

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»**

**Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу**

До захисту допущено:
Завідувач кафедри
_____ Оксана ТИМОЩУК
«___» _____ 21__ р.

Дипломна робота
на здобуття ступеня бакалавра
за освітньо-професійною програмою «Системи і методи штучного
інтелекту»
спеціальності 122 «Комп'ютерні науки»
на тему: «Методи штучного інтелекту для виявлення та запобігання
шахрайству в банківському секторі»

Виконала:

студентка IV курсу, групи КА-76

Пашнєва Ольга Миколаївна _____

Керівник: професор, д.т.н., Зайченко Олена Юріївна _____

Консультант з економічного розділу:

доцент, к.е.н., Рощина Надія Василівна _____

Консультант з нормконтролю:

доцент, к.т.н., Коваленко Анатолій Єпіфанович _____

Рецензент:

доцент кафедри АПЕПС, к.т.н., Шаповалова Світлана Ігорівна _____

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів
без відповідних посилань.

Студентка _____

Київ – 2021 року
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 «Комп’ютерні науки»

Освітня програма «Системи і методи штучного інтелекту»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Оксана ТИМОЩУК

«26» травня 2021 р.хххs

ЗАВДАННЯ
на дипломну роботу студентці
Пашнєвій Ользі Миколаївні

1. Тема роботи «Методи штучного інтелекту для виявлення та запобігання шахрайству в банківському секторі», керівник роботи Зайченко Олена Юріївна, професор, д.т.н., затверджені наказом по університету від «26» травня 2021 р. № 1344-с.
2. Термін подання студентом роботи: 8.06.2021.
3. Вихідні дані до роботи: вибірка банківських транзакцій з шахрайськими та звичайними операціями за 2 дні у вересні 2013 р.
4. Зміст роботи: робота складається з трьох частин. ЧАСТИНА 1 – постановка проблеми, актуальність роботи та теоретичні відомості з кількості та динаміки

банківського шахрайства. ЧАСТИНА 2 – дослідження математичних основ роботи з незбалансованими даними та побудови класифікатора. ЧАСТИНА 3 – огляд результатів роботи побудованих моделей машинного навчання. ЧАСТИНА 4 – функціонально-вартісний аналіз програмного продукту.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): принцип роботи програмного продукту, результати прогнозування моделей, багатошарова нейрона мережа, ілюстрації вхідних даних, презентація дипломної роботи.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Рощина Н.В.		

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Формування теми БДР	15.04.2021-16.04.2021	
2	Узгодження теми БДР з керівником	16.04.2021-21.04.2021	
3	Огляд літератури за тематикою роботи та її опрацювання	18.04.2021-23.04.2021	
4	Написання першого розділу БДР	20.04.2021-05.04.2021	
5	Програмна реалізація моделі	05.04.2021-25.04.2021	

6	Написання другого розділу БДР	06.04.2021- 14.04.2021	
7	Написання третього розділу БДР	14.04.2021- 26.04.2021	
8	Написання четвертого розділу БДР	26.04.2021- 29.04.2021	

Студент

Ольга ПАШНЄВА

Керівник

Олена ЗАЙЧЕНКО

РЕФЕРАТ

Дипломна робота містить: 115 с., 47 рис., 9 табл., 2 додатки, 19 джерел.

МЕТОДИ ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ ВИЯВЛЕННЯ ТА ЗАПОБІГАННЯ ШАХРАЙСТВУ В БАНКІВСЬКОМУ СЕКТОРІ

Об'єкт даного дослідження – Навчальні вибірки даних з звичайними та шахрайськими банківськими транзакціями.

Мета та цілі роботи – Дослідити існуючі підходи для роботи з незбалансованими даними, різні архітектури моделей машинного навчання та їх ефективність для виявлення банківського шахрайства.

Предмет дослідження – математичні методи аугментації даних, статистичні моделі, методи машинного навчання (зокрема глибокого машиного навчання).

Результатом роботи є система, що здатна робити передбачення є транзакція звичайною, або шахрайською.

ABSTRACT

Thesis: 115 pages, 47 figures, 9 tables, 2 appendices, 19 sources.

ARTIFICIAL INTELLIGENCE FOR FRAUD DETECTION AND PREVENTION IN THE BANKING SECTOR

The study uses a dataset featuring fraud and non-fraud banking transactions to evaluate models' performances.

The main aim of the paper is to explore existing methods for dealing with highly imbalanced data, different machine learning models and their aptness to detect fraudulent transactions.

The study objects are mathematical methods for data augmentation, statistical and machine learning (including deep learning) models.

As a result, a system that is capable of making predictions on whether the transaction is fraudulent or not was developed.

ЗМІСТ

ВСТУП.....	9
РОЗДІЛ 1 ОСНОВНІ ТИПИ БАНКІВСЬКОГО ШАХРАЙСТВА ТА ПІДХОДИ ДО ЙОГО ДО ЙОГО ВИЯВЛЕННЯ	11
1.1 Банківське шахрайство. Основні поняття та актуальність проблеми 11	
1.2 Підходи до виявлення банківського шахрайства.....	13
1.3 Виклики у розробці ML моделей для запобігання шахрайства	15
1.3.1. Складнощі навчання на незбалансованих даних	16
1.3.2 Необхідність регулярного донавчання моделі.....	21
1.4 Висновки до розділу 1	22
РОЗДІЛ 2 МАТЕМАТИЧНІ ОСНОВИ РОБОТИ.....	24
2.1 Стратегії семплінгу в умовах незбалансованості даних.....	24
2.2 Синтетичне наповнення датасету за допомогою SMOTE.....	25
2.1.1 Математичні методи видалення прикладів мажоритарного класу .	29
2.1.2 Плюси та мінуси використання андерсемплінгу та оверсемплінгу	32
2.2 Методи прогнозування	34
2.2.1 Логістична регресія	34
2.2.2 LightGBM.....	37
2.2.3 Нейронна мережа.....	39
2.3 Оцінка якості отриманого результату	41
2.4 Висновки до розділу 2	43
РОЗДІЛ 3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНІ РЕЗУЛЬТАТИ.....	45
3.1 Навчальна вибірка.....	45
3.2 Обґрунтування вибору мови та платформи програмування.....	46
3.3 Результати роботи	48
3.4 Висновки до розділу 3	55

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ.....	57
4.1 Постановка задачі проектування	57
4.2 Обґрунтування функцій програмного продукту	58
4.3 Обґрунтування системи параметрів ПП	61
4.4 Аналіз експертного оцінювання параметрів.....	63
4.5 Аналіз рівня якості варіантів реалізації функцій	68
4.6 Економічний аналіз варіантів розробки ПП	69
4.7 Вибір кращого варіанту ПП техніко-економічного рівня	75
4.8 Висновки до розділу 4	76
ВИСНОВКИ.....	77
ЛІТЕРАТУРА.....	78
ДОДАТОК А.....	81
ДОДАТОК Б.....	94

ВСТУП

Шахрайські операції у банківському секторі приносять колосальні збитки банкам та, у найкращому випадку, дискомфорт пов'язаний витратою часу на повернення коштів клієнтам, адже всього 25% збитків від шахрайства можна повернути [1]. Банківські ж установи зазнали збитків на 32.39 млрд доларів тільки у 2020 році. Динаміка також невтішна – за прогнозами експертів у 2027 р. банки втратять 40.62 млрд доларів, в той час як ще у 2011 р. зафіксовано лише 9.84 млрд прямих збитків, що втричі менше, аніж станом на 2020 рік [2].

Таким чином, найкращою стратегією боротьби з шахрайством є своєчасне виявлення шахрайських транзакцій та їх блокування. Складність полягає у тому, що виявляти шахрайські операції потрібно у режимі реального часу; часто на те, що це дії саме зловмисника вказують десятки факторів, які людина опрацювати та створити ефективну систему правил з їх виявлення не може. Крім того, банківські установи діляться між собою тільки обмеженою інформацією про клієнта, що створює додаткові труднощі, адже банківська установа не може аналізувати усю історію операцій клієнтів інших банків, коли ті здійснюють транзакцію.

Враховуючи ще й той факт, що кожної секунди через кожен банк проходять десятки операцій, тобто об'єм даних надзвичайно великий, виявлення та запобігання шахрайства в банківському секторі є класичною задачею для машинного навчання з учителем.

Задачі роботи:

- 1) Проаналізувати різні методи семплінга в умовах незбалансованих класів (адже шахрайських транзакцій суттєво менше, аніж звичайних, що сильно впливає на роботу алгоритмів машинного навчання), дослідити вплив передобробки даних на ефективність побудованих моделей.
- 2) Визначити метрики для оцінки моделей, порівняти різні архітектури для вирішення задачі виявлення шахрайства, підібрати оптимальні параметри.

- 3) 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 з року в рік залишається актуальною, адже кількість та відсоток таких операцій ростуть (рис. 1.1). Станом на 2018 р. найпоширенішими типами шахрайства є операції без наявності картки (79.5%) та шахрайство у ATM або POS терміналах з використанням викрадених або загублених (11.2%) та фальсифікованих (5.39%) карт [3].

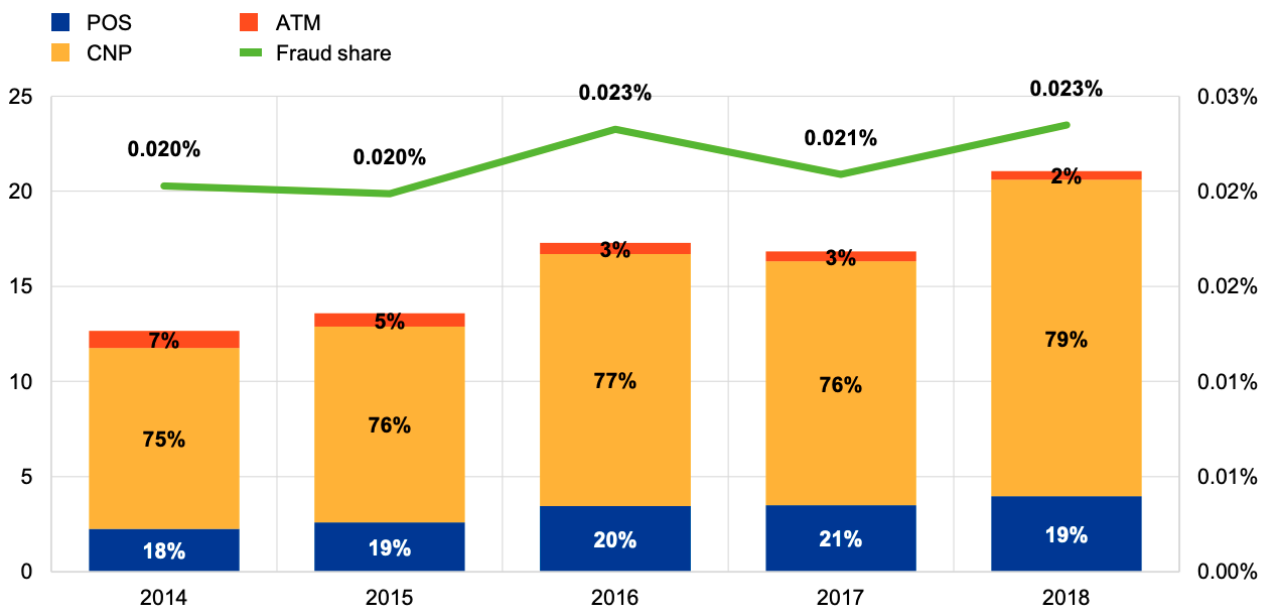


Рисунок 1.1 – Динаміка кількості банківського шахрайства у Європейському Союзі за категоріями [3]

Операції без наявності картки (CNP – card-not-present) – банківські операції, які відбуваються без присутності людини, що здійснює оплату, на місці проведення оплати. Прикладами використання CNP є оплата в інтернет-магазині або оплата за допомогою телефону. У наведених вище випадках продавець товару не може перевірити чи дійсно покупець є власником картки. Перевіряють тільки номер картки, дату завершення її дії, PIN та код безпеки, нанесений на звороті.

Через високий ризик шахрайських транзакцій (рис. 1.2) банк-емітент стягує вищу комісію за такі операції у організації, яка регулярно проводить такі платежі. Адже на відміну від транзакцій з присутністю клієнта на місці оплати, банківський еквайер, що розміщує торговий рахунок, має обов'язок повернути втрачене власнику картки, у випадку, якщо транзакція була шахрайською [4].

(left-hand scale: total value of CNP fraud (EUR millions); right-hand scale: value of CNP fraud as share of value of card fraud (percentages))

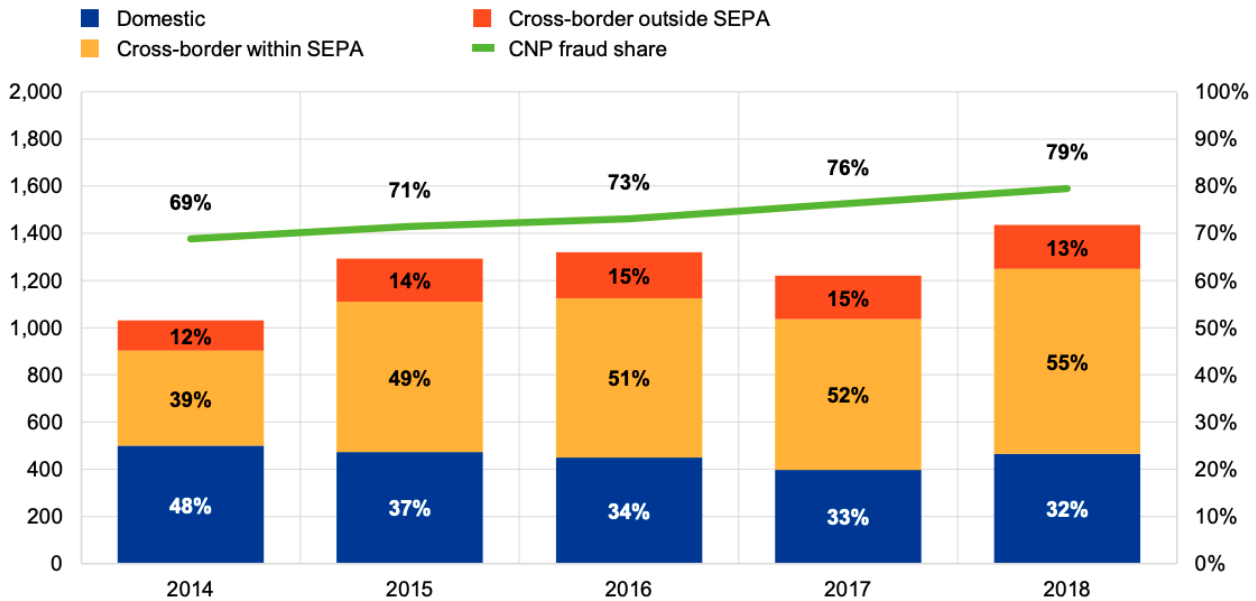


Рисунок 1.2 – Збитки та динаміка CNP шахрайства у Європейському Союзі у млн євро [3]

Другим типом шахрайства є фрод у операціях з наявністю картки (CP – card-present). Збитки клієнту у випадку CP фрода, відшкодовує банк, який обслуговує даний рахунок [5].

Використання викрадених або загублених карток для операцій через АТМ або POS є наймасовішим типом CP шахрайства, хоча за останні роки частка таких махінацій зменшилась з 25% у 2014 р. до 20% у 2018 р. [3].

Використання підроблених карток все ще залишається проблемою, але частка такого шахрайства продовжує зменшуватись через покращення систем захисту банківських терміналів, на які шахраї встановлюють зчитувачі даних з карток.

До інших типів СР шахрайства відносяться викрадення або злом акаунту та отримання доступу до логіна і пароля на вхід до банківської системи (АТО – account takeover), злом самої системи для банкінгу. Як видно з рис 1.3 (категорія «інше»), такі сценарії шахрайства стають більш популярними. Головною причиною росту є витоки даних, які трапляються частіше, в той час як користувачі інтернету нерідко використовують однакові або недостатньо видозмінені паролі для більшості сервісів, та розвиток даркнету.

Збитки від СР шахрайства падають (рис. 1.3), в той час як від CNP тільки ростуть.

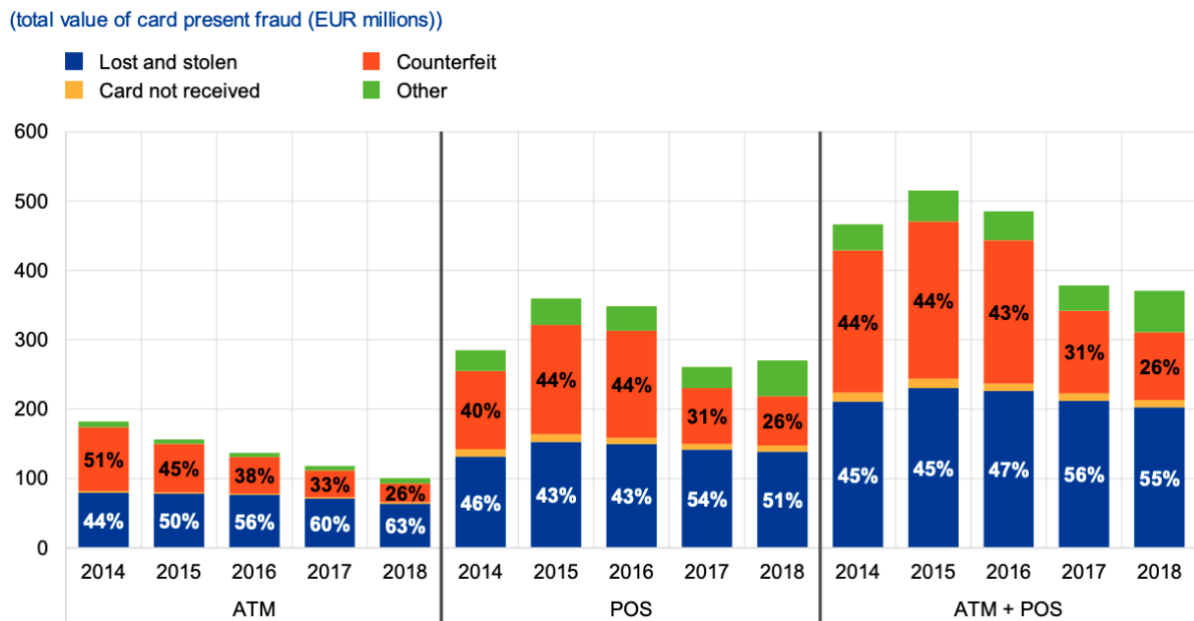


Рисунок 1.3 – Збитки від СР шахрайства у Європейському Союзі у млн євро [3]

1.2 Підходи до виявлення банківського шахрайства

Системи виявлення банківського шахрайства можуть бути розділені на 2 групи:

- 1) ML (або Data driven) системи, які ґрунтуються на аналізі даних (дата проведення транзакції, сума переводу, дані про кредитну історію клієнта і т. п.) про велику кількість транзакцій. В основі лежить алгоритм машинного навчання,

який навчається на вже розмічених даних про тип транзакції, або алгоритм, що без розмітки розділяє транзакції на кластери, виділяючи підозрілу активність;

2) Експертні системи, розроблені групою експертів тої чи іншої банківської установи, які на основі власного досвіду складають набір правил по виявленню шахрайства.

Переваги експертних систем очевидні:

- простота та швидкість розробки і наповнення
- можливість відслідкування точної причини блокування

Системи, створені на основі ML моделі не можуть похизуватися ні однією з цих переваг, крім того часто детектор на основі правил працює швидше, аніж модель, адже показники повинні бути приведені до стандартного вигляду, а далі слідує множення на матрицю вагів. Проте, мінуси експертних систем набагато більш вагомі:

- виявлення та блокування тільки раніше відомих сценаріїв фроду
- висока влучність, але мізерна повнота
- на довгій дистанції виявиться, що розробка таких систем – це достатньо довгий та неефективний процес, адже експерт має регулярно моніторити нові випадки шахрайства та вносити відповідні зміни
- працюють лише з очевидними для людини кореляціями між певними показниками та шахрайством, тому можуть легко відслідковуватися і обходитись шахраями

Усі ці недоліки повною мірою відсутні у ML системах. Тому має сенс або поєднувати ці два типи систем (використовувати правила як первинний фільтр) для покращення швидкості та прозорості виявлення шахрайства, або використовувати тільки ML системи, адже в банківській сфері дуже важливо виявляти як можна більший відсоток фроду, так як будь-яка операція може обернутися великими прямими втратами для компанії.

Кейси шахрайства можуть сильно відрізнятися у банках по всьому світу, а значить і правила для його визначення теж. Крім того, як було визначено раніше,

основна складність полягає саме у розробці Data driven систем для протидії фродовим транзакціям. Тому далі буде йти мова виключно про виявлення шахрайства за допомогою ML систем.

1.3 Виклики у розробці ML моделей для запобіганню шахрайства

Більшість проблем, з якими зіштовхуються інженери, які займаються розробкою моделей для виявлення банківського шахрайства, не унікальні, а типові для виявлення шахрайства в цілому. Так, можна виділити такі головні виклики:

- 1) Дані вкрай незбалансовані – 99% транзакцій звичайні;
- 2) Шахраї починають вигадувати та впроваджувати нові схеми з фроду після того, як розуміють, що їх транзакції блокуються. Таким чином, еволюція їх методів відбувається досить швидко;
- 3) Характер транзакцій звичайних користувачів та шахраїв може бути сезонним – покупки перед святами або на чорну п'ятницю відрізняються за багатьма факторами від звичайних. Тобто, потрібно розмічати та навчати модель на великих часових проміжках.
- 4) Необхідність регулярно проводити пост-аналіз (рис. 1.4) для виявлення нових сценаріїв та поповнення ними датасету (витікає з пункту 2).
- 5) Затримка в декілька днів у представленні актуальних даних для моделі, адже скарги від клієнтів на шахрайство доходять не одразу.

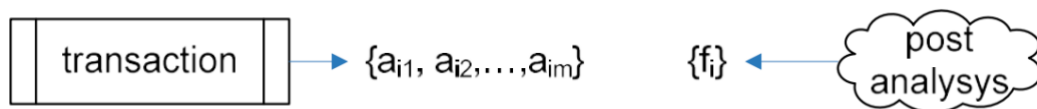


Рисунок 1.4 – Процес пост-аналізу

З п'яти розглянутих вище проблем побудови ML моделі для виявлення шахрайства розробник найбільше контролює та може вирішити тільки першу. А також може вплинути на процес збору нових даних. Тому розглянемо докладніше

як саме незбалансовані дані впливають на точність класифікації, та процес поповнення даних для дотренування моделі.

1.3.1. Складнощі навчання на незбалансованих даних

Дані вважаються незбалансованими, коли спостерігаються такі розподіли класів: 1:2, 1:10, 1:100, 1:1000 і т. п. У датасеті Credit Fraud [6], який був обраний для цієї роботи, 284807 транзакцій, але лише 492 з них шахрайські. Таким чином маємо 99.83% звичайних транзакцій (рис. 1.5).

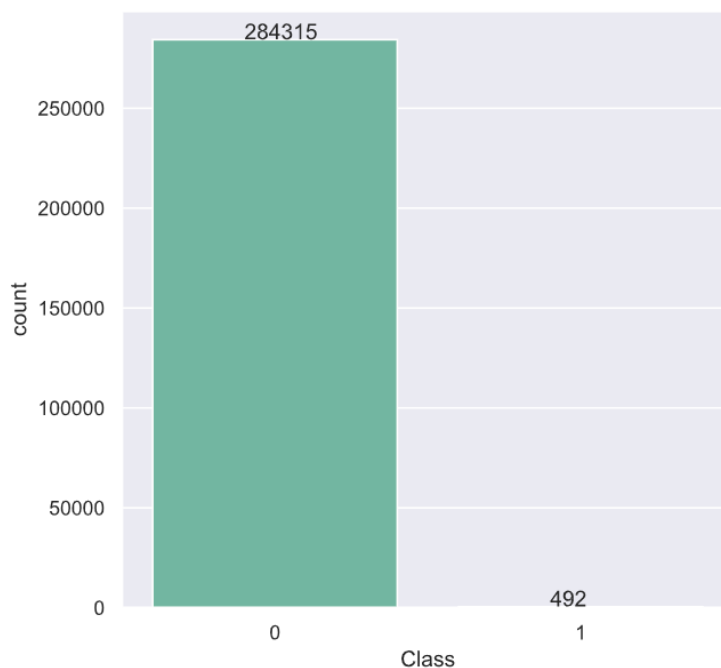


Рисунок 1.5 – Кількість звичайних та фродових транзакцій

Такий розподіл призводить до «упередженості» натренованих моделей – коли усі транзакції будуть класифікуватися як нормальні, і при цьому отримаємо точність 99.83%, що було б чудово для моделі на збалансованих даних. З цього витікає, що звичайні метрики для класифікації такі як точність та ROC AUC не репрезентативні для даного класу задач. Крім того, через невелику кількість даних мінорного класу або відсутність даних, зіштовхуємося з проблемою "відсутності щільності" або "недостатності інформації" [7, с. 261].

Також постає проблема у кількості даних, якщо ми до менш представленого класу застосовуємо різні методи збільшення кількості спостережень такі як SMOTE або oversampling, та збільшуємо кількість прикладів до співвідношення 1:1, то отримуємо майже вдвічі більший датасет, що негативно позначається на швидкості навчання. Наприклад, для звичайної задачі класифікації зі збалансованим розподілом класів ми можемо необхідні тисячі чи десятки тисяч прикладів для того, щоб розробити, оцінити та вибрати модель. Обравши же для аналізу транзакції за декілька днів, або місяців, отримаємо десятки мільйонів прикладів для тренування. Отже, працюючи з задачею виявлення шахрайства необхідно дослідити методи не тільки збільшення кількості прикладів мінорного класу для тренування моделі, а й методи зменшення кількості прикладів домінуючого класу такі як Tomek та random undersampling.

Дисбаланс класів є визнаним ускладнюючим фактором для класифікації. Хоча, деякі дослідження вказують на те, що висока диспропорція у класах не є єдиною причиною сильного падіння якості моделі при навчанні [7, с. 253]. Існують багато погіршуючих факторів серед яких:

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

Шуми у характеристиках є у прикладах, опис яких містить певні помилки або шум (рис. 1.6). Виділяють два типи шумів: шум в атрибутах та шум у цільовій змінній. Загалом вважається, що шум у цільовій змінній завдає набагато більше шкоди моделі, аніж шум атрибутів навіть для звичайних збалансованих моделей. Вплив на незбалансовану модель, коли прикладів одного з класів і так мало, помилки у атрибутах чи класі прикладу завдають руйнівної шкоди.

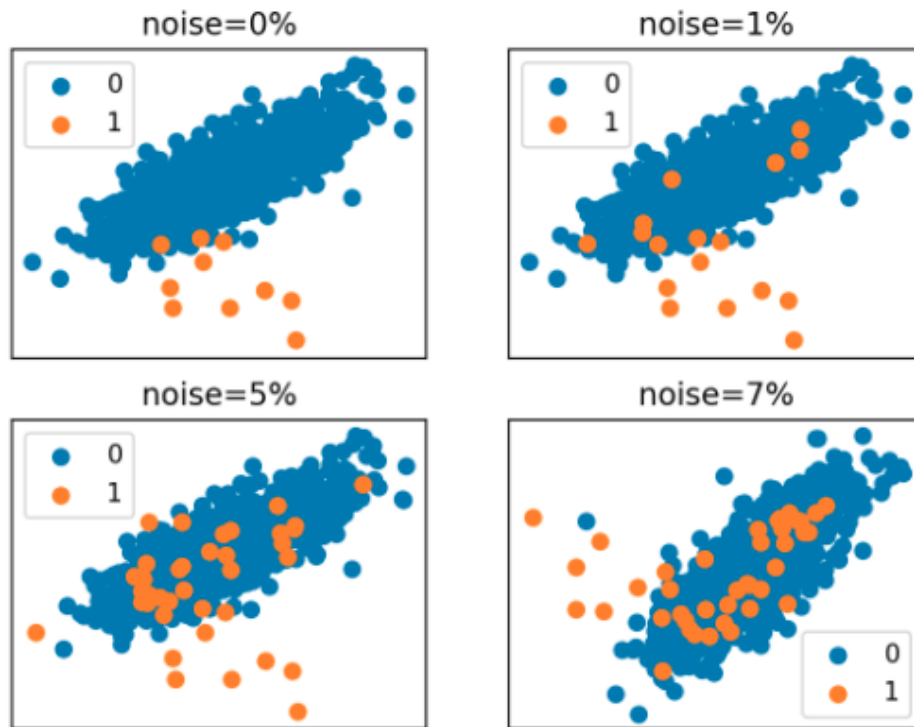


Рисунок 1.6 – Штучно зашумлені дані [8]

Другим важливим ускладнюючим фактором є розподіл прикладів у просторі ознак. Так, якщо зобразити візуально простір ознак (просто вивести на графік, якщо маємо 2-3 ознаки, або якщо ознак багато, то використати один з алгоритмів машинного навчання для візуалізації даних, наприклад PCA чи t-SNE), то найкращим варіантом буде чіткий розподіл на кластери (рис. 1.7). Тоді, одразу можемо сказати, що вдасться побудувати класифікатор, який досягає гарних результатів. Це можна сказати і про данні зі збалансованим, і про дані з незбалансованим розподілом.

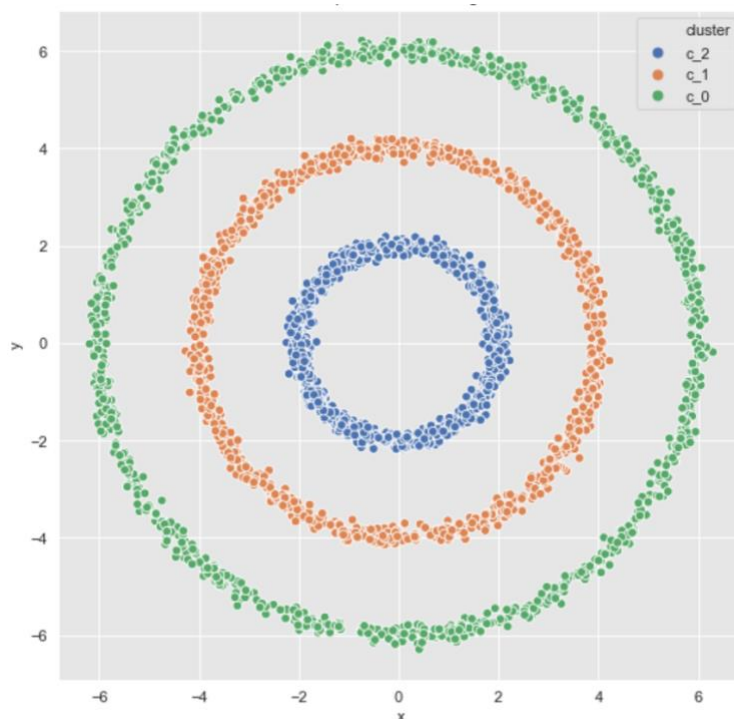


Рисунок 1.7 – Чіткий розподіл ознак x , y на кластери за класом [9]

У реальних, а не навчальних задачах це трапляється нечасто, тобто немає чіткого візуального розподілу на класи за візуалізацією ознак, і, швидше за все, кожен клас можна додатково розділити на декілька підкласів, що призводить до створення декількох різних груп або кластерів прикладів у просторі ознак. Наприклад, на рис. 1.8 видно, що клас 0 сам розділений на 2 кластера, а також між класами 0 та 1 немає чіткого розподілу на просторі ознак.

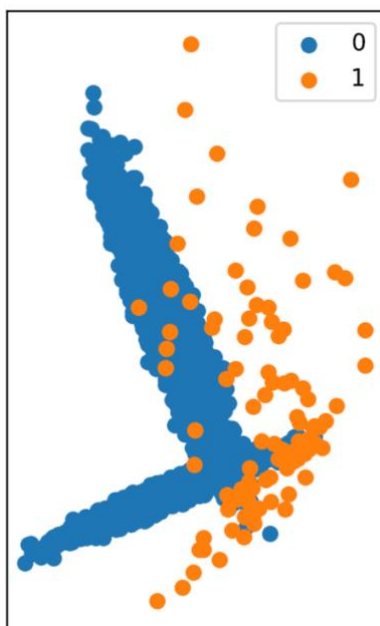


Рисунок 1.8 – Нечіткий розподіл ознак x, y на кластери за цільовою змінною [8]

Такі підгрупи формально називають диз'юнктами. Диз'юнктами можуть бути різними за розміром. Невеликим диз'юнктом називають той диз'юнкт, який охоплює всього декілька прикладів з навчального набору даних [8].

Цей розподіл ускладнює відокремлюваність класів для моделі, адже необхідно ідентифікувати та включити кожну підгрупу або кластер до визначення межі класу (рис. 1.9).

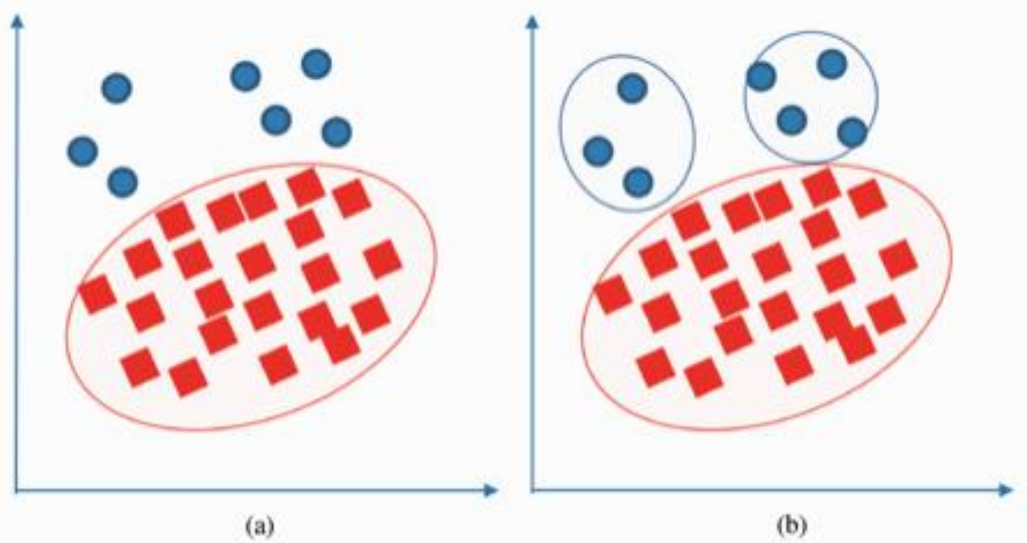


Рисунок 1.9 – Підгрупи у меншому класі [10]

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

Така недостатність щільності має особливий вплив в алгоритмах, які використовують підхід «розділяй та володарюй» (такі як дерева рішень) та в задачах покриття множин (наприклад індукція правил), в яких наявність підгруп веде до створення невеликих диз'юнктів, адже у алгоритмах покриття множин і «розділяй та володарюй» моделі розділені на декілька частин і концепція класу представлена як диз'юнкт цих частин для кожного класу [10].

1.3.2 Необхідність регулярного донавчання моделі

Як було описано вище, нові сценарії шахрайства з'являються регулярно, як і поступово змінюється характер звичайних транзакцій, тому з'являється необхідність дотреновувати модель з частотою від декількох днів до місяця. Така необхідність також створює значні ускладнення, адже виникає питання регулярної розмітки даних. У більшості фінансових компаній є окремий відділ, який займається боротьбою з шахрайством, спеціалісти якого регулярно здійснюють моніторинг та проводять аналітику дій шахраїв, натомість зазвичай шахрайських операцій так багато, що спеціалісти «полюють» виключно на великих шахраїв, а втрату невисоких сум і так покриває підвищена банківська комісія для CNP операцій. Тому інформація представлена подібним відділом була б неповною і недостатньою, враховуючи те, що необхідно регулярно збирати інформацію і про звичайні транзакції, а не тільки шахрайські.

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

- 1) Автоматичний збір скарг клієнтів на шахрайство (затримка 1-2 дні);
- 2) Автоматична обробка звітів, складених відділом з протидії шахрайству, у яких міститься інформація про шахрайські операції, що призвели до значущих збитків (затримка 2-7 днів);
- 3) Пост-аналіз (ручний чи напів-автоматичний), спрямований на виявлення нових сценаріїв шахрайських дій (затримка в 30 днів), в результаті якого датасет може бути поповнений не тільки шахрайськими прикладами, а й прикладами звичайних операцій, що також дозволяє корегувати модель на сезонність.

Процес дотренування моделі зображений на рис. 1.10.

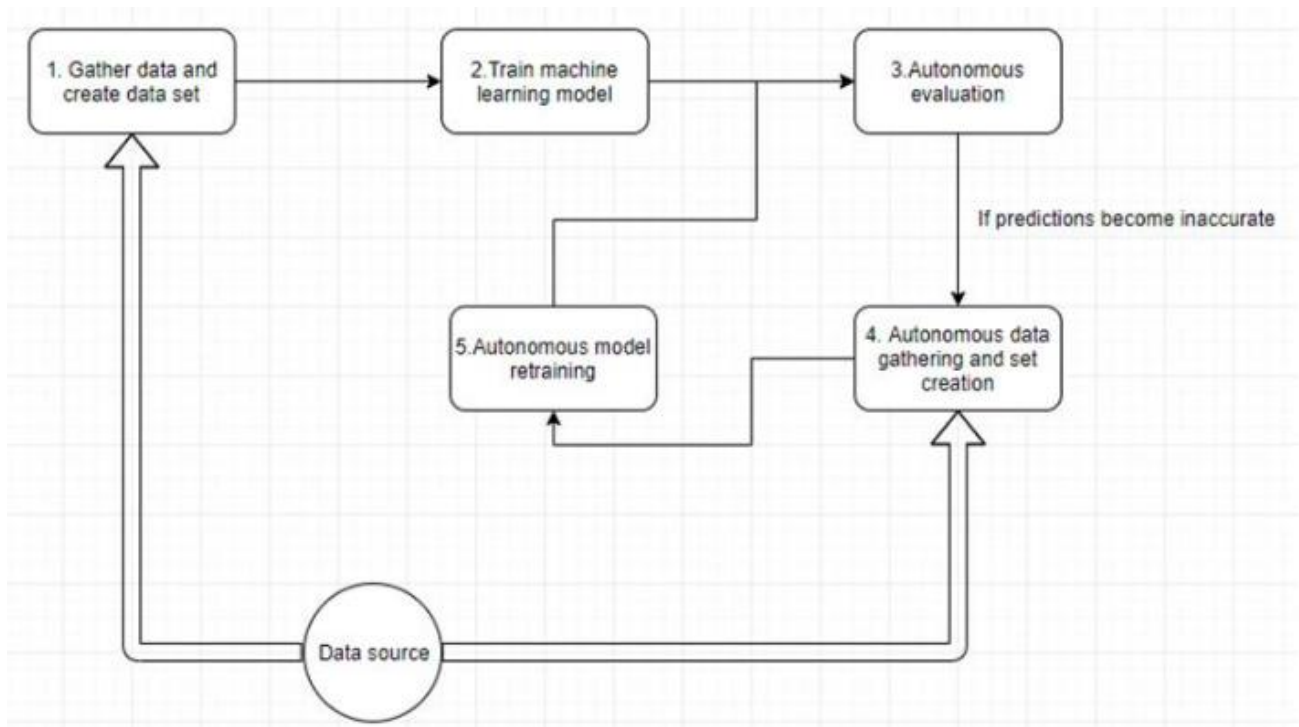


Рисунок 1.10 – Автоматичне дотренування моделі [11, с. 446]

Після того як була розроблена первинна модель та процес був автоматизований, оновлення результатів відбувається таким чином: до тестового набору датасету додаються нові приклади і отримується передбачення за допомогою наявної моделі. Якщо показник обраної для оцінки моделі метрики або метрик не змінюється суттєво, то для виявлення шахрайства залишаємо стару модель. В іншому випадку, вона має бути перетренована, для чого існують декілька зручних та швидких бібліотек мовою Python. В результаті, отримаємо нову модель для класифікації без втручання людини.

1.4 Висновки до розділу 1

Отже, враховуючи ріст як абсолютної кількості шахрайських транзакцій у банківській сфері, так і збільшення таких транзакцій у процентному співвідношенні до звичайних операцій, проблема розробки автоматизованої системи для їх виявлення залишається актуальною. Також були проаналізовані

основні виклики при розробці такої моделі – регулярна поява нових сценаріїв шахрайства, висока незбалансованість даних, та шляхи їх подолання.

РОЗДІЛ 2 МАТЕМАТИЧНІ ОСНОВИ РОБОТИ

2.1 Стратегії семплінгу в умовах незбалансованості даних

Оверсемплінг (oversampling) та андерсемплінг (undersampling) – це важливі кроки при роботі з незбалансованими даними, які дозволяють “вирівняти” розподіл кількості прикладів різних класів до 50:50.

Найпростіша стратегія оверсемплінгу – це копіювання наявних прикладів меншого класу, а андерсемплінгу – випадкове видалення елементів більшого класу (random undersampling). Проте, такі прості підходи не працюють для банківських даних, адже хоч у випадку з random undersampling доведеться видалити важливі дані про нешахрайські транзакції. А описаний вище підхід до оверсемплінгу не принесе нової інформації та не вирішить проблему розподілу даних описану у розділі 1.3.1. Наприклад, випадково можуть бути видалені майже усі приклади звичайних транзакцій на суму > 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).

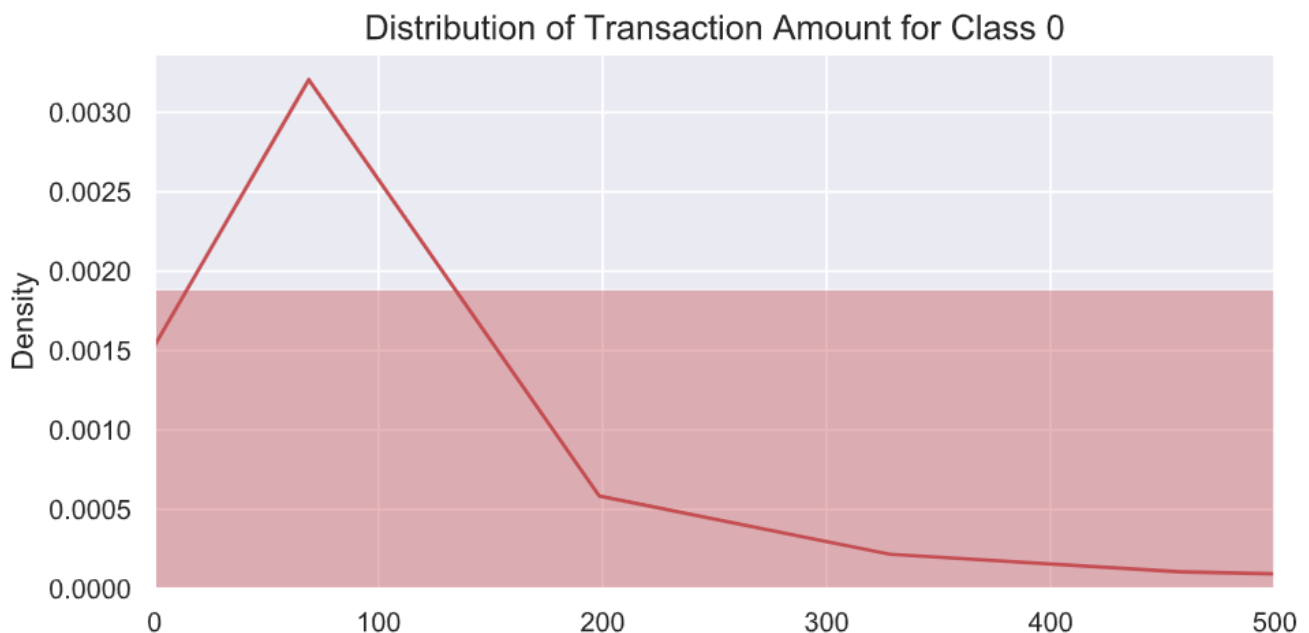


Рисунок 2.1 – Розподіл звичайних транзакцій за сумою

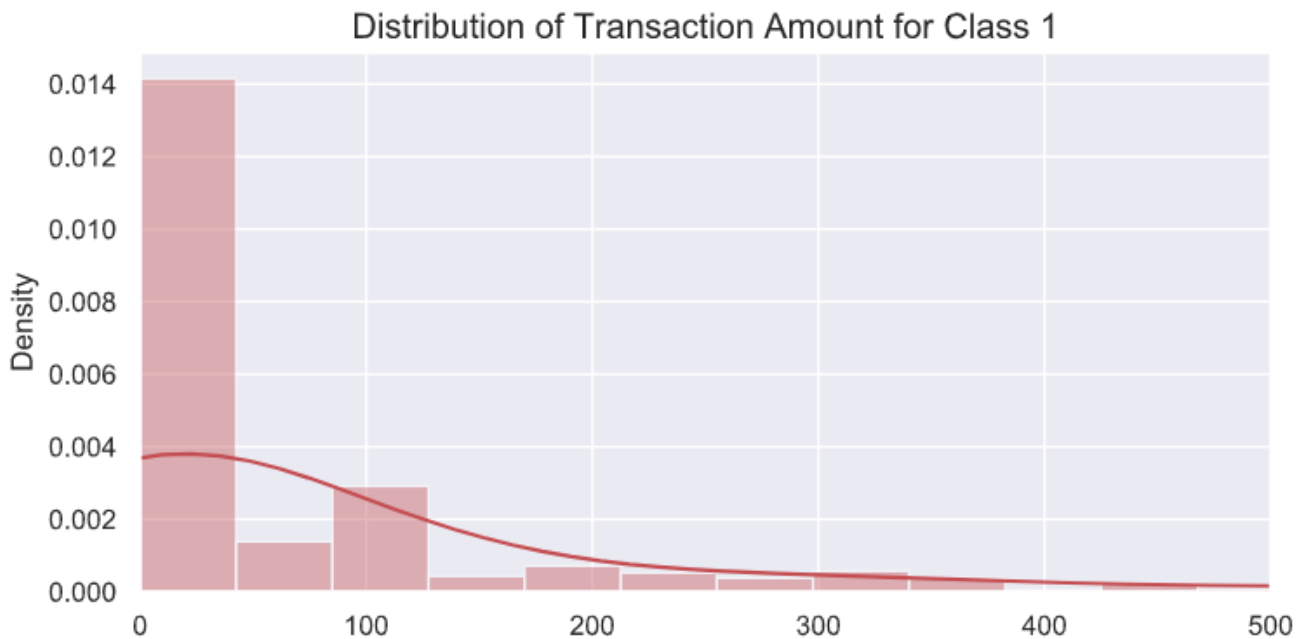


Рисунок 2.2 – Розподіл шахрайських транзакцій за сумою

В результаті, модель буде мати високу точність, але при цьому не буде придатною до використання через упередженість до великих за сумою транзакцій. Тому має сенс дослідити і використовувати у роботі математичні методи оверсамплінгу та андерсамплінгу, а не випадкові видалення та копіювання даних.

2.2 Синтетичне наповнення датасету за допомогою SMOTE

Алгоритм Synthetic Minority Oversampling Technique дозволяє замість копіювання наявних прикладів, на їх основі створювати нові «синтетичні» приклади. На створення цього підходу вплинув успішний досвід аугментації даних, використаний для задачі класифікації датасету написаних від руки символів. Тоді додаткові дані для тренування створювались за допомогою повороту, розтягування та інших перетворень фотографій, що позитивно вплинуло на точність класифікації. Проте, на відміну від попереднього прикладу SMOTE синтезує нові приклади, оперуючи простором однак, а не простором даних.

Більша кількість меншого класу синтезується, використовуючи алгоритм K-NN (k найближчих сусідів), який застосовується до усіх прикладів класу, а після створюються нові приклади, які лежать на сегментах ліній, які з'єднують найближчих сусідів класу. Зважаючи на те, яка кількість нових прикладів потрібна, сусіди з k найближчих сусідів обираються випадковим чином. Більшість реалізацій SMOTE використовують 5 найближчих сусідів. Так, якщо потрібно збільшити вибірку на 400%, обираються 4 сусіди з 5-ти найближчих, і один новий генерується по кожному напрямку.

Нові приклади створюються за таким алгоритмом:

- 1) Вираховується різниця між вектором ознак (прикладом), що розглядається, та його найближчим вектором;
- 2) Отриману різницю множать на випадкове число від (0; 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].

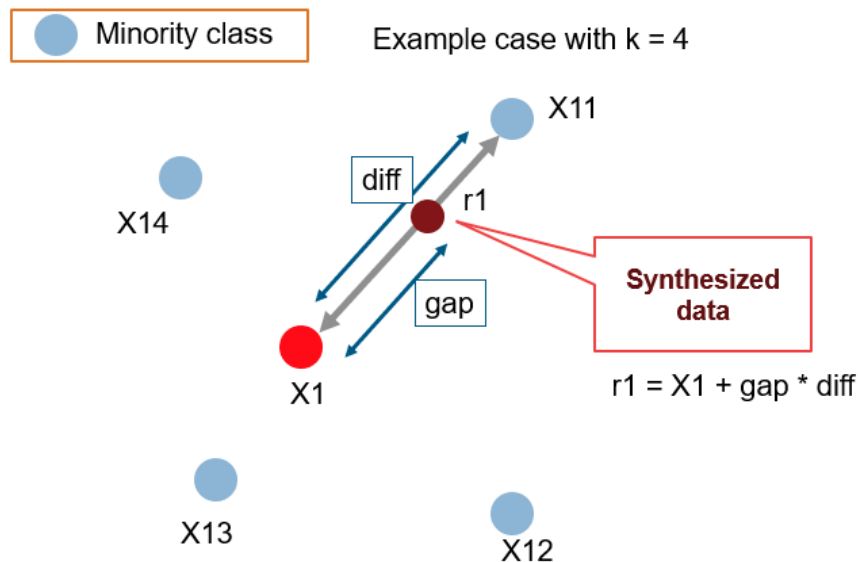


Рисунок 2.3 – Генерація нової точки у просторі ознак [13]

Псевдокод SMOTE:

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:  $diff = Sample[nnarray[nn]][attr] - Sample[i][attr]$ 
21.         Compute:  $gap$  = random number between 0 and 1
22.          $Synthetic[newindex][attr] = Sample[i][attr] + gap * diff$ 
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]

В результаті, завдяки згенерованим прикладам класифікатор створює більші та менш специфічні регіони рішень (рис. 2.5, (с)). Це допомагає дереву не перенавчатися. Точність коректного розпізнавання меншого класу теж зростає при оверсамплінгу за допомогою SMOTE > 200% в порівнянні з простим копіюванням прикладів (рис. 2.6).

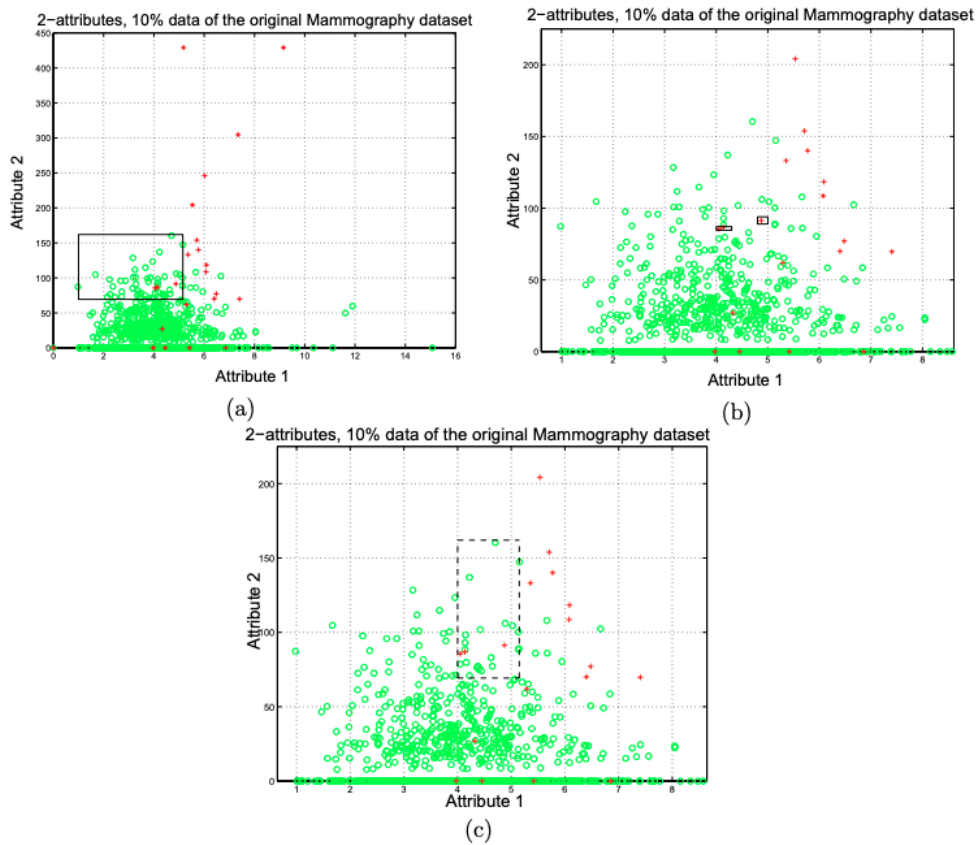


Рисунок 2.5 – вплив згенерованих за допомогою SMOTE прикладів (позначені на с) [12, с. 327]

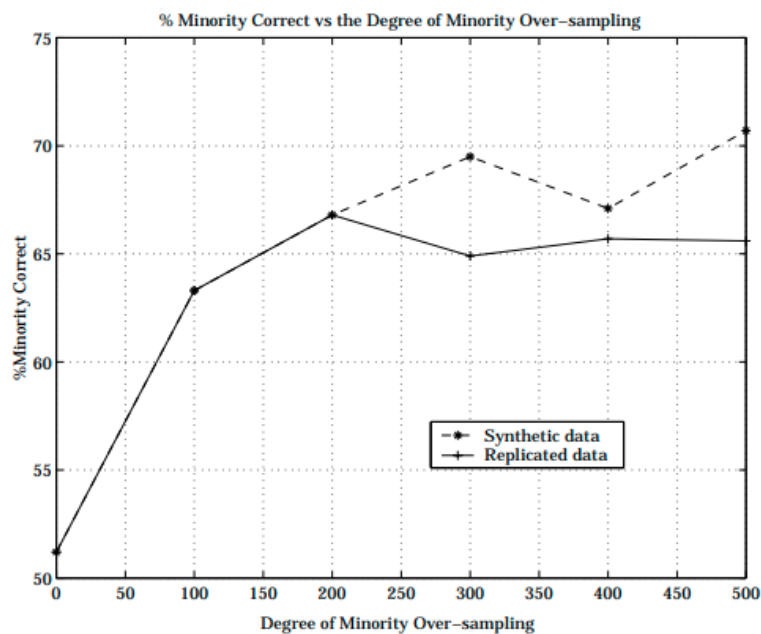


Рисунок 2.6 – Порівняння точності розпізнання меншого класу з використанням дублювання та SMOTE [12, с. 321]

Крім того, експерименти показали, що якщо комбінувати такі методи оверсемплінгу як SMOTE та андерсемплінг, то це позитивно впливає на якість вихідної моделі, так як таким чином модель повністю звільнюється від упередженості до більшого класу.

2.1.1 Математичні методи видалення прикладів мажоритарного класу

Один з найпопулярніших підходів до андерсемплінгу – це використання зв'язків Томека (Tomek Links), який був представлений у 1976 р., однак залишається актуальним на сьогодні через свою простоту та ефективність.

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

$$\begin{cases} d(E_i, E_m) < d(E_i, E_j) \\ d(E_j, E_m) < d(E_i, E_j) \end{cases} \quad (2.1)$$

Відповідно до даного підходу, всі мажоритарні записи, що входять в зв'язку Томека, повинні бути видалені з набору даних. Цей спосіб добре видаляє записи, які можуть виявитися шумом (рис. 2.7) [14].

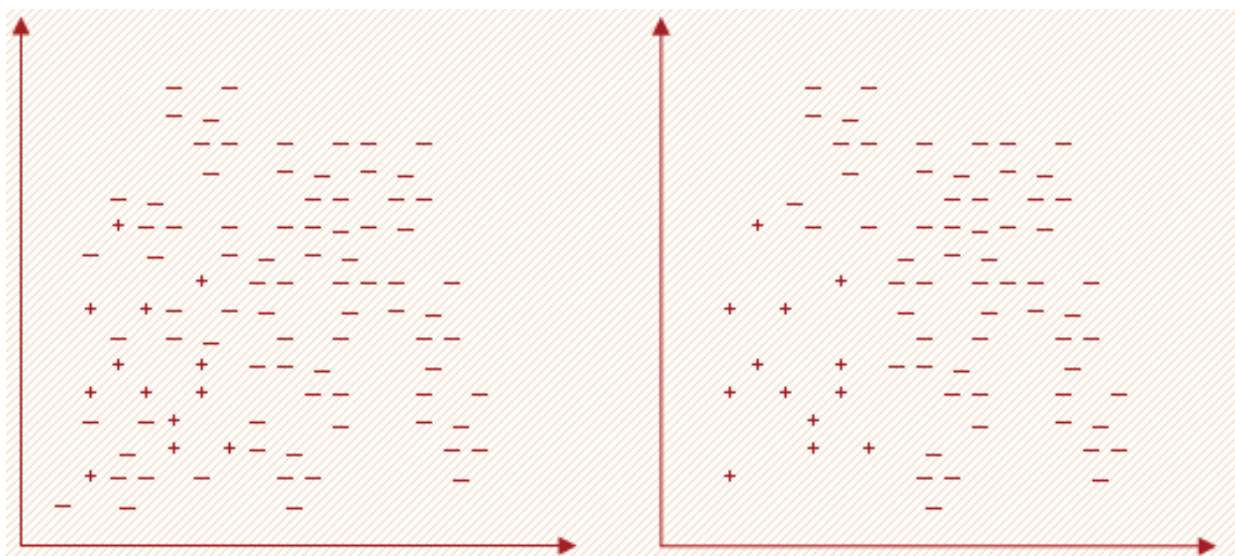


Рисунок 2.7– Датасет до видалення зв'язків Томека та після [14]

Правило найближчого ущільненого сусіда (Condensed Nearest Neighbor Rule Undersampling, правило US-CNN) – правило, яке використовують для андерсемплінгу, і кінцева ціль застосування якого – вибрати такий набір прикладів, який не буде спричиняти падіння у ефективності моделі. Таку вибірку називають мінімальним консистентним набором.

Такого результату досягають шляхом перебору усіх прикладів датасету і доданням їх до мінімально консистентного набору, якщо вони не можуть бути правильно класифіковані, використовуючи приклади, які вже наявні у наборі. Спершу, такий підхід був запропонований та використаний, щоб зменшити кількість пам'яті комп'ютера, яку необхідно виділити для роботи алгоритму k найближчих сусідів (KNN).

Для використання цього підходу для задачі незбалансованої класифікації, мінімальний консистентний набір складається з усіх прикладів меншого класу і тільки приклади більшого класу, які не вдається коректно класифікувати, послідовно додаються до набору. Під час роботи алгоритму використовується KNN, щоб класифікувати приклади, що розглядаються, і вирішити чи потрібно їх додавати до набору [15]. Таким чином, на відміну від методу зв'язків Томека, де обирають приклади, які необхідно видалити, правило найближчого ущільненого сусіда обирає, які приклади варто обрати для подальшої класифікації.

Робота алгоритму на простому датасеті з двома ознаками (рис. 2.8, рис. 2.9):

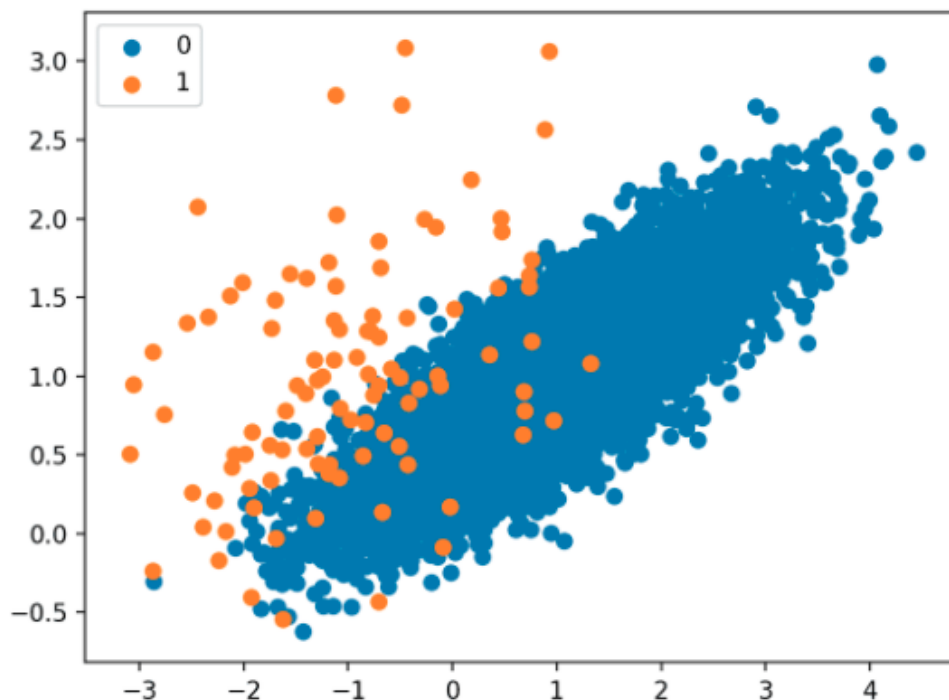


Рисунок 2.8 – Набір даних до андерсемплінгу US-CNN [15]

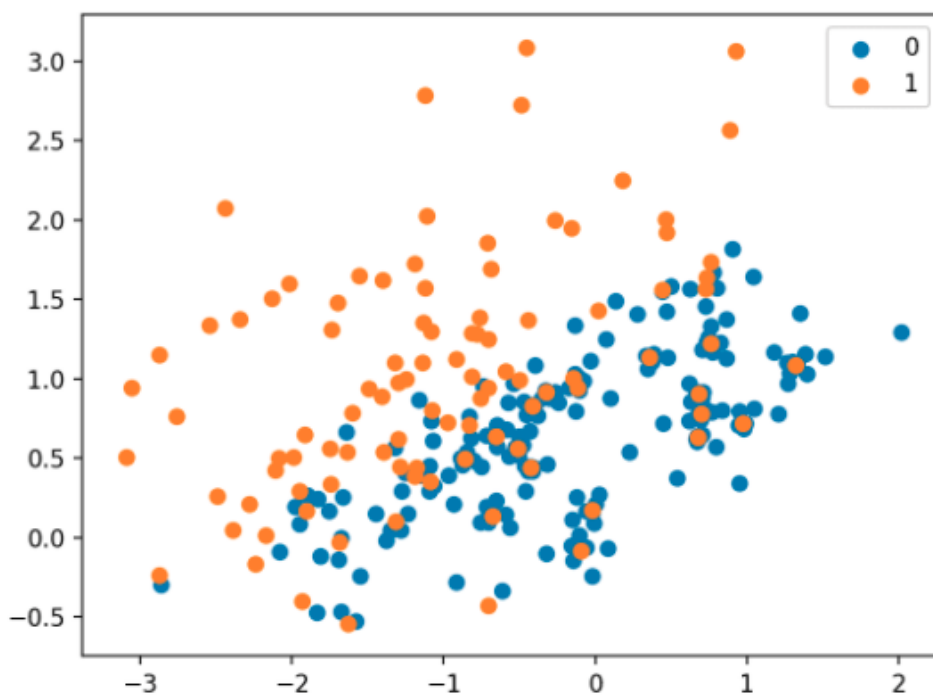


Рисунок 2.9 – Набір даних після андерсемплінгу US-CNN [15]

Дійсно, залишаються найважливіші приклади для класифікації на поточних даних. Проте варто звернути увагу на те, що усі точки далі по осі X ($X \in [2.5, 4.5]$) класу 0 були видалені. Тобто, залишаються лише приклади більшого класу «навколо» прикладів меншого.

Метод одностороннього вибору (One-sided Selection, OSS) поєднує два вище згадані алгоритми – зв’язки Томека та US-CNN, а точніше полягає у послідовному застосуванні першого та другого алгоритмів (рис. 2.10).

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

Приклади, які є результатом роботи алгоритму, – це усі приклади меншого класу та незашумлені і релевантні приклади більшого класу [7, с. 84].

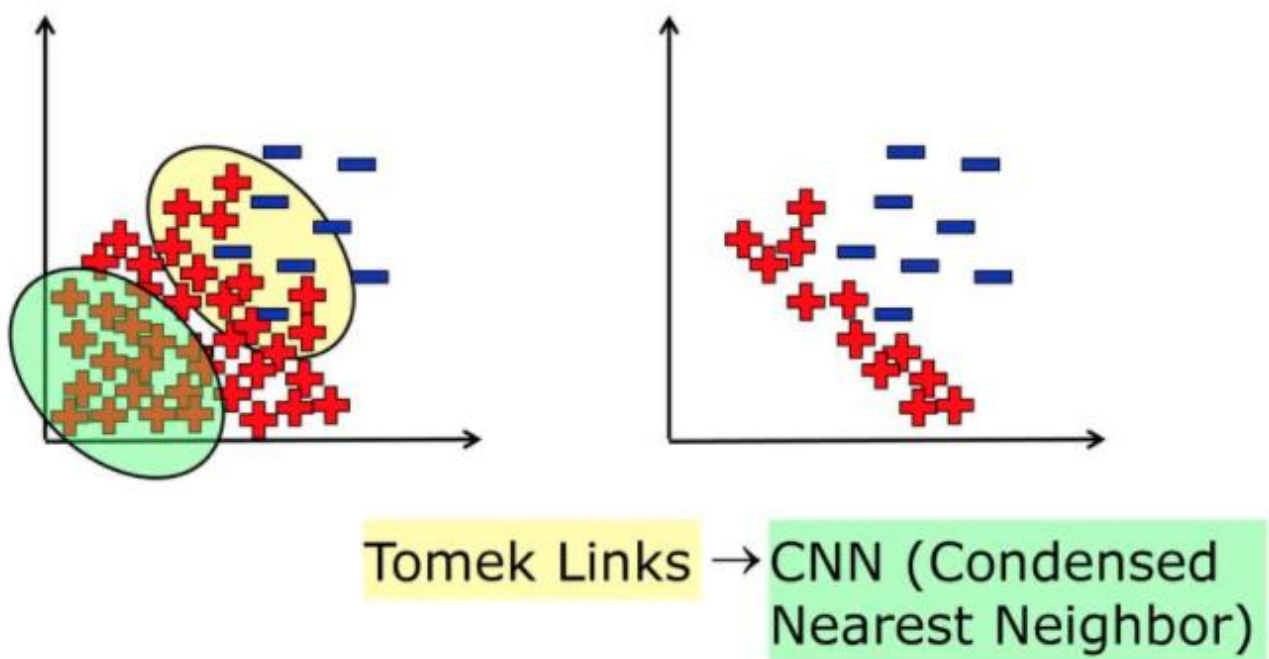


Рисунок 2.10– Алгоритм OSS [16]

2.1.2 Плюси та мінуси використання андерсемплінгу та оверсемплінгу

Хоча описані вище підходи (SMOTE, зв’язки Томека, US-CNN, OSS) працюють з даними набагато “розумніше”, аніж random oversampling та простий

undersampling, все ж недоліки використання цих підходів не нівелюються повністю. Основні недоліки наведено у таблиці 2.1:

Таблиця 2.1 – Недоліки використання оверсемплінгу та андерсемплінгу

Андерсемплінг	Оверсемплінг
- Можуть бути видалені потенційно корисні дані	- Перенавчання - Значно збільшує час, який витрачається 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, адже зазначені підходи мають ряд переваг, які список яких наведено у таблиці 2.2.

Таблиця 2.2 – Переваги використання оверсемплінгу та андерсемплінгу

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

2.2 Методи прогнозування

Існує безліч алгоритмів машинного навчання, які добре підійдуть для задачі класифікації шахрайська/звичайна транзакція, тому були обрані декілька алгоритмів з різних груп (статистичний алгоритм – логістична регресія; градієнтний бустинг, в основі дерева рішень – LightGBM, проста нейронна мережа) для дослідження якості класифікатора побудованого на різних архітектурах.

2.2.1 Логістична регресія

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

В результаті передбачення логістичної регресії отримують вірогідність p від 0 до 1. Що є результатом використання сигмоїдної функції для передбачення та значною перевагою в порівнянні зі схожим алгоритмом – лінійною регресією, результат передбачення якої може виходити за 0 чи 1 (рис. 2.11).

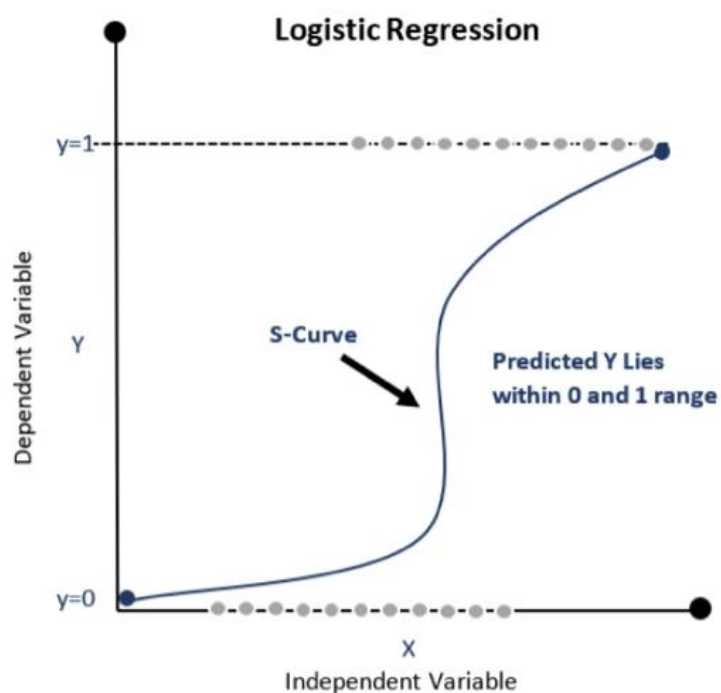


Рис. 2.11 – Передбачення в алгоритмі логістичної регресії [17]

Приналежність же до класу визначається таким чином:

$$\begin{cases} p \geq 0.5, \text{ клас} = 1 \\ p < 0.5, \text{ клас} = 0 \end{cases} \quad (2.2)$$

Формула сигмоїдної функції:

$$S(z) = \frac{1}{1 + e^{-z}} \quad (2.3)$$

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

$$J(\theta) = - \sum_{i=1}^n y_i * \ln(z_i) + (1 - y_i) * \ln(1 - z_i) \quad (2.4)$$

Де

- θ – ваги
- n – кількість прикладів
- $z = h_{\theta}(x) = S(\theta_0 + \theta_1 x_1 + \dots + \theta_b x_b)$
- x – вхідний вектор
- y – вихідний клас

Функція 2.3 була спеціально виведена для використання с сигмоїдною функцією активації, і є інтуїтивно зрозумілою (наприклад, якщо клас $y=1$, то найкращий варіант – щоб z_i (передбачення для поточних ваг) було як можна ближче до 1, тоді $\ln(z_i)$ буде близьким до 0 та значення всієї функції буде вищим, тобто кращим; аналогічно для $y=0$), що є ще одним плюсом використання логістичної регресії.

Як було зазначено раніше, оптимізація 2.3 відбувається за допомогою градієнтного спуску (рис. 2.12). Правило оновлення вагів:

Повторювати поки не зійдеться {

$$\theta_j := \theta_j - \alpha * \frac{\partial}{\partial \theta_j} * J(\theta) \quad (2.5)$$

}

Де α – крок навчання.

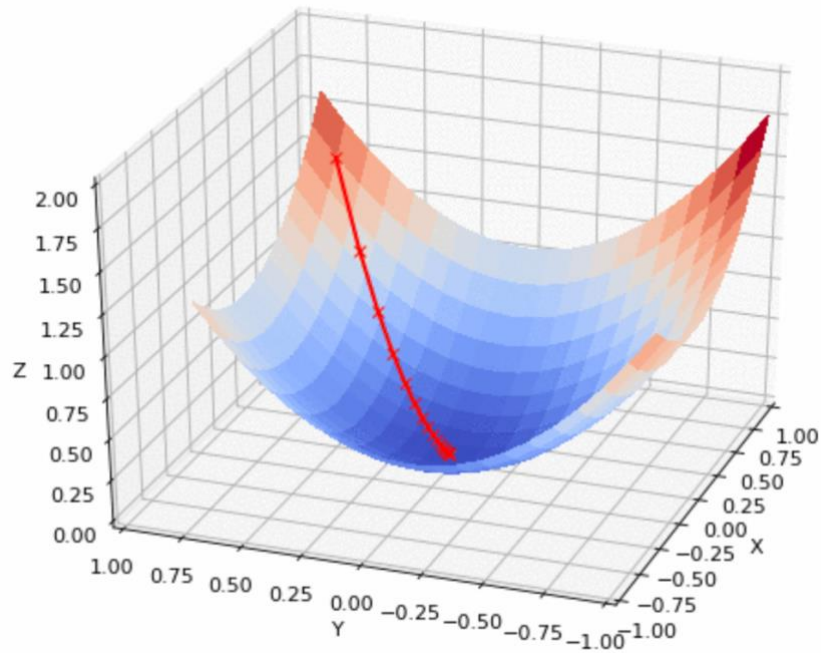


Рисунок 2.12 – Пошук оптимуму градієнтним спуском [18]

2.2.2 LightGBM

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

Підсилювальні (boosting) алгоритми – це сімейство алгоритмів, які з декількох слабких учнів складають сильного учня. Тобто натреновані алгоритми зі слабкими правилами комбінуються у один алгоритм з сильними правилами. Є багато підходів та методів до формування сильних учнів. Один з них – Gradient Boosting Decision Tree, який в якості слабких учнів використовує дерева рішень однакового розміру у ансамблі, щоб зробити передбачення.

Gradient Boosting Decision Tree (GBDT) - це алгоритм машинного навчання, який має багато якісних та ефективних імплементацій, таких як XGBoost та pGBRT. І хоча в XGBoost, pGBRT та інших був зроблений ряд оптимізаційних рішень, такі алгоритми все одно не рекомендується застосовувати на великих об'ємах даних.

Головна причина такої рекомендації – наявні GBDT алгоритми мають проскановувати усі ознаки кожного прикладу, щоб оцінити приріст інформації для усіх можливих точок розбиття, і це забирає дуже багато часу. Щоб вирішити цю проблему, авторами LightGBM були запропоновані дві нові техніки:

- Gradient-based One-Side Sampling (GOSS)
- Exclusive Feature Bundling (EFB)

GOSS (рис. 2.13) використовується для того, щоб виключити більшу кількість прикладів з малими градієнтами, і використовувати тільки залишене для сканування на приріст інформації. Адже, так як саме приклади з найбільшими градієнтами грають важливу роль в підрахунку приросту інформації, виключення інших прикладів з підрахунку не грає великої ролі, і у кінці результат підрахунку є доволі точним.

З EFB (рис. 2.14) знаходять взаємовиключні ознаки (тобто ті, які у сумі рідко дають ненульове значення), щоб зменшити кількість ознак. Пошук таких ознак – це NP повна задача, проте жадібний алгоритм може досягти гарного коефіцієнта наближення.

Як показують експерименти, LightGBM досягає майже однакової точності класифікації як і інші GBDT алгоритми, але навчається в 20 разів швидше (рис. 2.15) [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
 $models \leftarrow \{ \}$ ,  $fact \leftarrow \frac{1-a}{b}$ 
 $topN \leftarrow a \times \text{len}(I)$ ,  $randN \leftarrow b \times \text{len}(I)$ 
for  $i = 1$  to  $d$  do
     $preds \leftarrow models.predict(I)$ 
     $g \leftarrow loss(I, preds)$ ,  $w \leftarrow \{1, 1, \dots\}$ 
     $sorted \leftarrow \text{GetSortedIndices}(\text{abs}(g))$ 
     $topSet \leftarrow sorted[1:topN]$ 
     $randSet \leftarrow \text{RandomPick}(sorted[topN:\text{len}(I)],$ 
     $randN)$ 
     $usedSet \leftarrow topSet + randSet$ 
     $w[randSet] \times = fact \triangleright$  Assign weight  $fact$  to the
    small gradient data.
     $newModel \leftarrow L(I[usedSet], -g[usedSet],$ 
     $w[usedSet])$ 
     $models.append(newModel)$ 

```

Рисунок 2.13. Алгоритм GOSS [19]

Algorithm 3: Greedy Bundling Input: F : features, K : max conflict count Construct graph G $searchOrder \leftarrow G.sortByDegree()$ $bundles \leftarrow \{\}, bundlesConflict \leftarrow \{\}$ for i in $searchOrder$ do $needNew \leftarrow True$ for $j = 1$ to $len(bundles)$ do $cnt \leftarrow ConflictCnt(bundles[j], F[i])$ if $cnt + bundlesConflict[i] \leq K$ then $bundles[j].add(F[i])$, $needNew \leftarrow False$ 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 $binRanges \leftarrow \{0\}$, $totalBin \leftarrow 0$ for f in F do $totalBin += f.numBin$ $binRanges.append(totalBin)$ $newBin \leftarrow new\ Bin(numData)$ for $i = 1$ to $numData$ do $newBin[i] \leftarrow 0$ for $j = 1$ to $len(F)$ do if $F[j].bin[i] \neq 0$ then $newBin[i] \leftarrow F[j].bin[i] + binRanges[j]$ Output: $newBin, binRanges$
---	---

Рисунок 2.14. Алгоритм EFB [19]

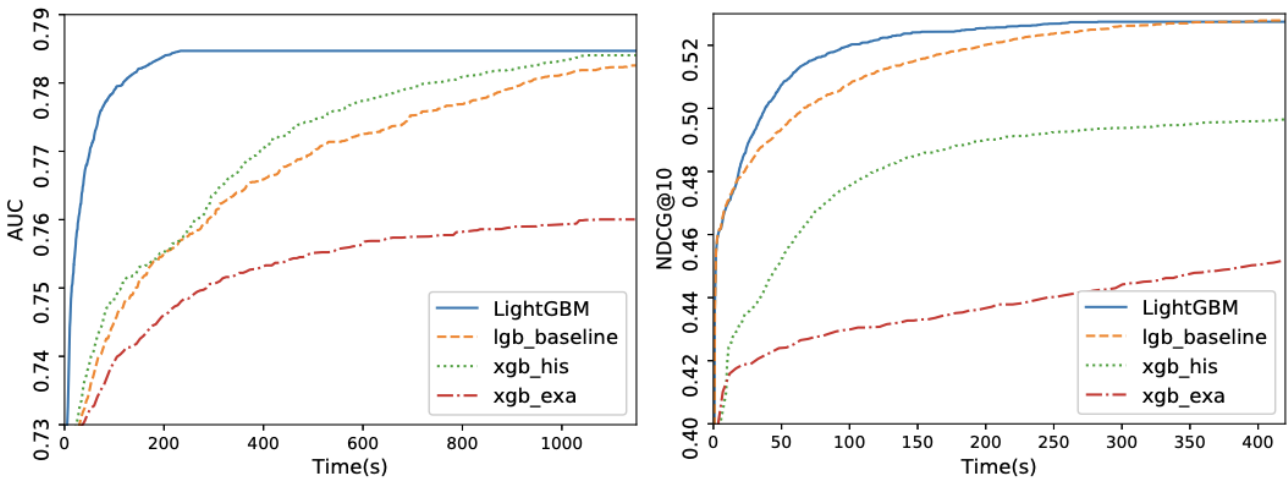


Figure 1: Time-AUC curve on Flight Delay.

Figure 2: Time-NDCG curve on LETOR.

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

2.2.3 Нейронна мережа

Існує безліч алгоритмів навчання заснованих на концепції нейронних мереж, адаптованих під різні задачі, як CNN для розпізнавання зображень, LSTM для роботи з часовими даними. Натомість, для задачі створення класифікатора для запобігання шахрайства у банківському секторі достатньо класичного

алгоритму, який застосовує штучні нейронні мережі, – багатошарового перцептрона (MLP) (рис. 2.16).

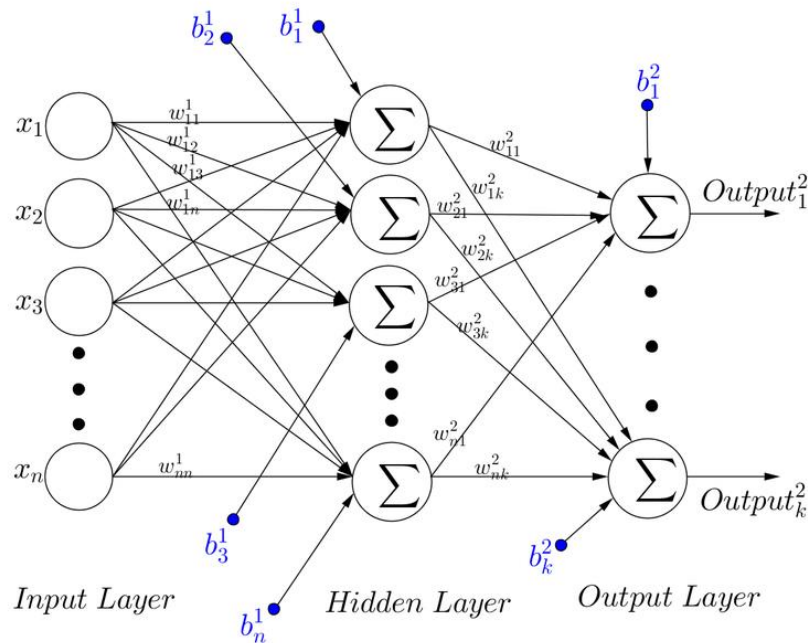


Рисунок 2.16 – Графічне представлення архітектури багатошарової нейронної мережі

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

Спрощено, процес навчання виглядає так:

- 1) Початкові ваги обираються випадково
- 2) Навчальні дані перший раз проходять через мережу та отримуємо перший результат
- 3) Вираховується помилка у обчисленнях, використовуючи певну функцію втрат
- 4) Відштовхуючись від помилки та функції вираховуються часткові похідні, методом градієнтного спуску вирішується в яку сторону рухатись, щоб мінімізувати помилку

5) Увесь процес повторюється до того часу поки помилка не буде мінімальною

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

- кількість прихованих шарів мережі – слід їх додавати поки помилка на тестових даних не перестане значно покращуватись
- виключення (dropout) – деякий відсоток нод нейронної мережі можна вимкнути, щоб не перенавчитися на вхідних даних і побудувати в міру глибоку мережу
- швидкість навчання – більша швидкість змушує модель навчатися швидше, проте можна «перескочити» оптимум
- функція активації – додає нелінійний компонент до моделі, вибір залежить від типу задачі. Наприклад, для бінарної класифікації використовують сигмоїдну функцію активації.

2.3 Оцінка якості отриманого результату

Вибір правильних метрик оцінки якості моделі – надзвичайно важливий крок, коли мова йде про класифікатор, який працює з незбалансованими даними. Адже точність або ROC AUC, які часто використовуються для оцінки класифікаторів, дадуть спотворене представлення про результат, а точніше дадуть занадто позитивну його оцінку. Хибне розуміння метрик призведе до некоректно натренованої моделі.

Для високо незбалансованих даних використовують такі метрики:

- влучність (precision)
- повнота (recall)
- F1-міра (F1-score)
- ВП крива (PR curve, precision recall curve)

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

Влучність намагається відповісти на питання «який відсоток відмічених одиницею на тесті прикладів був відмічений правильно». Модель, яка при передбаченні не здійснила жодної помилки першого роду, матиме влучність 1.0.

Повнота намагається відповісти на питання «який відсоток всіх тестових прикладів класу 1 був відмічений правильно». Модель, яка при передбаченні не здійснила жодної помилки другого роду, матиме повноту 1.0.

$$Precision = \frac{TP}{TP + FP} \quad (2.6)$$

$$Recall = \frac{TP}{TP + FN} \quad (2.7)$$

Де:

- TP (true positives) – к-сть правильно зпрогнозованих прикладів класу 1
- FP (false positives) – к-сть прикладів класу 0, які зпрогнозовані як клас 1
- FN (false negatives) – к-сть прикладів класу 1, які зпрогнозовані як клас 0

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

Усереднену оцінку між влучністю та повнотою можна отримати за допомогою міри F1:

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} \quad (2.8)$$

ВП крива представляє собою графічне зображення компромісу між влучністю та повнотою для певної моделі (рис. 2.17).

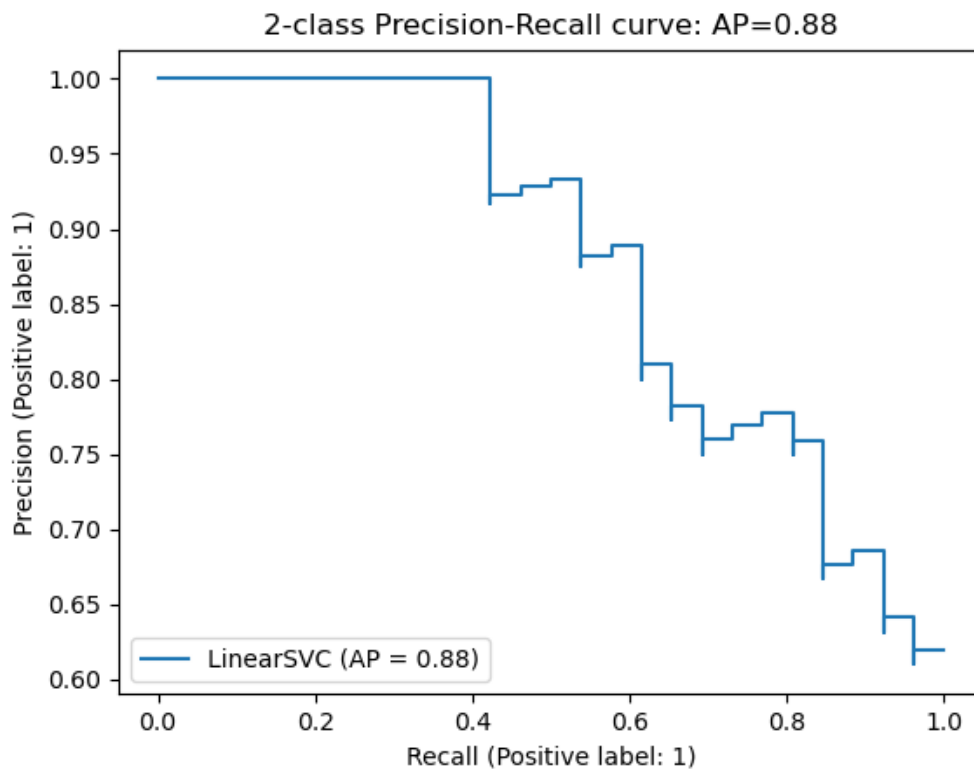


Рисунок 2.17 – ВП крива

Щоб досягти балансу між влучністю та повнотою у задачах з незбалансованими даними необхідно розуміти бізнес завдання. Що важливіше – ідентифікувати більше прикладів меншого класу або менше випадків, але при цьому і менше помилятися. Для задачі з боротьби з шахрайством, на мою думку, вища повнота важливіша за вищу влучність.

2.4 Висновки до розділу 2

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

Було визначено, що для створення вище зазначеної системи, будуть використовуватись 3 алгоритми – логістична регресія, LightGBM, багатошарова нейронна мережа.

Для оцінки якості моделей, зважаючи на незбалансовані дані, обрані такі метрики: влучність, повнота, міра F1.

Для збалансування даних обране поєднання передискретизації за допомогою SMOTE та субдискретизації One-sided Selection.

РОЗДІЛ 3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНІ РЕЗУЛЬТАТИ

3.1 Навчальна вибірка

Обраний для побудови моделі датасет містить транзакції, проведені за допомогою кредитних карток європейськими користувачами у вересні 2013 р. протягом двох днів. Числові дані трансформовані за допомогою PCA налічують 492 шахрайські транзакції та 284315 звичайних.

Кожна транзакція має 30 ознак (рис. 3.1) та значення цільової змінної. Ознаки V1-V28 були трансформовані за допомогою PCA з причин анонімізації даних, ознаки Time (час) та Amount (кількість) (рис. 3.2) залишились незмінними [6].

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	0.078714	0.532028	-0.519511	...	0.052989	0.260903	-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	-1.323866	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	0.100228	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	-0.070549	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

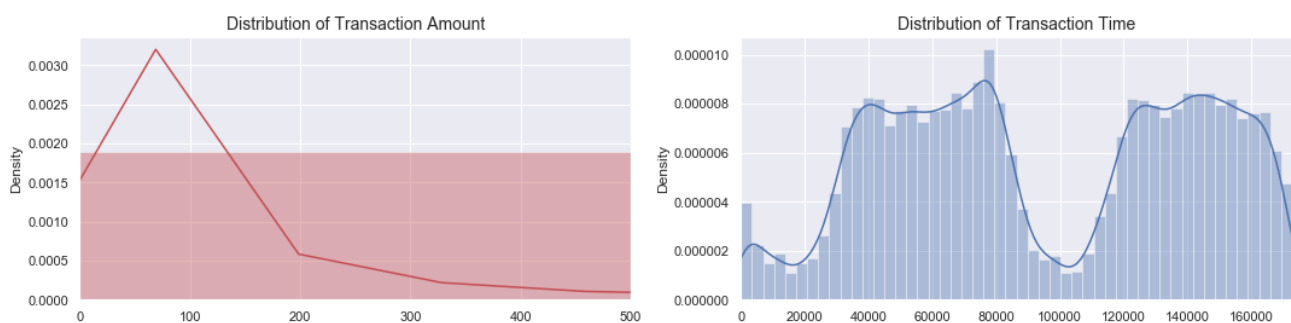


Рисунок 3.2 – Розподіл транзакцій за кількістю та часом

Анонімізовані та трансформовані ознаки, а також мала кількість прикладів шахрайських даних значно ускладнює класифікацію, але більшість датасетів, які використовують у банківських установах для навчання моделей мають саме такий вигляд.

3.2 Обґрунтування вибору мови та платформи програмування

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

Мова Python хоча і посувається швидкістю виконання коду таким популярним мовам як Java, C++ та ін., але має ряд важливих переваг – компактність та простота синтаксису, а значить швидкість розробки, підтримка багатьох парадигм програмування, наявність високої кількості бібліотек для машинного навчання та візуалізації даних.

Jupyter Notebook – це середовище розробки, яке дозволяє виконувати код покроково, причому порядок кроків може змінюватися, а результати виконання кожного кроку записуються глобально, таким чином 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).

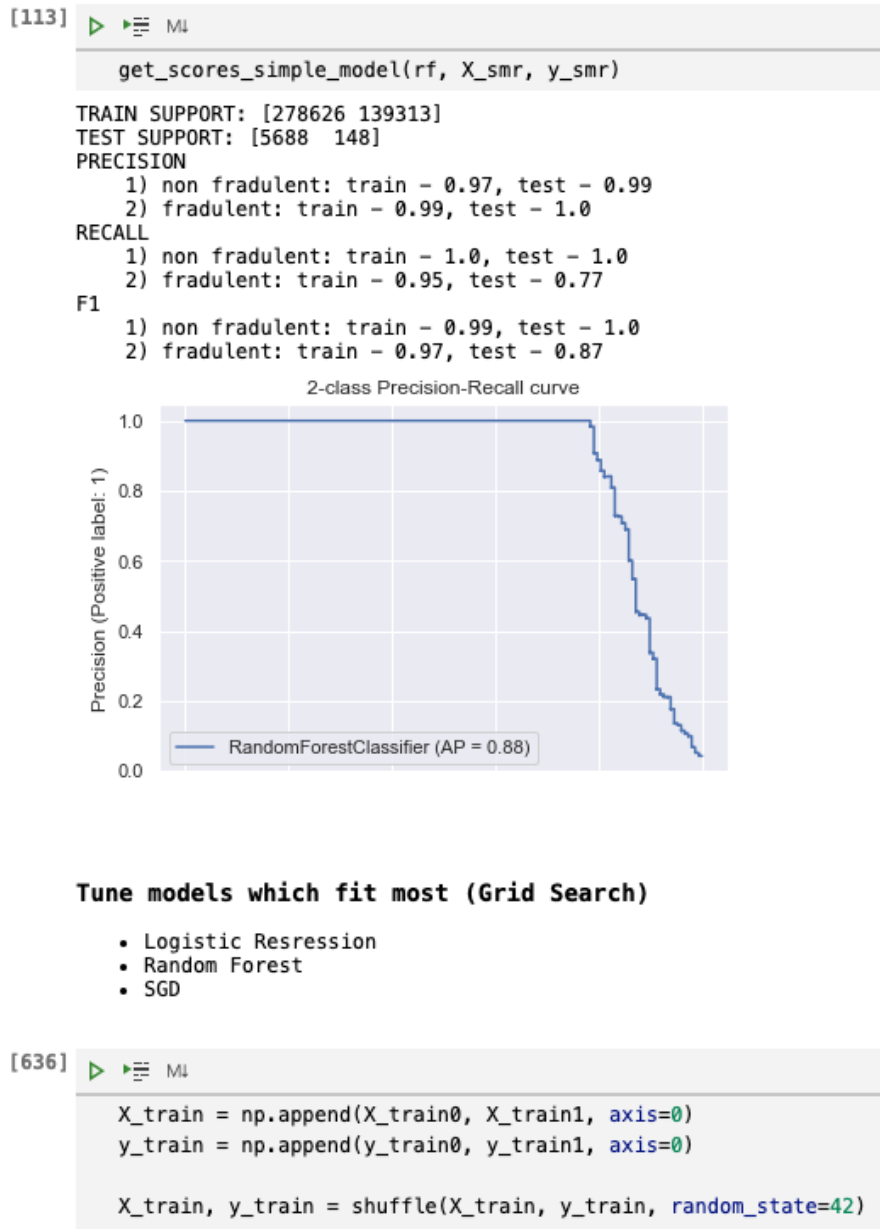


Рисунок 3.3 – Jupyter Notebook та Python 3

Для розробки програмного продукту були використані такі бібліотеки:

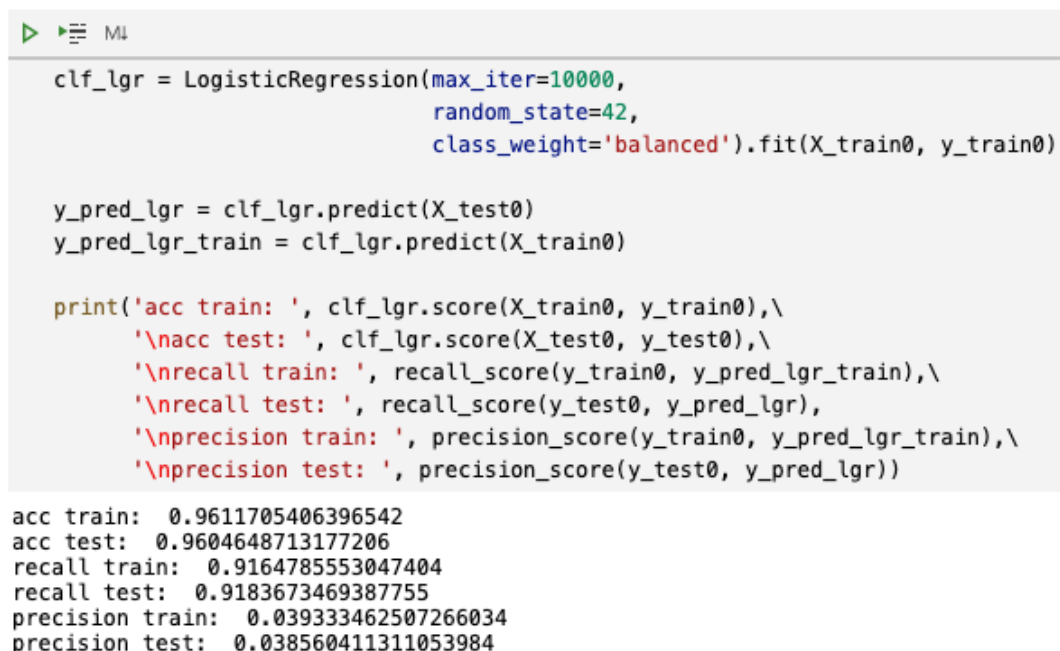
- Matplotlib та Seaborn для візуалізації результатів
- Pandas для зручної роботи з таблицями та обробки csv файлів
- Numpy для швидкої та ефективної роботи з числовими масивами
- Imblearn для оверсамплінгу та андерсамплінгу, побудови пайплайну моделі
- Sklearn для обрахунку метрик, пошуку оптимальних параметрів, побудови моделей

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

3.3 Результати роботи

Для побудови системи з запобігання шахрайству, як зазначено у розділі 2, були обрані декілька відомих архітектур моделей машинного навчання – логістична регресія, lightgbm, нейронна мережа. Тож ці архітектури були взяті за основу досліджень. Також проведені експерименти з різними методами оверсемплінгу та андерсемплінгу, які відіграли важливу роль у збільшенні завданих метрик моделей – влучність, повнота, F1-міра.

З моделлю логістичної регресії на необроблених даних отримані погані показники, зображені на 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:  0.9164785553047404
recall test:  0.9183673469387755
precision train:  0.039333462507266034
precision test:  0.038560411311053984

```

Рисунок 3.4 – Результати тренування логістичної регресії на необроблених даних

Наглядно видно, що попри надвисоку точність (0.96) ця модель є непридатною для використання, адже влучність становить всього 3%. Також відразу видно, що використання логістичної регресії скоріше за все не принесе найкращих результатів, адже показник влучності 3% за використання параметру `class_weight='balanced'` (врахування нерівномірного розподілу класів під час знаходження ваг для моделі) занадто низький навіть враховуючи той факт, що дані не передоброблені.

Для покращення результатів тренування побудована система з підбору оптимальних параметрів заснована на GridSearchCV та комбінації андерсамплінгу та оверсамплінгу SMOTE + RandomUndersampling, схема якої міститься на рис. 3.5.



Рисунок 3.5 – Система з підбору параметрів для моделей

На рис. 3.6 зображений процес підбору параметрів для логістичної регресії:

```

> * ML
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()

> * ML
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
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:

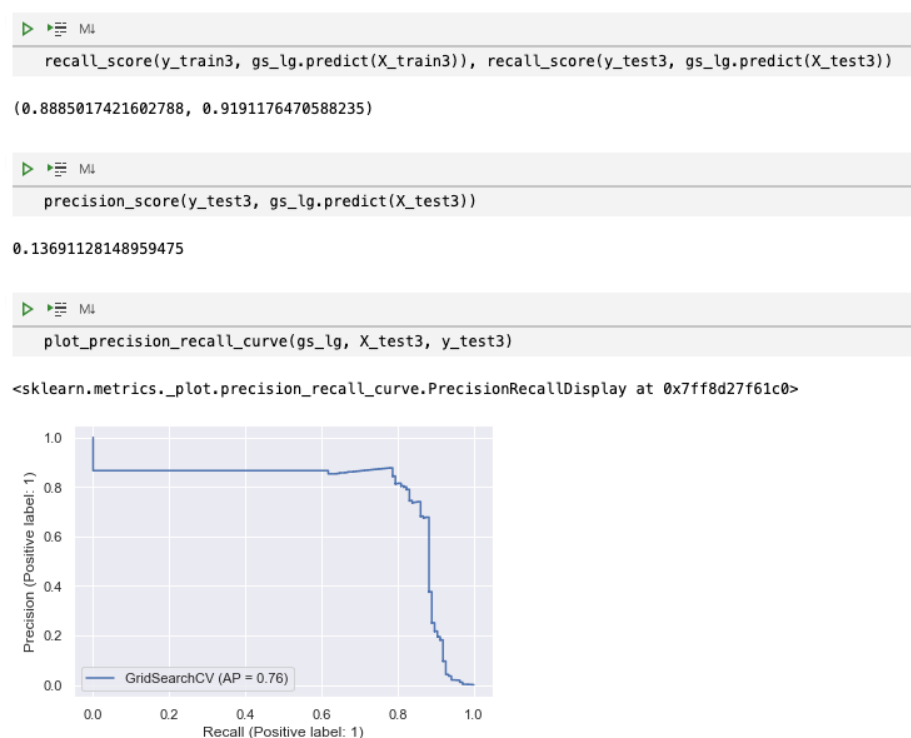


Рисунок 3.7 – Точність, повнота та F1-міра на тренувальній та тестовій вибірках для логістичної регресії

Як і показала попередня модель, очевидно що логістична регресія, на жаль, не підходить як єдиний класифікатор для поставленої задачі. Хоч трохи адекватний результат можна отримати тільки змістивши вірогідність, необхідну для прийняття рішення з 0.5 до 0.99 (рис. 3.8).

```

> ▶ M4
probs_lg = gs_lg.predict_proba(X_test3)

> ▶ M4
def to_labels(pos_probs, threshold):
    return (pos_probs >= threshold).astype('int')

> ▶ M4
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

Підбір параметрів для LightGBM (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_params)

```

Рисунок 3.9 – Підбір параметрів на прикладі LightGBM

Метрики для найкращого LightGBM класифікатора (рис. 3.10):

```

> ▶ M4
recall_score(y_train3, gs_gbm2.predict(X_train3)), recall_score(y_test3, gs_gbm2.predict(X_test3))

(1.0, 0.8823529411764706)

> ▶ M4
precision_score(y_train3, gs_gbm2.predict(X_train3)), precision_score(y_test3, gs_gbm2.predict(X_test3))

(0.9318181818181818, 0.7643312101910829)

> ▶ M4
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 непоганий результат, проте попри намагання уникнути перенавчання регулюванням `num_leaves`, `feature_fraction` та `max_depth` модель сильно перенавчена (особливо на прикладах класу шахрайських транзакцій, що відображає показник повноти 1.0). Отже, й таку модель складно порекомендувати до використання, проте важливим висновком стало те, що бустинги і ансамблі засновані на деревах рішень гарно працюють для даного датасету, потрібно тільки уникнути перенавчання.

Також була побудована нейронна мережа архітектури зображеної на рис. 3.11, яка показала хоч і непоганий, але все ж посередній результат і не мала успіхів у навчанні (рис. 3.12).

Model: "sequential_12"

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

Проаналізувавши попередні результати було вирішено створити ансамбль класифікаторів, у який були би включені алгоритми, які працюють на основі дерев рішень, а попередній досвід у тренуванні LightGBM дозволив визначити параметри класифікаторів не вдаючись до перебору GridSearchCV.

До ансамблю було включено такі моделі:

- LightGBM
- ExtraTreesClassifier
- CatBoostClassifier
- RandomForestClassifier
- LogisticRegression як мета-модель

Схема побудови ансамблю з базових моделей та мета-моделі під назвою out-of-fold stacking (рис. 3.13):

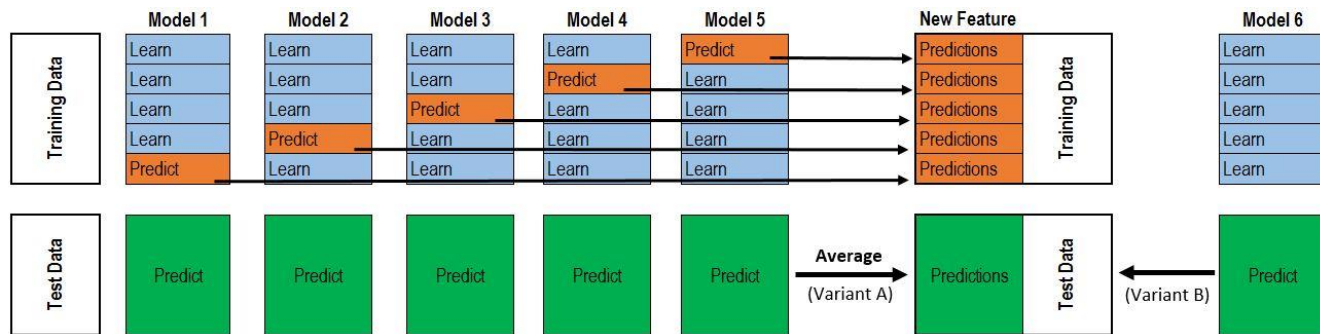


Рисунок 3.13 – OOF-Stacking

Використані параметри для 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,
}

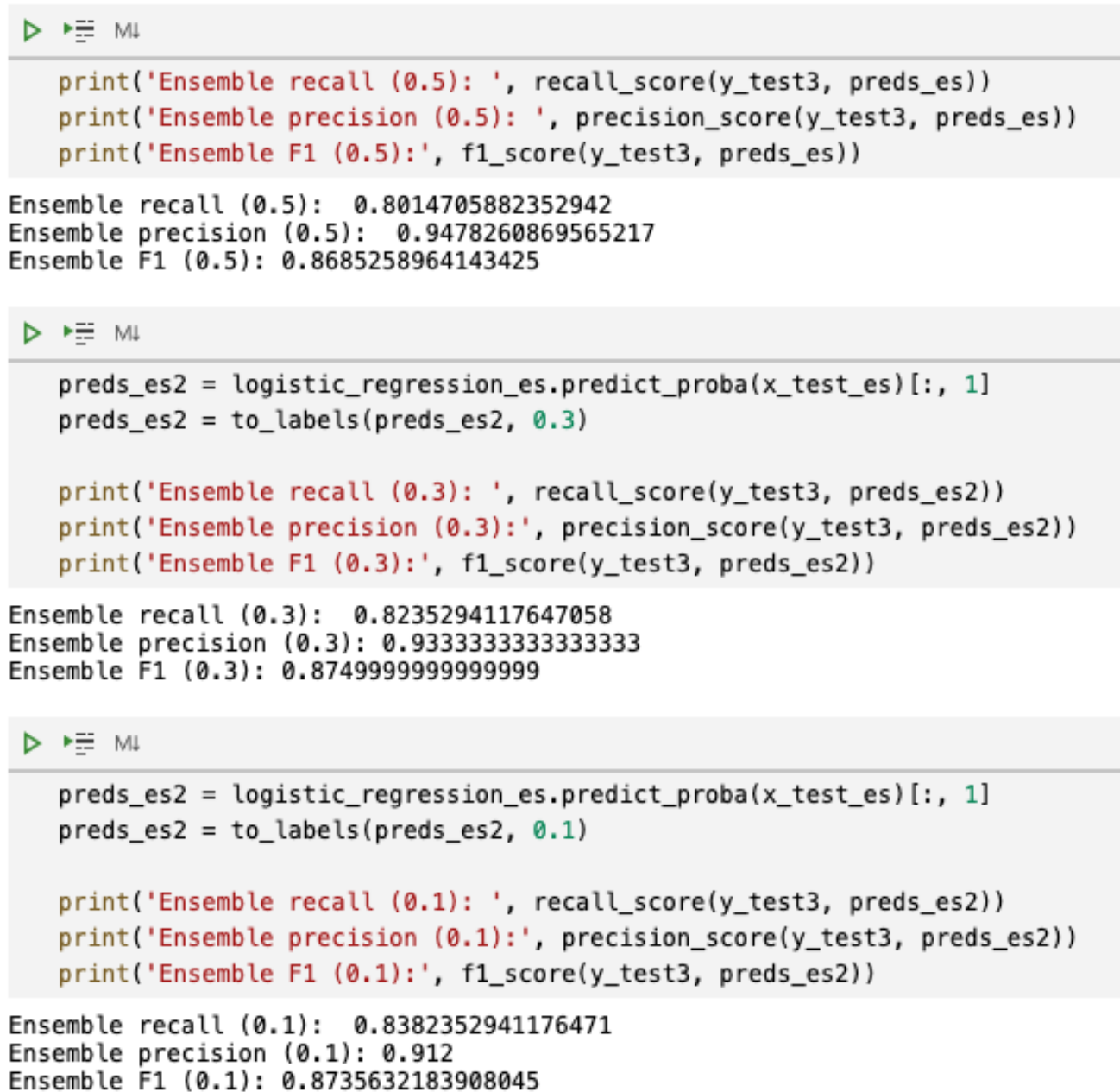
rf_params = { # RandomForestClassifier
    'n_jobs': 16,
    'n_estimators': 200,
    'max_features': 0.2,
    'max_depth': 10,
    'min_samples_leaf': 2,
}

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) дійсно виявилися найкращими з усіх розроблених моделей, і є достатньо гарними, щоб використовувати таку модель у реальному проєкті, відрегулювавши в залежності від потреби поріг прийняття рішень.



```

> ▶ M4

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

> ▶ M4

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

> ▶ M4

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 була створена програмна реалізація для задачі виявлення шахрайства методами штучного інтелекту. Були досліджені різні підходи до побудови такої системи – статистична модель логістична регресія, нейронна мережа, бустинг LightGBM. На основі аналізу їх ефективності за трьома показниками – влучність, повнота, F1-міра, визначено, що для тестових даних найкраще приймає рішення саме LightGBM – бустинг на основі дерев рішень. Модель, однак, сильно перенавчалась навіть попри велику кількість класифікаторів та невелику їх глибину.

Тому був побудований OOF-stacking ансамбль з алгоритмів з деревами рішень в основі. В результаті, отримана влучність у 93%, повнота у 82%.

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У даному розділі проводиться оцінка основних характеристик програмного продукту, розробленого для створення системи для виявлення та запобігання шахрайству в банківському секторі за допомогою методів штучного інтелекту.

Нижче наведено аналіз різних варіантів реалізації модулю з метою вибору оптимальної, з огляду при цьому як на економічні фактори, так і на характеристики продукту, що впливають на продуктивність роботи і на його сумісність з апаратним забезпеченням. Для цього було використано апарат функціонально-вартісного аналізу.

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

Алгоритм функціонально-вартісного аналізу включає в себе визначення послідовності етапів розробки продукту, визначення повних витрат (річних) та кількості робочих часів, визначення джерел витрат та кінцевий розрахунок вартості програмного продукту.

4.1 Постановка задачі проектування

У роботі застосовується метод ФВА для проведення техніко-економічного аналізу розробки системи виявлення та запобігання банківському шахрайству. Оскільки рішення стосовно проектування та реалізації компонентів, що розробляється, впливають на всю систему, кожна окрема підсистема має її

задовольняти. Тому фактичний аналіз представляє собою аналіз функцій програмного продукту, призначеного для збору, обробки та проведення аналізу даних по коронавірусу.

Технічні вимоги до програмного продукту є наступні:

- функціонування на персональних комп'ютерах із стандартним набором компонентів;
- зручність та зрозумілість для користувача;
- швидкість обробки даних та доступ до інформації в реальному часі;
- можливість зручного масштабування та обслуговування;
- мінімальні витрати на впровадження програмного продукту.

4.2 Обґрунтування функцій програмного продукту

Головна функція F_0 – розробка програмного продукту, який вирішує задачу короткострокового прогнозування COVID-19 та будує його модель. Беручи за основу цю функцію, можна виділити наступні:

F_1 – вибір мови програмування;

F_2 – вибір фреймворку машинного навчання;

F_3 – вибір середовища розробки.

Кожна з цих функцій має декілька варіантів реалізації:

Функція F_1 :

а) Python

б) C++

Функція F_2 :

а) Tensorflow;

б) Caffe

Функція F_3 :

а) Jupyter Notebook;

6) Visual Studio.

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

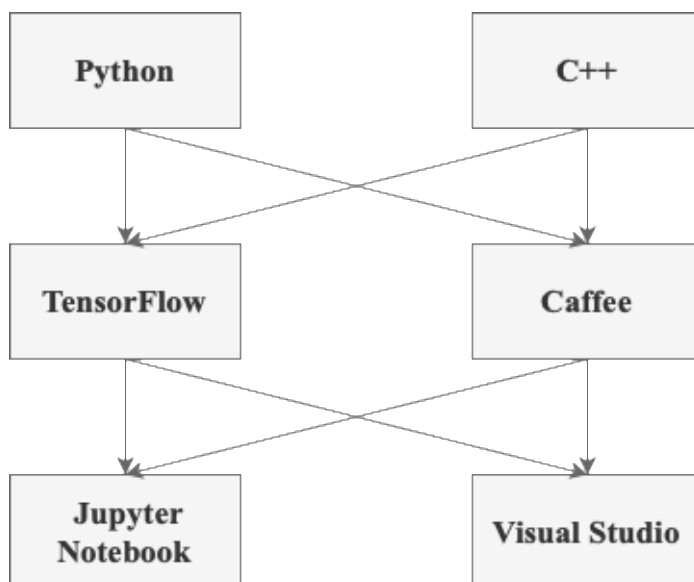


Рисунок 4.1 – Морфологічна карта

Морфологічна карта відображає множину всіх можливих варіанти основних функцій.

Таблиця 4.2 - Позитивно-негативна матриця

Функції	Варіанти реалізації	Переваги	Недоліки
F_1	A	Швидка розробка програми, доступність бібліотек, кросплатформеність	Низька швидкість роботи, особливо, якщо потрібно обробляти велику кількість даних
	B	Код швидко виконується	Іде багато часу на розробку програми

F_2	A	Надійно працює з складними проектами	Не підтримується багатьма мовами
	B	Надійність	Додатковий час на інсталяцію та вивчення
F_3	A	Підтримується багатьма мовами програмування, легко запускається на будь-якому сервері	Відсутня можливість роботи без інтернету
	B	Багато інструментів, безпечна	Підтримує одночасно лише одну мову програмування

На основі цієї карти будемо позитивно-негативну матрицю варіантів основних функцій (Таб.4.2). Робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

Функція F_1 :

Перевагу віддаємо швидкості вивчення, простоті використання та наявності стандартних бібліотек для обчислення. Для спрощення роботи по написанню коду варіант Б має бути відкинтий.

Функція F_2 :

Обидва варіанти можна використовувати в розробці.

Функція F_3 :

Віддаємо перевагу варіанту А в разі вибору мови програмування Python.

Таким чином, будемо розглядати такий варіанти реалізації ПП:

$$F_{1a} - F_{2a} - F_{3a}$$

$$F_{1a} - F_{2б} - F_{3a}$$

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

4.3 Обґрунтування системи параметрів ПП

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:

- X_1 – швидкодія мови програмування;
- X_2 – об'єм пам'яті для обчислень та збереження даних;
- X_3 – час навчання даних;
- X_4 – потенційний об'єм програмного коду.

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію ПП як показано у таблиці 4.4.

Таблиця 4.4 - Основні параметри ПП

Назва Параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Швидкодія мови програмування	X_1	оп/мс	10000	14000	19000
Об'єм пам'яті	X_2	Мб	420	128	64
Час попередньої обробки даних	X_3	мс	4	3	2

Потенційний об'єм програмного коду	X4	кількість рядків коду	4000	2500	1000
------------------------------------	----	-----------------------	------	------	------

За даними таблиці 4.3 будуються графічні характеристики параметрів – рис. 4.2 – рис. 4.5.

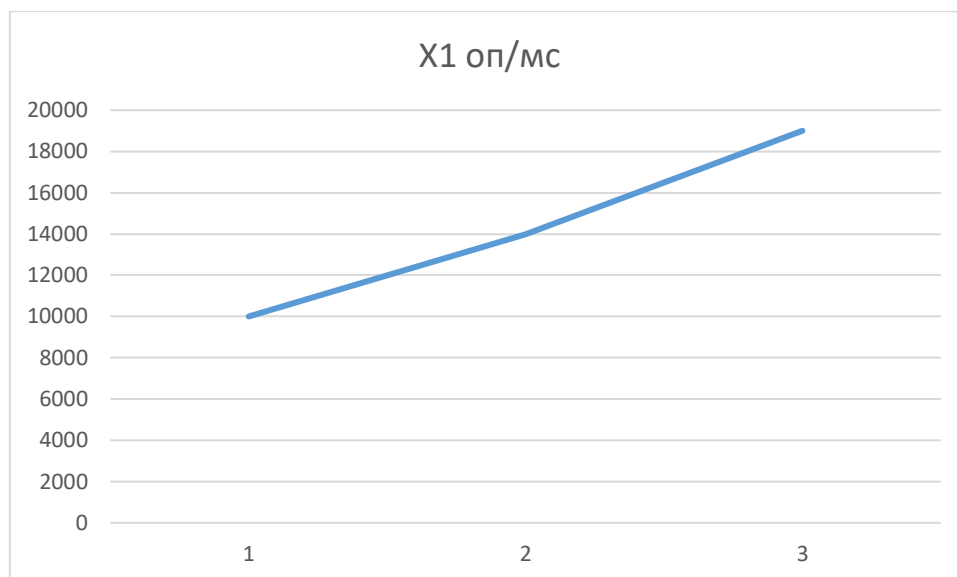


Рисунок 4.2 – X1, швидкодія мови програмування

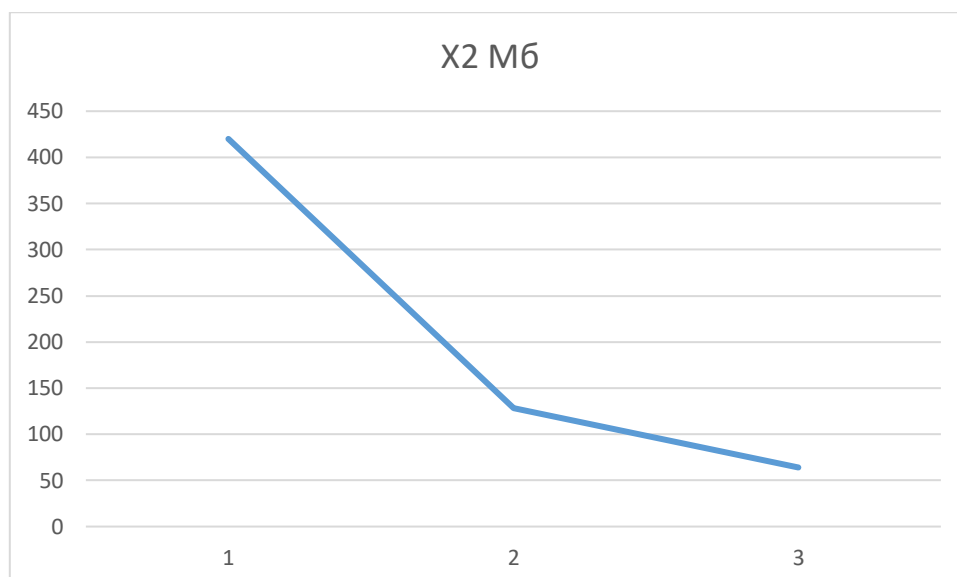


Рисунок 4.3 – X2, об'єм пам'яті

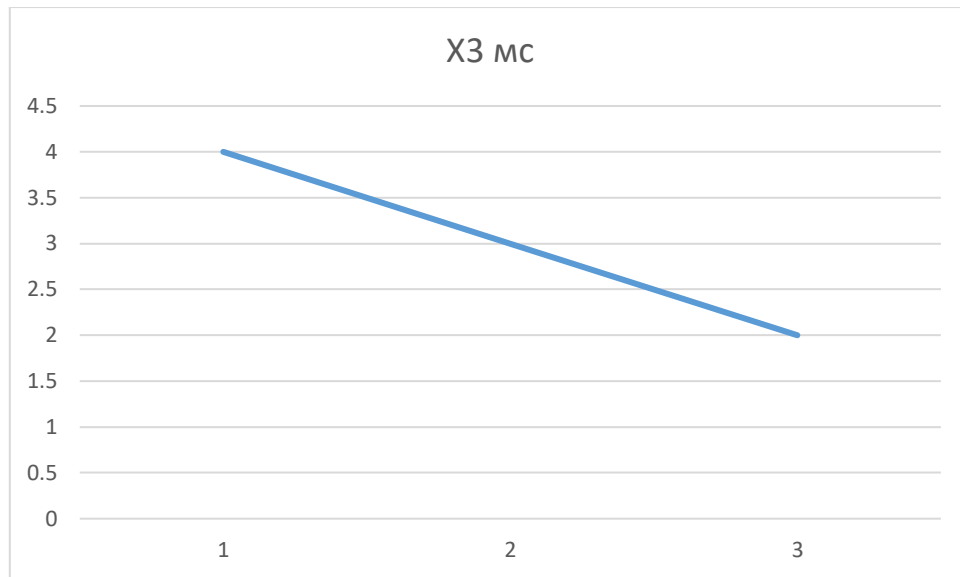


Рисунок 4.4 – X3, час попередньої обробки даних

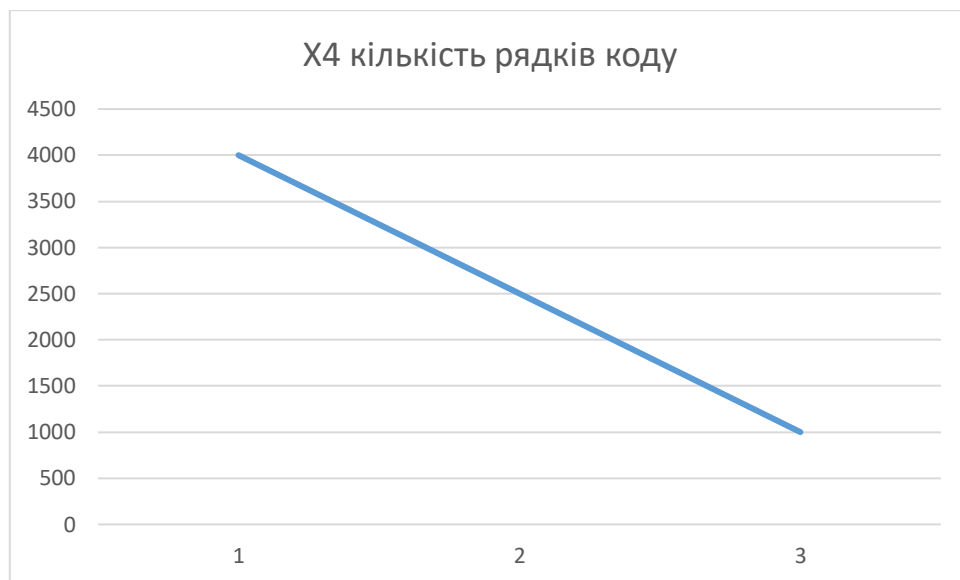


Рисунок 4.5 – X4, потенційний об'єм програмного коду

4.4 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який дає найбільш точні результати при знаходженні параметрів моделей адаптивного прогнозування і обчислення прогнозних значень.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 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.

Результати експертного ранжування наведені у таблиці 4.4.

Таблиця 4.4 - Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X1	Швидкодія мови програмування	Оп/мс	4	5	2	5	3	4	5	28	3,5	12,25
X2	Об'єм пам'яті	Мб	2	1	3	1	2	1	2	12	-12,5	156,25
X3	Час попередньої обробки даних	мс	5	3	5	5	4	5	3	30	5,5	30,25

X4	Потенційний об'єм програмного коду	Кількість рядків коду	3	5	4	3	5	4	4	28	3,5	12,25
	Разом		14	14	14	14	14	14	14	98	0	211

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^N r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 98, \quad (4.1)$$

де N – число експертів,

n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 24,5 \quad (4.2)$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T. \quad (4.3)$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г) загальна сума квадратів відхилення:

$$S = \sum_{i=1}^N \Delta_i^2 = 211. \quad (4.4)$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 211}{7^2(4^3 - 4)} = 0,86 > W_k = 0,67. \quad (4.5)$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 4.5.

Таблиця 4.5 - Попарне порівняння параметрів.

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
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

Числове значення, що визначає ступінь переваги i -го параметра над j -тим, a_{ij} визначається по формулі:

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases} \quad (4.6)$$

З отриманих числових оцінок переваги складемо матрицю $A = \| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості K_{Bi} за наступними формулами:

$$K_{Bi} = \frac{b_i}{\sum_{i=1}^n b_i} \quad (4.7)$$

$$b_i = \sum_{j=1}^N a_{ij} \quad (4.8)$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятись від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{Bi} = \frac{b'_i}{\sum_{i=1}^n b'_i}, \quad (4.9)$$

$$b'_i = \sum_{j=1}^N a_{ij} b_j \quad (4.10)$$

Як видно з таблиці 4.6, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.6 - Розрахунок вагомості параметрів

Параметри x_i	Параметри x_j				Перша ітер.		Друга ітер.		Третя ітер	
	X1	X2	X3	X4	b_i	K_{Bi}	b_i^1	K_{Bi}^1	b_i^2	K_{Bi}^2
X1	1,0	1,5	0,5	1,5	4,5	0,36	17,75	0,25	73,38	0,25
X2	0,5	1,0	1,5	0,5	3,5	0,28	15,75	0,22	65,63	0,23
X3	1,5	1,5	1,0	1,5	5,5	0,44	22,75	0,33	93,6	0,32
X4	0,5	1,5	0,5	1,0	3,5	0,28	13,75	0,2	57,63	0,2

Всього:		12,5	1	70	1	290,25	1
---------	--	------	---	----	---	--------	---

4.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів X_2 (Об'єм пам'яті), X_3 (час попередньої обробки даних) та X_4 (потенційний об'єм програмного коду) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра X_1 (швидкість роботи мови програмування) обрано не найгіршим.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 4.7):

$$K_K(j) = \sum_{i=1}^n K_{ei,j} B_{i,j}, \quad (4.11)$$

де n – кількість параметрів;

K_{ei} – коефіцієнт вагомості i -го параметра;

B_i – оцінка i -го параметра в балах.

Таблиця 4.7 - Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Осн овні функції	Вар іант реалізації функції	Пара метри	Абсол ютне значення параметра	Ба льна оцінка параметр а	Коефі цієнт вагомості параметра	Коефі цієнт рівня якості
F1	A	X1	10000	6	0,25	1,47

F2	A	X2	64	5	0,23	0,98
	Б	X2	128	2	0,32	0,4
F3	A	X3	1000	8	0,2	2,14

За даними з таблиці 4.7 за формулою:

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \dots + K_{\text{ТУ}}[F_{zk}], \quad (4.12)$$

визначаємо рівень якості кожного з варіантів:

$$K_{K1} = 1,47 + 0,98 + 2,14 = 4,59,$$

$$K_{K2} = 1,47 + 0,4 + 2,14 = 4,01.$$

Як видно з розрахунків, кращим є перший варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.6 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Розробка проекту програмного продукту;
2. Розробка програмної оболонки;

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

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

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Загальна трудомісткість обчислюється як

$$T_0 = T_P \cdot K_P \cdot K_{СК} \cdot K_M \cdot K_{СТ} \cdot K_{СТ.М}, \quad (4.13)$$

де T_P – трудомісткість розробки ПП;

K_P – поправочний коефіцієнт;

$K_{СК}$ – коефіцієнт на складність вхідної інформації;

K_M – коефіцієнт рівня мови програмування;

$K_{СТ}$ – коефіцієнт використання стандартних модулів і
прикладних програм;

$K_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: $T_P = 90$ людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $K_P = 1.7$. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $K_{СК} = 1$. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $K_{СТ} = 0.8$. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$T_1 = 90 \cdot 1.7 \cdot 0.8 = 122.4 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $T_P = 27$ людино-днів, $K_P = 0.9$, $K_{СК} = 1$, $K_{СТ} = 0.8$:

$$T_2 = 27 \cdot 0.9 \cdot 0.8 = 19.44 \text{ людино-днів.}$$

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$$T_I = (122.4 + 19.44 + 4.8 + 19.44) \cdot 8 = 1328,64 \text{ людино-годин.}$$

$$T_{II} = (122.4 + 19.44 + 6.91 + 19.44) \cdot 8 = 1345.52 \text{ людино-годин.}$$

Найбільш високу трудомісткість має варіант II.

В розробці беруть участь два програмісти з окладом 27000 грн., один аналітик в області даних з окладом 31500. Визначимо середню зарплату за годину за формулою:

$$C_q = \frac{M}{T_m \cdot t} \text{ грн.,} \quad (4.14)$$

де M – місячний оклад працівників;

T_m – кількість робочих днів тиждень;

t – кількість робочих годин в день.

$$C_q = \frac{27000 + 31500 + 27000}{3 \cdot 21 \cdot 8} = 169,64 \text{ грн.} \quad (4.15)$$

Тоді, розрахуємо заробітну плату за формулою:

$$C_{зп} = C_q \cdot T_i \cdot K_d, \quad (4.16)$$

де C_q – величина погодинної оплати праці програміста;

T_i – трудомісткість відповідного завдання;

K_d – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

$$\text{I.} \quad C_{зп} = 169,64 \cdot 1328.64 \cdot 1.2 = 270468,59 \text{ грн.}$$

$$\text{II.} \quad C_{зп} = 169,64 \cdot 1345.52 \cdot 1.2 = 273904,85 \text{ грн.}$$

Відрахування на єдиний соціальний внесок становить 22%:

$$\text{I. } C_{\text{ВІД}} = C_{\text{ЗП}} \cdot 0.22 = 270468,59 \cdot 0.22 = 59503,09 \text{ грн.}$$

$$\text{II. } C_{\text{ВІД}} = C_{\text{ЗП}} \cdot 0.22 = 273904,85 \cdot 0.22 = 60259,07 \text{ грн.}$$

Тепер визначимо витрати на оплату однієї машино-години. (C_M)

Так як одна ЕОМ обслуговує одного програміста з окладом 27000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$C_G = 12 \cdot M \cdot K_3 = 12 \cdot 27000 \cdot 0,2 = 64800 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$C_{\text{ЗП}} = C_G \cdot (1 + K_3) = 64800 \cdot (1 + 0.2) = 77760 \text{ грн.}$$

Відрахування на соціальний внесок:

$$C_{\text{ВІД}} = C_{\text{ЗП}} \cdot 0.22 = 77760 \cdot 0,22 = 17107,2 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 32000 грн.

$$C_A = K_{\text{ТМ}} \cdot K_A \cdot Ц_{\text{ПР}} = 1.15 \cdot 0.25 \cdot 32000 = 9200 \text{ грн.,}$$

де $K_{\text{ТМ}}$ – коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

K_A – річна норма амортизації;

$Ц_{\text{ПР}}$ – договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$C_P = K_{TM} \cdot C_{ПР} \cdot K_P = 1.15 \cdot 32000 \cdot 0.05 = 11840 \text{ грн.},$$

де K_P – відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$T_{\text{ЕФ}} = (D_K - D_B - D_C - D_P) \cdot t_z \cdot K_B = (365 - 104 - 12 - 16) \cdot 8 \cdot 0.9 = \\ = 1677.6 \text{ годин},$$

де D_K – календарна кількість днів у році;

D_B, D_C – відповідно кількість вихідних та святкових днів;

D_P – кількість днів планових ремонтів устаткування;

t – кількість робочих годин в день;

K_B – коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$C_{\text{ЕЛ}} = T_{\text{ЕФ}} \cdot N_C \cdot K_3 \cdot C_{\text{ЕН}} = 1677.6 \cdot 0.3 \cdot 0.9 \cdot 3.52 = 1594.39 \text{ грн.},$$

де N_C – середньо-споживча потужність приладу;

K_3 – коефіцієнтом зайнятості приладу;

$C_{\text{ЕН}}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$C_H = C_{ПР} \cdot 0.67 = 32000 \cdot 0.67 = 21440 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть:

$$C_{\text{ЕКС}} = C_{\text{ЗП}} + C_{\text{ВІД}} + C_{\text{А}} + C_{\text{Р}} + C_{\text{ЕЛ}} + C_{\text{Н}}, \quad (4.17)$$

$$C_{\text{ЕКС}} = 77760 + 17107,2 + 9200 + 11840 + 1594,39 + 21440 = 138941,59 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$C_{\text{М-Г}} = C_{\text{ЕКС}} / T_{\text{ЕФ}} = 138941,59 / 1677,6 = 82,82 \text{ грн/год.}$$

Оскільки в даному випадку всі роботи, які пов'язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$C_{\text{М}} = C_{\text{М-Г}} \cdot T, \quad (4.18)$$

$$\text{I. } C_{\text{М}} = 82,82 \cdot 1328,64 = 110037,97 \text{ грн.}$$

$$\text{II. } C_{\text{М}} = 82,82 \cdot 1345,52 = 111435,97 \text{ грн.}$$

Накладні витрати складають 67% від заробітної плати:

$$C_{\text{Н}} = C_{\text{ЗП}} \cdot 0,67, \quad (4.19)$$

$$\text{I. } C_{\text{Н}} = 270468,59 \cdot 0,67 = 181213,95 \text{ грн.}$$

$$\text{II. } C_{\text{Н}} = 273904,85 \cdot 0,67 = 183516,25 \text{ грн.}$$

Отже, вартість розробки ПП за варіантами становить:

$$C_{\text{ПП}} = C_{\text{ЗП}} + C_{\text{ВІД}} + C_{\text{М}} + C_{\text{Н}}, \quad (4.20)$$

$$\text{I. } C_{\text{ПП}} = 270468,59 + 59503,09 + 110037,97 + 181213,95 = 621223,6 \text{ грн.}$$

$$\text{II. } C_{\text{ПП}} = 273904,85 + 60259,07 + 111435,97 + 183516,25 = 629116,14 \text{ грн.}$$

4.7 Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$K_{\text{ТЕР}j} = K_{\text{К}j} / C_{\text{Ф}j}, \quad (4.21)$$

$$K_{\text{ТЕР}1} = 4,59/621223,6 = 7,39 \cdot 10^{-6},$$

$$K_{\text{ТЕР}2} = 4,01/629116,14 = 6,37 \cdot 10^{-6}.$$

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $K_{\text{ТЕР}1} = 7,39 \cdot 10^{-6}$.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишилися після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $K_{\text{ТЕР}} = 7,36 \cdot 10^{-6}$.

Цей варіант реалізації програмного продукту має такі параметри:

- мова програмування – Python;
- Використання моделей з великою ємністю
- Використання стандартного інтерфейсу візуалізації, швидкість розробки

Даний варіант виконання програмного комплексу дає користувачу зручний інтерфейс, непоганий функціонал і швидкодію.

4.8 Висновки до розділу 4

Проведено повний функціонально-вартісний аналіз програмного продукту. Визначено та проведено оцінку основних функцій програмного продукту. Визначено параметри, які характеризують програмний продукт. Проведено експертне оцінювання параметрів та аналіз якості варіантів реалізації функцій.

Проведено економічний аналіз варіантів розробки – трудомісткість, витрати на заробітну плату та інші витрати.

На основі аналізу вибрано варіант реалізації програмного продукту.

ВИСНОВКИ

У роботі була визначена актуальність та необхідність дослідження способів створення системи з виявлення та запобігання банківського шахрайства. Були проаналізовані труднощі роботи з даними, які містять, шахрайські та звичайні банківські транзакції. Зокрема, це висока незбалансованість даних та низька кількість прикладів шахрайства, що значно ускладнює побудову моделі машинного навчання. Досліджені методи боротьби з зазначеними проблемами та різні архітектури моделей, які підходять для створення системи з виявлення шахрайства. Визначені методи оцінки для складених моделей машинного навчання.

У практичній частині на прикладі обраного датасету побудовані такі моделі:

- Логістична регресія
- LightGBM
- Нейронна мережа
- Ансамбль з LightGBM, ExtraTreesClassifier, CatBoostClassifier, випадкового лісу, а також логістичної регресії

Ансамбль має найкращі показники: влучність 93%, повнота 82%.

ЛІТЕРАТУРА

1. The multi-faceted threat of fraud. KPMG. URL: <https://home.kpmg/xx/en/home/insights/2019/05/the-multi-faceted-threat-of-fraud-are-banks-up-to-the-challenge-fs.html> (дата звернення: 16.05.2021).
2. Payment Fraud Statistics, Trends & Forecasts (2020). Merchant Savvy. URL: <https://www.merchantsavvy.co.uk/payment-fraud-statistics/> (дата звернення: 16.05.2021).
3. European Central Bank. URL: <https://www.ecb.europa.eu/pub/pdf/cardfraud/ecb.cardfraudreport202008~521edb602b.en.pdf> (дата звернення: 16.05.2021).
4. 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: 16.05.2021).
5. Identity theft and scams: how to get your money back. URL: <https://www.moneyadviceservice.org.uk/en/articles/identity-theft-and-scams-what-you-are-liable-for> (дата звернення: 16.05.2021).
6. Credit Card Fraud Detection. Kaggle: Your Machine Learning and Data Science Community. URL: <https://www.kaggle.com/mlg-ulb/creditcardfraud> (дата звернення: 16.05.2021).
7. 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: 16.05.2021).
8. Brownlee J. Why Is Imbalanced Classification Difficult? Machine Learning Mastery. URL: <https://machinelearningmastery.com/imbalanced-classification-is-hard/> (дата звернення: 16.05.2021).

9. Orduz J. C. Getting Started with Spectral Clustering - KDnuggets. KDnuggets. URL: <https://www.kdnuggets.com/2020/05/getting-started-spectral-clustering.html> (дата звернення: 16.05.2021).
10. 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: 16.05.2021).
11. 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.I3322.1081219> (дата звернення: 16.05.2021).
12. 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: 16.05.2021).
13. minoue-xx/Oversampling-Imbalanced-Data. GitHub. URL: <https://github.com/minoue-xx/Oversampling-Imbalanced-Data> (дата звернення: 16.05.2021).
14. Различные стратегии сэмпинга в условиях несбалансированности классов. BaseGroup Labs. URL: <https://basegroup.ru/community/articles/imbalance-datasets> (дата звернення: 16.05.2021).
15. Brownlee J. Undersampling Algorithms for Imbalanced Classification. *Machine Learning Mastery*. URL: <https://machinelearningmastery.com/undersampling-algorithms-for-imbalanced-classification/> (дата звернення: 16.05.2021).
16. Fernández G. Balancing Techniques. SlidePlayer. URL: <https://slideplayer.com/slide/13128031/> (дата звернення: 16.05.2021).
17. What is the Logistic Regression algorithm and how does it work? *Medium*. URL: <https://medium.com/analytics-vidhya/what-is-the-logistic-regression-algorithm-and-how-does-it-work-92f7394ce761> (дата звернення: 16.05.2021).

18. Kathuria A. Intro to optimization in deep learning: Gradient Descent. Paperspace. URL: <https://blog.paperspace.com/intro-to-optimization-in-deep-learning-gradient-descent/> (дата звернення: 16.05.2021).

19. 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: 16.05.2021).

ДОДАТОК А

Методи штучного інтелекту для виявлення та запобігання шахрайству в банківському секторі

Виконала:

Пашнєва Ольга Миколаївна

Керівник:

д. т. н., проф. Зайченко Олена Юріївна

Постановка задачі

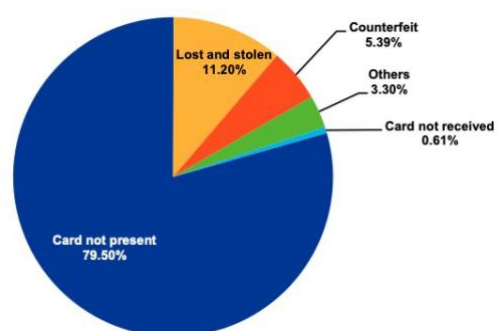
1. Дослідити різні методи семплінга в умовах незбалансованих класів та їх вплив на ефективність моделей
2. Визначити метрики для оцінки моделей, порівняти різні архітектури для вирішення задачі виявлення шахрайства, підібрати оптимальні гіперпараметри
3. Обрати найвдалішу архітектуру, зробити висновки щодо можливості використання отриманої моделі для комерційних задач, описати сильні сторони та недоліки отриманого рішення

Об'єкт дослідження – виявлення шахрайства.

– Мета та цілі роботи – Дослідити існуючі підходи для роботи з незбалансованими даними, різні архітектури моделей машинного навчання та їх ефективність для виявлення банківського шахрайства.

Предмет дослідження – математичні методи аугментації даних, статистичні моделі, методи машинного навчання (зокрема глибокого машинного навчання).

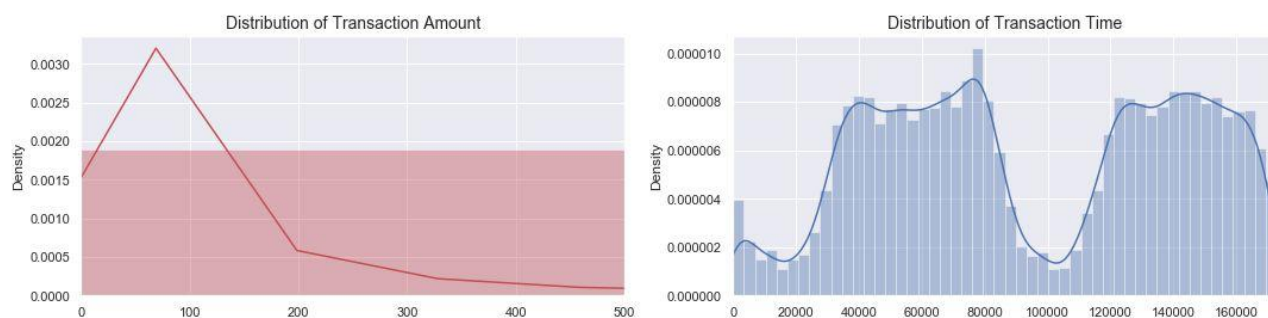
Актуальність роботи



Збитки від банківського шахрайства у Європейському Союзі та розповсюдженість типів шахрайства

Опис даних

Датасет містить транзакції, проведені за допомогою кредитних карток європейськими користувачами у вересні 2013 р. протягом двох днів. Числові дані трансформовані за допомогою PCA налічують 492 шахрайські транзакції та 284315 звичайних. Всього 20 ознак.



Метрики для дослідження

Обрані метрики - Влучність, Повнота, F1-міра

$$Precision = \frac{TP}{TP + FP} \quad (2.5)$$

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

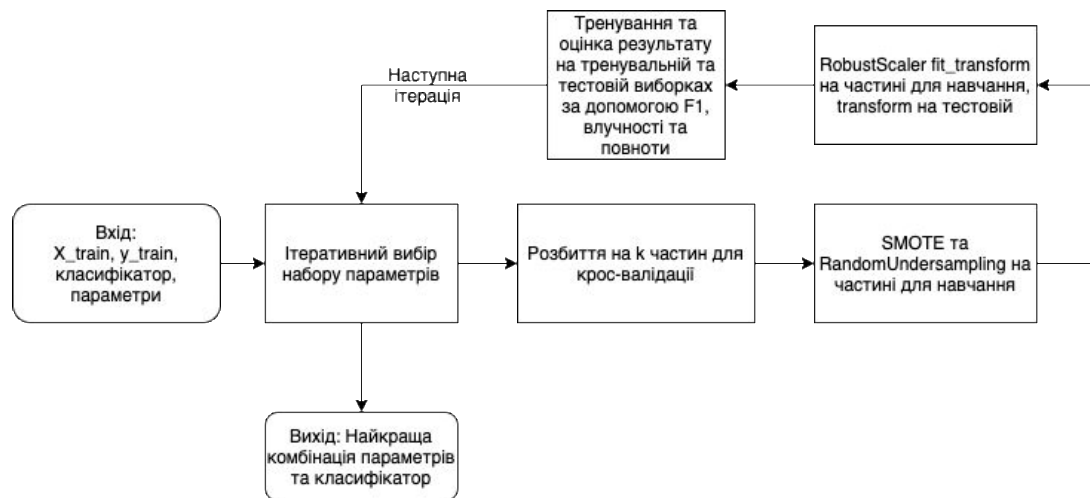
Де:

- TP (true positives) – к-сть правильно спрогнозованих прикладів класу 1
- FP (false positives) – к-сть прикладів класу 0, які спрогнозовані як клас 1
- FN (false negatives) – к-сть прикладів класу 1, які спрогнозовані як клас 0

Усереднену оцінку між влучністю та повнотою можна отримати за допомогою міри F1:

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} \quad (2.7)$$

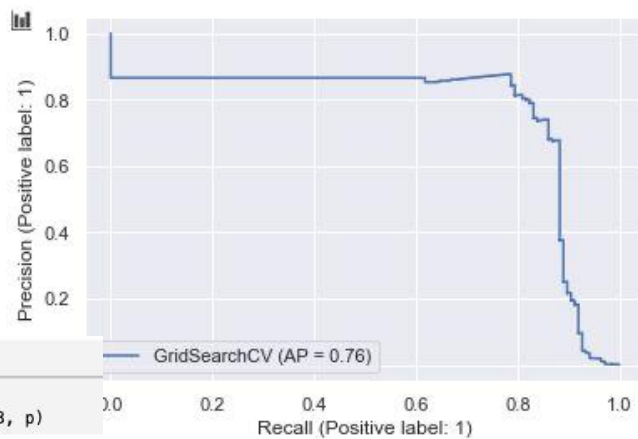
Пошук оптимальних
параметрів



Логістична Регресія

Recall = 0.91

Precision = 0.14



```

> M4
p = to_labels(probs_lg[:, 1], 0.999)
precision_score(y_test3, p), recall_score(y_test3, p)

```

(0.7371794871794872, 0.8455882352941176)

LightGBM

Метрики на навчальній вибірціМетрики на тестовій вибірці

Recall = 1.0

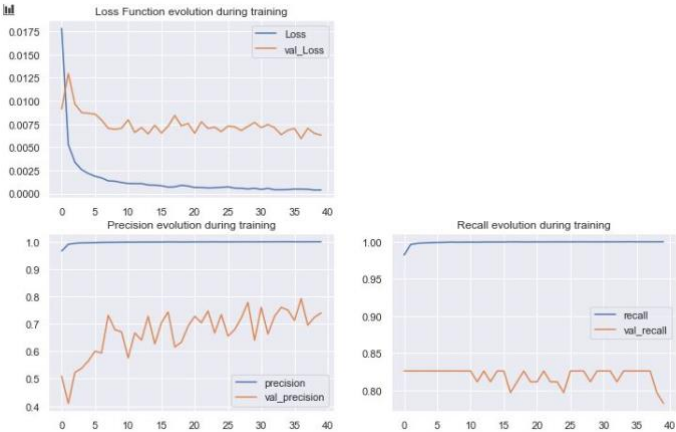
Precision = 0.93

Recall = 0.88

Precision = 0.76

```
gs_gbm2.best_params_  
  
{'clf__boosting_type': 'goss',  
 'clf__class_weight': 'balanced',  
 'clf__feature_fraction': 0.3,  
 'clf__learning_rate': 0.1,  
 'clf__max_depth': 10,  
 'clf__n_estimators': 200,  
 'clf__num_leaves': 70}
```

Нейронна Мережа



Model: "sequential_12"

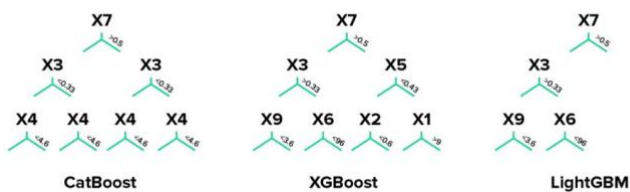
Layer (type)	Output Shape	Param #
dense_60 (Dense)	(None, 256)	7936
batch_normalization_48 (Batch Normalization)	(None, 256)	1024
dropout_41 (Dropout)	(None, 256)	0
dense_61 (Dense)	(None, 256)	65792
batch_normalization_49 (Batch Normalization)	(None, 256)	1024
dropout_42 (Dropout)	(None, 256)	0
dense_62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

Ансамбль

Моделі використані для ансамблю

Tree growth examples:

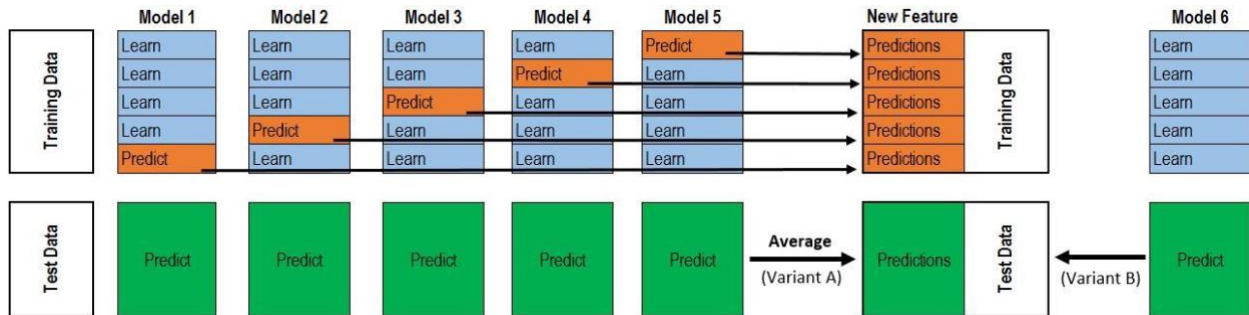
- 1) LightGBM
- 2) ExtraTrees
- 3) RandomForest
- 4) CatBoost



-
- 5) Logistic Regression

	Decision Tree	Random Forest	Extra Trees
Number of trees	1	Many	Many
No of features considered for split at each decision node	All Features	Random subset of Features	Random subset of Features
Boostrapping(Drawing Sampling without replacement)	Not applied	Yes	No
How split is made	Best Split	Best Split	Random Split

OOF- Stacking



1. Ймовірнісний поріг для шахрайських транзакцій > 0.5

Влучність: 0.95

Повнота: 0.8

2. Ймовірнісний поріг для шахрайських транзакцій > 0.3

Влучність: 0.93

Повнота: 0.82

3. Ймовірнісний поріг для шахрайських транзакцій > 0.1

Влучність: 0.91

Повнота: 0.84

```
> %M4
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
```

```
> %M4
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
```

```
> %M4
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
```

Висновки

У роботі були проаналізовані труднощі роботи з даними, які містять, шахрайські та звичайні банківські транзакції. Зокрема, це висока незбалансованість даних, низька кількість прикладів шахрайства, анонімізовані ознаки, що значно ускладнює побудову моделі машинного навчання.

Було розроблено декілька моделей різних архітектур та ансамблю. Визначено, що найкращими методами оцінки моделей є влучність та повнота.

В результаті проведення експериментів визначено, що для наведеної задачі головним фактором, який заважає побудові якісної моделі у випадку незбалансованих даних, є саме недостатня кількість прикладів меншого класу, а не перенавчання на більшому класі як вважається багатьма дослідниками.

На основі ансамблю моделей, що має найкращі показники влучності у 93%, повноти у 82%, створений програмний продукт для розпізнавання та виявлення шахрайства у банківському секторі.

Подальші дослідження

Подальші дослідження включають:

1. Проведення більшої кількості експериментів з методами андерсамплінгу даних
2. Розробку більш складної нейронної мережі з кращими показниками
3. Використання розроблених методів на різних датасетах з даними про банківське шахрайство

Дякую за увагу

ДОДАТОК Б

```
import pandas as pd
import numpy as np
import seaborn as sns
from matplotlib import pyplot as plt
import time
sns.set_theme()

df = pd.read_csv('creditcard.csv')

X = df.drop('Class', axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import recall_score, precision_score

X_train0, X_test0, y_train0, y_test0 = train_test_split(df.drop('Class', axis=1).values,
df['Class'].values,          random_state=42,          shuffle=True,          test_size=0.1,
stratify=df['Class'].values)
X_train0.shape, X_test0.shape
pd.Series(y_test0).value_counts()

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))

# ### Derive test data

from sklearn.utils import shuffle

df0 = df.loc[df['Class']==0]
df1 = df.loc[df['Class']==1]

test_size0 = 0.01
test_size1 = 0.15

X_train0, X_test0, y_train0, y_test0 = train_test_split(df0.drop('Class', axis=1).values,
df0['Class'].values,

                shuffle=False, test_size=test_size0)

X_train0, X_val0, y_train0, y_val0 = train_test_split(X_train0, y_train0, shuffle=False,
test_size=test_size0/(1-test_size0))

X_train1, X_test1, y_train1, y_test1 = train_test_split(df1.drop('Class', axis=1).values,
df1['Class'].values,

                shuffle=False, test_size=test_size1)

X_train1, X_val1, y_train1, y_val1 = train_test_split(X_train1, y_train1, shuffle=False,
test_size=test_size1/(1-test_size1))


X_train0.shape,  X_test0.shape,  X_val0.shape,  X_train1.shape,  X_test1.shape,
X_val1.shape

X_train = np.append(X_train0, X_train1, axis=0)
y_train = np.append(y_train0, y_train1, axis=0)
X_train, y_train = shuffle(X_train, y_train, random_state=42)

```

```

X_test = np.append(X_test0, X_test1, axis=0)
y_test = np.append(y_test0, y_test1, axis=0)
X_test, y_test = shuffle(X_test, y_test, random_state=42)

X_val = np.append(X_val0, X_val1, axis=0)
y_val = np.append(y_val0, y_val1, axis=0)
X_val, y_val = shuffle(X_val, y_val, random_state=42)
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from imblearn.combine import SMOTEENN
from imblearn.pipeline import Pipeline as ImblearnPipeline

df_sme = pd.read_csv('data_sme.csv', index_col=0)
X_sme = df_sme.drop('Class', axis=1).values
y_sme = df_sme['Class'].values
pd.Series(y_sme).value_counts()
over = SMOTE(sampling_strategy=0.5)
under = RandomUnderSampler(sampling_strategy=0.5)
steps = [('o', over), ('u', under)]
pipeline = ImblearnPipeline(steps=steps)
X_smr, y_smr = pipeline.fit_resample(X_train, y_train)

pd.Series(y_smr).value_counts()

from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import RobustScaler
ct = ColumnTransformer([
    ('scale', RobustScaler(), [0, 29])

```

```

    ], remainder='passthrough')
ct = ct.fit(X_train)

X_train = ct.transform(X_train)
X_test = ct.transform(X_test)
X_val = ct.transform(X_val)

X_train.shape, X_test.shape, X_val.shape
X_sme = ct.transform(X_sme)
X_smr = ct.transform(X_smr)

X_test_no_vl = np.append(X_test, X_val, axis=0)
y_test_no_vl = np.append(y_test, y_val, axis=0)

from sklearn.pipeline import Pipeline
from sklearn.model_selection import cross_validate
from sklearn.metrics import precision_recall_fscore_support, average_precision_score,
plot_precision_recall_curve

def print_results(model, X_train, X_test, y_train, y_test):
    y_pred_train = model.predict(X_train)
    y_pred_test = model.predict(X_test)
    scores_train = precision_recall_fscore_support(y_train, y_pred_train)
    scores_test = precision_recall_fscore_support(y_test, y_pred_test)

    print(f'TRAIN SUPPORT: {scores_train[-1]}')
    print(f'TEST SUPPORT: {scores_test[-1]}')

    print('PRECISION')

```

```

print(f'      1) non fraudulent: train - {round(scores_train[0][0], 2)}, test -
{round(scores_test[0][0], 2)}')
print(f'      2) fraudulent: train - {round(scores_train[0][1], 2)}, test -
{round(scores_test[0][1], 2)}' )

print('RECALL')
print(f'      1) non fraudulent: train - {round(scores_train[1][0], 2)}, test -
{round(scores_test[1][0], 2)}')
print(f'      2) fraudulent: train - {round(scores_train[1][1], 2)}, test -
{round(scores_test[1][1], 2)}' )

print('F1')
print(f'      1) non fraudulent: train - {round(scores_train[2][0], 2)}, test -
{round(scores_test[2][0], 2)}')
print(f'      2) fraudulent: train - {round(scores_train[2][1], 2)}, test -
{round(scores_test[2][1], 2)}' )

# print('AVERAGE PRECISION SCORE')
# print(f' - train {round(average_precision_score(y_train, y_pred_train), 2)}')
# print(f' - test {round(average_precision_score(y_test, y_pred_test), 2)}')

disp = plot_precision_recall_curve(model, X_test, y_test)
disp.ax_.set_title(f'2-class Precision-Recall curve')
plt.show()

def get_scores_simple_model(model, X_train, y_train, X_test=X_test_no_vl,
y_test=y_test_no_vl, print_scores=True):
    model.fit(X_train, y_train)
    if print_scores:

```

```

    print_results(model, X_train, X_test, y_train, y_test)
    return model, model.predict(X_train), model.predict(X_test)

# %% [markdown]
# ##### Logistic Regression

# %%
lg = LogisticRegression(max_iter=10000, random_state=42, class_weight='balanced')

# %%
lg_fitted, y_pred_train, y_pred_test = get_scores_simple_model(lg, X_sme, y_sme)

# %%
y_pred_test[0:30]

# %%
y_test[0:30]

# %%
get_scores_simple_model(lg, X_smr, y_smr)

# %% [markdown]

# %%
from sklearn.ensemble import GradientBoostingClassifier

```

```
# %%
```

```
gbc = GradientBoostingClassifier(n_estimators=10)
```

```
# %%
```

```
get_scores_simple_model(gbc, X_sme, y_sme)
```

```
# %%
```

```
get_scores_simple_model(gbc, X_smr, y_smr)
```

```
# %% [markdown]
```

```
# ##### Random Forest
```

```
# %%
```

```
from sklearn.ensemble import RandomForestClassifier
```

```
# %%
```

```
rf = RandomForestClassifier(max_depth=5, random_state=0,
class_weight='balanced_subsample')
```

```
# %%
```

```
get_scores_simple_model(rf, X_sme, y_sme)
```

```

# %%
get_scores_simple_model(rf, X_smr, y_smr)

# %% [markdown]
# ### Tune models which fit most (Grid Search)

# %%
d2 = pd.read_csv("creditcard.csv")

scalar = StandardScaler()

X3 = d2.drop('Class', axis=1)
y3 = d2.Class

X_train_v3, X_test3, y_train_v3, y_test3 = train_test_split(X3, y3,
                                                            test_size=0.3, random_state=42)
X_train3, X_validate3, y_train3, y_validate3 = train_test_split(X_train_v3, y_train_v3,
                                                                test_size=0.2, random_state=42)

over = SMOTE(sampling_strategy=0.4)
steps = [('o', over)]
pipeline = ImblearnPipeline(steps=steps)
X_train3_smote, y_train3_smote = pipeline.fit_resample(X_train3, y_train3)

X_train3 = scalar.fit_transform(X_train3)
X_validate3 = scalar.transform(X_validate3)
X_test3 = scalar.transform(X_test3)
X_train3_smote = scalar.transform(X_train3_smote)

```

```

w_p = y_train3_smote.value_counts()[0] / len(y_train3_smote)
w_n = y_train3_smote.value_counts()[1] / len(y_train3_smote)

print(f"Fraudulent transaction weight: {w_n}")
print(f"Non-Fraudulant transaction weight: {w_p}")
# make cross validation pipeline (SMOTE with RANDOM undersampling)! + grid
search
from sklearn.metrics import make_scorer, accuracy_score, auc, f1_score
from sklearn.model_selection import GridSearchCV, StratifiedKFold

def build_smote_pipeline(clf, X_train, y_train, X_test, y_test, param_grid):
    ct = ColumnTransformer([
        ('scale', RobustScaler(), [0, 29]) # scale only Time and Amount, as the rest have
been scaled initially
    ], remainder='passthrough')
    oversample = SMOTE(sampling_strategy=0.3, random_state=1)
    undersample = RandomUnderSampler(sampling_strategy=0.5, random_state=1)

    pipeline_model = ImblearnPipeline(steps = [('smote', oversample),
                                                ('undersample', undersample),
                                                ('scaler', ct),
                                                ('clf', clf)])

    stratified_kfold = StratifiedKFold(n_splits=2)

    scoring = {'f1': make_scorer(f1_score),
               'precision': make_scorer(precision_score),
               'recall': make_scorer(recall_score),
               'accuracy': make_scorer(accuracy_score)}

```

```

grid_search = GridSearchCV(estimator=pipeline_model,
                           param_grid=param_grid,
                           scoring=scoring,
                           refit='f1',
                           cv=stratified_kfold,
                           error_score=0.0,
                           n_jobs=1,
                           verbose=10,
                           return_train_score=True)

```

```

grid_search.fit(X_train, y_train)

```

```

cv_best_score = grid_search.best_score_

```

```

test_score = grid_search.score(X_test, y_test)

```

```

print(f'Cross-validation score: {cv_best_score}\nTest score: {test_score}')

```

```

return grid_search, grid_search.cv_results_

```

```

def plot_grid_search(cv_results, grid_param_1, grid_param_2, name_param_1,
                    name_param_2, mask):

```

```

    # Get Test Scores Mean and std for each grid search

```

```

    scores_recall = cv_results['mean_test_recall']

```

```

    scores_recall =
```

```

    np.array(scores_recall)[mask].reshape(len(grid_param_2),len(grid_param_1))

```

```

    scores_precision = cv_results['mean_test_accuracy']

```

```

    scores_precision =
```

```

    np.array(scores_precision)[mask].reshape(len(grid_param_2),len(grid_param_1))

```

```

# Plot Grid search scores
_, ax = plt.subplots(1,1)

# Param1 is the X-axis, Param 2 is represented as a different curve (color line)
for idx, val in enumerate(grid_param_2):
    s = scores_recall[idx,:]
    s[s == 0] = np.nan
    ax.scatter(grid_param_1, scores_recall[idx,:], label= name_param_2 + ': ' +
str(val))

ax.set_title("Grid Search Scores Recall", fontsize=20, fontweight='bold')
ax.set_xlabel(name_param_1, fontsize=16)
ax.set_ylabel('CV Average Score', fontsize=16)
ax.legend(loc="best", fontsize=15)
ax.grid('on')

# Plot Grid search scores
_, ax2 = plt.subplots(1,1)

# Param1 is the X-axis, Param 2 is represented as a different curve (color line)
for idx, val in enumerate(grid_param_2):
    s = scores_precision[idx,:]
    s[s == 0] = np.nan
    ax2.scatter(grid_param_1, scores_precision[idx,:], label= name_param_2 + ': ' +
str(val))

ax2.set_title("Grid Search Scores Accuracy", fontsize=20, fontweight='bold')
ax2.set_xlabel(name_param_1, fontsize=16)

```

```

ax2.set_ylabel('CV Average Score', fontsize=16)
ax2.legend(loc="best", fontsize=15)
ax2.grid('on')
# #### LogReg
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)
param_clf__C_l1_mask = np.ma.masked_where(gs_lg_report['param_clf__penalty']
== 'l1', gs_lg_report['param_clf__C']).mask
param_clf__solver_l2_mask =
np.ma.masked_where(gs_lg_report['param_clf__penalty'] == 'l2',
gs_lg_report['param_clf__solver']).mask
gs_lg_report['mean_test_recall'][param_clf__C_l1_mask]

plot_grid_search(gs_lg_report, log_reg_params['clf__C'],
log_reg_params['clf__solver'], 'clf__C', 'clf__solver', param_clf__C_l1_mask)
plot_grid_search(gs_lg_report, log_reg_params['clf__C'],
log_reg_params['clf__solver'], 'clf__C', 'clf__solver', param_clf__solver_l2_mask)
gs_lg.best_params_

lg_best = gs_lg.best_estimator_
recall_score(y_train3, gs_lg.predict(X_train3)), recall_score(y_test3,
gs_lg.predict(X_test3))

```

```

precision_score(y_test3, gs_lg.predict(X_test3))

plot_precision_recall_curve(gs_lg, X_test3, y_test3)
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)
# ##### LightGBM
from lightgbm import LGBMClassifier
gs_gbm2.best_params_

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_params)
```

```
gs_gbm2
```

```
gs_gbm_report2
```

```
gs_gbm2.best_params_
```

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

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

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

```
probs_gbm = gs_gbm2.best_estimator_.predict_proba(X_test3)
```

```
p = to_labels(probs_gbm[:, 1], 0.9)
```

```
precision_score(y_test3, p), recall_score(y_test3, p)
```

```
# ##### Neural Networks
```

```
from tensorflow import keras
```

```
from sklearn.preprocessing import StandardScaler
```

```
model = keras.Sequential([
```

```
    keras.layers.Dense(256, activation='relu', input_shape=(X_train3_smote.shape[-
1])),
```

```
    keras.layers.BatchNormalization(),
```

```
    keras.layers.Dropout(0.3),
```

```
    keras.layers.Dense(256, activation='relu'),
```

```
    keras.layers.BatchNormalization(),
```

```
    keras.layers.Dropout(0.3),
```

```

keras.layers.Dense(256, activation='relu'),
keras.layers.BatchNormalization(),
keras.layers.Dropout(0.3),
keras.layers.Dense(256, activation='relu'),
keras.layers.BatchNormalization(),
keras.layers.Dense(1, activation='sigmoid'),
])

```

```

model.summary()
METRICS = [
    keras.metrics.FalseNegatives(name='fn'),
    keras.metrics.FalsePositives(name='fp'),
    keras.metrics.TrueNegatives(name='tn'),
    keras.metrics.TruePositives(name='tp'),
    keras.metrics.Precision(name='precision'),
    keras.metrics.Recall(name='recall')
]

```

```

model.compile(optimizer=keras.optimizers.Adam(1e-3), loss='binary_crossentropy',
metrics=METRICS)

```

```

callbacks = [keras.callbacks.ModelCheckpoint('fraud_model_at_epoch_{epoch}.h5')]

```

```

r = model.fit(
    X_train3_smote, y_train3_smote,
    validation_data=(X_validate3, y_validate3),
    batch_size=1024,
    epochs=40,
    callbacks=callbacks,

```

```

class_weight={0:w_n, 1:w_p}
)

score = model.evaluate(X_test3, y_test3)
print(score)
plt.figure(figsize=(12, 16))

plt.subplot(4, 2, 1)
plt.plot(r.history['loss'], label='Loss')
plt.plot(r.history['val_loss'], label='val_Loss')
plt.title('Loss Function evolution during training')
plt.legend()

plt.subplot(4, 2, 3)
plt.plot(r.history['precision'], label='precision')
plt.plot(r.history['val_precision'], label='val_precision')
plt.title('Precision evolution during training')
plt.legend()

plt.subplot(4, 2, 4)
plt.plot(r.history['recall'], label='recall')
plt.plot(r.history['val_recall'], label='val_recall')
plt.title('Recall evolution during training')
plt.legend()

# #### Try ensembles
from scipy.stats import skew
import xgboost as xgb
from sklearn.model_selection import KFold

```

```
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import mean_squared_error
from math import sqrt
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import KFold
from catboost import CatBoostClassifier
import gc
```

```
SEED = 42
```

```
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
}

```

```

kf = KFold(n_splits = 2, shuffle=True, random_state=SEED)

```

```

class SklearnWrapper(object):
    def __init__(self, clf, seed=SEED, params=None):
        params['random_state'] = seed
        self.clf = clf(**params)

    def train(self, x_train, y_train):

```

```
self.clf.fit(x_train, y_train)
```

```
def predict(self, x):
    return self.clf.predict_proba(x)[:,-1]
```

```
class CatboostWrapper(object):
    def __init__(self, clf, seed=SEED, params=None):
        params['random_seed'] = seed
        self.clf = clf(**params)
```

```
def train(self, x_train, y_train):
    self.clf.fit(x_train, y_train)
```

```
def predict(self, x):
    return self.clf.predict_proba(x)[:,-1]
```

```
class LightGBMWrapper(object):
    def __init__(self, clf, seed=SEED, params=None):
        params['feature_fraction_seed'] = seed
        params['bagging_seed'] = seed
        self.clf = clf(**params)
```

```
def train(self, x_train, y_train):
    self.clf.fit(x_train, y_train)
```

```
def predict(self, x):
    return self.clf.predict_proba(x)[:,-1]
```

```
def get_oof(clf, x_train, y_train, x_test, y_test):
```

```

oof_train = np.zeros((x_train.shape[0],))
oof_test = np.zeros((y_test.shape[0],))
oof_test_skf = np.empty((2, y_test.shape[0]))

```

```

for i, (train_index, test_index) in enumerate(kf.split(x_train)):

```

```

    x_tr = x_train.loc[train_index]

```

```

    y_tr = y_train.loc[train_index]

```

```

    x_te = x_train.loc[test_index]

```

```

    clf.train(x_tr, y_tr)

```

```

    oof_train[test_index] = clf.predict(x_te)

```

```

    oof_test_skf[i, :] = clf.predict(x_test)

```

```

oof_test[:] = oof_test_skf.mean(axis=0)

```

```

return oof_train.reshape(-1, 1), oof_test.reshape(-1, 1)

```

```

et = SklearnWrapper(clf=ExtraTreesClassifier, seed=SEED, params=et_params)

```

```

rf = SklearnWrapper(clf=RandomForestClassifier, seed=SEED, params=rf_params)

```

```

cb = CatboostWrapper(clf= CatBoostClassifier, seed = SEED,
params=catboost_params)

```

```

lg = LightGBMWrapper(clf = LGBMClassifier, seed = SEED, params =
lightgbm_params)

```

```

et_oof_train, et_oof_test = get_oof(et, pd.DataFrame(np.append(X_train3,
X_validate3, axis=0)), pd.Series(np.append(y_train3, y_validate3, axis=0)), X_test3,
y_test3)

```

```

rf_oof_train, rf_oof_test = get_oof(rf, pd.DataFrame(np.append(X_train3, X_validate3,
axis=0)), pd.Series(np.append(y_train3, y_validate3, axis=0)), X_test3, y_test3)

```

```

cb_oof_train, cb_oof_test = get_oof(cb, pd.DataFrame(np.append(X_train3,
X_validate3, axis=0)), pd.Series(np.append(y_train3, y_validate3, axis=0)), X_test3,
y_test3)
lg_oof_train, lg_oof_test = get_oof(lg, pd.DataFrame(np.append(X_train3,
X_validate3, axis=0)), pd.Series(np.append(y_train3, y_validate3, axis=0)), X_test3,
y_test3)
print("ET-CV: {}".format(sqrt(mean_squared_error(pd.Series(np.append(y_train3,
y_validate3, axis=0)), et_oof_train))))
print("RF-CV: {}".format(sqrt(mean_squared_error(pd.Series(np.append(y_train3,
y_validate3, axis=0)), rf_oof_train))))
print("CB-CV: {}".format(sqrt(mean_squared_error(pd.Series(np.append(y_train3,
y_validate3, axis=0)), cb_oof_train))))
print("LG-CV: {}".format(sqrt(mean_squared_error(pd.Series(np.append(y_train3,
y_validate3, axis=0)), lg_oof_train))))
x_train_es = np.concatenate((lg_oof_train, et_oof_train, rf_oof_train, cb_oof_train),
axis=1)
x_test_es = np.concatenate((lg_oof_test, et_oof_test, rf_oof_test, cb_oof_test), axis=1)
print("{}{}".format(x_train_es.shape, x_test_es.shape))
logistic_regression_es = LogisticRegression()
logistic_regression_es.fit(x_train_es, pd.Series(np.append(y_train3, y_validate3,
axis=0)))
preds_es = logistic_regression_es.predict(x_test_es)
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))
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))
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))
```