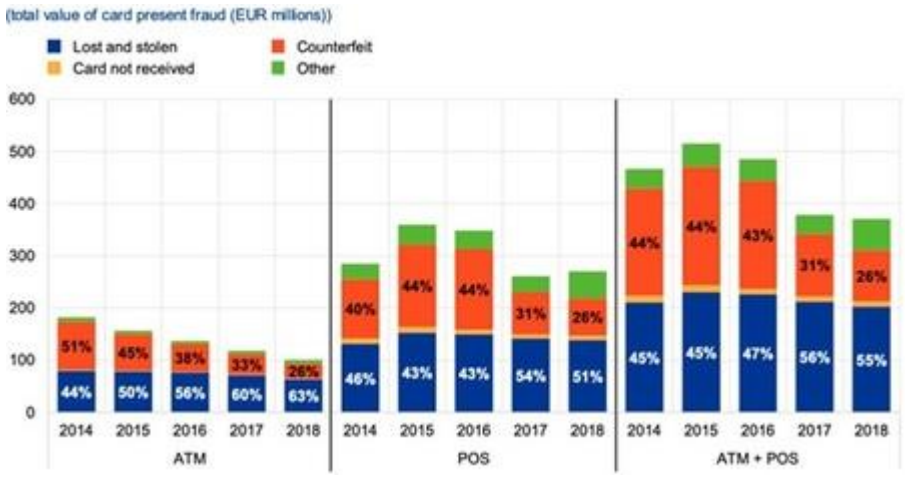


Рисунок 1.3 – Збитки від CP шахрайства в Європейському Союзі у млн євро [3]

1.2 Підходи до виявлення банківського шахрайства

Системи для виявлення банківського шахрайства можна розділити на дві групи:

1. ML (або Data driven) системи: Ці системи ґрунтуються на аналізі великих обсягів даних, таких як дата здійснення транзакції, сума переказу, інформація про кредитну історію клієнта тощо. В основі цих систем лежать алгоритми машинного

навчання, які можуть навчатись на вже розмічених даних про тип транзакції або без розмітки розподіляти транзакції на кластери, виявляючи підозрілу активність.

2. Експертні системи: Ці системи розроблені групою експертів конкретної банківської установи, які на основі свого досвіду створюють набір правил для виявлення шахрайства. Переваги експертних систем:

– простота та швидкість розробки і наповнення;
– можливість відстеження точної причини блокування.

Системи, створені на основі ML моделей, не можуть похвалитися такими перевагами. Крім того, детектор на основі правил часто працює швидше, оскільки показники повинні бути приведені до стандартного вигляду, а потім слідує множення на матрицю ваг. Однак, недоліки експертних систем значно переважають їхні переваги:

– виявлення та блокування тільки раніше відомих сценаріїв шахрайства;
– висока точність, але низька повнота;
– довгострокова розробка таких систем є тривалим та неефективним процесом, оскільки експерт повинен постійно відстежувати нові випадки шахрайства та вносити відповідні корективи;
– працюють тільки з явними для людини кореляціями між певними показниками та шахрайством, тому можуть бути легко виявлені та обійдені зловмисниками.

Усі ці недоліки повністю відсутні у ML системах. Тому має сенс поєднувати ці два типи систем (використовуючи правила як первинний фільтр) для покращення швидкості та прозорості виявлення шахрайства або використовувати тільки ML системи, оскільки у банківській сфері дуже важливо виявляти якомога більший відсоток шахрайства, адже будь-яка операція може призвести до значних прямих втрат для компанії.

Кейси шахрайства можуть значно відрізнятися у банках по всьому світу, а значить, і правила для його визначення також. Крім того, як було визначено раніше, основна складність полягає саме у розробці Data driven систем для протидії шахрайським транзакціям. Тому далі буде розглянуто виключно виявлення шахрайства за допомогою ML систем.

1.3 Виклики у розробці ML моделей для запобігання шахрайству

Більшість проблем, з якими стикаються інженери, що займаються розробкою моделей для виявлення банківського шахрайства, не є унікальними, а є типовими для виявлення шахрайства загалом. Серед головних викликів можна виділити такі:

Незбалансованість даних: Переважна більшість транзакцій (99%) є звичайними, що створює значну диспропорцію у даних. Це ускладнює навчання моделей, оскільки алгоритми можуть бути схильні до ігнорування меншості (шахрайських транзакцій).

Еволюція шахрайських методів: Шахраї постійно вигадують та впроваджують нові схеми шахрайства після того, як їхні попередні методи починають блокуватися. Це призводить до швидкої зміни характеру шахрайських операцій, і моделі повинні швидко адаптуватися до нових схем.

Сезонність транзакцій: Поведінка звичайних користувачів та шахраїв може змінюватися в залежності від сезону. Наприклад, покупки перед святами або на Чорну п’ятницю відрізняються за багатьма факторами від звичайних днів. Тому моделі потребують розмітки та навчання на даних, що охоплюють великі часові проміжки, щоб враховувати сезонні коливання.

Необхідність регулярного пост-аналізу: Постійний аналіз є необхідним для виявлення нових сценаріїв шахрайства та поповнення ними датасету. Це витікає з необхідності швидкої адаптації до нових методів шахраїв.

Затримка в оновленні даних: Існує затримка в кілька днів у представленні актуальних даних для моделі, оскільки скарги від клієнтів на шахрайство (рис. 1.4) надходять не одразу. Це створює розрив між поточними даними та реальними подіями, що ускладнює оперативне реагування на нові шахрайські транзакції.

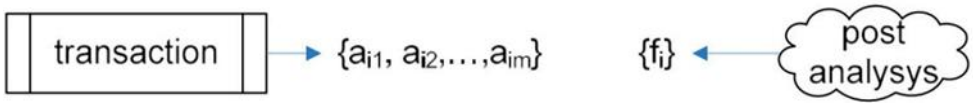


Рисунок 1.4 – Пост-аналіз для виявлення нових сценаріїв шахрайства

Ці виклики вимагають розробки комплексних стратегій для ефективного виявлення та запобігання шахрайству, використовуючи сучасні методи машинного навчання.

1.4 Складнощі навчання на незбалансованих даних

Дані вважаються незбалансованими, коли розподіл класів є таким: 1:2, 1:10, 1:100, 1:1000 і т.д. У датасеті Credit Fraud [6], який використовується в цій роботі, міститься 284807 транзакцій, з яких лише 492 є шахрайськими. Це означає, що 99,83% транзакцій є звичайними (рис. 1.5).

Такий розподіл призводить до «упередженості» моделей, коли всі транзакції можуть бути класифіковані як нормальні, і при цьому модель досягає точності 99,83%, що було б чудово для моделі на збалансованих даних. З цього випливає, що звичайні метрики для класифікації, такі як точність та ROC AUC, не є репрезентативними для даного класу задач. Крім того, через невелику кількість даних мінорного класу або їх відсутність виникає проблема "недостатності інформації" [7, с. 261].

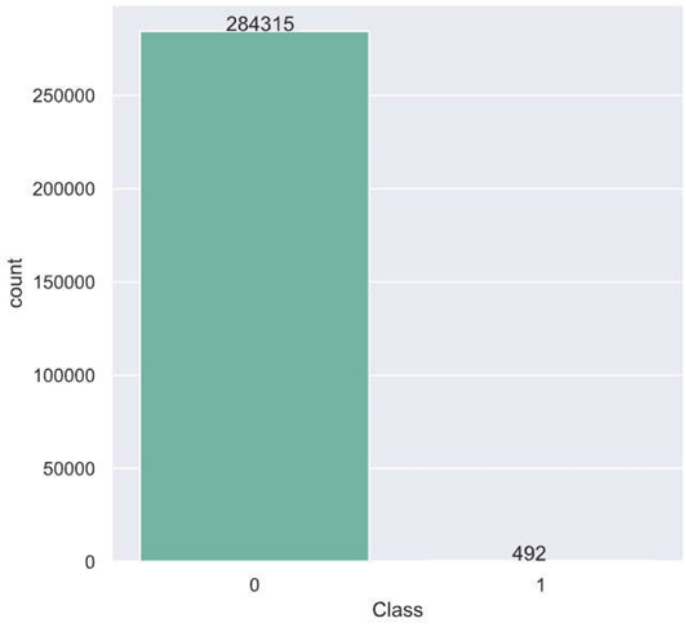


Рисунок 1.5 – Кількість звичайних та фродових транзакцій

Ще одна проблема полягає в тому, що при застосуванні методів збільшення кількості спостережень менш представленого класу, таких як SMOTE або oversampling, ми збільшуємо кількість прикладів до співвідношення 1:1, що призводить до значного збільшення розміру датасету і негативно впливає на швидкість навчання. Наприклад, для звичайної задачі класифікації зі збалансованим розподілом класів необхідні тисячі чи десятки тисяч прикладів для розробки, оцінки та вибору моделі. Для аналізу транзакцій за декілька днів або місяців отримаємо десятки мільйонів прикладів для тренування. Отже, працюючи з виявленням шахрайства, потрібно досліджувати не лише методи збільшення кількості прикладів мінорного класу, а й методи зменшення кількості прикладів домінуючого класу, такі як TOMEK та random undersampling.

Дисбаланс класів є визнаним ускладнюючим фактором для класифікації. Проте деякі дослідження вказують, що висока диспропорція у класах не є єдиною причиною зниження якості моделі під час навчання [7, с. 253]. Серед інших факторів:

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

Шуми у характеристиках виникають через помилки або шум в описі прикладів (рис. 1.6). Існують два типи шумів: шум в атрибутах та шум у цільовій змінній. Шум у цільовій змінній завдає набагато більше шкоди моделі, ніж шум в атрибутах, навіть для звичайних збалансованих моделей. У випадку незбалансованої моделі, коли прикладів одного з класів і так мало, помилки у атрибутах чи класі прикладу завдають руйнівної шкоди.

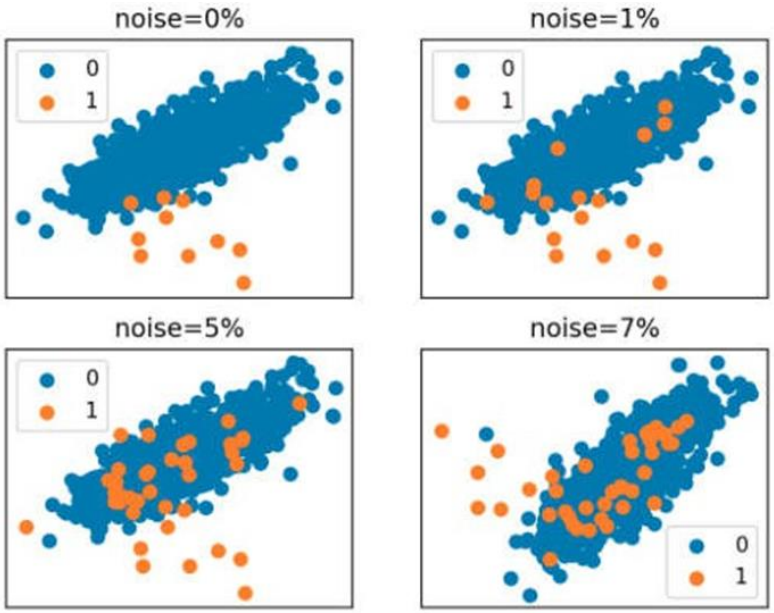


Рисунок 1.6 – Штучно зашумлені дані [8]

Другим важливим фактором є розподіл прикладів у просторі ознак. Якщо візуалізувати простір ознак (на графіку, якщо є 2-3 ознаки, або використовуючи алгоритми машинного навчання, такі як PCA чи t-SNE), то ідеальним варіантом буде чіткий розподіл на кластери (рис. 1.7). У такому випадку легко побудувати

класифікатор, який досягає гарних результатів як на збалансованих, так і на незбалансованих даних.

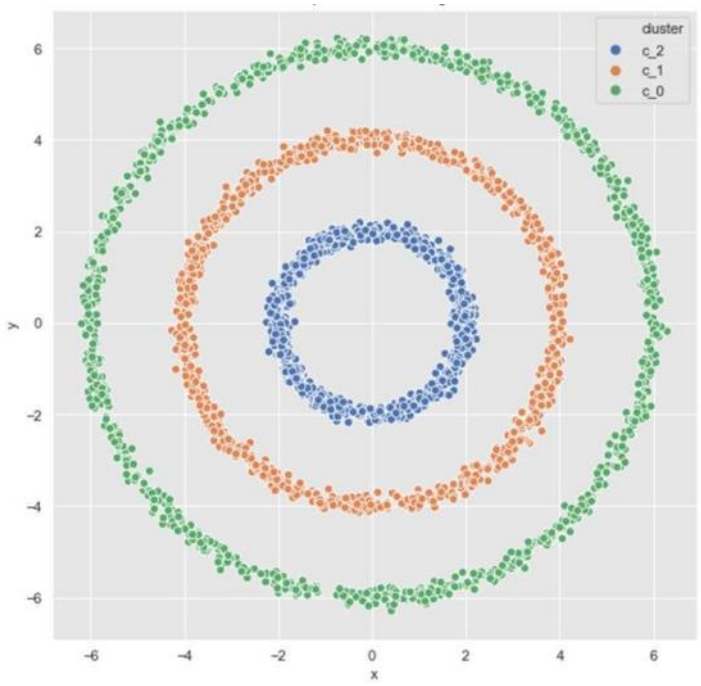


Рисунок 1.7 – Чіткий розподіл ознак x, y на кластери за класом [9]

У реальних задачах чіткий візуальний розподіл на класи трапляється нечасто, і кожен клас можна додатково розділити на декілька підкласів, що призводить до створення кількох різних груп або кластерів прикладів у просторі ознак. Наприклад, на (рис. 1.8) видно, що клас 0 сам розділений на 2 кластери, і немає чіткого розподілу між класами 0 та 1 у просторі ознак.

Такі підгрупи називають диз'юнктами. Невеликий диз'юнкт охоплює лише кілька прикладів з навчального набору даних [8]. Цей розподіл ускладнює відокремлення класів для моделі, оскільки необхідно ідентифікувати та включити кожну підгрупу або кластер до визначення межі класу (рис. 1.9).

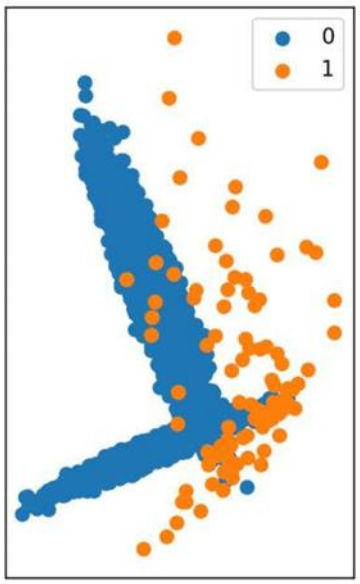


Рисунок 1.8 – Нечіткий розподіл ознак x, y на кластери за цільовою змінною [8]

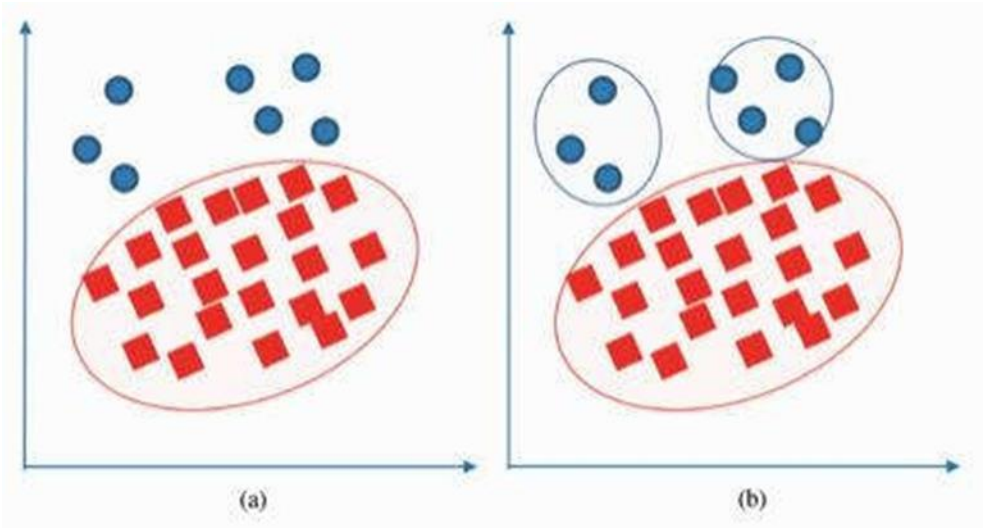


Рисунок 1.9 – Підгрупи у меншому класі [10]

Коли ми маємо дуже незбалансовані дані з невеликою кількістю прикладів меншого класу, розподіл класів на підгрупи стає яскравою проблемою. Невелика щільність даних робить кожен приклад важливим, що ускладнює уникнення перенавчання моделі та одночасне розпізнавання невеликих диз'юнктів. Тут може знадобитися консультація експерта щодо предметної області.

Недостатність щільності має особливий вплив на алгоритми, що використовують підхід «розділяй та володарюй» (такі як дерева рішень) та у задачах покриття множин (наприклад, індукція правил), де наявність підгруп веде до створення невеликих диз'юнктів. У таких алгоритмах моделі розділені на кілька частин, і концепція класу представлена як диз'юнкт цих частин для кожного класу [10].

1.5 Необхідність регулярного донавчання моделі

Як було описано вище, нові сценарії шахрайства з'являються регулярно, і характер звичайних транзакцій також поступово змінюється. Це створює необхідність регулярного донавчання моделі з частотою від декількох днів до місяця. Така необхідність спричиняє значні ускладнення, адже виникає питання регулярної розмітки даних. У більшості фінансових компаній є окремий відділ, що займається боротьбою з шахрайством. Спеціалісти цього відділу регулярно здійснюють моніторинг та проводять аналіз дій шахраїв. Однак шахрайських операцій зазвичай настільки багато, що спеціалісти «полюють» виключно на великих шахраїв, а втрату невеликих сум покриває підвищена банківська комісія для CNP операцій. Тому інформація, представлена таким відділом, була б неповною і недостатньою, враховуючи необхідність регулярного збору інформації як про звичайні транзакції, так і про шахрайські.

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

– автоматичний збір скарг клієнтів на шахрайство: цей процес має затримку в 1-2 дні;

– автоматична обробка звітів від відділу з протидії шахрайству: ці звіти містять інформацію про шахрайські операції, що призвели до значних збитків, з затримкою в 2-7 днів;

– пост-аналіз (ручний або напів-автоматичний): він спрямований на виявлення нових сценаріїв шахрайських дій з затримкою в 30 днів. В результаті датасет може бути поповнений не тільки шахрайськими прикладами, а й звичайними операціями, що також дозволяє корегувати модель на сезонність.

Процес донавчання моделі зображений на рис. 1.10.

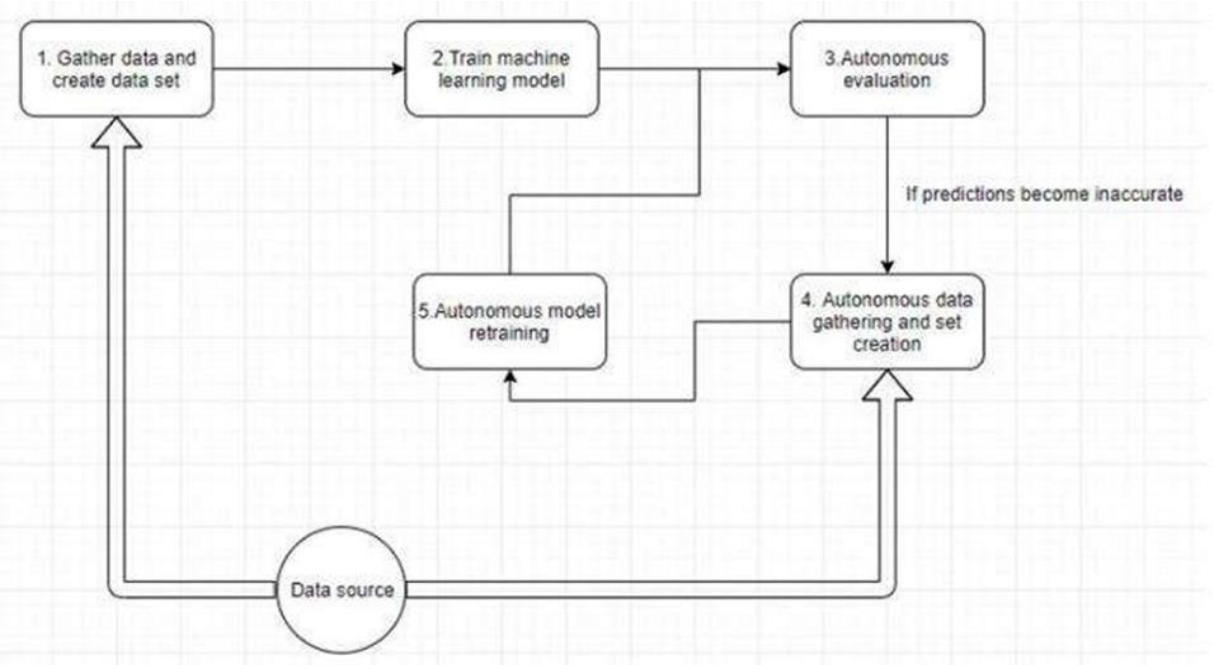


Рисунок 1.10 – Автоматичне дотренування моделі [11, с. 446]

Після розробки первинної моделі та автоматизації процесу, оновлення результатів відбувається таким чином: до тестового набору датасету додаються нові приклади і здійснюється передбачення за допомогою наявної моделі. Якщо показник обраної для оцінки моделі метрики або метрик не змінюється суттєво, стару модель залишають для виявлення шахрайства. В іншому випадку, модель має бути перетренована. Існує кілька зручних та швидких бібліотек для мови Python, що

дозволяють швидко виконувати це завдання. В результаті, отримуємо нову модель для класифікації без втручання людини.

1.6 Висновки до розділу 1

Отже, враховуючи зростання як абсолютної кількості шахрайських транзакцій у банківській сфері, так і збільшення їх частки відносно звичайних операцій, проблема розробки автоматизованої системи для їх виявлення залишається актуальною. У цьому розділі були проаналізовані основні виклики при розробці таких моделей, включаючи:

– регулярну появу нових сценаріїв шахрайства, що вимагає частого оновлення та донавчання моделей;

– високу незбалансованість даних, яка створює труднощі в навчанні моделей та потребує застосування спеціальних методів для корекції дисбалансу.

Необхідність розробки ефективних стратегій для обробки даних, що включають автоматичний збір скарг клієнтів, обробку звітів від відділів протидії шахрайству та регулярний пост-аналіз.

Розглянуті виклики та методи їх подолання підкреслюють важливість використання машинного навчання та комплексних підходів для успішного виявлення шахрайських транзакцій у банківській сфері.

РОЗДІЛ 2 МАТЕМАТИЧНІ ОСНОВИ РОБОТИ

2.1 Стратегії семплінгу в умовах незбалансованості даних

Оверсемплінг (oversampling) та андерсемплінг (undersampling) є важливими кроками при роботі з незбалансованими даними, оскільки вони дозволяють "вирівняти" розподіл кількості прикладів різних класів до співвідношення 50:50.

Найпростіша стратегія оверсемплінгу полягає в копіюванні наявних прикладів меншого класу, тоді як андерсемплінг передбачає випадкове видалення елементів більшого класу (random undersampling). Проте такі прості підходи не працюють для банківських даних, оскільки, у випадку random undersampling, можуть бути видалені важливі дані про нешахрайські транзакції. З іншого боку, простий оверсемплінг не додає нової інформації та не вирішує проблему розподілу даних, описану в розділі 1.4.

Наприклад, випадково можуть бути видалені майже всі приклади звичайних транзакцій на суму понад 330 доларів, яких дуже мало (рис. 2.1), водночас різко збільшиться кількість прикладів з шахрайськими транзакціями, які мають трохи інший ціновий розподіл (рис. 2.2).

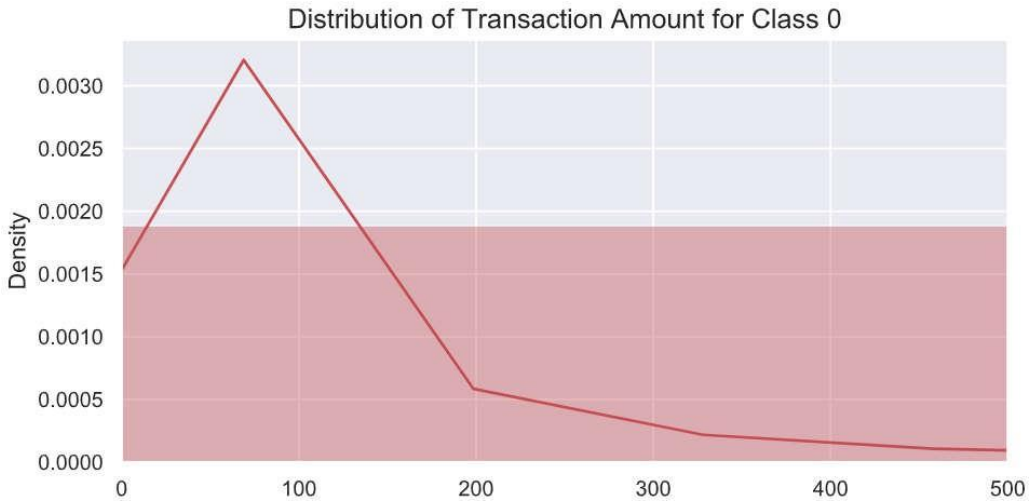


Рисунок 2.1 – Розподіл звичайних транзакцій за сумою

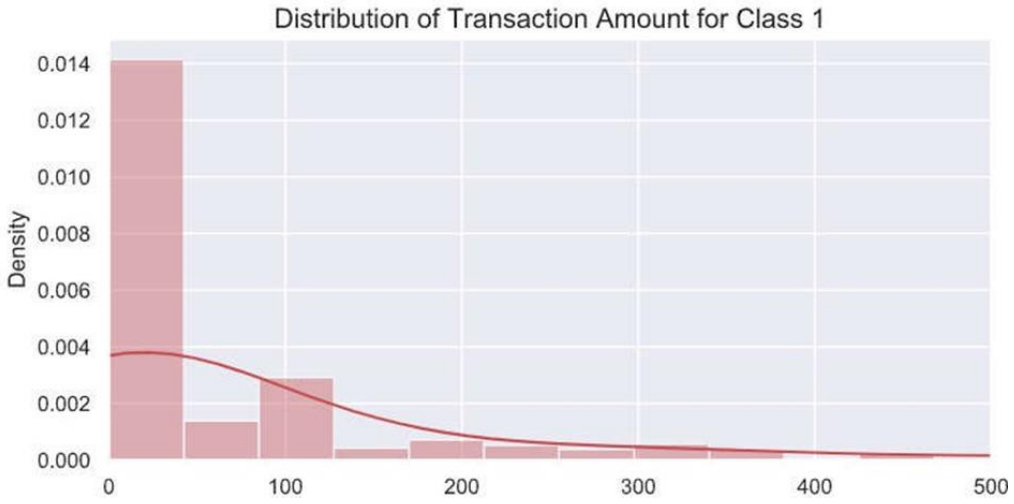


Рисунок 2.2 – Розподіл шахрайських транзакцій за сумою

У результаті модель буде мати високу точність, але не буде придатною для використання через упередженість до великих за сумою транзакцій. Тому має сенс дослідити та використовувати математичні методи оверсемплінгу та андерсемплінгу, а не випадкові видалення та копіювання даних.

Математичні методи оверсемплінгу та андерсемплінгу:

– SMOTE (Synthetic Minority Over-sampling Technique): цей метод створює нові приклади меншого класу шляхом інтерполяції між існуючими прикладами. SMOTE допомагає зменшити проблему перенавчання, яка може виникати через копіювання тих самих прикладів;

– ADASYN (Adaptive Synthetic Sampling): подібно до SMOTE, але адаптивний підхід створює більше нових прикладів для тих областей, де менше прикладів меншого класу. Це дозволяє краще покривати розріджені області даних;

– TOMEK Links: метод для андерсемплінгу, який видаляє зразки з домінуючого класу, що утворюють пари з найближчими зразками меншого класу, які вони можуть неправильно класифікувати. Це допомагає очистити межі між класами;

– Cluster-based Undersampling: замість випадкового видалення, цей метод кластеризує дані більшого класу і вибирає представників кожного кластера для видалення, що дозволяє зберігати інформацію про розподіл даних.

Ці методи допомагають покращити якість моделей для виявлення шахрайства, забезпечуючи краще представлення як звичайних, так і шахрайських транзакцій у навчальних даних. Таким чином, модель стає більш стійкою та точнішою у виявленні шахрайства в реальних умовах.

2.2 Синтетичне наповнення датасету за допомогою SMOTE

Алгоритм Synthetic Minority Oversampling Technique (SMOTE) дозволяє створювати нові «синтетичні» приклади на основі наявних даних замість простого копіювання. На розробку цього підходу вплинув успішний досвід аугментації даних, використаний для класифікації набору рукописних символів. Тоді додаткові дані для навчання створювалися за допомогою обертання, розтягування та інших перетворень зображень, що позитивно позначилося на точності класифікації. Проте, на відміну від цього прикладу, SMOTE синтезує нові приклади в просторі ознак, а не в просторі даних.

Більша кількість прикладів меншого класу синтезується за допомогою алгоритму KNN (k найближчих сусідів), який застосовується до всіх прикладів класу. Після цього створюються нові приклади, які лежать на сегментах ліній, що з'єднують найближчих сусідів класу. Враховуючи кількість нових прикладів, сусіди з k найближчих обираються випадковим чином. Більшість реалізацій SMOTE використовують 5 найближчих сусідів. Наприклад, якщо потрібно збільшити вибірку на 400%, обираються 4 сусіди з 5 найближчих, і один новий приклад генерується для кожного з напрямків.

Нові приклади створюються за таким алгоритмом:

– вираховується різниця між вектором ознак (зразком), що розглядається, та його найближчим вектором;

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

В результаті отримують випадкову точку на лінії між найближчим сусідом та прикладом, що розглядається (рис. 2.3) [12, с. 328].

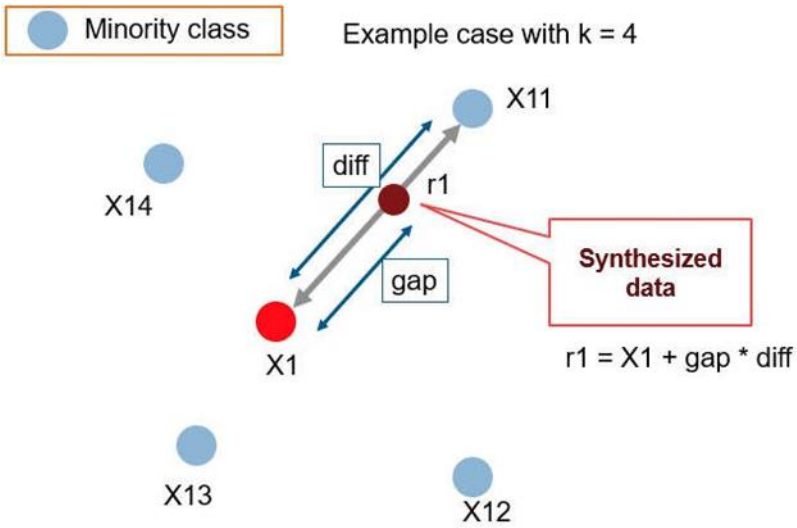


Рисунок 2.3 – Генерація нової точки у просторі ознак [13]

Псевдокод SMOTE (рис. 2.4) [12].

Завдяки згенерованим прикладам класифікатор створює більші та менш специфічні регіони рішень (рис. 2.5, (c)). Це допомагає дереву рішень уникнути перенавчання. Точність коректного розпізнавання меншого класу також зростає при оверсамплінгу за допомогою SMOTE більше ніж на 200% у порівнянні з простим копіюванням прикладів (рис. 2.6).

Крім того, експерименти показали, що комбінування методів оверсемплінгу, таких як SMOTE, та андерсемплінгу позитивно впливає на якість моделі. Таким чином

модель звільнюється від упередженості до більшого класу, що покращує загальну продуктивність.

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 2.4 – Алгоритм SMOTE [12, с. 329]

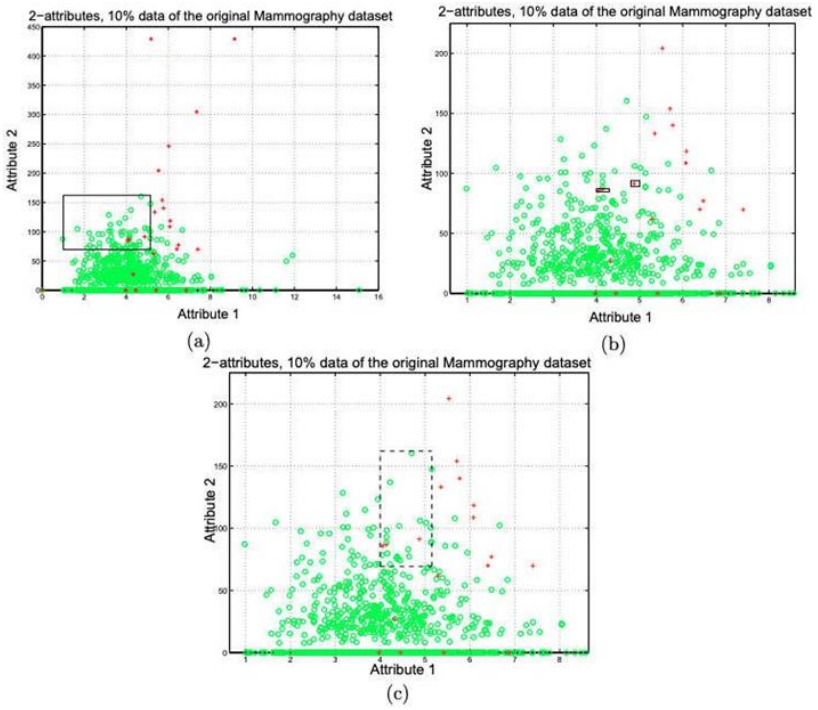


Рисунок 2.5 – Вплив згенерованих за допомогою SMOTE прикладів (позначені на (c)) [12, с. 327]

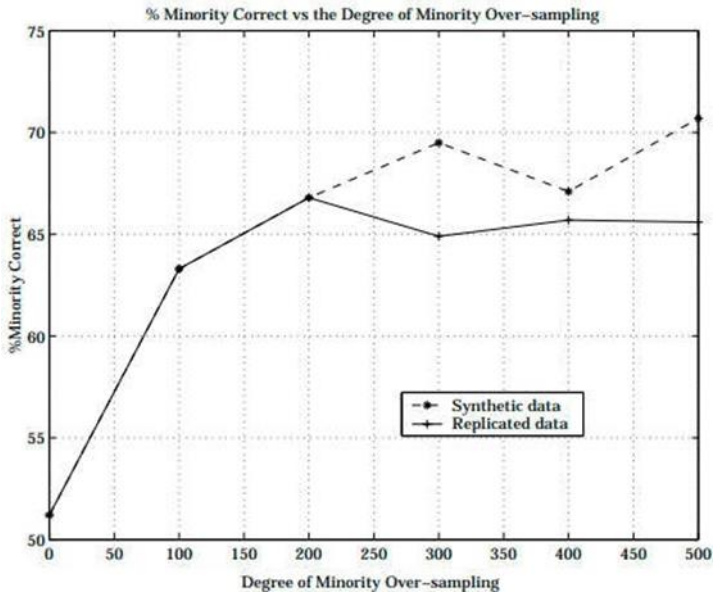


Рисунок 2.6 – Порівняння точності розпізнання меншого класу з використанням дублювання та SMOTE [12, с. 321]

2.3 Математичні методи видалення прикладів мажоритарного класу

Одним із найпопулярніших підходів до андерсемплінгу є використання зв’язків Томека (Tomek Links), який був представлений у 1976 році, але залишається актуальним і сьогодні через свою простоту та ефективність. Нехай приклади Ei і Ej належать до різних класів, а d(Ei, Ej) - відстань між цими прикладами. Пара (Ei, Ej) називається зв'язком Томека, якщо не знайдеться жодного прикладу Em, для якого буде справедлива сукупність нерівностей (2.1):

$$\begin{cases} d(Ei, Em) < d(Ei, Ej) \\ d(Ej, Em) < d(Ei, Ej) \end{cases}$$

Згідно з цим підходом, всі мажоритарні записи, що входять у зв'язок Томека, повинні бути видалені з набору даних. Цей метод ефективно видаляє записи, які можуть бути шумом (рис. 2.7) [14].

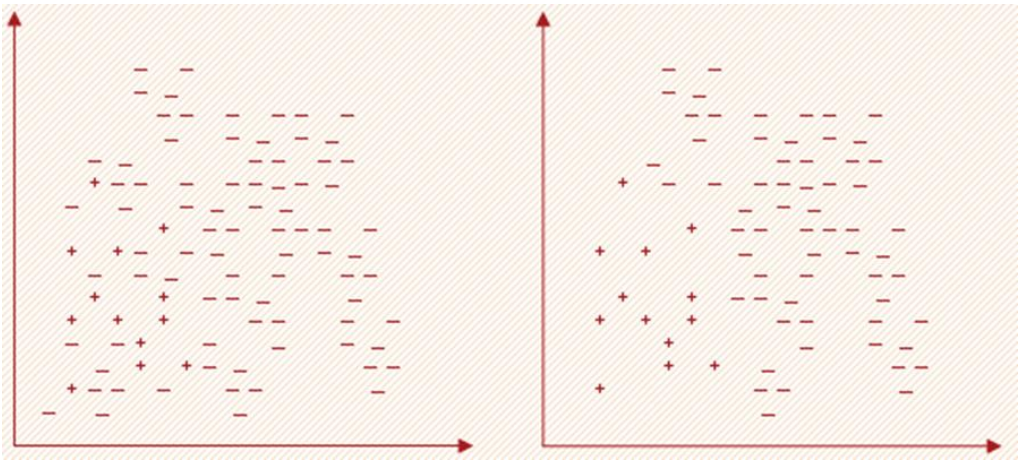


Рисунок 2.7 – Датасет до видалення зв’язків Томека та після [14]

Правило найближчого ущільненого сусіда (Condensed Nearest Neighbor Rule Undersampling, US-CNN) – це метод андерсемплінгу, метою якого є вибір такого набору прикладів, який не знижує ефективність моделі. Цей набір називають мінімальним консистентним набором.

Цей результат шляхом перебору всіх прикладів датасету і додавання їх до мінімально консистентного набору, якщо вони не можуть бути правильно

класифіковані за допомогою прикладів, які вже є в наборі. Спершу цей підхід був запропонований для зменшення кількості пам’яті, необхідної для роботи алгоритму k найближчих сусідів (KNN).

Щоб застосувати цей підхід до задач незбалансованої класифікації, мінімальний консистентний набір складається з усіх прикладів меншого класу, і лише приклади більшого класу, які не можуть бути правильно класифіковані, поступово додаються до набору. Алгоритм використовує KNN для класифікації прикладів і вирішує, чи потрібно їх додавати до набору (рис. 2.8-2.9) [15].

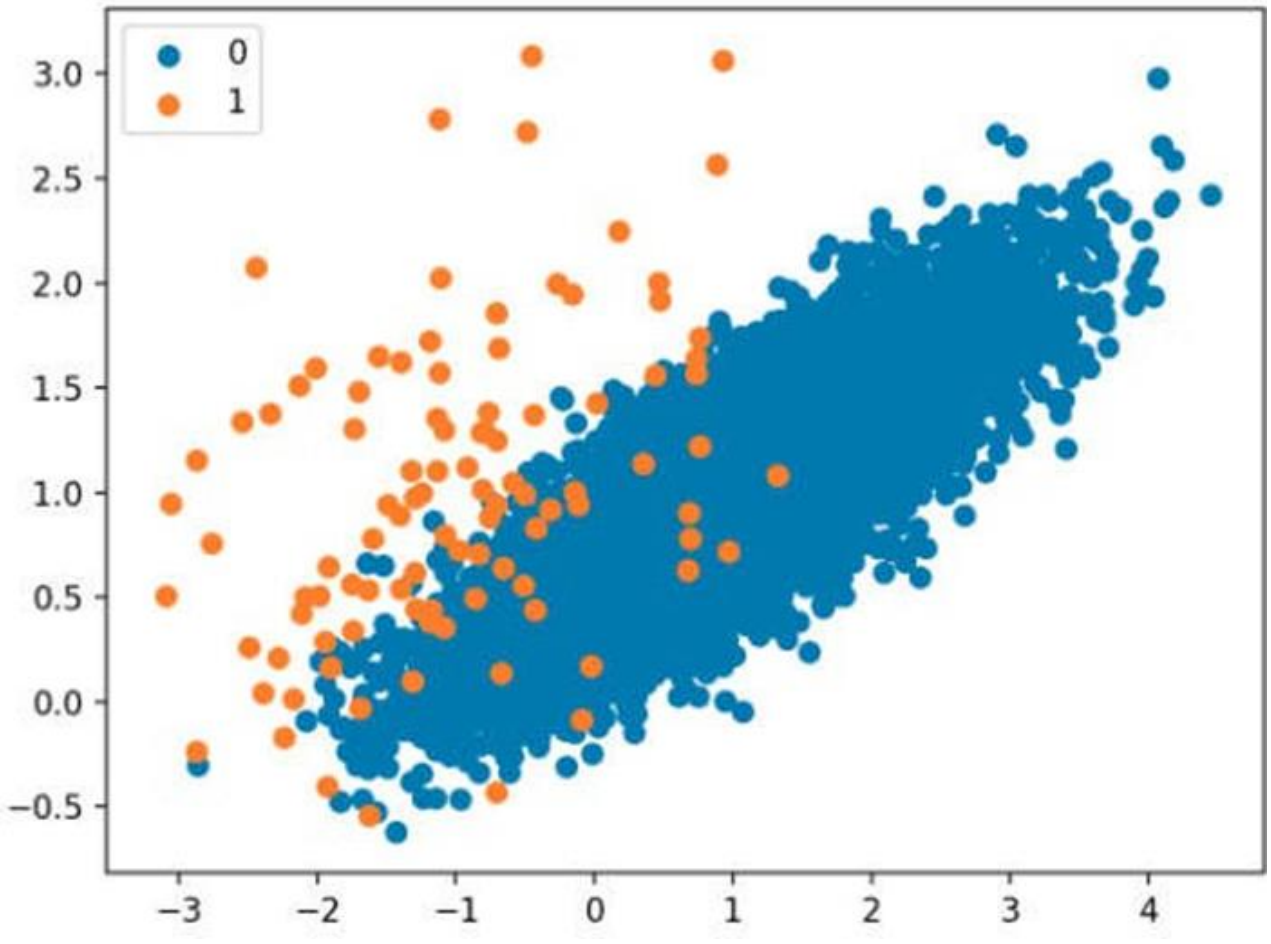


Рисунок 2.8 – Набір даних до андерсемплінгу US-CNN [15]

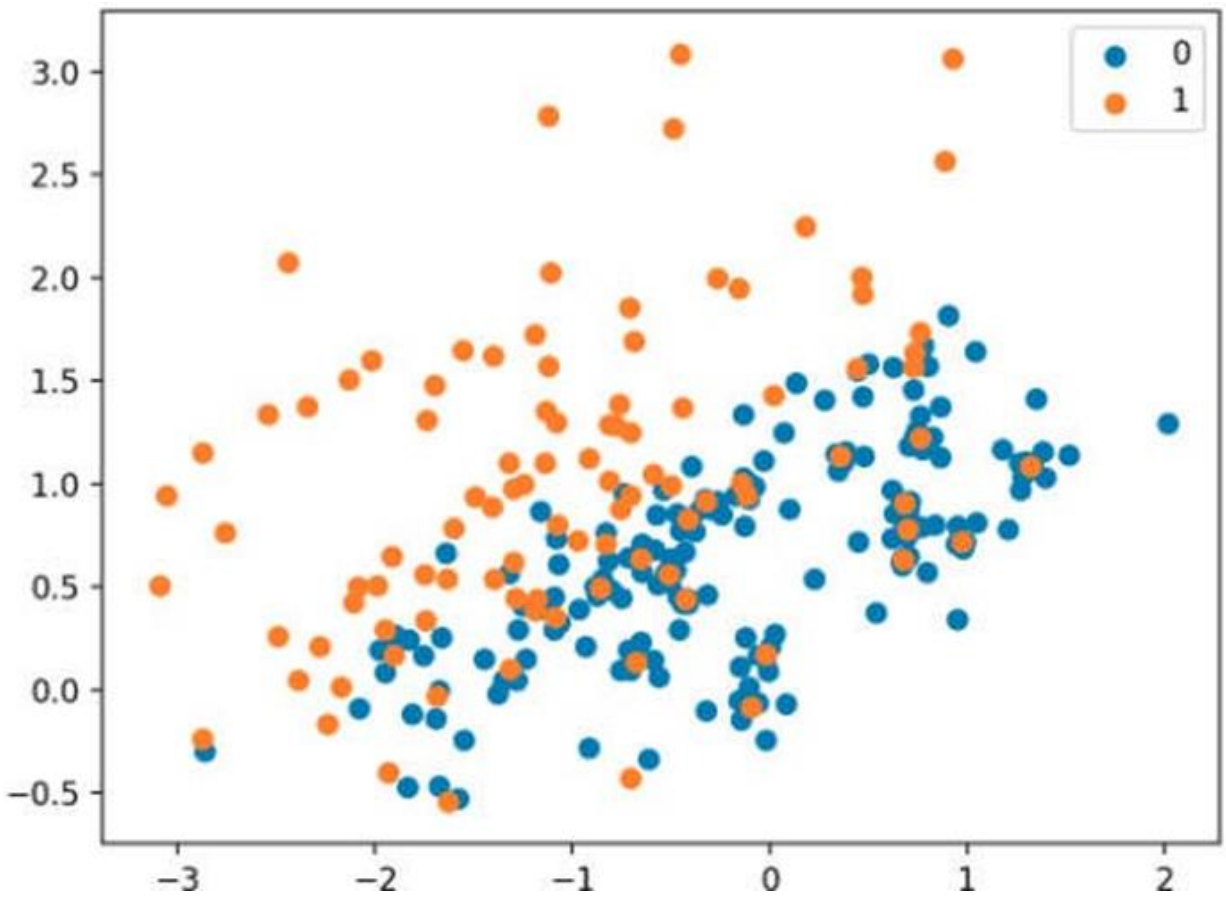


Рисунок 2.9 – Набір даних після андерсемплінгу US-CNN [15]

Як видно з рис. 2.8 та рис. 2.9, залишаються найважливіші приклади для класифікації на поточних даних. Однак варто звернути увагу на те, що всі точки далі по осі X (X є [2.5, 4.5]) класу 0 були видалені. Залишаються лише приклади більшого класу «навколо» прикладів меншого. Метод одностороннього відбору (One-sided Selection, OSS) поєднує два вищезгадані алгоритми – зв'язки Томека та US-CNN. Точніше, він полягає у послідовному застосуванні першого та другого алгоритмів (рис. 2.10) [16]. Приклади одного класу, що лежать дуже близько до прикладів іншого, виявляються за зв'язками Томека та видаляються, оскільки вони розглядаються як шум і не погіршують якість класифікатора. Після цього мета US-CNN – видалити приклади з більшого класу, які розташовані далеко від межі рішень (decision border).

Результатом роботи алгоритму є всі приклади меншого класу та незашумлені і релевантні приклади більшого класу [7, с. 84]..

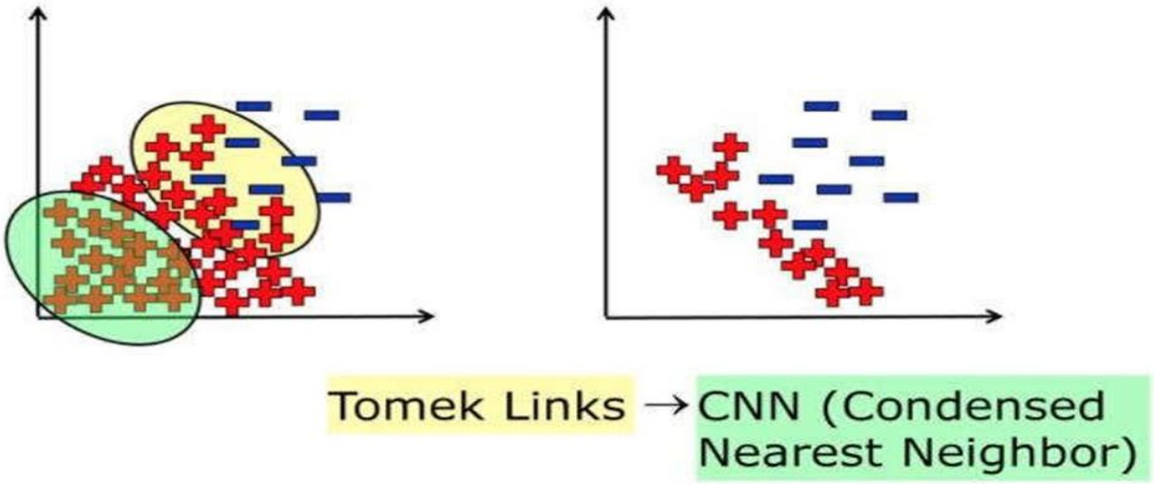


Рисунок 2.10 – Алгоритм OSS [16]

2.4 Плюси та мінуси використання андерсемплінгу та оверсемплінгу

Хоча згадані вище підходи (SMOTE, зв’язки Томека, US-CNN, OSS) працюють з даними значно ефективніше, ніж випадкове надзразкове збільшення та просте зменшення вибірки, їхні недоліки все ж таки не усуваються повністю. Основні недоліки наведені в таблиці 2.1.

Таблиця 2.1 – Недоліки використання оверсемплінгу та андерсемплінгу

Андерсемплінг	**Оверсемплінг**
- Можуть бути видалені потенційно корисні дані	- Перенавчання
	- Значно збільшує час, витрачений на обробку даних та навчання
	- Можливе копіювання або генерування нових прикладів на основі шуму

Проте, не використовуючи ці підходи, якість вихідної моделі буде значно нижчою. Оскільки зазначені підходи мають ряд переваг, їх список наведено у таблиці 2.2.

Таблиця 2.2 – Переваги використання оверсемплінгу та андерсемплінгу

Андерсемплінг	**Оверсемплінг**
- Значно зменшує час, витрачений на обробку даних та навчання	- Немає втрати інформації
- Знищує частковий збіг між класами на користь меншого класу, правильно визначити який важливіше, ніж приклади більшого класу	- Можливість наповнити датасет копіями або навіть штучно згенерованими прикладами, що дуже важливо в умовах недостатку даних
- Знищує дані, які точно не будуть корисними при класифікації	

Зважаючи на вищезазначене, можна зробити висновок, що використання методів оверсемплінгу та андерсемплінгу, незважаючи на їхні недоліки, значно покращує якість моделей, зменшуючи проблеми, пов'язані з незбалансованими даними. Методи, такі як SMOTE, зв’язки Томека, US-CNN та OSS, дозволяють ефективніше обробляти дані, знижуючи ризик перенавчання та втрати важливої інформації, а також забезпечуючи краще представлення меншого класу.

2.5 Методи прогнозування

Існує безліч алгоритмів машинного навчання, які добре підходять для задачі класифікації шахрайських та звичайних транзакцій. У цьому дослідженні були обрані декілька алгоритмів з різних груп для оцінки якості класифікатора, побудованого на різних архітектурах. Розглянуті алгоритми включають:

– логістична регресія (статистичний алгоритм);
– LightGBM (градієнтний бустинг, що використовує дерева рішень);

– проста нейронна мережа.

2.6 Логістична регресія

Логістична регресія – це статистичний метод бінарної класифікації, який застосовує логістичну (сигмоїдну) функцію для створення прогнозів. Логістична регресія є досить простим алгоритмом, але часто демонструє хороші результати класифікації, а також є швидкою у реалізації та навчанні.

Результат передбачення логістичної регресії – це ймовірність p від 0 до 1, що є результатом використання сигмоїдної функції для передбачення та значною перевагою в порівнянні з лінійною регресією, результат передбачення якої може виходити за межі від 0 до 1 (рис. 2.11) [17].

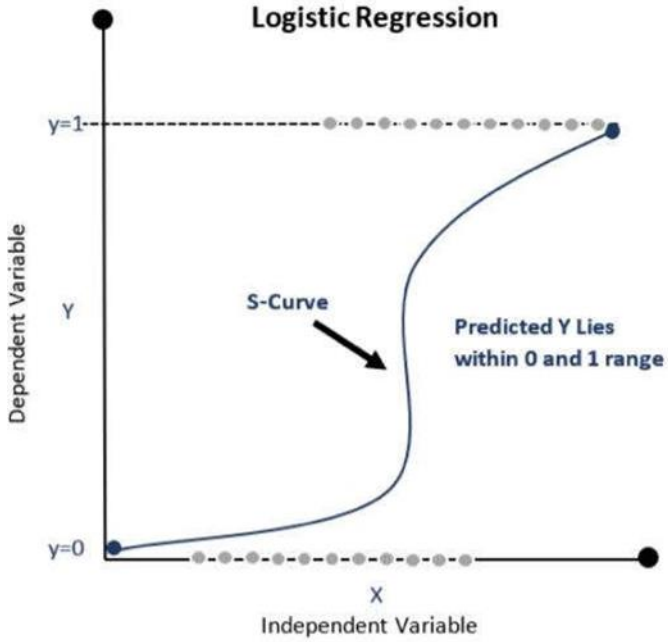


Рис. 2.11 – Передбачення в алгоритмі логістичної регресії [17]

Приналежність до класу визначається таким чином:

$$\text{Якщо } p \geq 0.5, \text{тоді клас } = 1$$
$$\text{Якщо } p < 0.5, \text{тоді клас } = 0$$

Формула сигмоїдної функції:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

При навчанні для знаходження оптимальних ваг використовується така функція втрат, значення якої ми оптимізуємо за допомогою градієнтного спуску:

$$J(\theta) = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(h_\theta(x_i)) + (1 - y_i)\log(1 - h_\theta(x_i))\right]$$

Де:

θ – ваги

n – кількість прикладів

$h_\theta(x_i)$ – передбачення для вхідного вектора

y_i – вихідний клас

Оптимізація функції втрат відбувається за допомогою градієнтного спуску (рис. 2.12). Правило оновлення вагів:

$$\theta := \theta - \alpha \nabla_\theta J(\theta)$$

Де α – крок навчання.

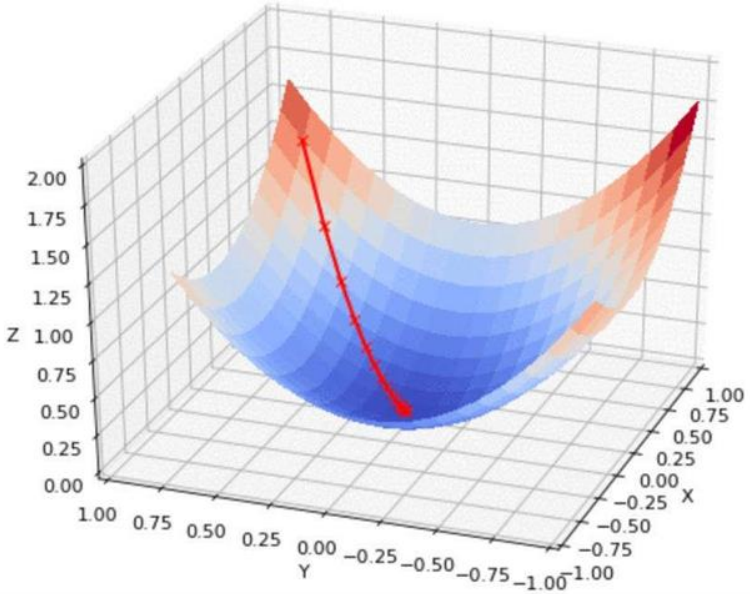


Рис. 2.12 – Пошук оптимуму градієнтним спуском [18]

2.7 LightGBM

LightGBM (Light Gradient Boosting Machine) – це підсилювальний алгоритм з сімейства Gradient Boosting Decision Tree.

Підсилювальні алгоритми – це сімейство алгоритмів, які з декількох слабких учнів створюють сильного учня. У випадку Gradient Boosting Decision Tree (GBDT), слабкими учнями є дерева рішень, які комбінуються в ансамблі для підвищення точності передбачень.

GBDT має багато ефективних імплементацій, таких як XGBoost та pGBRT. Однак, навіть з оптимізаціями, ці алгоритми все одно не рекомендується застосовувати на великих обсягах даних через високу обчислювальну складність. LightGBM вирішує цю проблему завдяки двом новим технікам:

– Gradient-based One-Side Sampling (GOSS);

– Exclusive Feature Bundling (EFB).

GOSS використовується для виключення більшої кількості прикладів з малими градієнтами, залишаючи лише важливі приклади для сканування на приріст інформації. EFB знаходить взаємовиключні ознаки, щоб зменшити кількість ознак, які потрібно обробляти (рис. 2.13) [19] та (рис. 2.14) [19].

Input: I: training data, d: iterations
Input: a: sampling ratio of large gradient data
Input: b: sampling ratio of small gradient data
Input: $loss$: loss function, L: weak learner
models ← { }, fact ← $\frac{1-a}{b}$
topN ← a × len(I) , randN ← b × len(I)
for $i = 1$ ***to*** d **do**
  preds ← models.predict(I)
  g ← $loss$(I, preds), w ← {1,1,...}
  sorted ← GetSortedIndices(abs(g))
  topSet ← sorted[1:topN]
  randSet ← RandomPick(sorted[topN:len(I)], randN)
  usedSet ← topSet + randSet
  w[randSet] × = fact ▷ Assign weight $fact$ to the small gradient data.
  newModel ← L(I[usedSet], − g[usedSet], w[usedSet])
  models.append(newModel)

Рисунок 2.13. Алгоритм GOSS [19]

Algorithm 3: Greedy Bundling

Input: F: features, K: max conflict count
Construct graph G
searchOrder ← G.sortByDegree()
bundles ← {}, bundlesConflict ← {}
for i ***in*** *searchOrder* **do**
  needNew ← True
  for $j = 1$ ***to*** *len(bundles)* **do**
    cnt ← ConflictCnt(bundles[j],F[i])
    if *cnt + bundlesConflict[i]* $\leq K$ **then**
      bundles[j].add(F[i]), needNew ← False
      break
  if *needNew* **then**
    Add $F[i]$ as a new bundle to *bundles*
Output: *bundles*

Algorithm 4: Merge Exclusive Features

Input: $numData$: number of data
Input: F: One bundle of exclusive features
binRanges ← {0}, totalBin ← 0
for f ***in*** F **do**
  totalBin += f.numBin
  binRanges.append(totalBin)
newBin ← new Bin(numData)
for $i = 1$ ***to*** $numData$ **do**
  newBin[i] ← 0
  for $j = 1$ ***to*** *len*(F) **do**
    if $F[j].bin[i] \neq 0$ **then**
      newBin[i] ← F[j].bin[i] + binRanges[j]
Output: $newBin, binRanges$

Рисунок 2.14. Алгоритм EFB [19]

Як показують експерименти, LightGBM досягає майже такої ж точності класифікації, як і інші GBDT алгоритми, але навчається в 20 разів швидше (рис. 2.15).

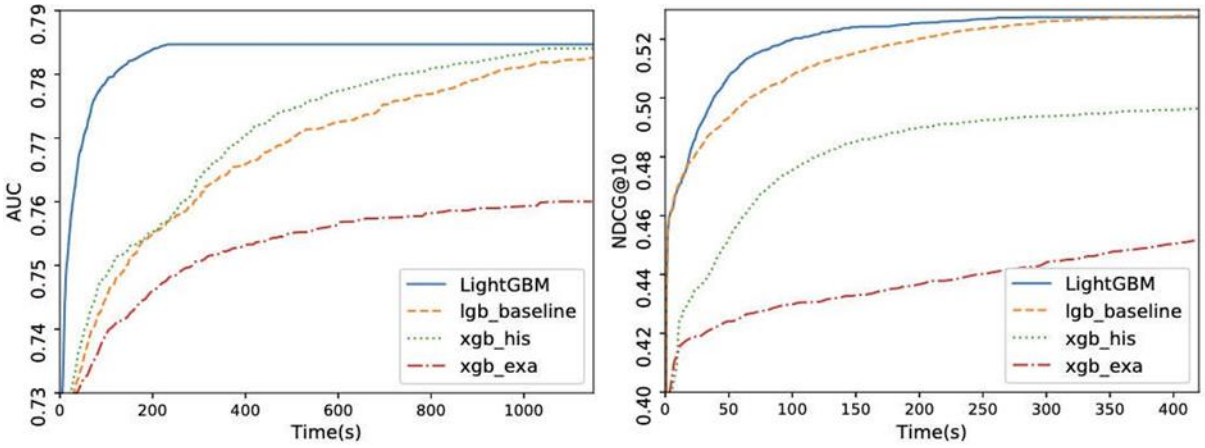


Figure 1: Time-AUC curve on Flight Delay.

Figure 2: Time-NDCG curve on LETOR.

Рисунок 2.15. Порівняння LightGBM з іншими GBDT на основі AUC та NDCG [19]

2.8 Нейронна мережа

Існує безліч алгоритмів навчання, заснованих на концепції нейронних мереж, адаптованих під різні задачі, такі як CNN для розпізнавання зображень та LSTM для роботи з часовими даними. Для задачі класифікації шахрайських транзакцій достатньо класичного алгоритму на основі штучних нейронних мереж – багатошарового перцептрона (MLP) (рис. 2.16).

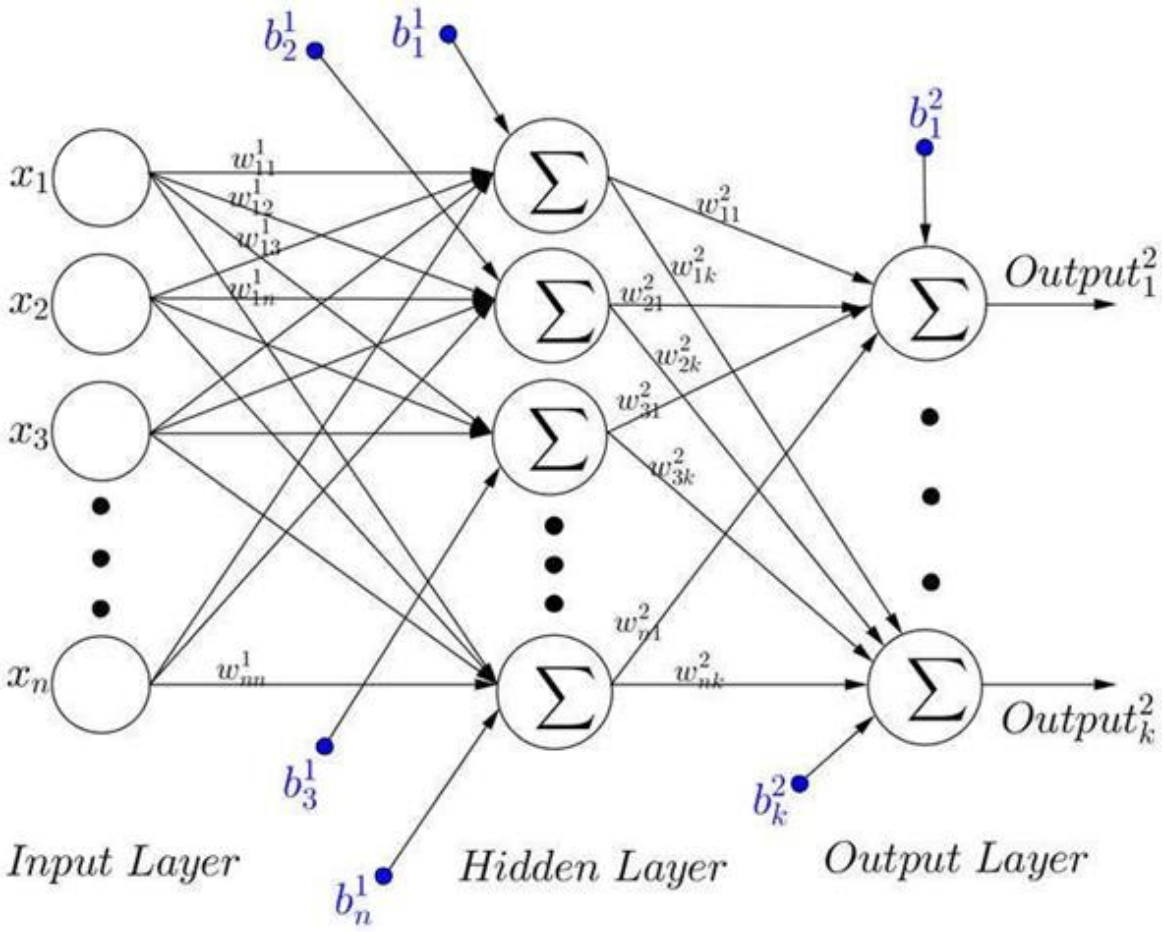


Рисунок 2.16 – Графічне представлення архітектури багатошарової нейронної мережі

MLP можна розглядати як алгоритм апроксимації функції. У теорії, якщо нейронна мережа має достатню кількість шарів, добре спроектована і обмежень у обчислювальних можливостях немає, будь-яка функція може бути змодельована за допомогою MLP.

Процес навчання виглядає так:

– початкові ваги обираються випадково;

– навчальні дані проходять через мережу, отримується перший результат;

– вираховується помилка у обчисленнях, використовуючи певну функцію втрат;

– відштовхуючись від помилки та функції, вираховуються часткові похідні, методом градієнтного спуску вирішується напрямок руху для мінімізації помилки;

– процес повторюється до досягнення мінімальної помилки.

При розробці алгоритму на основі нейронної мережі важливо звертати увагу на підбір гіперпараметрів – змінних, що визначають структуру мережі та процес її навчання. Найважливіші параметри:

– кількість прихованих шарів мережі – слід їх додавати доти, поки помилка на тестових даних не перестане значно зменшуватись;

– виключення (dropout) – деякий відсоток вузлів нейронної мережі можна вимкнути, щоб уникнути перенавчання на вхідних даних;

– швидкість навчання – вища швидкість дозволяє моделі навчатися швидше, але може призвести до "перескакування" оптимуму;

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

2.9 Оцінка якості отриманого результату

Вибір правильних метрик оцінки якості моделі є надзвичайно важливим кроком при роботі з класифікатором, що працює з незбалансованими даними. Звичайні метрики, такі як точність або ROC AUC, часто дають спотворене уявлення про результат і занадто позитивно оцінюють модель. Неправильне розуміння метрик може призвести до некоректно натренованої моделі.

Для сильно незбалансованих даних використовують такі метрики:

– точність (precision);

– повнота (recall);

– F1-міра (F1-score);

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

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

Точність намагається відповісти на питання «який відсоток прикладів, позначених як одиниця, був позначений правильно». Модель, яка не здійснила жодної помилки першого роду (false positive), матиме влучність 1.0.

$$Precision = \frac{TP}{\mathrm{TP} + \mathrm{FP}}$$

Повнота намагається відповісти на питання «який відсоток всіх тестових прикладів класу 1 був позначений правильно». Модель, яка не здійснила жодної помилки другого роду (false negative), матиме повноту 1.0.

$$Recall = \frac{TP}{\mathrm{TP} + \mathrm{FN}}$$

Де:

– TP (true positives) – кількість правильно спрогнозованих прикладів класу 1;
– FP (false positives) – кількість прикладів класу 0, спрогнозованих як клас 1;
– FN (false negatives) – кількість прикладів класу 1, спрогнозованих як клас 0.

Щоб оцінити модель, необхідно аналізувати і точність, і повноту. Модель з високою повнотою, але низькою точністю поверне багато результатів класу 1, але більшість з них будуть неправильними. Модель з високою точністю, але низькою повнотою поверне мало результатів класу 1, хоча більшість з них будуть класифіковані правильно. Усереднену оцінку між точністю та повнотою можна отримати за допомогою міри F1:

$$F1 = 2 \times \frac{Precision \times Recall}{\mathrm{Precision} + \mathrm{Recall}}$$

PR крива представляє собою графічне зображення компромісу між точністю та повнотою для конкретної моделі (рис. 2.17).

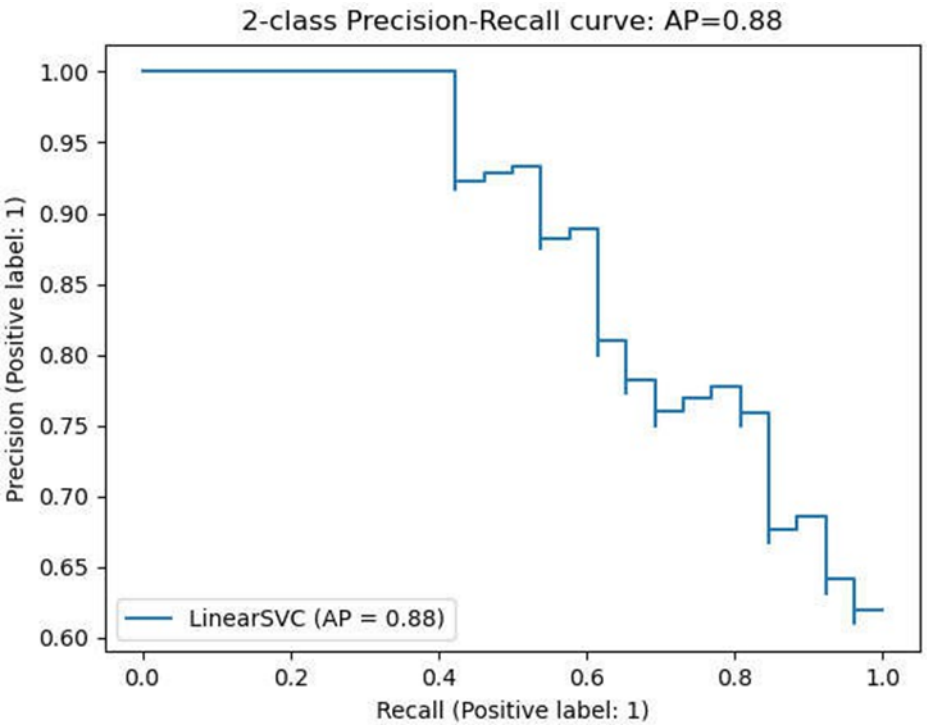


Рисунок 2.17 – ВП крива

Щоб досягти балансу між точністю та повнотою у задачах з незбалансованими даними, необхідно розуміти бізнес-завдання. Що важливіше – ідентифікувати більше прикладів меншого класу або мати менше помилок, але при цьому менше виявлених випадків. Для задачі боротьби з шахрайством, вища повнота важливіша за вищу влучність. Це пояснюється тим, що виявлення більшої кількості шахрайських транзакцій, навіть якщо це включає деяку кількість помилкових спрацювань, є критичним для мінімізації фінансових втрат банківської установи.

2.10 Висновки до розділу 2

У розділі 2 проведено детальний аналіз математичних методів, що використовуються при підготовці даних, побудові та аналізі результатів системи для виявлення та запобігання шахрайству.

Основні висновки розділу:

Алгоритми: Для створення системи виявлення шахрайських транзакцій були обрані три алгоритми:

– логістична регресія;
– LightGBM;
– багатошарова нейронна мережа.

Оцінка якості моделей: Зважаючи на незбалансованість даних, для оцінки якості моделей були обрані наступні метрики:

– точність (precision);
– повнота (recall);
– міра F1 (F1-score).

Балансування даних: Для збалансування даних було обрано поєднання методів:

– передискретизація за допомогою SMOTE;
– субдискретизація за допомогою One-sided Selection.

Застосування цих методів та алгоритмів дозволяє створити ефективну систему для виявлення та запобігання шахрайству, що забезпечує високу точність і надійність моделі навіть при роботі з незбалансованими даними.

РОЗДІЛ 3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНІ

3.1 Навчальна вибірка

Для побудови моделі був обраний датасет, що містить транзакції, проведені за допомогою кредитних карток європейськими користувачами у вересні 2013 року протягом двох днів. Числові дані були трансформовані за допомогою методу головних компонент (PCA) і включають 492 шахрайські транзакції та 284,315 звичайних.

Кожна транзакція має 30 ознак (рис. 3.1) та значення цільової змінної. Ознаки V1-V28 були трансформовані за допомогою PCA з метою анонімізації даних, тоді як ознаки Time (час) та Amount (сума) залишились незмінними (рис. 3.2).

Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
81056.0	-1.066270	0.875439	1.067624	1.446671	0.220971	-0.204483	0.070714	0.532028	-0.519511	...	0.052989	0.260983	-0.112073	0.116830	-0.054372	-0.227716	0.316994	0.070870	8.59	0
86356.0	-0.530874	0.430370	2.440461	0.237931	-0.668095	-0.249169	-0.078105	0.247998	0.525167	...	-0.013615	0.030937	-0.116513	0.387681	-0.330486	0.274465	0.103611	0.123728	3.01	0
137409.0	-0.809497	0.897605	-0.113028	-1.107324	1.153097	0.009334	0.813692	0.082092	0.228997	...	0.092786	0.171141	-0.388818	0.007484	0.141047	0.487137	-0.361147	0.146626	8.43	0
63337.0	1.016673	1.033790	-1.690823	1.316207	1.697548	0.529130	0.332909	0.364305	-0.711904	...	-0.039692	0.095697	-0.093342	-1.345618	0.509974	-0.182770	0.106888	0.071975	1.79	0
63643.0	-2.441716	-2.236823	-0.159494	-1.039938	-1.885127	-0.815075	1.631079	0.197389	-1.390132	...	0.767344	0.541343	1.337561	0.307461	0.151808	-0.345942	0.021726	0.020431	635.00	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
153037.0	1.083711	-1.812910	-3.148691	0.094672	0.473993	-0.487318	1.332628	-0.477055	0.098840	...	0.395588	-0.061858	-0.666866	-1.323066	0.275248	0.178251	-0.196435	-0.001987	570.06	0
40688.0	-1.697503	-2.116995	1.035712	-0.247337	-3.143273	1.513667	2.499200	0.180228	0.029980	...	0.688645	0.567173	1.851676	0.201772	-0.147425	1.427395	-0.323313	0.059751	832.00	0
126353.0	2.026226	0.152387	-1.740240	1.181536	0.637319	-0.611756	0.487696	-0.214799	0.051524	...	0.065237	0.372803	-0.051319	-0.475570	0.465457	-0.479689	-0.021258	-0.078549	1.99	0
154939.0	-1.123193	0.080612	-1.577233	-1.243715	3.076750	3.141938	-0.252252	1.225136	-0.385105	...	-0.082127	-0.505945	0.343995	0.706818	-0.984795	0.132432	-0.248594	0.250852	1.29	0
5441.0	-1.463977	-0.716653	1.638066	-0.173094	1.852650	0.612182	2.494759	-2.224434	2.539634	...	-1.301480	-1.386607	0.205892	-1.378139	-0.643976	0.439735	-2.118056	-1.245902	121.90	0

Рисунок 3.1 – Зовнішній вигляд даних

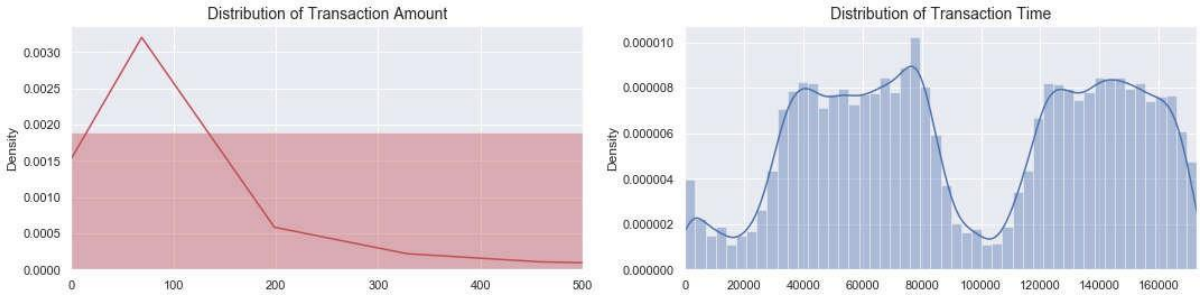


Рисунок 3.2 – Розподіл транзакцій за сумою та часом

Анонімізовані та трансформовані ознаки, а також мала кількість прикладів шахрайських транзакцій значно ускладнюють класифікацію, однак більшість датасетів, які використовуються в банківських установах для навчання моделей, мають саме такий вигляд.

Особливості датасету:

– анонімізація: ознаки V1-V28 були анонімізовані за допомогою PCA для захисту конфіденційності користувачів;

– час: ознака Time відображає кількість секунд, що пройшли з моменту першої транзакції у датасеті;

– сума: ознака Amount показує суму транзакції, що є важливим фактором для виявлення шахрайства;

– незбалансованість: Датасет сильно незбалансований, оскільки шахрайські транзакції становлять лише близько 0.17% від загальної кількості транзакцій.

Ці особливості роблять цей датасет хорошим прикладом для розробки та тестування моделей виявлення шахрайства у банківському секторі, з урахуванням реальних умов та викликів, що стоять перед такими системами.

3.2 Обґрунтування вибору мови та платформи програмування

Для розробки програми, яка вирішує поставлену задачу, були обрані мова Python 3 та середовище Jupyter Notebook.

Переваги мови Python - мова Python, хоча і поступається швидкістю виконання коду таким популярним мовам як Java, C++ та інші, має ряд важливих переваг:

– компактність та простота синтаксису: Python має зрозумілий та лаконічний синтаксис, що дозволяє швидко писати та читати код, зменшуючи час розробки;

– підтримка багатьох парадигм програмування: Python підтримує об'єктно-орієнтоване, функціональне та процедурне програмування, що робить його універсальним інструментом для різних задач;

– велика кількість бібліотек: Python має потужну екосистему бібліотек для машинного навчання та візуалізації даних, що значно спрощує процес розробки та впровадження моделей.

Переваги середовища Jupyter Notebook

Jupyter Notebook – це середовище розробки, яке дозволяє виконувати код покроково, при цьому порядок кроків може змінюватися, а результати виконання кожного кроку зберігаються глобально. Це середовище є надзвичайно зручним для візуалізацій та машинного навчання, адже можна швидко перетреновувати моделі або додавати нові, використовуючи попередній результат або попередньо оброблені дані (рис. 3.3).

Для розробки програмного продукту були використані наступні бібліотеки:

– Matplotlib та Seaborn: для візуалізації результатів. Вони дозволяють створювати різноманітні графіки та діаграми для аналізу даних;

– Pandas: для зручної роботи з таблицями та обробки CSV файлів. Ця бібліотека забезпечує потужні інструменти для маніпуляції даними;

– Numpy: для швидкої та ефективної роботи з числовими масивами. Numpy забезпечує багатовимірні масиви та математичні функції для роботи з ними;

– Imblearn: для оверсамплінгу та андерсемплінгу, побудови пайплайну моделі. Ця бібліотека спеціалізується на методах обробки незбалансованих даних;

– Sklearn: для обчислення метрик, пошуку оптимальних параметрів та побудови моделей. Sklearn є стандартною бібліотекою для машинного навчання в Python;

– Tensorflow/Keras: для побудови нейронної мережі. Keras є зручним інтерфейсом для Tensorflow, однієї з найдосконаліших бібліотек, розроблених Google для машинного навчання;

– LightGBM: для побудови моделі LightGBM. Ця бібліотека забезпечує швидке та ефективне навчання моделей градієнтного бустингу.

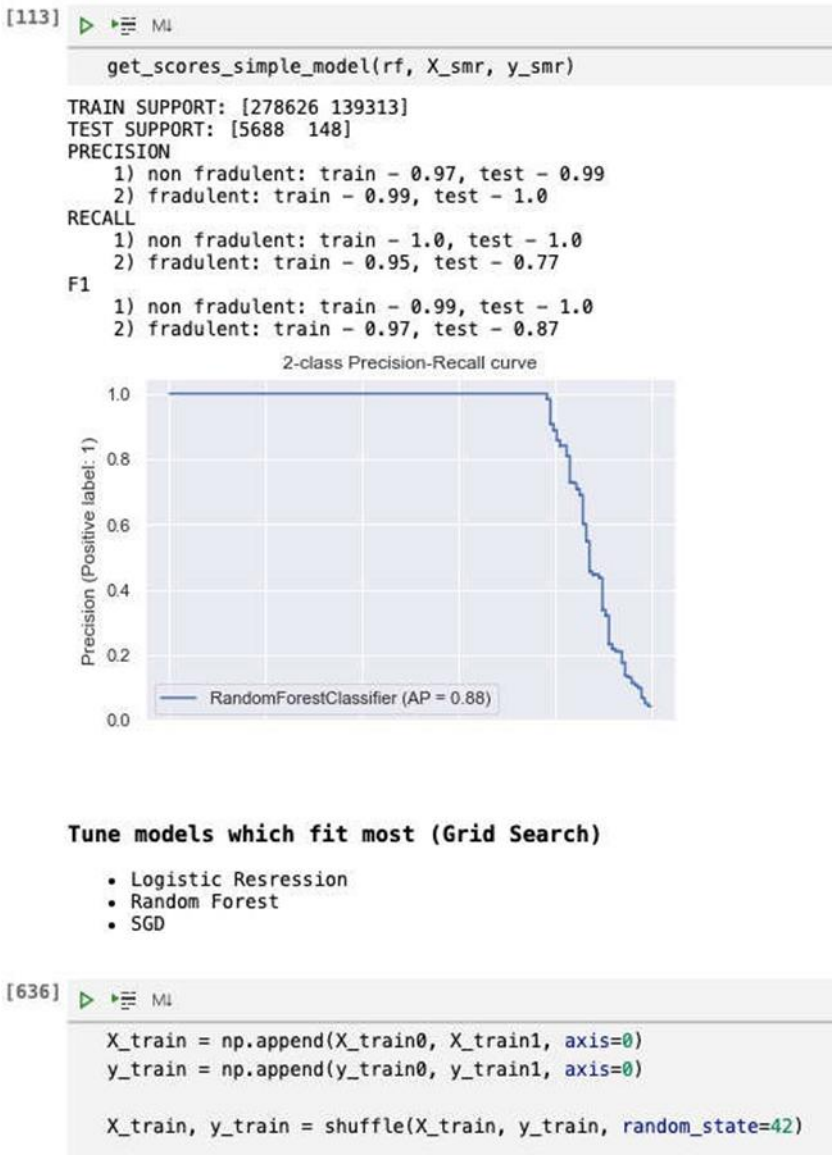


Рисунок 3.3 – Jupyter Notebook та Python 3

Вибір Python та Jupyter Notebook обґрунтований їхніми перевагами в простоті використання, багатій екосистемі бібліотек та зручності для інтерактивної розробки та аналізу моделей машинного навчання.

3.3 Результати роботи

Для побудови системи запобігання шахрайству, як зазначено в розділі 2, були обрані кілька відомих архітектур моделей машинного навчання: логістична регресія, LightGBM та нейронна мережа. Ці архітектури були використані як основа досліджень. Також проведено експерименти з різними методами оверсемплінгу та андерсемплінгу, які відіграли важливу роль у підвищенні метрик моделей – влучності, повноти та F1-міри.

На початковому етапі модель логістичної регресії була натренована на необроблених даних, що дало погані результати (рис. 3.4).

```
clf_lgr = LogisticRegression(max_iter=10000,
                             random_state=42,
                             class_weight='balanced').fit(X_train0, y_train0)

y_pred_lgr = clf_lgr.predict(X_test0)
y_pred_lgr_train = clf_lgr.predict(X_train0)

print('acc train: ', clf_lgr.score(X_train0, y_train0),\
      '\nacc test: ', clf_lgr.score(X_test0, y_test0),\
      '\nrecall train: ', recall_score(y_train0, y_pred_lgr_train),\
      '\nrecall test: ', recall_score(y_test0, y_pred_lgr),
      '\nprecision train: ', precision_score(y_train0, y_pred_lgr_train),\
      '\nprecision test: ', precision_score(y_test0, y_pred_lgr))
```

```
acc train:  0.9611705406396542
acc test:  0.9604648713177206
recall train:  0.916478553047404
recall test:  0.9183673469387755
precision train:  0.039333462507266034
precision test:  0.038560411311053984
```

Рисунок 3.4 – Результати тренування логістичної регресії на необроблених даних

Хоча точність моделі була надвисокою (0.96), влучність становила лише 3%, що робить модель непридатною для використання. Це показує, що логістична регресія не є оптимальним рішенням для поставленої задачі.

Для покращення результатів була побудована система підбору оптимальних параметрів на основі GridSearchCV та комбінації андерсемплінгу та оверсемплінгу (SMOTE + RandomUndersampling) (рис. 3.5).

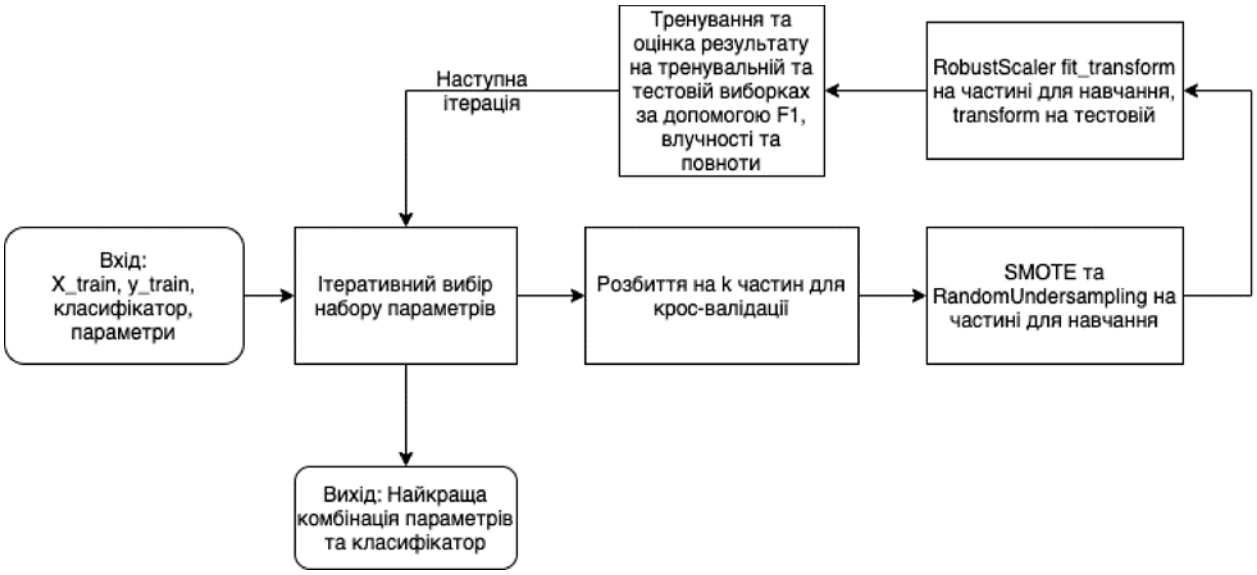


Рисунок 3.5 – Система підбору параметрів для моделей

Процес підбору параметрів для логістичної регресії показаний на рис. 3.6.

```
log_reg_params = {'clf__penalty': ['l2'],
                  'clf__C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],
                  'clf__max_iter': [10000],
                  'clf__solver': ['lbfgs'],
                  'clf__class_weight': ['balanced']}

clf_lgr = LogisticRegression()
```

```
gs_lg, gs_lg_report = build_smote_pipeline(clf_lgr, X_train3, y_train3, X_test3, y_test3, log_reg_params)
```

```
[CV 2/2; 3/7] END clf__C=0.1, clf__class_weight=balanced, clf__max_iter=10000, clf__penalty=l2, clf__solver=lbfgs; accuracy: (train=0.990, test=0.98
ecall: (train=0.889, test=0.874) total time=   1.5s
[CV 1/2; 4/7] START clf__C=1, clf__class_weight=balanced, clf__max_iter=10000, clf__penalty=l2, clf__solver=lbfgs
[CV 1/2; 4/7] END clf__C=1, clf__class_weight=balanced, clf__max_iter=10000, clf__penalty=l2, clf__solver=lbfgs; accuracy: (train=0.991, test=0.992)
all: (train=0.916, test=0.861) total time=   2.3s
[CV 2/2; 4/7] START clf__C=1, clf__class_weight=balanced, clf__max_iter=10000, clf__penalty=l2, clf__solver=lbfgs
[CV 2/2; 4/7] END clf__C=1, clf__class_weight=balanced, clf__max_iter=10000, clf__penalty=l2, clf__solver=lbfgs; accuracy: (train=0.990, test=0.989)
```

Рисунок 3.6 – Підбір параметрів на прикладі логістичної регресії

Найкращі параметри:

```
{
    'clf__C': 1000,
        'clf__class_weight': 'balanced',
        'clf__max_iter': 10000,
        'clf__penalty': 'l2',
        'clf__solver': 'lbfgs'
}
```

Метрики для отриманого класифікатора містяться на (рис. 3.7).

```
recall_score(y_train3, gs_lg.predict(X_train3)), recall_score(y_test3, gs_lg.predict(X_test3))
```

```
(0.8885017421602788, 0.9191176470588235)
```

```
precision_score(y_test3, gs_lg.predict(X_test3))
```

```
0.13691128148959475
```

```
plot_precision_recall_curve(gs_lg, X_test3, y_test3)
```

```
<sklearn.metrics._plot.precision_recall_curve.PrecisionRecallDisplay at 0x7ff8d27f61c0>
```

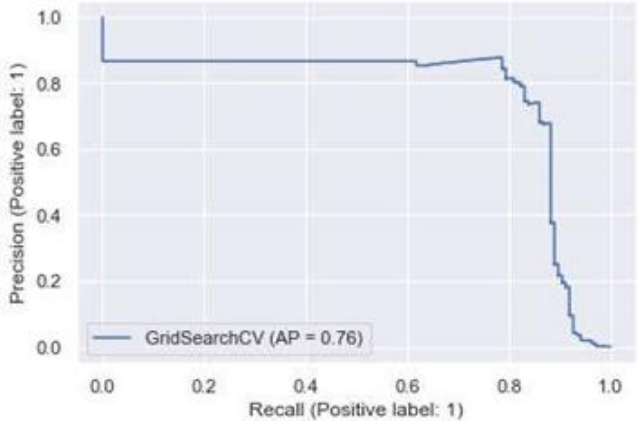


Рисунок 3.7 – Точність, повнота та F1-міра на тренувальній та тестовій вибірках для логістичної регресії

Попри певне покращення, логістична регресія, на жаль, не підходить як єдиний класифікатор для поставленої задачі. Адекватний результат можна отримати тільки змістивши ймовірність, необхідну для прийняття рішення, з 0.5 до 0.99 (рис. 3.8).

```
probs_lg = gs_lg.predict_proba(X_test3)
```

```
def to_labels(pos_probs, threshold):
    return (pos_probs >= threshold).astype('int')
```

```
p = to_labels(probs_lg[:, 1], 0.999)
precision_score(y_test3, p), recall_score(y_test3, p)
```

```
(0.7371794871794872, 0.8455882352941176)
```

Рисунок 3.8 – Влучність та повнота для логістичної регресії при необхідній вірогідності для прийняття рішення 0.99

Процес підбору параметрів для LightGBM наведений на рис. 3.9.

```
lightgbm_params = {'clf__max_depth': [10],
                   'clf__n_estimators': [200],
                   'clf__boosting_type': ['goss'],
                   'clf__learning_rate': [0.1],
                   'clf__class_weight': ['balanced'],
                   # 'clf__colsample_bytree': [0.5, 0.8],
                   'clf__num_leaves': [70],
                   # 'clf__reg_alpha': [1.1],
                   # 'clf__reg_lambda': [1.1],
                   # 'clf__min_split_gain': [0.1, 0.2],
                   # 'clf__subsample': [0.4]
                   'clf__feature_fraction': [0.2, 0.3]
                   }

clf_gbm = LGBMClassifier()

gs_gbm2, gs_gbm_report2 = build_smote_pipeline(clf_gbm, np.append(X_train3, X_validate3, axis=0), np.append(y_train3, y_validate3, axis=0), X_test3, y_test3, lightgbm_param
```

Рисунок 3.9 – Підбір параметрів на прикладі LightGBM

Метрики для найкращого LightGBM класифікатора представлені на рис. 3.10.

```
recall_score(y_train3, gs_gbm2.predict(X_train3)), recall_score(y_test3, gs_gbm2.predict(X_test3))
```

```
(1.0, 0.8823529411764706)
```

```
precision_score(y_train3, gs_gbm2.predict(X_train3)), precision_score(y_test3, gs_gbm2.predict(X_test3))
```

```
(0.9318181818181818, 0.7643312101910829)
```

```
f1_score(y_test3, gs_gbm2.predict(X_test3))
```

```
0.8191126279863482
```

Рисунок 3.10 – Точність, повнота та F1-міра на тренувальній та тестовій вибірках для LightGBM

Отримана модель показує непогані результати, але перенавчання залишається проблемою, особливо на прикладах класу шахрайських транзакцій (повнота = 1.0). Це означає, що бустингові та ансамблеві методи на основі дерев рішень добре працюють для цього датасету, але потрібно уникати перенавчання.

Було побудовано нейронну мережу, архітектуру якої зображено на рис. 3.11, але результати були посередні (рис. 3.12).

Проаналізувавши попередні результати, було вирішено створити ансамбль класифікаторів, включаючи алгоритми на основі дерев рішень. Попередній досвід з LightGBM дозволив визначити параметри класифікаторів без перебору GridSearchCV. До ансамблю увійшли такі моделі:

- LightGBM;
- ExtraTreesClassifier;
- CatBoostClassifier;
- RandomForestClassifier;
- Logistic Regression як мета-модель.

Схема побудови ансамблю з базових моделей та мета-моделі out-of-fold stacking зображена на рис. 3.13.

Model: "sequential_12"

Layer (type)	Output Shape	Param #
dense_60 (Dense)	(None, 256)	7936
batch_normalization_48 (Batc	(None, 256)	1024
dropout_41 (Dropout)	(None, 256)	0
dense_61 (Dense)	(None, 256)	65792
batch_normalization_49 (Batc	(None, 256)	1024
dropout_42 (Dropout)	(None, 256)	0
dense_62 (Dense)	(None, 256)	65792
batch_normalization_50 (Batc	(None, 256)	1024
dropout_43 (Dropout)	(None, 256)	0
dense_63 (Dense)	(None, 256)	65792
batch_normalization_51 (Batc	(None, 256)	1024
dense_64 (Dense)	(None, 1)	257

Рисунок 3.11 – Шари нейронної мережі

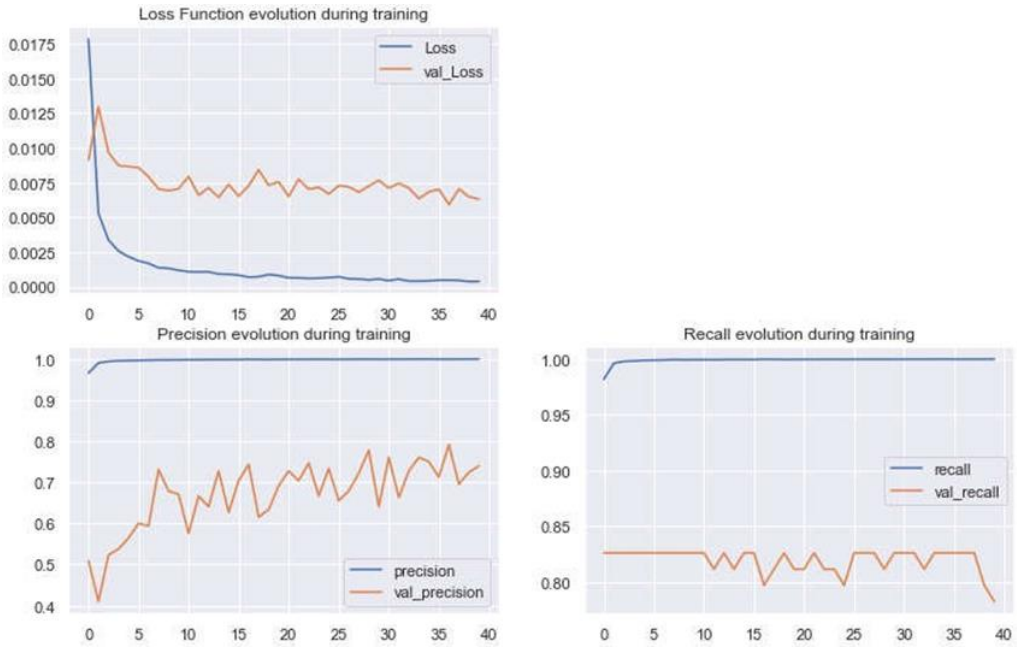


Рисунок 3.12 – Навчання нейронної мережі з рис. 3.11

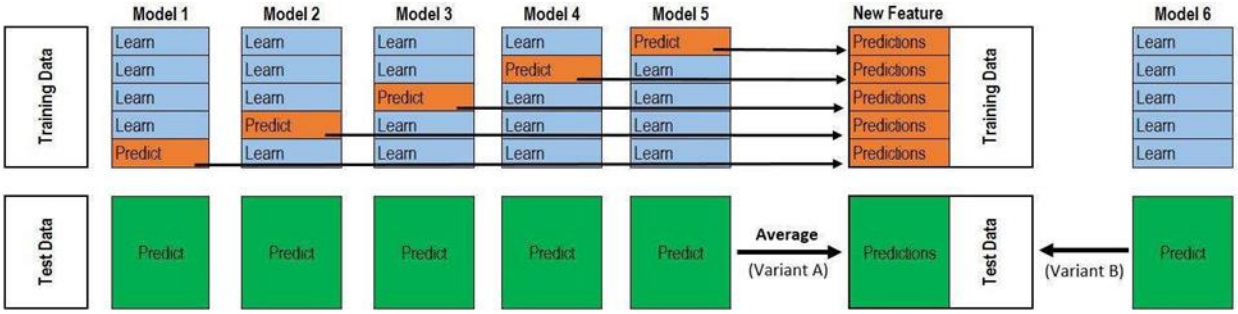


Рисунок 3.13 – OOF-Stacking

Параметри базових моделей показані на рис. 3.14.

```
et_params = { # ExtraTreesClassifier
    'n_jobs': 16,
    'n_estimators': 200,
    'max_features': 0.3,
    'max_depth': 10,
    'min_samples_leaf': 2,
}

rf_params = { # RandomForestClassifier
    'n_jobs': 16,
    'n_estimators': 200,
    'max_features': 0.2,
    'max_depth': 10,
    'min_samples_leaf': 2,
}

catboost_params = { # CatBoostClassifier
    'iterations': 200,
    'learning_rate': 0.5,
    'depth': 3,
    'l2_leaf_reg': 40,
    'bootstrap_type': 'Bernoulli',
    'subsample': 0.7,
    'scale_pos_weight': 5,
    'eval_metric': 'F1',
    'od_type': 'Iter',
    'allow_writing_files': False
}

lightgbm_params = { # LightGBM Classifier
    'clf__boosting_type': 'goss',
    'n_estimators':200,
    'learning_rate':0.1,
    'num_leaves':70,
    'subsample':0.9,
    'max_depth':10,
    'class_weight': 'balanced',
    'feature_fraction': 0.3
}
```

Рисунок 3.14 – Параметри базових моделей у ансамблі

Отримані результати (рис. 3.15) виявилися найкращими з усіх розроблених моделей і є достатньо гарними для використання у реальному проекті, відрегулювавши поріг прийняття рішень залежно від потреби.

```
print('Ensemble recall (0.5): ', recall_score(y_test3, preds_es))
print('Ensemble precision (0.5): ', precision_score(y_test3, preds_es))
print('Ensemble F1 (0.5):', f1_score(y_test3, preds_es))
```

```
Ensemble recall (0.5):  0.8014705882352942
Ensemble precision (0.5):  0.9478260869565217
Ensemble F1 (0.5): 0.8685258964143425
```

```
preds_es2 = logistic_regression_es.predict_proba(x_test_es)[:, 1]
preds_es2 = to_labels(preds_es2, 0.3)

print('Ensemble recall (0.3): ', recall_score(y_test3, preds_es2))
print('Ensemble precision (0.3):', precision_score(y_test3, preds_es2))
print('Ensemble F1 (0.3):', f1_score(y_test3, preds_es2))
```

```
Ensemble recall (0.3):  0.8235294117647058
Ensemble precision (0.3): 0.9333333333333333
Ensemble F1 (0.3): 0.8749999999999999
```

```
preds_es2 = logistic_regression_es.predict_proba(x_test_es)[:, 1]
preds_es2 = to_labels(preds_es2, 0.1)

print('Ensemble recall (0.1): ', recall_score(y_test3, preds_es2))
print('Ensemble precision (0.1):', precision_score(y_test3, preds_es2))
print('Ensemble F1 (0.1):', f1_score(y_test3, preds_es2))
```

```
Ensemble recall (0.1):  0.8382352941176471
Ensemble precision (0.1): 0.912
Ensemble F1 (0.1): 0.8735632183908045
```

Рисунок 3.15 – Метрики для ансамблю з різними порогами прийняття рішень

3.4 Висновки до розділу 3

У розділі 3 було створено програмну реалізацію для задачі виявлення шахрайства методами штучного інтелекту. Розглянуто та досліджено різні підходи до

побудови такої системи: статистична модель логістичної регресії, нейронна мережа та бустинг LightGBM. На основі аналізу їх ефективності за трьома показниками – точність, повнота та F1-міра – було визначено, що для тестових даних найкращі результати демонструє LightGBM – бустинг на основі дерев рішень.

Модель LightGBM показала високу точність, але сильно перенавчалась навіть попри використання великої кількості класифікаторів та невеликої їх глибини. Для подолання цієї проблеми був побудований ансамбль OOF-stacking, що включає алгоритми на основі дерев рішень. В результаті, отримані показники досягли високих значень: влучність – 93%, повнота – 82%.

Основні висновки:

– логістична регресія не показала достатньо високих результатів для виявлення шахрайства, навіть після підбору оптимальних параметрів;

– нейронна мережа продемонструвала посередні результати і не змогла забезпечити належний рівень ефективності для даної задачі;

– LightGBM показав найкращі результати серед окремих моделей, проте був схильний до перенавчання;

– ансамбль OOF-stacking з алгоритмів на основі дерев рішень значно покращив загальну ефективність системи, досягнувши високих показників влучності та повноти.

Таким чином, використання ансамблів моделей є ефективним підходом для задачі виявлення шахрайства, забезпечуючи високу точність та надійність класифікації у реальних умовах.

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У цьому розділі виконується оцінка ключових характеристик програмного продукту, розробленого для створення системи моніторингу фінансових операцій для виявлення та запобігання шахрайству. Тут проведено аналіз різних варіантів реалізації модуля з метою вибору оптимального, враховуючи як економічні фактори, так і характеристики продукту, що впливають на його продуктивність та сумісність з апаратним забезпеченням. Для цього використано метод функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) є технологією, яка дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. ФВА проводиться з метою виявлення можливостей зниження витрат за рахунок ефективніших варіантів виробництва та покращення співвідношення між споживчою вартістю виробу та витратами на його виготовлення. Для проведення аналізу використовується економічна, технічна та конструкторська інформація.

4.1 Визначення задачі проектування

У роботі застосовується метод ФВА для проведення техніко-економічного аналізу розробки системи виявлення та запобігання банківському шахрайству. Оскільки рішення щодо проектування та реалізації компонентів впливають на всю систему, кожна окрема підсистема повинна її задовольняти. Тому фактичний аналіз полягає в оцінці функцій програмного продукту, призначеного для збору, обробки та аналізу даних про фінансові операції.

Технічні вимоги до програмного продукту включають наступне:

– робота на персональних комп'ютерах зі стандартним набором компонентів;
– зручний та зрозумілий інтерфейс для користувача;

– швидка обробка даних і доступ до інформації в реальному часі;
– можливість легкого масштабування та технічного обслуговування;
– мінімальні витрати на впровадження програмного продукту.

4.2 Обґрунтування функціональних можливостей програмного продукту

Головна функція – розробка програмного продукту, який вирішує задачу моніторингу фінансових операцій та виявлення шахрайства. Виходячи з цієї функції, можна виділити наступні:

– вибір мови програмування;
– вибір фреймворку машинного навчання;
– вибір середовища розробки.

Кожна з цих функцій має декілька варіантів реалізації:

Функція вибору мови програмування:

– Python;
– C++.

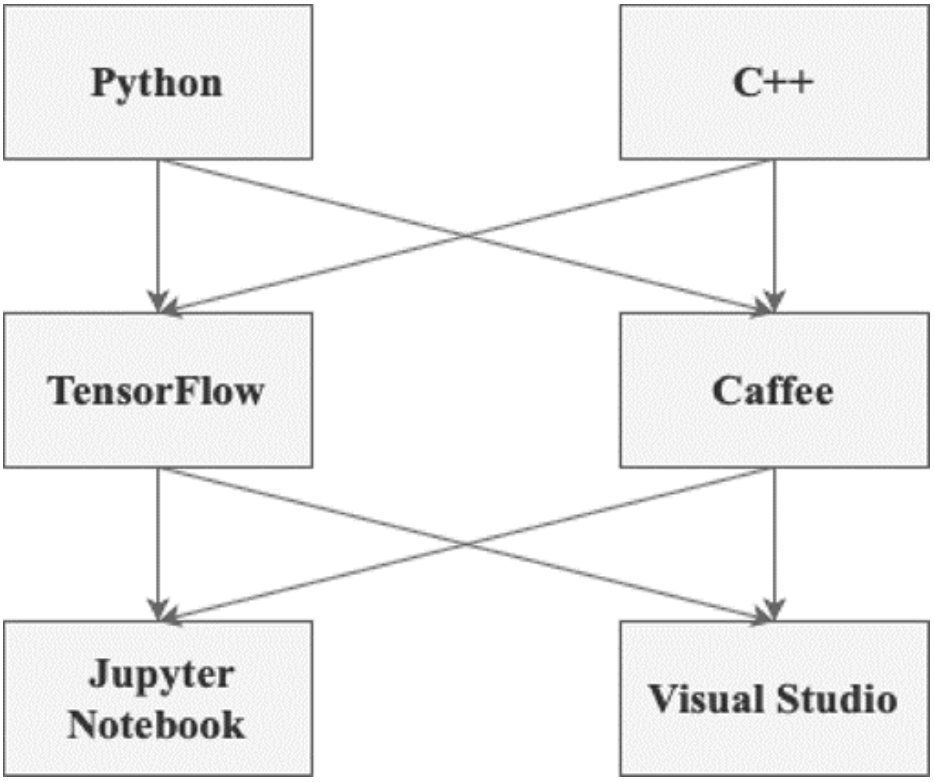
Функція вибору фреймворку машинного навчання:

– Tensorflow;
– Caffe .

Функція вибору середовища розробки:

– Jupyter Notebook;
– Visual Studio.

Візуалізація процесу вибору мови програмування, фреймворку машинного навчання, та середовища розробки (рис. 4.1).



Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 4.1).

4.3 Визначення системи параметрів програмного забезпечення

На основі даних, розглянутих вище, визначаються основні параметри вибору, які будуть використані для розрахунку коефіцієнта технічного рівня. Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

- X1 – швидкодія мови програмування;
- X2 – об’єм пам’яті для обчислень та збереження даних;
- X3 – час навчання даних;
- X4 – потенційний об’єм програмного коду.

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника та умов, що характеризують експлуатацію ПП.

Таблиця 4.4 - Основні параметри ПП

Назва параметра	Умовні позначення	Одиниці виміру	Гірші	Середні	Кращі
Швидкодія мови програмування	X1	оп/мс	10000	14000	19000
Об’єм пам’яті	X2	Мб	420	128	64
Час попередньої обробки даних	X3	мс	4	3	2
Потенційний об’єм програмного коду	X4	кількість рядків коду	4000	2500	1000

4.4 Аналіз експертної оцінки параметрів

Після ретельного обговорення та аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який дає найбільш точні результати при моніторингу фінансових операцій і виявленні шахрайських дій.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із семи людей. Визначення коефіцієнтів значимості передбачає:

– визначення рівня значимості параметра шляхом присвоєння різних рангів;
– перевірку придатності експертних оцінок для подальшого використання;
– визначення оцінки попарного пріоритету параметрів;
– обробку результатів та визначення коефіцієнту значимості.

У результаті порівняння було визначено ступінь переваги одного параметра над іншим. На основі числових оцінок переваги складено матрицю, що містить числові значення для кожної пари параметрів. Відносні оцінки параметрів розраховувалися

кілька разів до досягнення стабільності результатів (різниця між оцінками на наступних кроках не перевищувала 2%).

Таблиця 4.5 - Попарне порівняння параметрів.

Параметри	Експерти	1	2	3	4	5	6	Кінцева оцінка	Числове значення
X1 і X2		>	>	<	<	>	>	>	1.5
X1 і X3		<	>	<	=	<	<	<	0.5
X1 і X4		>	>	<	=	<	=	>	1.5
X2 і X3		<	<	<	<	<	<	<	0.5
X2 і X4		<	<	<	<	<	<	<	0.5
X3 і X4		>	<	>	>	<	>	>	1.5

4.5 Оцінка якості реалізації функцій

Після визначення коефіцієнтів значимості для кожного параметра було проведено аналіз рівня якості різних варіантів виконання основних функцій програмного продукту.

Абсолютні значення параметрів, таких як обсяг пам'яті, час попередньої обробки даних та потенційний обсяг програмного коду, були порівняні з технічними вимогами умов функціонування програмного продукту. Швидкість роботи мови програмування вибиралася таким чином, щоб відповідати не найгіршим показникам.

Для кожного варіанта реалізації функцій програмного продукту були визначені коефіцієнти технічного рівня, що дозволяє оцінити загальний рівень якості. Найвищий коефіцієнт технічного рівня був виявлений у першому варіанті, що свідчить про його перевагу над іншими.

Таблиця 4.6 – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	**Варіант реалізації функції**	**Параметри**	**Абсолютне значення параметра**	**Бальна оцінка параметра**	**Коефіцієнт вагомості параметра**	**Коефіцієнт рівня якості**
F1	А	X1	10000	6	0.25	1.47
F2	А	X2	64	5	0.23	0.98
F2	Б	X2	128	2	0.32	0.40
F3	А	X3	1000	8	0.20	2.14

За даними з таблиці 4.6, рівень якості кожного з варіантів визначається за сумою коефіцієнтів рівня якості:

$$КК1=1.47+0.98+2.14=4.59$$

$$КК2=1.47+0.40+2.14=4.01$$

Перший варіант має найбільший коефіцієнт технічного рівня, отже, є кращим.

4.6 Тестування розробленого програмного забезпечення

Тестування є одним з найважливіших етапів при розробки програмного забезпечення. Метою тестування є виявлення помилок або некоректну поведінку програмної системи при тих чи інших обставин, якої не описано у специфікації. Якісне тестування є одним з критеріїв успішності системи. Відсутність дефектів та несправності у роботі програмного забезпечення є задачею як тестувальників так і розробників програмної системи.

Серед основних видів тестування, які мають бути проведені слід відмітити модульне тестування (Unit - тестування), інтеграційне тестування та ad-hoc тестування. Ці види тестування найбільш розповсюджені та широко використовуються у сучасних програмних системах.

Ad-hoc тестування – при такому тестуванні не потрібно надавати точну документацію та тест-кейси. Тестувальники покладається на своє знання програмного забезпечення та намагається знайти дефекти у системи випадковим мануальним шляхом. Цей вид тестування зазвичай найбільш ефективний у пошуку дефектів та несправності так як не існує чіткого сценарію для тестувальника, в основному це проходить випадковим тестуванням функцій програмної системи.

Модульне тестування (Unit - тестування) – це вид тестування при якому складаються так названі Unit -тести, які допомагають протестувати внутрішній код та модулі програмної системи. Зазвичай такі тести знаходяться у вигляді коду у програмному забезпеченні та дозволяє розробнику виявити помилку до того, як буде проведен основний етап тестування. Unit -тестами найчастіше покривають функції, методи та класи програмної системи, яке допоможе у дальнішому переконатися розробнику що змінення коду саме у цей функції не змінить основну поведінку та логіку для чого була ця функція створена. Для написання таких тестів слід переконатися, що програмна система відповідає усім принципам об’єктно -орієнтованого програмування та має малу зчепленість між компонентами та модулями програмної системи. Також Unit -тест допомагають виявити міста у коді де була порушена основна архітектура та потребує рефакторінгу. Дані та середовища у цьому виду тестування є несправжніми та по суті являються так названими заглушками або мок об’єктами.

Інтеграційне тестування – також являється одним з видів тестування який розміщується у коді. Але на відміну від модульного тестування яке тестується у штучному середовищі, з несправжніми даними. Цей вид тестування використовує справжнє середовище системи, а саме створює базу даних та тестує усі функції у ньому. Найпоширенішим використанням такого виду тестування є операції додавання, оновлення та видалення даних з бази даних так як за допомогою Unit -

тестування цього не зробити. Також тестуються взаємодія окремих модулів системи між собою та інтеграцію цих модулів у єдину систему. Таке тестування допомагає системам виявити проблеми при інтеграції з реальними даними та покращує покриття коду тестами так як інтеграційні тести покривають взаємодію купу компонентів системи за раз. Недоліком цього виду тестування є час виконання цих тестів та складність написання.

В результаті були розроблені та написані Unit та інтеграційні тести для програмного забезпечення. Більшу частину кода було покрито Unit тестами та інтеграційними тестами. Також було проведено ad-hoc тестування у якому був протестований основний функціонал програмної системи.

4.7 Економічний аналіз варіантів розробки програмного забезпечення

Щоб визначити вартості розробки програмного продукту спочатку проведемо розрахунок трудомісткості. Всі варіанти включають у себе два окремих завдання:

– розробка проекту програмного продукту;
– розробка програмної оболонки.

Завдання 1 за ступенем новизни відноситься до групи А, а завдання 2 – до групи Б. За складністю алгоритми, що використовуються у завданні 1, належать до групи 1, а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовується довідкова інформація, тоді як завдання 2 використовує інформацію у вигляді даних.

Трудомісткість кожного завдання обчислюється з урахуванням норм часу, поправочних коефіцієнтів на вид нормативно-довідкової інформації, складність контролю вхідної та вихідної інформації, рівень мови програмування, використання стандартних модулів та стандартного математичного забезпечення.

Складаючи трудомісткість відповідних завдань для кожного з варіантів реалізації програми, ми отримуємо такі результати:

– варіант I: 1328.64 людино-годин;
– варіант II: 1345.52 людино-годин.

Далі визначається середня зарплата за годину, виходячи з окладу програмістів та аналітиків, і розраховується загальна заробітна плата за кожним варіантом.

Додатково враховуються відрахування на єдиний соціальний внесок (22%).

Також проводяться розрахунки витрат на оплату машино-годин, з урахуванням заробітної плати програмістів, амортизаційних відрахувань, витрат на ремонт і профілактику, а також витрат на електроенергію. Накладні витрати складають 67% від заробітної плати.

Вартість розробки ПП за варіантами становить:

– варіант I: 621223.6 грн;
– варіант II: 629116.14 грн.

4.8 Вибір найкращого варіанту програмного забезпечення за техніко-економічними показниками

Для вибору оптимального варіанту програмного продукту проводиться оцінка техніко-економічного рівня кожного варіанта. Враховуються коефіцієнти технічного рівня, отримані в попередньому розділі, та загальна вартість розробки.

Перший варіант реалізації програми показав кращий коефіцієнт техніко-економічного рівня, що свідчить про його ефективність. Цей варіант реалізації програмного продукту має такі параметри:

– мова програмування – Python;
– застосування моделей з великою ємністю;
– використання стандартного інтерфейсу візуалізації;

– висока швидкість розробки.

Цей варіант забезпечує користувачу зручний інтерфейс, високу продуктивність та швидкодію, що є важливими факторами для успішного функціонування системи моніторингу фінансових операцій та виявлення шахрайства.

4.9 Висновки до розділу 4

Проведено детальний функціонально-вартісний аналіз програмного продукту. Визначено та оцінено ключові функції програмного продукту, а також визначено параметри, що характеризують його продуктивність. Експерти оцінили ці параметри, і було проведено аналіз якості різних варіантів реалізації функцій. Крім того, здійснено економічний аналіз варіантів розробки, який включав трудомісткість, витрати на заробітну плату та інші супутні витрати. На основі цього всебічного аналізу був обраний оптимальний варіант реалізації програмного продукту.

ВИСНОВКИ

У роботі була визначена актуальність та необхідність дослідження способів створення системи з виявлення та запобігання банківському шахрайству. Проведено аналіз труднощів роботи з даними, що містять шахрайські та звичайні банківські транзакції, зокрема висока незбалансованість даних та низька кількість прикладів шахрайства, що значно ускладнює побудову моделі машинного навчання. Було досліджено методи боротьби з зазначеними проблемами та різні архітектури моделей, які підходять для створення системи з виявлення шахрайства. Визначено методи оцінки для складених моделей машинного навчання.

У практичній частині на прикладі обраного датасету побудовано такі моделі:

– логістична регресія;
– LightGBM;
– нейронна мережа;
– ансамбль з LightGBM, ExtraTreesClassifier, CatBoostClassifier, RandomForestClassifier, а також логістичної регресії.

Результати досліджень показали, що ансамбль має найкращі показники, досягаючи влучності 93% та повноти 82%. Це демонструє, що використання ансамблів моделей є ефективним підходом для задачі виявлення шахрайства, забезпечуючи високу точність та надійність класифікації в реальних умовах.

Таким чином, дана робота показує важливість та ефективність застосування сучасних методів машинного навчання та функціонально-вартісного аналізу для розробки системи виявлення та запобігання банківському шахрайству. Проведений аналіз та отримані результати можуть бути використані для подальших досліджень та впровадження таких систем у банківському секторі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Г.В. Табунщик, Р.К. Кудерметов, Т.І. Брагіна Інженерія якості програмного забезпечення. // https://library.kre.dp.ua/Books/2-4%20kurs/Основи%20програмної%20інженерної/Інженерія%20якості%20ПЗ.pdf , (дата звернення: 20.05.2024)..

2. Грешилов А. А. Математичні методи прийняття рішень / А. А. Грешилов. - М., 2014. - 648с. (дата звернення: 17.05.2024).

3. Лотов А. В. Багатокритеріальні задачі прийняття рішень. Метод досяжних цілей / А. В. Лотов, І. І. Поспєлова. - М.: МАКС Прес, 2008. - 197 с. (дата звернення: 02.05.2024).

4. Кіні Р. Л. Ухвалення рішень за багатьох критеріїв: уподобання та заміщення /Р. Л. Кіні, Х. Райфа – М., 1981. - 560 с. (дата звернення: 09.06.2024).

5. Айзерман М. А. Вибір варіантів. Основи теорії / М. А. Айзерман, Ф. Т. Алескеров - М. : Наука, 1990. - 237 с. (дата звернення: 09.05.2024).

6. Березовський Б. А. Задача найкращого вибору / Б. А. Березовський, О. В. Гнєдін - М., 1984. - 196 с. (дата звернення: 01.06.2024).

7. The multi-faceted threat of fraud. KPMG. URL: https://home.kpmg/xx/en/home/insights/2019/05/the-multi-faceted-threat-of-fraud-arebanks-up-to-the-challenge-fs.html (дата звернення: 31.05.2024).

8. Payment Fraud Statistics, Trends & Forecasts (2020). Merchant Savvy. URL: https://www.merchantsavvy.co.uk/payment-fraud-statistics/ (дата звернення: 18.06.2024).

9. European Central Bank. URL: https://www.ecb.europa.eu/pub/pdf/cardfraud/ecb.cardfraudreport202008~521edb602b.en.pdf (дата звернення: 09.06.2024).

10. Stross R. $9 Here, 20 Cents There and a Credit-Card Lawsuit (Published 2010). The New York Times. URL:

https://www.nytimes.com/2010/08/22/business/22digi.html?_r=1&src=me&ref=business (дата звернення: 11.06.2024).

11. Identity theft and scams: how to get your money back. URL: https://www.moneyadviceservice.org.uk/en/articles/identity-theft-and-scamswhat-you-are-liable-for (дата звернення: 18.06.2024).

12. Credit Card Fraud Detection. Kaggle: Your Machine Learning and Data Science Community. URL: https://www.kaggle.com/mlg-ulb/creditcardfraud (дата звернення: 10.06.2024).

13. Learning from Imbalanced Data Sets / A. Fernández та ін. Cham : Springer International Publishing, 2018. 377 с. URL: https://doi.org/10.1007/978-3-319-98074-4 (дата звернення: 15.06.2024).

14. Brownlee J. Why Is Imbalanced Classification Difficult? Machine Learning Mastery. URL: https://machinelearningmastery.com/imbalancedclassification-is-hard/ (дата звернення: 16.06.2024).

15. Orduz J. C. Getting Started with Spectral Clustering - KDnuggets. KDnuggets. URL: https://www.kdnuggets.com/2020/05/getting-startedspectral-clustering.html (дата звернення: 17.06.2024).

16. Menardi G., Torelli N. Training and assessing classification rules with imbalanced data. Data Mining and Knowledge Discovery. 2012. Т. 28, № 1. С. 92–122. URL: https://doi.org/10.1007/s10618-012-0295-5 (дата звернення: 17.06.2024).

17. Automated Retraining of Machine Learning Models. International Journal of Innovative Technology and Exploring Engineering. 2019. Т. 8, № 12. С. 445–452. URL: https://doi.org/10.35940/ijitee.l3322.1081219 (дата звернення: 10.06.2024).

18. SMOTE: Synthetic Minority Over-sampling Technique / N. V. Chawla та ін. Journal of Artificial Intelligence Research. 2002. Т. 16. С. 321–357. URL: https://doi.org/10.1613/jair.953 (дата звернення: 18.06.2024).

19. minoue-xx/Oversampling-Imbalanced-Data. GitHub. URL: https://github.com/minoue-xx/Oversampling-Imbalanced-Data (дата звернення:

20. 03.06.2024).

21. Различные стратегии сэмплинга в условиях несбалансированности классов. BaseGroup Labs. URL: https://basegroup.ru/community/articles/imbalancedatasets(дата звернення: 16.06.2024).

22. Brownlee J. Undersampling Algorithms for Imbalanced Classification. Machine Learning Mastery. URL: https://machinelearningmastery.com/undersampling-algorithms-for-imbalancedclassification/ (дата звернення: 06.06.2024).

23. Fernández G. Balancing Techniques. SlidePlayer. URL: https://slideplayer.com/slide/13128031/ (дата звернення: 10.06.2024).

24. What is the Logistic Regression algorithm and how does it work? Medium. URL: https://medium.com/analytics-vidhya/what-is-the-logistic-regression-algorithmand-how-does-it-work-92f7394ce761 (дата звернення: 12.06.2024).

25. Kathuria A. Intro to optimization in deep learning: Gradient Descent. Paperspace. URL: https://blog.paperspace.com/intro-to-optimization-in-deeplearning-gradient-descent/ (дата звернення: 14.06.2024).

26. LightGBM: A highly efficient gradient boosting decision tree / K. Guolin, M. Qi та ін. NIPS 2017, С. 3149–3157. URL:https://papers.nips.cc/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf (дата звернення: 14.06.2024).