Міністерство освіти і науки України

Український державний університет науки і технологій

Факультет Комп'ютерні технології та системи
Кафедра Комп'ютерні інформаційні технології

Пояснювальна записка

до кваліфікаційної роботи
магістра

на тему: «Дослідження алгоритмів рекомендаційних систем»

за освітньою програмою **12 Інженерія програмного забезпечення**
зі спеціальності: **121 Інженерія програмного забезпечення**

Виконав: студент групи «ПЗ2321» _____ /Олександр КЕСАР/

Керівник: _____ /доц. Тетяна ГРИШЕЧКІНА/

Нормоконтролер: _____ /Світлана ВОЛКОВА/

Засвідчую, що у цій роботі немає запозичень з праць інших авторів без відповідних посилань
Студент _____
(підпис)

Дніпро – 2025 рік

Ministry of Education and Science of Ukraine

Ukrainian State University of Science and Technologies

Faculty Computer technologies and systems
Department Computer information technology

Explanatory Note

to Master's Thesis

on the topic: «Research of recommender system algorithms»

according to educational curriculum **12 software engineering**
in the Speciality: **121 software engineering**

Done by the student of the group PZ2222: Oleksandr KESAR/

Scientific Supervisor: /Tetyana Grischechkina/

Normative controller: /Svitlana VOLKOVA/

Dnipro – 2024

Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет: Комп'ютерних технологій і систем
Кафедра: Комп’ютерні інформаційні технології
Рівень вищої освіти: магістр
Освітня програма: Інженерія програмного забезпечення
Спеціальність: Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ____КІТ
_____ Вадим ГОРЯЧКІН
_____ грудня 2023 р.

ЗАВДАННЯ

На кваліфікаційну роботу ____Магістр____
студенту_Кесар Олександру Романовичу____.
1. Тема дипломної роботи: «Дослідження алгоритмів рекомендаційних систем».
Керівник роботи:Гришечкіна Тетяна Сергіївна
затверджені наказом 1196 ст від 05.12.2022 року
2. Строк подання студентом роботи ___.01.2024 року
3. Вихідні дані до дипломної роботи:
Дані з платформи MyAnimeList, що включають рейтинги та взаємодії.
4. Зміст пояснювальної записки (перелік питань до розробки):
4.1. Аналітична частина: Аналіз сучасного стану дослідження за науковими літературними джерелами;
4.2. Основна частина: Часове дослідження роботи ДОБД;
5. Перелік демонстраційного матеріалу:
5.1. презентація;
5.2. демонстраційне відео.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз сучасного стану дослідження проблеми за науковими літературними джерелами	01.09.23 – 01.10.23	10%
2	Аналіз сучасного стану програмно-апаратного забезпечення, яке потребує вдосконалення для вирішення проблем дослідження	01.10.22 – 10.10.22	
3	Постановка задачі, технічне завдання	01.09.23 – 15.09.23	30%
4	Розробка інструментальних засобів дослідження	15.09.23 – 01.11.23	
5	Виконання досліджень	01.10.23 – 01.11.23	60%
6	Оформлення пояснювальної записки	01.11.23 – 01.01.24	
7	Розробка демонстраційних матеріалів	10.01.24 – 15.01.24	100%
8	Подання кваліфікаційної роботи до кафедри	18.01.24	
9	Захист кваліфікаційної роботи на засіданні Екзаменаційної комісії	25.01.24	

Студент: _____ Олександр КЕСАР

Керівник роботи: _____ доц. Тетяна ГРИШЕЧКІНА

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи магістра складається з 89 с., 33 рис. , 12 табл., 3 додатків, 27 джерел.

Об'єктом дослідження алгоритми рекомендаційних систем.

Мета роботи: дослідити ефективність роботи алгоритмів рекомендаційних систем у різних умовах. Провести аналіз їхньої продуктивності, точності рекомендацій, часу тренування та використання обчислювальних ресурсів. Розробити рекомендації щодо вибору алгоритму залежно від специфіки завдань і обсягу даних.

Методика дослідження: теоретичний аналіз літературних джерел, розробка програмного забезпечення з використанням обраних алгоритмів, тестування моделей на реальних даних, порівняння їхніх кількісних та якісних характеристик.

Перелік ключових слів: рекомендаційні системи, градієнтний бустинг, матрична факторизація, швидке дерево Твідді, продуктивність алгоритмів, генерація рекомендацій.

ЗМІСТ

ВСТУП

Рекомендаційні системи сьогодні відіграють ключову роль у цифровому світі, впливаючи на спосіб прийняття рішень користувачами в різних галузях – від електронної комерції до онлайн-освіти. Зі збільшенням обсягів даних та різноманітності користувацьких уподобань постала проблема ефективного підбору релевантної інформації для кожного окремого користувача. Традиційні методи фільтрації даних часто демонструють низьку продуктивність у випадках, коли доступні дані є фрагментарними, а уподобання користувачів змінюються динамічно.

Сучасні підходи, зокрема алгоритми колаборативної фільтрації, контентного підходу та їх гібридні варіанти, мають значний потенціал, проте стикаються з низкою обмежень [1]. Наприклад, колаборативна фільтрація залежить від наявності великої кількості взаємодій між користувачами і об'єктами, а методи на основі контенту вразливі до проблеми недостатності даних (cold-start). Гібридні підходи намагаються поєднати сильні сторони цих методів, але їх реалізація вимагає складних обчислень і масштабованих рішень.

У контексті України ця проблематика набуває особливого значення з огляду на стрімкий розвиток цифрових платформ, включаючи електронну торгівлю, медіа-сервіси та освітні ресурси. Ефективні рекомендаційні системи можуть суттєво покращити конкурентоспроможність українських компаній на глобальному ринку, а також сприяти підвищенню якості сервісів для місцевих споживачів.

Існуючі дослідження в цій галузі демонструють багатообіцяючі результати, але потребують адаптації до специфічних умов України, включаючи мовний бар’єр, культурні особливості та економічні реалії [2]. Тому подальше вдосконалення алгоритмів з урахуванням локальних викликів та потреб є не лише науково значущим, а й практично доцільним.

Об’єкт дослідження – алгоритми рекомендаційних систем, такі як градієнтний бустинг, матрична факторизація та швидке дерево Твідді, зокрема

їх ефективність у задачах генерації рекомендацій на основі великих наборів даних.

Предмет дослідження – особливості роботи різних алгоритмів рекомендаційних систем, їх застосування в залежності від типу даних, а також підходи до їхньої програмної реалізації та тестування в умовах практичних завдань.

Мета дослідження – дослідити ефективність роботи алгоритмів рекомендаційних систем у різних умовах. Провести аналіз їхньої продуктивності, точності рекомендацій, часу тренування та використання обчислювальних ресурсів.

Для досягнення поставленої мети, необхідно виконати наступні завдання:

– провести огляд алгоритмів рекомендаційних систем, проаналізувати їх класифікацію, особливості роботи та практичні сфери застосування;

– вивчити переваги та недоліки різних алгоритмів (градієнтного бустингу, матричної факторизації, швидкого дерева Твідді) з урахуванням специфіки оброблюваних даних;

– розробити програмне забезпечення для реалізації та тестування обраних алгоритмів, забезпечивши їхню адаптацію до реальних умов;

– виконати порівняльний аналіз результатів роботи алгоритмів за критеріями точності, швидкості роботи та вимог до ресурсів, а також сформулювати рекомендації щодо їх застосування в різних завданнях.

РОЗДІЛ 1 ОГЛЯД АЛГОРИТМІВ РЕКОМЕНДАЦІЙНИХ СИСТЕМ

1.1 Сутність рекомендаційних систем та їх роль у сучасних технологіях

Рекомендаційні системи є складовою частиною сучасних інформаційних технологій, які забезпечують автоматизований підбір релевантного контенту, продуктів чи послуг для користувачів. Сутність цих систем полягає у зборі, обробці та аналізі даних про користувачів і предмети, щоб сформувати рекомендації, які максимально відповідають вподобанням та потребам кожного конкретного споживача. Рекомендаційні системи є ключовим інструментом персоналізації в цифровому середовищі, спрямованим на зменшення інформаційного перевантаження і підвищення задоволеності користувачів.

Основний принцип роботи рекомендаційних систем полягає у використанні спеціалізованих алгоритмів для аналізу поведінки користувачів, їхніх уподобань, а також характеристик об’єктів, які підлягають рекомендації. Ці алгоритми дозволяють системі виявляти закономірності в поведінці користувачів та ефективно підбирати найбільш релевантний контент, продукти або послуги [3].

Рекомендаційні системи зазвичай використовують два основних типи даних:

– історичні дані – це інформація про минулі дії користувача: перегляди товарів, історія покупок, поставлені оцінки чи залишені відгуки. Наприклад, у сервісі Amazon система аналізує історію покупок користувача і пропонує товари, схожі на вже придбані, або ті, які купували інші користувачі з подібною поведінкою;

– поточні дії – це інформація про дії користувача в реальному часі, як-от додавання товарів до кошика, пошукові запити чи клік на певні елементи інтерфейсу. Наприклад, YouTube аналізує відео, які переглядає користувач, щоб одразу пропонувати схожий контент [4].

У сучасних технологіях роль рекомендаційних систем виходить за межі простого інструменту покращення користувацького досвіду. Вони є невід'ємною частиною бізнес-процесів яких наведено на рис. 1.1.

Рис. 1.1 – Галузі рекомендаційних систем

До основних галузей рекомендаційних систем належать:

– електронна комерція. Платформи, такі як Amazon чи Rozetka, використовують рекомендації для збільшення продажів через персоналізовані пропозиції товарів;

– медіасервіси. Сервіси, такі як Netflix чи YouTube, пропонують контент на основі переглядів користувачів, що сприяє тривалому утриманню аудиторії.

– освітні платформи. Coursera чи Khan Academy використовують рекомендаційні системи для підбору курсів, які відповідають рівню знань та інтересам користувачів;

– соціальні мережі. Facebook, Instagram та TikTok активно застосовують рекомендаційні алгоритми для формування стрічки новин або рекомендацій друзів.

Роль рекомендаційних систем є особливо важливою у сучасному цифровому світі, де обсяги доступної інформації та даних постійно зростають.

Сучасні користувачі стикаються з викликом вибору серед безлічі варіантів, які пропонують електронна комерція, медіасервіси, освітні платформи та соціальні мережі. У таких умовах простий пошук інформації може перетворитися на тривалий і ресурсозатратний процес. Рекомендаційні системи вирішують цю проблему шляхом автоматизації пошуку релевантного контенту, що дозволяє значно зменшити час, необхідний для прийняття рішень, і водночас зробити їх більш обґрунтованими [5].

Наприклад, в електронній комерції такі системи аналізують історію покупок і взаємодії користувачів із платформою, щоб запропонувати товари, які максимально відповідають їхнім вподобанням. Це дозволяє компаніям не тільки збільшувати кількість транзакцій, але й стимулювати покупців до повторних замовлень. У медіасервісах, таких як Netflix або Spotify, рекомендаційні алгоритми допомагають формувати персоналізовані списки відтворення або каталоги фільмів, підвищуючи лояльність користувачів і тривалість їх взаємодії із сервісом.

Згідно з дослідженням, опублікованим на платформі ScienceDirect, впровадження таких систем може призвести до збільшення обсягів продажів на 20-30% [5]. Це досягається завдяки персоналізованим рекомендаціям, які стимулюють споживачів до здійснення додаткових покупок. Окрім того, рекомендаційні системи сприяють зниженню витрат на залучення нових клієнтів. Використовуючи наявні дані про поведінку та вподобання споживачів, компанії можуть ефективніше таргетувати свою аудиторію, що зменшує потребу в дорогих маркетингових кампаніях.

У конкурентному середовищі рекомендаційні системи надають стратегічну перевагу, дозволяючи компаніям виділятися серед конкурентів та швидше адаптуватися до змін у поведінці споживачів. Це підтверджується дослідженням, яке підкреслює важливість таких систем для підвищення задоволеності клієнтів та лояльності до бренду.

Більше того, рекомендаційні системи стають важливим інструментом у вирішенні інформаційної складності. У сучасних умовах надлишку даних вони

не тільки допомагають користувачам приймати рішення, але й сприяють підвищенню ефективності бізнес-процесів, створюючи умови для максимальної реалізації потенціалу цифрових платформ. Таким чином, роль рекомендаційних систем виходить за межі персоналізації – вони стають ключовим елементом екосистеми цифрових технологій, що забезпечують ефективну взаємодію між користувачами і сервісами в умовах стрімкого розвитку інформаційного середовища.

1.2 Огляд сучасних підходів до розробки рекомендаційних систем

Розробка рекомендаційних систем є однією з найважливіших галузей досліджень у сфері штучного інтелекту та машинного навчання. Існує декілька основних підходів, які визначають спосіб формування рекомендацій, їх точність і здатність адаптуватися до змінних умов. Розглянемо найпоширеніші сучасні підходи до створення рекомендаційних систем.

Колаборативна фільтрація є одним із найпоширеніших підходів у розробці рекомендаційних систем. Вона базується на аналізі вподобань користувачів і спрямована на пошук закономірностей у їхній поведінці. Основна ідея цього методу полягає у використанні схожих профілів користувачів або об'єктів для формування персоналізованих рекомендацій. Наприклад, якщо два користувачі демонструють схожі вподобання до певних товарів чи фільмів, система може рекомендувати їм об'єкти, які сподобалися одному з них, але ще не оцінені іншим. Цей метод знайшов застосування у сферах, де немає явних характеристик об'єктів, таких як медіа-платформи, освітні ресурси чи онлайн-магазини [6].

На рис. 1.2 наведено схему реалізації колаборативної фільтрації, яка ілюструє, як взаємодія між користувачами та об’єктами використовується для побудови рекомендацій. У цій схемі зображено процес, у якому система аналізує зв'язки між користувачами (User A, User B, User C) та об'єктами (Item A, Item B, Item C, Item D) для виявлення схожості й генерації рекомендацій.

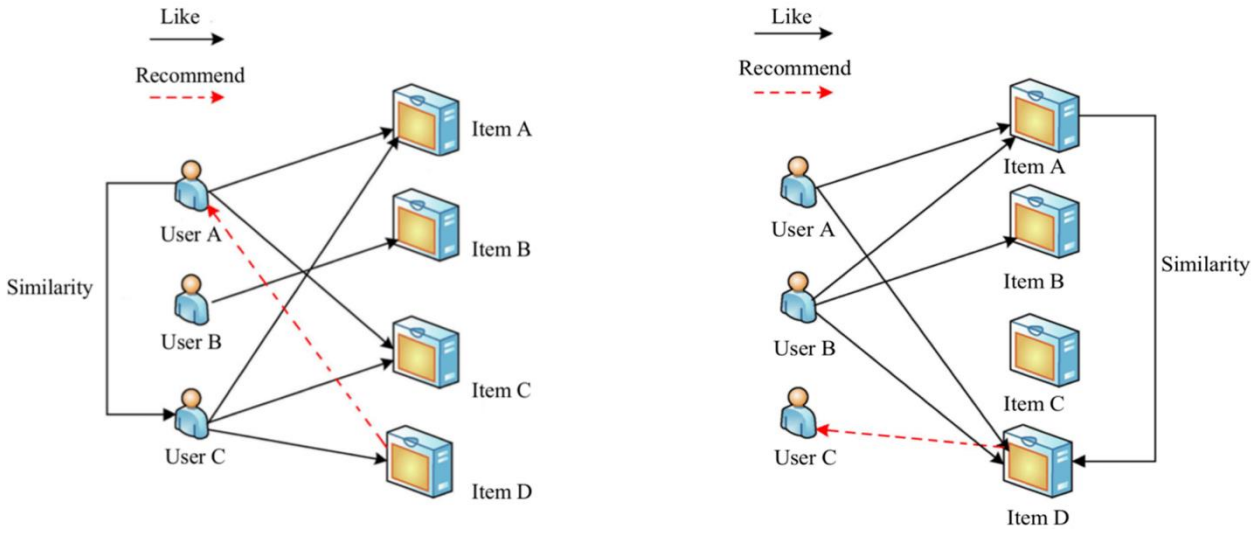

Рис. 1.2 – Аналіз поведінки за допомогою колаборативної фільтрації [7]

На лівій частині рисунка показано, як користувач A оцінює Item A та Item B, створюючи базу для аналізу. Оскільки User B має схожі вподобання, система припускає, що User A може зацікавитися Item C, рекомендуючи його. У правій частині рисунка демонструється альтернативний підхід, де об’єкти аналізуються на основі схожості. Наприклад, якщо Item C схожий на Item D, система може порекомендувати Item D користувачу C на основі вподобань інших користувачів.

Колаборативна фільтрація є універсальним інструментом для побудови рекомендацій у системах, що працюють із великими обсягами даних. Вона дозволяє формувати рекомендації, не вимагаючи детальної інформації про об'єкти, та адаптуватися до змін у вподобаннях користувачів. Однак цей метод залежить від обсягу даних: за відсутності достатньої кількості інформації про дії користувачів (проблема холодного старту) його ефективність може знижуватися.

Кластеризація є методом машинного навчання, який використовується для групування даних на основі їхніх схожих характеристик. Цей підхід дозволяє системі автоматично виявляти групи (або кластери) у великих наборах даних, де об’єкти всередині одного кластеру мають бути максимально схожими, а між різними кластерами – максимально відмінними. У контексті рекомендаційних систем кластеризація може використовуватись для

сегментації користувачів або об’єктів, що дозволяє створювати рекомендації на основі поведінкових або інших ознак, властивих певній групі [8]. Наприклад, у платформі електронної комерції кластеризація допомагає групувати користувачів за їх уподобаннями, такими як вибір товарів, частота покупок чи географічне розташування.

На рис. 1.3 зображено принцип роботи кластеризації, який демонструє процес розподілу неструктурованих даних на окремі групи. На початковому етапі дані (об'єкти) представлені у вигляді нерозподілених точок, які не мають чіткої структури. Після застосування алгоритму кластеризації всі об'єкти групуються у три різні класи, які позначені червоним, синім та зеленим кольорами.

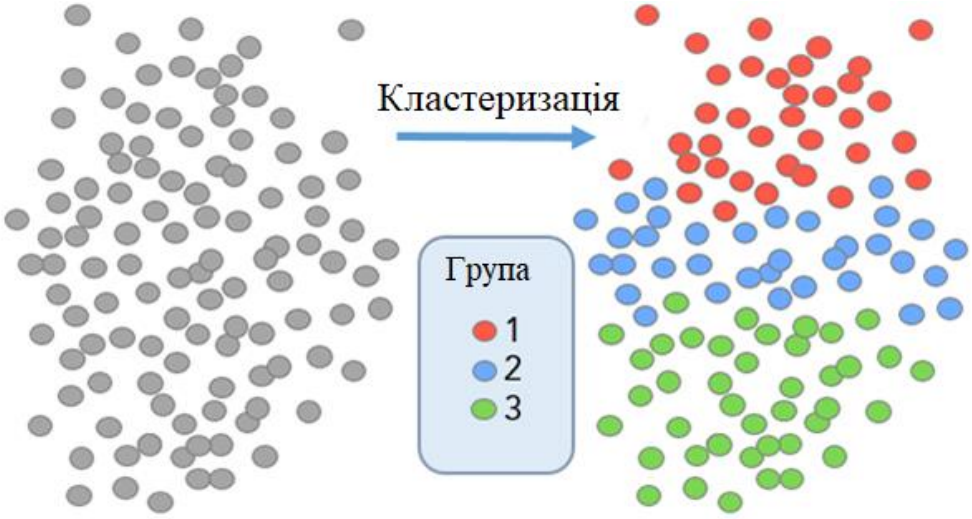

Рис. 1.3 – Принцип роботи кластеризації [9]

Ця схема ілюструє, як алгоритм кластеризації виявляє структуру у даних, автоматично формуючи групи на основі певних метрик схожості. Наприклад, точки, що мають подібні значення за певними характеристиками, об’єднуються у групи. У рекомендаційних системах це може означати групування об'єктів, таких як фільми чи товари, на основі схожості жанру, цінового діапазону або рейтингу.

Отже, кластеризація є важливим інструментом для попередньої обробки даних і формування рекомендацій. Вона дозволяє спростити аналіз великих наборів даних, забезпечуючи ефективну сегментацію, що особливо корисно у

сценаріях із високою варіативністю користувачів чи об'єктів. Метод не потребує попередньо підготовлених міток, що робить його гнучким і застосовним у багатьох задачах.

Останнім часом глибоке навчання набуває все більшої популярності у сфері персоналізації завдяки своїй здатності працювати з великими обсягами даних і виявляти складні, приховані взаємозв’язки, які важко визначити традиційними методами. Зокрема, рекурентні нейронні мережі застосовуються для аналізу послідовностей дій користувачів, таких як історія переглядів або покупок, що дає змогу передбачати майбутню активність із високою точністю. Конволюційні нейронні мережі використовуються для обробки зображень або тексту, що робить їх незамінними у медіа-рекомендаціях, де аналіз мультимедійного контенту є ключовим для формування точних пропозицій.

Сучасні рекомендаційні системи все частіше використовують комбіновані підходи, які інтегрують різні методи машинного навчання для досягнення найкращих результатів. Наприклад, комбінація колаборативної фільтрації з класифікаційними моделями дозволяє враховувати не лише індивідуальні вподобання користувачів, а й загальні тенденції поведінки великих груп. Таке комбінування сприяє більш точній і швидкій роботі системи, оскільки сильні сторони одного алгоритму компенсують слабкі місця іншого.

Завдяки гнучкості та універсальності сучасні підходи машинного навчання відкривають нові горизонти в персоналізації. Вибір відповідного методу або їхньої комбінації залежить від специфіки поставлених задач, доступних ресурсів та характеру даних, з якими працює система. Інновації в галузі машинного навчання забезпечують створення рекомендаційних систем, які здатні не лише краще розуміти потреби користувачів, але й адаптуватися до їхніх змін, забезпечуючи максимально релевантну взаємодію.

1.3 Особливості вибору алгоритмів рекомендацій залежно від типу даних

Рекомендаційні системи працюють із різноманітними типами даних, які безпосередньо впливають на вибір алгоритмів для їх реалізації. Тип даних

визначає, які характеристики можуть бути використані для аналізу, а також обмеження, які слід враховувати при розробці рекомендаційної системи. Нижче розглянуто основні типи даних і відповідні їм алгоритми рекомендацій.

Структуровані дані

Структуровані дані представляють собою інформацію у формі таблиць із чітко визначеними атрибутами, такими як категорія товару, ціна, рейтинг або кількість покупок. Для роботи з такими даними найбільш ефективними є алгоритми контентного фільтрування, які базуються на схожості між атрибутами об'єктів. Наприклад, якщо користувач придбав товар із категорії "смартфони" певної цінової категорії, система може запропонувати інші смартфони з аналогічними характеристиками.

Інший підхід для структурованих даних – використання градієнтного бустингу, який дозволяє моделювати складні взаємозв’язки між атрибутами. Такий метод може бути ефективним у системах електронної комерції, де структура даних чітко визначена [10].

Неструктуровані дані

Неструктуровані дані включають текст, зображення, аудіо та відео. Їх складність і відсутність чіткої структури вимагають використання методів глибокого навчання, зокрема нейронних мереж. Наприклад:

– текст. Обробка відгуків чи описів товарів може здійснюватися за допомогою рекурентних нейронних мереж (RNN) або трансформерів (наприклад, BERT). Це дозволяє виявляти емоційне забарвлення тексту та визначати його релевантність для користувача;

– зображення. Конволюційні нейронні мережі використовуються для аналізу візуальних даних. У платформі моди система може рекомендувати схожі за стилем товари, аналізуючи їх зображення;

– відео. Для аналізу відео або мультимедіа даних застосовуються гібридні моделі, які поєднують обробку зображень і тексту (наприклад, описів або субтитрів) [11].

Часові дані

Часові дані включають інформацію про послідовність взаємодії користувача з системою, як-от історія переглядів чи покупок. Такі дані є особливо цінними для прогнозування поведінки користувача. Для цього використовуються рекурентні нейронні мережі та їх покращені модифікації, як-от Long Short-Term Memory (LSTM) або Gated Recurrent Units (GRU). Вони дозволяють враховувати порядок дій користувача, що важливо для формування релевантних рекомендацій у таких сферах, як потокові сервіси (наприклад, Netflix або Spotify).

Часові дані також можуть оброблятися за допомогою методів колаборативної фільтрації, які адаптуються до змін у вподобаннях користувачів, враховуючи тимчасові закономірності [12].

Графові дані

Графові дані використовуються для моделювання зв'язків між користувачами та об'єктами. Наприклад, графи можуть представляти соціальні мережі, де вузли – це користувачі, а ребра – їхні взаємодії. Для аналізу графів використовуються графові нейронні мережі, які дозволяють виявляти приховані закономірності у зв’язках. Наприклад, у системах рекомендацій друзів у соціальних мережах графові алгоритми враховують, із ким користувач взаємодіє найчастіше, щоб запропонувати нові знайомства [13].

Гібридні дані

У реальних системах часто зустрічаються комбіновані типи даних, наприклад, структуровані характеристики товарів і текстові відгуки. У таких випадках доцільно використовувати гібридні підходи, які інтегрують різні алгоритми. Наприклад, Amazon застосовує одночасно колаборативну фільтрацію для аналізу вподобань користувачів і контентне фільтрування для врахування характеристик товарів.

Гібридні моделі також можуть комбінувати методи глибокого навчання (для обробки тексту чи зображень) із традиційними підходами, що дозволяє максимально ефективно використовувати наявні дані.

Особливості вибору алгоритмів рекомендаційних систем значною мірою залежать від типу даних, з якими працює система [14]. Структуровані дані краще обробляти традиційними методами, такими як контентне фільтрування чи градієнтний бустинг, тоді як для неструктурованих даних необхідні нейронні мережі. Часові та графові дані вимагають спеціалізованих підходів, таких як рекурентні моделі або графові нейронні мережі. Гібридні підходи дозволяють ефективно комбінувати різні методи для забезпечення максимальної точності рекомендацій, враховуючи всі доступні типи даних.

1.4 Популярні моделі та методи побудови рекомендаційних систем

Рекомендаційні системи є багатогранною галуззю досліджень, яка включає в себе широкий спектр моделей і методів, спрямованих на забезпечення персоналізованих рекомендацій. Вибір конкретної моделі залежить від типу даних, поставленої задачі та вимог до точності й продуктивності системи. Розглянемо найбільш популярні моделі та методи, які активно використовуються для побудови сучасних рекомендаційних систем.

Матрична факторизація є одним із найпоширеніших підходів у колаборативній фільтрації, який застосовується для побудови персоналізованих рекомендацій. Цей метод базується на представленні взаємодії між користувачами і об’єктами у вигляді матриці, де рядки відповідають користувачам, а стовпці – об’єктам (товарам, фільмам, пісням тощо). Значення матриці зазвичай відображають рейтинг або взаємодію користувача з певним об'єктом. Основна ідея методу полягає в розкладанні цієї матриці на дві менші матриці: одну, що представляє користувачів, і другу, що представляє об’єкти. Це дозволяє виявити приховані фактори, які впливають на взаємодію, наприклад, жанрові вподобання або популярність серед певних груп користувачів.

Матрична факторизація використовується у багатьох системах, таких як Netflix, Amazon чи Spotify. Наприклад, у стрімінгових платформах цей метод дозволяє виявити, які жанри фільмів чи музики є цікавими для конкретного

користувача, навіть якщо він раніше не взаємодіяв з певним контентом. Це досягається завдяки прихованим характеристикам, які визначають, чому користувач надає перевагу.

На рис. 1.4 зображено приклад використання матричної факторизації для передбачення вподобань користувачів. Схема демонструє матрицю, у якій рядки відповідають користувачам, а стовпці – об’єктам (наприклад, фільмам). У ній показані як відомі оцінки користувачів (зелені галочки), так і відсутні значення, які необхідно передбачити (червона рамка).

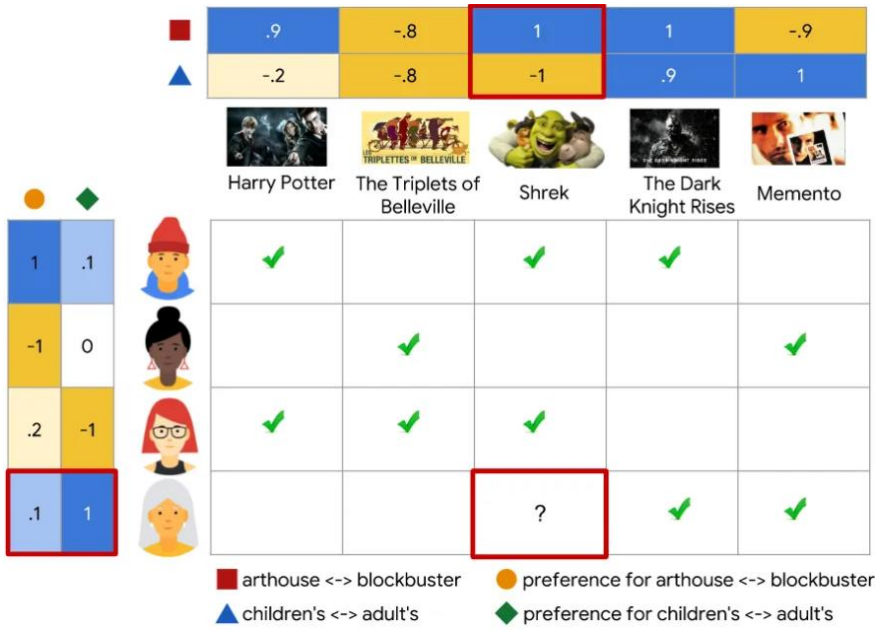

Рис. 1.4 – Застосування алгоритму матричної факторизації [15]

Механізм роботи матричної факторизації полягає у тому, що матриця рейтингу розкладається на дві менші матриці: одну, що містить приховані характеристики користувачів, і другу – з характеристиками об’єктів. Наприклад, користувачі можуть мати вподобання до певного жанру (дитячі чи дорослі фільми, артхаус чи блокбастери), а об’єкти матимуть відповідні атрибути. На основі подібності між характеристиками користувачів і об’єктів можна передбачити невідомі значення матриці, як-от рейтинг, який користувач, ймовірно, поставить фільму «Shrek» на основі своїх вподобань і оцінок інших користувачів.

Матрична факторизація є потужним інструментом для роботи з великими обсягами даних. Вона дозволяє виявляти приховані закономірності у

взаємодіях, які неможливо побачити безпосередньо. Однак метод потребує достатньої кількості даних для ефективного навчання та є вразливим до проблеми «холодного старту», коли в системі немає інформації про нових користувачів чи об'єкти. У сучасних рекомендаційних системах матрична факторизація часто використовується у поєднанні з іншими методами для досягнення максимальної точності.

Випадковий ліс (Random Forest) – це ансамблевий метод машинного навчання, який об'єднує кілька дерев прийняття рішень для створення більш стабільної, точної та узагальненої моделі. Його основна ідея полягає у побудові багатьох дерев, які тренуються на різних підмножинах даних, а потім об'єднують свої прогнози через середнє значення (для регресії) або більшість голосів (для класифікації). Випадковий ліс ефективно працює з великими обсягами даних і складними наборами характеристик, що робить його популярним у рекомендаційних системах. Наприклад, у платформі електронної комерції цей метод може використовуватись для передбачення ймовірності покупки товару на основі поведінки користувачів.

На рис. 1.5 представлено схему роботи алгоритму випадкового лісу.

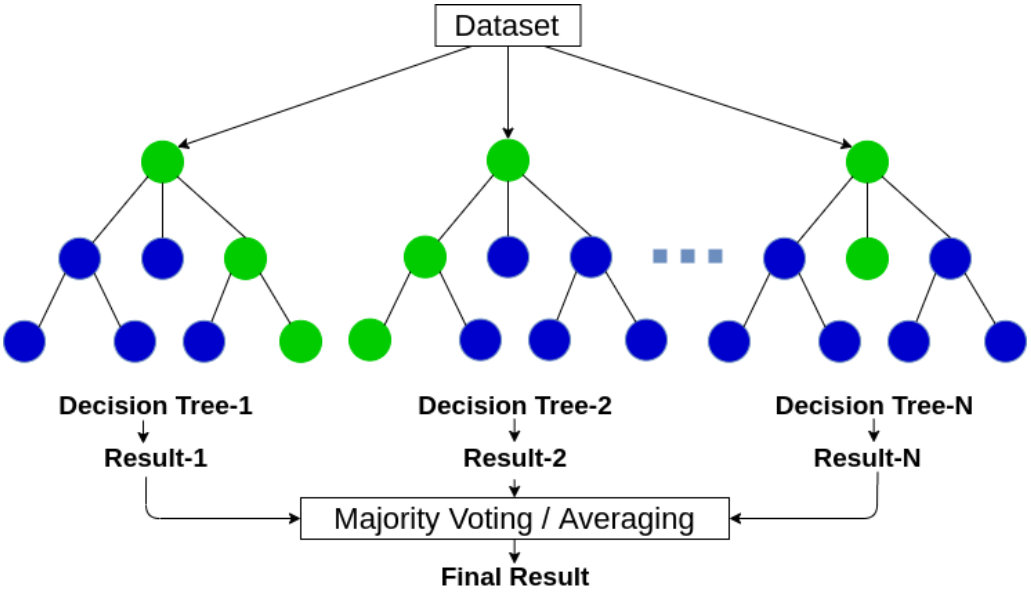

Рис. 1.5 – Принцип роботи випадкового лісу [17]

Цей алгоритм працює шляхом розбиття початкового набору даних на підмножини, які використовуються для тренування кількох дерев прийняття

рішень. Кожне дерево генерує свій власний прогноз (результат), і ці прогнози комбінуються для отримання фінального результату. Наприклад, якщо це задача класифікації, підсумковий результат визначається голосуванням більшості дерев, а для регресії – обчислюється середнє значення всіх результатів.

Схема демонструє, як початковий набір даних поділяється на кілька частин, кожна з яких використовується для тренування окремого дерева. На кожному етапі дерева приймають рішення на основі характеристик даних, а потім об'єднують свої результати. На виході ми отримуємо фінальний результат через механізм «голосування» або «усереднення».

Випадковий ліс є потужним методом для задач класифікації та регресії, який поєднує простоту дерев прийняття рішень із надійністю ансамблевого підходу. Його переваги включають високу стійкість до переобучення, здатність працювати з даними різної природи та простоту налаштування. Однак випадковий ліс може вимагати значних обчислювальних ресурсів, особливо для великих наборів даних [18].

Градієнтний бустинг – це ансамблевий метод машинного навчання, який будується шляхом послідовного додавання слабких моделей (зазвичай дерев рішень), кожна з яких намагається зменшити помилки попередніх. Основна ідея методу полягає у мінімізації функції втрат за допомогою градієнтного спуску, що дозволяє моделі ітеративно покращувати точність. Градієнтний бустинг широко використовується для задач класифікації, регресії, а також у рекомендаційних системах для прогнозування рейтингів або ймовірності вибору товарів. Наприклад, у платформі електронної комерції градієнтний бустинг може передбачати ймовірність покупки товару, враховуючи поведінку користувачів.

На рис. 1.6 показано схему роботи алгоритму градієнтного бустингу. Вона демонструє, як кожне дерево рішень навчається на залишкових помилках попереднього. На першому етапі алгоритм створює базову модель (Tree 1), яка робить первинний прогноз. Потім обчислюється різниця між реальними

значеннями (y) та прогнозованими ($\hat{y}_1$), яка називається залишками (r_1). Ці залишки використовуються як цільова змінна для наступного дерева (Tree 2), що навчається на цих помилках, намагаючись їх скоригувати. Процес повторюється для кожного наступного дерева, яке поступово зменшує залишкові помилки, додаючи свої прогнози до підсумкового результату.

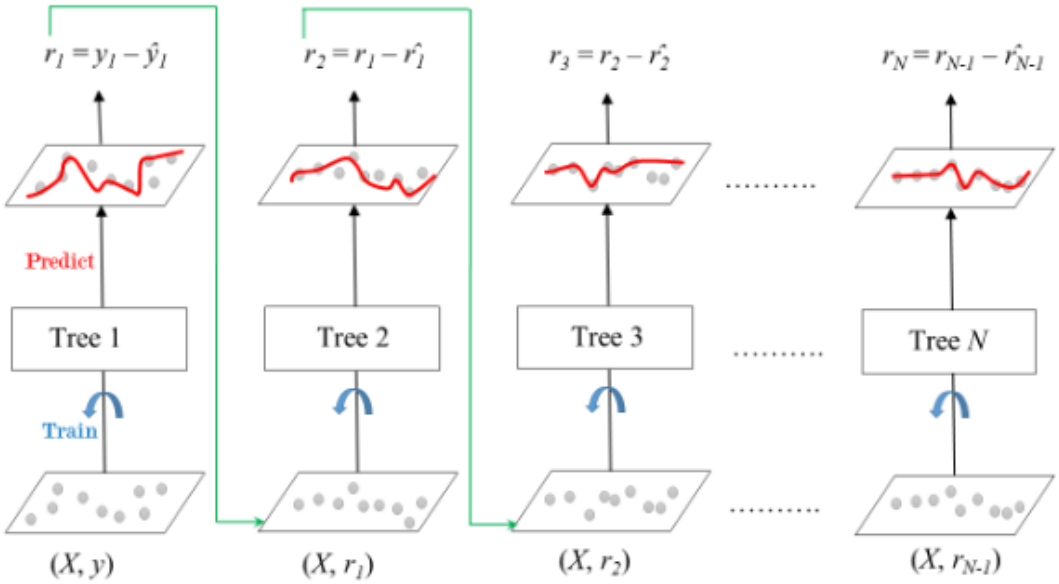

Рис. 1.6 – Схема роботи градієнтного бустингу [18]

Схема ілюструє, як початкові дані (X, y) проходять через перше дерево, яке генерує прогноз. Потім залишки (r_1, r_2, ...) використовуються як нові цільові змінні для тренування наступних дерев. Кожне дерево зосереджується на мінімізації помилок попередніх, а на фінальному етапі всі результати дерев комбінуються для отримання точного прогнозу.

Градієнтний бустинг є одним із найефективніших методів для роботи з великими й складними наборами даних. Він забезпечує високу точність завдяки ітеративному підходу до зменшення помилок, але водночас потребує значних обчислювальних ресурсів. У рекомендаційних системах градієнтний бустинг дозволяє враховувати багатофакторні залежності та створювати високоточні персоналізовані рекомендації.

Також існують гібридні моделі, які поєднують кілька різних алгоритмів для досягнення найкращих результатів у персоналізації. Наприклад, система може комбінувати колаборативну фільтрацію з контентно-орієнтованими

рекомендаціями, щоб враховувати як поведінкові патерни користувачів, так і характеристики об'єктів. Це дозволяє подолати обмеження кожного окремого підходу і створити більш точні та релевантні рекомендації.

Останнім часом гібридні моделі все частіше стають ключовим інструментом для сучасних рекомендаційних систем завдяки своїй здатності інтегрувати сильні сторони кількох підходів. Вони дозволяють ефективно поєднувати переваги різних алгоритмів, таких як колаборативна фільтрація, контентне фільтрування, моделі на основі знань чи методи глибокого навчання. Наприклад, у системах потокового відео чи музики, таких як Netflix або Spotify, комбінуються рекомендації на основі історії переглядів користувачів з аналізом характеристик контенту, наприклад жанру або ключових тем. Це не тільки покращує точність рекомендацій, але й забезпечує адаптацію до змінних вподобань користувачів.

Одним із популярних варіантів гібридизації є каскадний підхід, коли один алгоритм використовується для первинного відбору об'єктів, а інший уточнює результати. Наприклад, колаборативна фільтрація може визначити найбільш релевантних користувачів, а контентне фільтрування звузити рекомендації на основі їхніх індивідуальних уподобань. Інший підхід – змішування, коли результати кількох моделей об'єднуються шляхом усереднення або вагового голосування.

Такі системи також активно використовують глибокі нейронні мережі для аналізу текстів, зображень чи поведінкових патернів. Наприклад, у випадку рекомендацій книг можна одночасно врахувати схожість у рейтингах користувачів (колаборативна фільтрація) та аналіз текстового опису книг (контентне фільтрування). Це дозволяє враховувати не лише історичні дані, а й специфічні деталі об'єктів [20].

Важливим аспектом гібридних моделей є їх гнучкість: вони можуть адаптуватися до нових даних і задач, а також вирішувати класичні проблеми, такі як «холодний старт». Наприклад, для нових користувачів або об'єктів, у яких відсутня достатня кількість взаємодій, система може використовувати

контентні чи знаннєві методи, а для досвідчених користувачів – колаборативну фільтрацію. Таким чином, гібридні моделі здатні забезпечити універсальність та високу якість рекомендацій навіть у складних умовах.

Висновки до розділу 1

У рамках даного розділу проведено аналіз алгоритмів та підходів, що використовуються в сучасних рекомендаційних системах. Було розглянуто сутність рекомендаційних систем, їхню роль у різних галузях, таких як електронна комерція, медіасервіси, освітні платформи та соціальні мережі. Аналіз сучасних підходів до побудови рекомендаційних систем включав колаборативну фільтрацію, яка ефективно використовує взаємодії користувачів для формування персоналізованих рекомендацій, та кластеризацію, яка дозволяє групувати дані на основі схожих характеристик. Вивчено особливості вибору алгоритмів залежно від типу даних: структурованих, неструктурованих, часових, графових та комбінованих (гібридних), що дає змогу враховувати специфіку вхідної інформації під час вибору оптимальних методів.

Розглянуто популярні моделі та методи побудови рекомендаційних систем, такі як матрична факторизація, що дозволяє знаходити приховані закономірності у вподобаннях користувачів, алгоритм випадкового лісу, який забезпечує точність і надійність прогнозів, та градієнтний бустинг, який ітеративно покращує точність моделі. Гібридні моделі, які поєднують кілька підходів, були визначені як найбільш універсальні завдяки їхній здатності долати обмеження окремих методів і забезпечувати високу точність рекомендацій.

Результати цього розділу дозволили сформувати базу знань про сучасні алгоритми, їх переваги та недоліки, що є критично важливим для подальшого обґрунтування вибору алгоритмів у наступному розділі. Отримані висновки стануть основою для аналізу алгоритмів, вибору відповідного тренувального набору даних та побудови технологічного стеку, який буде використовуватися для реалізації рекомендаційної системи.

РОЗДІЛ 2 АНАЛІЗ АЛГОРИТМІВ, ОБГРУНТУВАННЯ ТРЕНУВАЛЬНОГО НАБОРУ ТА ВИБІР ТЕХНОЛОГІЧНОГО СТЕКУ

2.1 Алгоритми рекомендаційних систем

У процесі дослідження алгоритмів рекомендаційних систем ключовим завданням є аналіз їхніх характеристик, переваг та обмежень у контексті вирішення задач побудови персоналізованих рекомендацій. Такий аналіз є необхідним для розуміння особливостей кожного методу, визначення їхньої придатності до використання в різних умовах та порівняння їхньої продуктивності. Окрім теоретичного розгляду, важливим етапом є програмна реалізація та тестування алгоритмів, що дозволяє оцінити їхню ефективність на практичних прикладах і сформувати рекомендації щодо їх застосування. У цьому підрозділі буде проведено аналіз алгоритмів градієнтного бустингу, матричної факторизації та швидкого дерева Твідді, які є сучасними підходами до вирішення задач рекомендаційних систем.

2.1.1 Градієнтний бустинг

Градієнтний бустинг належить до ансамблевих методів машинного навчання, які базуються на поетапному об'єднанні кількох слабких моделей, здебільшого дерев рішень, у потужний прогнозуючий ансамбль. Кожна нова модель створюється з метою виправлення помилок, допущених попередніми моделями, шляхом мінімізації похибки за допомогою методу градієнтного спуску. У кінцевому підсумку отримується високоточна система прогнозування, здатна ефективно працювати з різноманітними типами даних.

Завдяки своїй здатності адаптуватися до складних закономірностей у даних, градієнтний бустинг знаходить застосування в задачах класифікації, регресії, виявлення аномалій та побудови рекомендацій. Незважаючи на високу точність, цей метод характеризується значними обчислювальними витратами. Щоб зменшити їх, використовують оптимізацію гіперпараметрів, скорочення розмірності даних, багатопоточні обчислення, а також спеціалізовані

бібліотеки, такі як XGBoost, LightGBM і CatBoost, які забезпечують високу продуктивність і швидкість навчання.

Ключовою особливістю цього алгоритму є його ітеративна природа, де кожен крок навчання спрямований на побудову моделі, що компенсує недоліки попередніх [21]. Це дозволяє створювати потужні комплексні моделі, які демонструють стабільно високі результати на різних наборах даних.

Нехай задано вибірку $\{(x_i, y_i)\}_{i=1}^{n}$, де x_i є набором ознак, а y_i – значенням цільової змінної (наприклад, класом або числовим значенням). Завдання алгоритму полягає у побудові функції $F(x)$, яка максимально точно відображає зв'язок між вхідними ознаками x_i та значеннями y_i. Модель $F(x)$ формується поступово, додаючи нові базові моделі:

$$F_M(x) = F_0(x) + \sum_{m=1}^{M} \gamma_m \cdot h_m(x) \qquad (2.1)$$

де $F_0(x)$ – початкове передбачення (наприклад, середнє значення цільової змінної), $h_m(x)$ – нова базова модель на кожній ітерації, а γ_m – коефіцієнт, що враховує внесок цієї моделі.

Для визначення оптимального напрямку покращення моделі на кожному кроці використовуються похідні функції втрат, які характеризують залишкову похибку. Під час навчання кожна нова базова модель налаштовується на ці залишки, зменшуючи їх.

З огляду на високу обчислювальну складність алгоритму, застосовуються такі підходи:

– паралельне обчислення. Сучасні бібліотеки, як-от XGBoost чи LightGBM, дозволяють будувати базові моделі паралельно, що суттєво скорочує час навчання;

– зменшення кроку (shrinkage). Внесок кожного базового алгоритму зменшується, щоб забезпечити стійкість моделі та уникнути перенавчання;

– рання зупинка. Процес навчання припиняється, якщо покращення функції втрат стає незначним.

– субсемплінг. Використання частини даних на кожній ітерації скорочує час навчання та сприяє регуляризації.

Градієнтний бустинг знаходить широке застосування в задачах персоналізації, таких як побудова рекомендаційних систем. Він дозволяє аналізувати поведінку користувачів, враховувати їхні вподобання та прогнозувати інтерес до певного контенту або товарів. Завдяки своїй здатності працювати з великими обсягами даних і забезпечувати точні результати, алгоритм є ключовим інструментом у створенні індивідуалізованого досвіду користувачів.

Переваги градієнтного бустингу:

– висока точність моделей. Завдяки послідовному покращенню прогнозів і врахуванню помилок попередніх ітерацій, градієнтний бустинг забезпечує високу точність для різних задач, таких як класифікація, регресія та рекомендаційні системи;

– гнучкість у роботі з різними типами даних. Алгоритм ефективно працює як із числовими, так і з категорійними ознаками без необхідності значної попередньої обробки;

– можливість інтеграції регуляризації. Вбудовані механізми, такі як проріджування дерев (shrinkage), субсемплінг і обмеження глибини дерев, зменшують ризик перенавчання, що робить метод стійким на складних наборах даних;

– адаптивність до змін у даних. Градієнтний бустинг може бути налаштований для швидкого врахування нових даних, що є особливо важливим для динамічних систем, таких як рекомендаційні або фінансові програми.

Недоліки градієнтного бустингу:

– високі обчислювальні витрати. Алгоритм є ресурсозатратним як за часом, так і за обсягом пам'яті, особливо на великих наборах даних;

– чутливість до вибору гіперпараметрів. Градієнтний бустинг має багато параметрів, таких як глибина дерев, швидкість навчання, кількість

ітерацій тощо. Неправильний вибір може призвести до низької точності або перенавчання;

– складність інтерпретації. У порівнянні з простішими моделями (наприклад, лінійною регресією чи окремими деревами рішень), результати градієнтного бустингу важче інтерпретувати, що ускладнює пояснення його рішень кінцевим користувачам;

– схильність до перенавчання на шумових даних. Незважаючи на наявність регуляризаційних механізмів, метод може перенавчатися, якщо дані містять значний рівень шуму або помилкові кореляції, особливо без належного налаштування параметрів.

Градієнтний бустинг є потужним та універсальним алгоритмом машинного навчання, який забезпечує високу точність завдяки послідовному покращенню прогнозів шляхом мінімізації похибок. Його гнучкість у роботі з різними типами даних і здатність адаптуватися до складних залежностей роблять його популярним у задачах класифікації, регресії та побудови рекомендацій. Водночас алгоритм має значні обчислювальні витрати, потребує ретельного налаштування гіперпараметрів і може бути складним для інтерпретації, що вимагає обережного підходу до його застосування в реальних проєктах.

2.1.2 Матрична факторизація

Матрична факторизація є методом, що дозволяє розкласти велику матрицю на дві або більше компактніші матриці, добуток яких наближено відтворює початкову матрицю. Цей підхід широко застосовується в задачах машинного навчання, зокрема в рекомендаційних системах, де він використовується для аналізу взаємодії між користувачами та об'єктами, такими як товари, фільми чи інший контент.

Основна ідея матричної факторизації полягає в тому, щоб представити складні взаємозв'язки у вигляді матриці, де рядки відповідають користувачам, а стовпці – об'єктам. Елементи матриці можуть відображати рівень взаємодії

(наприклад, рейтинг, перегляд або купівлю). Матрична факторизація дозволяє розділити цю матрицю на дві менші: матрицю характеристик користувачів і матрицю характеристик об'єктів [22]. Перемноження цих матриць дає змогу наближено відновити вихідну матрицю, включаючи відсутні значення.

Це означає, що алгоритм може передбачити, як користувачі взаємодіятимуть із об'єктами, з якими вони ще не мали справи. Таким чином, матрична факторизація використовується для створення персоналізованих рекомендацій, ґрунтуючись на історії взаємодії користувачів і схожості характеристик об'єктів. Такий підхід ефективний для роботи з великими наборами даних і дозволяє враховувати приховані закономірності, що впливають на взаємодію між користувачами та контентом.

Нехай $R \in R^{m \cdot n}$ – це матриця, яка відображає взаємодію між користувачами та товарами, де R_{ij} представляє рівень взаємодії (наприклад, рейтинг) користувача ii з товаром jj. Метою алгоритму є розкласти цю матрицю на дві менші: матрицю користувачів $P \in R^{m \cdot k}$ і матрицю товарів $Q \in R^{k \cdot n}$, де k – це кількість прихованих факторів, що відображають інтереси користувачів і характеристики товарів.

Алгоритм спрямований на мінімізацію похибки між елементами вихідної матриці R і відновленими значеннями через добуток P і Q, тобто:

$$\min_{P,Q} \sum_{(i,j \in \Omega)} (R_{i,j} - P_i^T Q_i)^2 + \lambda(\| P_i \|^2 + \| Q_j \|^2), \qquad (2.2)$$

де Ω – множина відомих взаємодій, а λ – регуляризаційний параметр, який запобігає перенавчанню, додаючи штраф за складність моделі.

Цей підхід є основою для багатьох систем рекомендацій, зокрема у методах спільної фільтрації (Collaborative Filtering). Завдяки можливості аналізувати приховані фактори, алгоритм дозволяє прогнозувати вподобання користувачів навіть у ситуаціях, коли дані взаємодії є обмеженими (проблема «рідких» даних).

Наприклад, у системах інтернет-магазинів матриця R може містити дані про товари, які користувачі оцінювали чи купували. Матрична факторизація

дозволяє передбачити товари, які ймовірно зацікавлять користувача, навіть якщо він раніше не взаємодіяв із цими товарами. Аналогічно, на стрімінгових платформах або новинних сайтах цей метод може бути використаний для аналізу поведінки користувачів і пропонування релевантного контенту, ґрунтуючись на їхніх попередніх діях.

Переваги матричної факторизації:

– ефективність роботи з великими даними. Матрична факторизація добре масштабується для роботи з великими наборами даних, забезпечуючи швидке навчання та генерацію рекомендацій навіть для великих систем із великою кількістю користувачів і об'єктів;

– виявлення прихованих закономірностей. Алгоритм знаходить приховані зв’язки між користувачами та об'єктами, аналізуючи латентні фактори, що дозволяє створювати точні й персоналізовані рекомендації;

– можливість роботи з рідкими даними. Навіть якщо матриця взаємодій містить велику кількість пропущених значень (що є типовим для реальних систем), алгоритм здатний будувати точні прогнози на основі існуючих даних;

– гнучкість застосування. Матричну факторизацію можна адаптувати для вирішення різноманітних задач у різних галузях, включаючи електронну комерцію, стрімінгові платформи, соціальні мережі та системи освіти.

Недоліки матричної факторизації:

– залежність від якості даних. Якщо дані взаємодії містять багато шуму, алгоритм може виробляти помилкові прогнози, оскільки приховані фактори можуть включати небажані кореляції;

– неможливість роботи з новими користувачами або об'єктами (Cold Start Problem). Алгоритм не може створювати рекомендації для нових користувачів чи об'єктів без наявних даних взаємодії, що обмежує його застосування в динамічних системах;

– обчислювальна складність. Хоча алгоритм ефективний для великих наборів даних, його навчання може займати значний час при використанні на масивних матрицях із високою розмірністю;

– недостатня інтерпретація результатів. Латентні фактори, на основі яких формуються рекомендації, важко інтерпретувати, що ускладнює пояснення їхньої логіки кінцевим користувачам або адміністраторам системи.

Матрична факторизація дозволяє будувати точні й персоналізовані прогнози, базуючись на аналізі прихованих закономірностей у даних взаємодії. Незважаючи на певні обмеження, такі як проблема холодного старту та залежність від якості даних, її гнучкість і здатність працювати з рідкими даними роблять цей метод надзвичайно корисним у багатьох сферах, де персоналізація є важливою складовою успіху.

2.1.3 Швидке дерево Твідді

Алгоритм швидкого дерева Твідді (ШДТ) є спеціалізованим підходом у рамках ансамблевих методів, який застосовується для задач регресії, коли цільова змінна відповідає розподілу Твідді. Цей тип розподілу поєднує властивості нормального та пуассонівського розподілів, що робить його особливо корисним для аналізу даних із нерівномірними або змішаними характеристиками [23]. Алгоритм був розроблений для роботи з великими обсягами даних і змішаними наборами ознак, забезпечуючи високу точність прогнозів у складних задачах.

Розглянемо вибірку $\{(x_i, y_i)\}_{i=1}^{n}$, де $x_i \in R^d$ – вектори ознак, а $y_i \in R$ – цільова змінна. ШДТ будує ансамбль дерев рішень, кожне з яких додається до моделі для зменшення похибки попередніх. Унікальною особливістю алгоритму є його функція втрат, адаптована для розподілу Твідді:

$$L\big(y, F(x)\big) = \frac{1}{\phi}\left(\frac{y^{2-p}}{2-p} - y \cdot F(x)^{1-p} + \frac{F(x)^{2-p}}{2-p}\right), \qquad (2.3)$$

де p – параметр, який визначає тип розподілу, а ϕ – масштабний параметр. Для p=1 функція втрат відповідає логарифмічній функції Пуассонівського розподілу, а для p=2 вона стає квадратичною, як у випадку нормального розподілу.

ШДТ найчастіше використовується в задачах, де цільова змінна має нерівномірний розподіл або складну структуру. Це може бути фінансове прогнозування, моделювання страхових ризиків, аналіз збитків чи обсягів продажів.

Переваги алгоритму:

– підтримка нерівномірних розподілів. Завдяки функції втрат, налаштованій для розподілу Твідді, алгоритм може працювати з даними, які мають складну структуру та значну варіативність;

– гнучкість у роботі з великими даними. ШДТ здатний ефективно працювати з великими наборами даних, зберігаючи високу продуктивність навіть для складних задач;

– можливість моделювання кількісних змінних. Алгоритм добре підходить для задач, де необхідно прогнозувати кількісні показники, наприклад, витрати чи обсяги продажів;

– адаптація до змішаних ознак. ШДТ може працювати як із числовими, так і з категорійними ознаками, що робить його універсальним для різних типів задач.

Недоліки алгоритму:

– складність налаштування параметрів. Робота алгоритму сильно залежить від вибору параметрів, таких як p і ϕ, що потребує експериментів і обчислювальних витрат;

– високі вимоги до ресурсів. Навчання моделі може бути тривалим, особливо на великих наборах даних, через використання ансамблів дерев;

– чутливість до шуму в даних. Алгоритм може погіршувати результати, якщо дані містять значну кількість аномалій або помилок;

– обмежена інтерпретованість. Як і в інших ансамблевих методах, результати ШДТ важко інтерпретувати, що може бути проблемою для розуміння моделі кінцевими користувачами.

Швидке дерево Твідді є потужним інструментом для задач регресії з нерівномірними чи змішаними даними, що використовує гнучкість розподілу Твідді для побудови точних прогнозів. Незважаючи на певні обмеження, такі як чутливість до параметрів і висока обчислювальна складність, цей алгоритм добре підходить для складних задач у фінансовій сфері, страхуванні та розробці рекомендаційних систем, де необхідно враховувати кількісні показники, такі як рейтинги, частота взаємодій або прогнозування вподобань користувачів.

2.2 Обґрунтування вибору тренувального набору даних

Для побудови рекомендаційної системи ключовим етапом є вибір якісного та репрезентативного набору даних, що відображає реальні взаємодії між користувачами і об'єктами рекомендацій. У даній роботі використовується датасет, створений на основі даних з платформи MyAnimeList, який містить інформацію про взаємодію користувачів із аніме-контентом [24]. Цей набір даних має ряд важливих характеристик, що роблять його придатним для дослідження алгоритмів рекомендаційних систем. У табл. 2.1 наведено опис основних атрибутів набору даних.

Таблиця 2.1 – Опис атрибутів датасету

№	Атрибут	Опис	Приклад значення
1	user_id	Унікальний ідентифікатор користувача, створений випадковим чином.	12345
2	anime_id	Унікальний ідентифікатор аніме на основі MyAnimeList.	6789
3	rating	Оцінка, виставлена користувачем для конкретного аніме (1–10).	8

Аналізуючи наведені атрибути, можна зробити висновок, що вони чітко відображають взаємодію користувачів із контентом. Така структура дозволяє використовувати дані для побудови як моделей класифікації (наприклад,

прогнозування вподобань користувачів), так і для рекомендацій на основі оцінок.

Табл. 2.2 містить основні статистичні показники, що дозволяють оцінити масштаб і якість інформації в наборі даних.

Таблиця 2.2 – Статистичний огляд набору даних

№	Показник	Значення
1	Загальна кількість користувачів	268 973
2	Загальна кількість аніме	16 867
3	Загальна кількість оцінок	50 000 000
4	Середня кількість оцінок на користувача	185.89
5	Середня кількість оцінок на аніме	2964.37

З таблиці видно, що середній користувач залишає близько 186 оцінок, що свідчить про активну взаємодію з платформою. У середньому кожне аніме отримує 2964 оцінки, що забезпечує достатню репрезентативність для аналізу та побудови рекомендацій. Великий обсяг даних дозволяє алгоритмам ефективно виявляти приховані закономірності між користувачами та об'єктами.

Даний набір даних було обрано для тренування моделей з наступних причин:

– масштабність. Велика кількість користувачів, аніме та оцінок забезпечує достатню кількість даних для навчання алгоритмів.

– різноманітність даних. Включення метаінформації про аніме (жанри, студії, статистика) дозволяє побудувати багатовимірні моделі;

– реальність сценаріїв. Дані відображають реальні взаємодії користувачів, що підвищує практичну цінність отриманих рекомендацій.

– структурованість. Чіткий розподіл атрибутів і значень полегшує обробку даних.

Таким чином, аналіз даного набору даних підтверджує його відповідність задачам дослідження алгоритмів рекомендаційних систем. Великий обсяг та

різноманітність інформації створюють умови для всебічного тестування й оцінки ефективності різних підходів до персоналізації контенту.

2.3 Технологічний стек для розробки рекомендаційних систем

Для ефективної розробки рекомендаційних систем важливим є вибір відповідного технологічного стеку, який забезпечує оптимальну продуктивність, масштабованість і гнучкість. У рамках даного дослідження особливу увагу приділяється мові програмування, інструментам для обробки даних та бібліотекам машинного навчання, які дозволяють реалізувати сучасні алгоритми рекомендаційних систем. Огляд та порівняння популярних мов програмування, таких як Python, Java та C#, допомагають визначити їхні переваги для використання у різних аспектах проєкту.

Python – це інтерпретована мова програмування, яка виділяється простим синтаксисом і широким набором бібліотек для обробки даних, машинного навчання та візуалізації [25]. Вона широко використовується для побудови прототипів рекомендаційних систем завдяки інструментам, таким як Scikit-learn, TensorFlow і Pandas. Python є ідеальним вибором для швидкої розробки алгоритмів і експериментів у наукових дослідженнях.

Java – це компільована мова програмування, що забезпечує високу продуктивність і платформну незалежність завдяки використанню віртуальної машини Java (JVM) [26]. Завдяки своїй стабільності та масштабованості Java є популярною для створення великих промислових систем, включаючи рекомендаційні системи. Інструменти, такі як Apache Mahout і Deeplearning4j, дозволяють реалізовувати алгоритми машинного навчання на цій мові.

C# – це об'єктно-орієнтована мова програмування, розроблена компанією Microsoft, яка забезпечує інтеграцію з екосистемою .NET [27]. Вона підходить для створення високопродуктивних додатків, зокрема вебсервісів і інтегрованих рекомендаційних систем у корпоративному середовищі. Завдяки бібліотекам, як-от ML.NET, C# дозволяє реалізовувати сучасні моделі машинного навчання у поєднанні з інструментами Microsoft Azure.

У табл. 2.3 наведені основні характеристики розглянутих мов програмування. Аналіз дозволяє оцінити сильні сторони кожної мови в контексті специфічних вимог розробки рекомендаційних систем.

Таблиця 2.3 – Порівняльний аналіз мов програмування

Характеристика	Python	Java	C#
Продуктивність (швидкість виконання)	Середня	Висока	Висока
Простота інтеграції з ML-фреймворками	Висока (TensorFlow, PyTorch, Scikit-learn)	Середня	Висока (ML.NET, Accord.NET)
Зручність розробки	Дуже висока	Середня	Висока
Підтримка бібліотек для великих даних	Дуже висока	Висока	Висока
Масштабованість	Середня	Висока	Висока
Інструменти для багатоплатформності	Обмежені (сторонні рішення)	Висока (JVM)	Висока (.NET, .NET Core)
Спільнота та підтримка	Дуже широка	Широка	Висока
Складність для початківців	Низька	Висока	Середня

Виходячи з аналізу у таблиці, мова програмування C# є оптимальним вибором для розробки системи дослідження алгоритмів рекомендаційних систем. Її висока продуктивність завдяки JIT-компіляції забезпечує швидке виконання алгоритмів, що є критичним для роботи з великими обсягами даних. Потужна підтримка машинного навчання через бібліотеки ML.NET і Accord.NET дозволяє реалізовувати сучасні алгоритми з мінімальними витратами на інтеграцію. Інструменти розробки, такі як Visual Studio, забезпечують зручність і ефективність роботи розробників. Завдяки .NET Core, C# також пропонує високу кросплатформну сумісність, що робить її універсальним вибором для створення систем, здатних працювати в різних середовищах.

Для максимально ефективного використання мови програмування C# у розробці системи дослідження алгоритмів рекомендаційних систем важливим є вибір відповідного інструментарію, що забезпечить зручність і продуктивність

роботи. Основним середовищем розробки виступає Microsoft Visual Studio, яке надає інтегровані засоби для написання, тестування та налагодження коду. Visual Studio також підтримує інтеграцію з ML.NET – сучасним фреймворком машинного навчання, який дозволяє реалізовувати складні моделі та адаптувати їх для виконання конкретних задач.

Технологічний стек розробки включає наступні компоненти:

– C# – основна мова програмування, яка забезпечує високу продуктивність та гнучкість для реалізації алгоритмів;

– Microsoft Visual Studio 2022 – інструмент розробки з широкими можливостями для тестування та аналізу роботи програмного забезпечення;

– ML.NET – фреймворк, що надає засоби для впровадження алгоритмів машинного навчання, таких як градієнтний бустинг, матрична факторизація та швидке дерево Твідді;

– .NET Framework – платформу для створення масштабованих і надійних додатків з багатим функціоналом і підтримкою хмарних технологій.

Застосування цього технологічного стеку дозволяє створити систему, яка поєднує в собі гнучкість, продуктивність та простоту впровадження алгоритмів. Завдяки інтеграції з Visual Studio, забезпечується оптимізація процесу розробки та підтримка складних обчислювальних операцій, що є необхідним для реалізації проєкту.

2.4 Постановка задачі

Для дослідження алгоритмів рекомендаційних систем буде створено віконний додаток, що дозволить проводити навчання, тестування та аналіз продуктивності моделей у різних умовах.

Основні функціональні вимоги:

– навчання та тестування моделей: додаток має підтримувати реалізацію та навчання алгоритмів, таких як градієнтний бустинг, матрична факторизація та швидке дерево Твідді, з можливістю тестування їх точності та швидкодії;

– робота з рекомендаціями: забезпечення функціоналу для генерації рекомендацій на основі введених даних та аналізу продуктивності моделей;

– аналіз використання ресурсів: реалізація інструментів для оцінки використання пам’яті, часу навчання та генерації рекомендацій у залежності від обсягу даних;

– авторизація та безпека даних: підтримка реєстрації та авторизації користувачів, а також фіксація подій, таких як навчання та використання моделей, для збереження історії дій;

– зберігання моделей: додаток має забезпечувати можливість збереження натренованих моделей для подальшого використання або перенавчання.

Вхідні дані для додатку включають:

– дані тренувального набору. Датасет з інформацією про взаємодію користувачів із аніме, включаючи рейтинги, ідентифікатори користувачів і об'єктів рекомендацій, жанри та додаткові характеристики;

– параметри алгоритмів. Конфігурації для навчання моделей, такі як кількість ітерацій, розмір тренувального набору та вибір цільових показників;

– сценарії тестування: Визначення умов, за яких проводиться тестування алгоритмів, включаючи кількість рекомендацій, обсяг вхідних даних та обчислювальні ресурси;

– параметри користувача. Інформація про дії користувачів для авторизації, запису логів та персоналізації результатів.

Вихідні дані додатку включають:

– результати навчання та тестування моделей: показники, такі як RMSE, MAE, R-Squared, час навчання та оцінки моделей;

– порівняльний аналіз алгоритмів: таблиці та графіки, які ілюструють продуктивність кожного алгоритму в залежності від обсягу даних і сценаріїв використання;

– рекомендації щодо використання алгоритмів: інформація про те, який алгоритм краще підходить для конкретних умов, з урахуванням обчислювальних ресурсів та вимог до точності;

– збережені моделі: файли з натренованими моделями, які можна використовувати для прогнозів або перенавчання.

Створення такого додатку забезпечить всебічний аналіз алгоритмів рекомендаційних систем, що дозволить отримати конкретні практичні результати та рекомендації для їх використання в реальних умовах.

Висновки до розділу 2

У рамках даного розділу проведено аналіз ключових компонентів, необхідних для розробки рекомендаційної системи, а також обґрунтовано вибір алгоритмів, тренувального набору даних та технологічного стеку. Зокрема, досліджено три основні алгоритми рекомендаційних систем: градієнтний бустинг, матричну факторизацію та швидке дерево Твідді. Для кожного алгоритму розглянуто математичний апарат, визначено їхні переваги та недоліки, що дозволило виділити найперспективніші методи для використання в системі.

Важливою частиною роботи стало обґрунтування вибору тренувального набору даних. Для реалізації системи використано датасет, сформований на основі платформи MyAnimeList, який містить понад 50 мільйонів оцінок аніме від майже 270 тисяч унікальних користувачів. Проведений аналіз набору даних показав його відповідність завданням побудови персоналізованих рекомендацій завдяки великому обсягу даних, структурованості та багатій метаінформації. Це дозволяє алгоритмам враховувати індивідуальні уподобання користувачів і специфіку контенту.

Окрім того, проведено детальне порівняння трьох мов програмування – Python, Java та C#. У результаті аналізу було обрано мову C#, яка забезпечує високу продуктивність, зручність у розробці та інтеграцію з екосистемою .NET. Для реалізації програмного забезпечення також обрано Microsoft Visual Studio

2022 як основне середовище розробки, ML.NET як фреймворк машинного навчання та .NET Framework, що забезпечує масштабованість і надійність системи.

Здійснено постановку задачі на розробку віконного застосунку, що передбачає навчання алгоритмів, тестування їхньої точності, збереження моделей для повторного використання, інтеграцію модуля авторизації користувачів та фіксацію подій для моніторингу активності. Це дозволить забезпечити зручність роботи із системою та зберегти історію її використання.

Отримані результати аналізу дозволяють сформувати чітку основу для реалізації рекомендаційної системи у наступному розділі, де буде здійснено її розробку, інтеграцію алгоритмів та тестування. Використання обґрунтованих алгоритмів, якісного тренувального набору даних і сучасного технологічного стеку забезпечить ефективність і надійність розробленого програмного забезпечення.

РОЗДІЛ 3 РОЗРОБКА ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Архітектура програмного забезпечення

Архітектура програмного забезпечення є ключовим етапом у розробці будь-якої інформаційної системи, адже від неї залежить продуктивність, гнучкість і масштабованість системи. У рамках даного підрозділу розглядається архітектурний підхід, який забезпечить ефективну реалізацію системи дослідження алгоритмів рекомендаційних систем. Особлива увага приділяється структурі компонентів, їх взаємодії та розподілу функцій, що дозволяє створити систему, здатну виконувати складні обчислення, інтегрувати алгоритми машинного навчання та забезпечувати зручний користувацький досвід.

Для розробки системи дослідження алгоритмів рекомендаційних систем було обрано трирівневу архітектуру через її здатність розділяти логічні рівні, що сприяє кращій структуризації та підтримуваності коду. Такий підхід дозволяє виділити три основні рівні: презентаційний, логічний (бізнес-логіки) та рівень даних. Презентаційний рівень забезпечує зручний інтерфейс для користувачів, що дозволяє легко взаємодіяти із системою, виконувати аналіз алгоритмів і переглядати результати. Логічний рівень відповідає за реалізацію алгоритмів рекомендаційних систем і обробку даних, забезпечуючи ефективне виконання складних математичних операцій. Рівень даних відповідає за зберігання та обробку великих обсягів інформації, включаючи взаємодію з тренувальним набором даних і метаданими.

Така архітектура забезпечує гнучкість у зміні окремих компонентів, що є важливим для систем, які працюють з експериментальними алгоритмами. Відокремлення рівнів також сприяє масштабованості, дозволяючи адаптувати систему для роботи з великими обсягами даних або розширювати її функціонал. Крім того, трирівнева архітектура полегшує тестування, що особливо важливо для забезпечення надійності при впровадженні нових алгоритмів або оновленні компонентів. Такий підхід дозволяє створити

стабільну, модульну і продуктивну систему, яка відповідає сучасним вимогам розробки програмного забезпечення.

На рис. 3.1 зображено діаграму класів рівня даних, яка демонструє структуру взаємодії між об'єктами системи. Ця діаграма відображає класи, їхні властивості, методи та зв'язки, що забезпечують функціонування рівня даних у системі дослідження алгоритмів рекомендаційних систем.

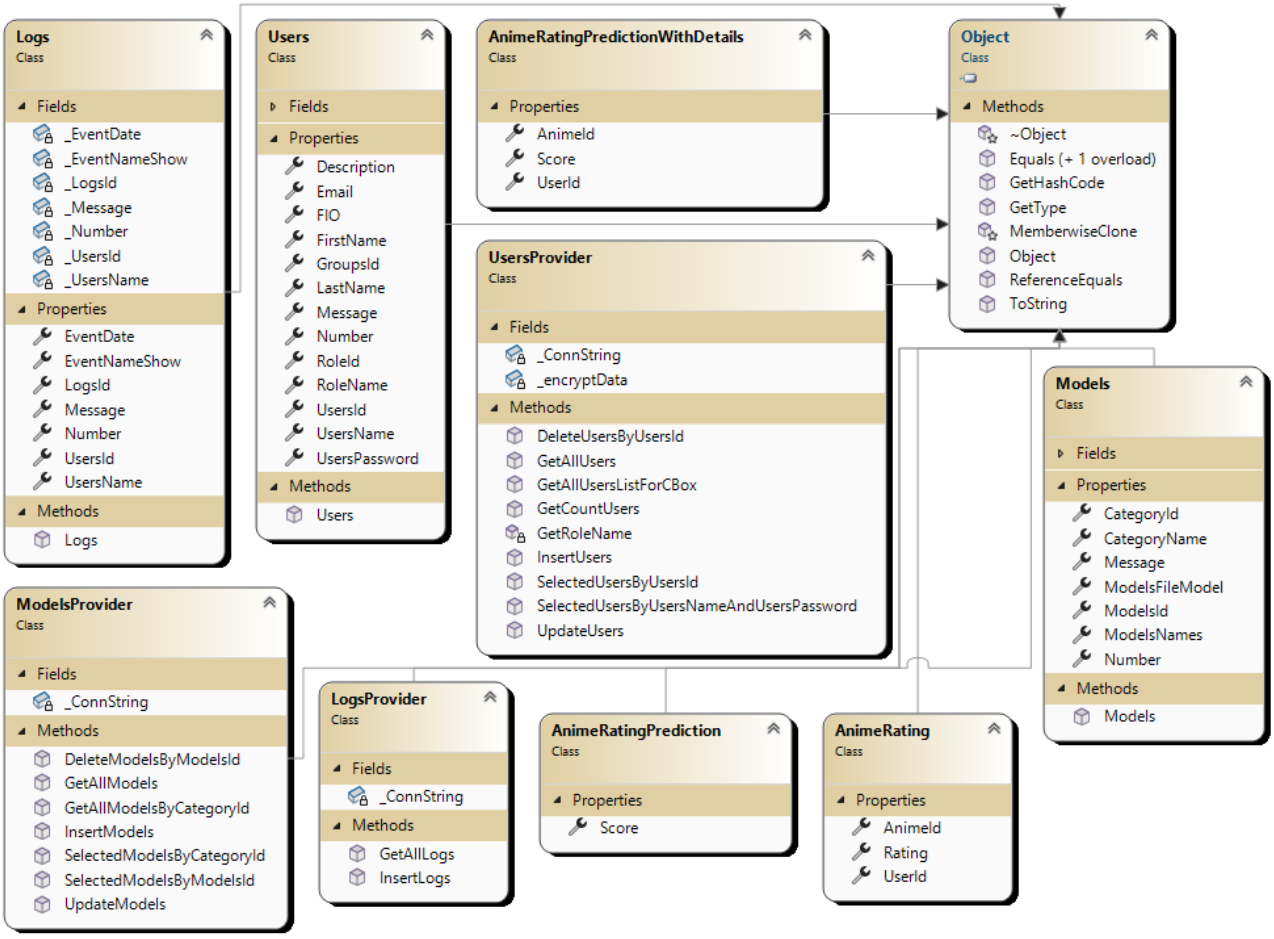

Рис. 3.1 – Діаграма класів рівня даних

Нижче розглянуто кожен із класів, представлений на діаграмі, відповідно до їх функціонального призначення:

– клас Logs відповідає за зберігання інформації про події у системі. Клас містить властивості, які відображають дату події (EventDate), її опис (EventNameShow), унікальний ідентифікатор події (LogsId), повідомлення (Message), а також ідентифікатор та ім'я користувача (UserId, UsersName).

Методи класу дозволяють отримувати або оновлювати інформацію про події в системі;

– клас Users використовується для управління інформацією про користувачів системи. Включає властивості, такі як ідентифікатор користувача (UserId), особисті дані (ім'я, прізвище, електронна пошта), а також інформацію про роль користувача (RoleId, RoleName). Методи забезпечують доступ до даних про користувачів;

– клас AnimeRatingPredictionWithDetails містить властивості для роботи з прогнозами рейтингу аніме. Включає ідентифікатор аніме (AnimeId), оцінку користувача (Score) та ідентифікатор користувача (UserId), забезпечуючи можливість аналізу прогнозів;

– клас UsersProvider відповідає за управління даними користувачів. Містить методи для додавання, оновлення, видалення та вибірки інформації про користувачів, зокрема на основі ідентифікатора або комбінації імені й пароля;

– клас ModelsProvider забезпечує доступ до даних, пов'язаних із моделями. Методи класу дозволяють отримувати, додавати, оновлювати та видаляти моделі, а також виконувати пошук за категоріями або ідентифікаторами;

– клас LogsProvider призначений для управління даними про події. Містить методи для отримання та додавання записів про події, що відображають діяльність системи;

– клас AnimeRatingPrediction відображає дані про передбачення рейтингів аніме. Містить властивості, які включають ідентифікатор аніме (AnimeId), оцінку (Score) та ідентифікатор користувача (UserId);

– клас Models відповідає за структуру моделі. Містить властивості для роботи з ідентифікаторами, назвами, категоріями моделей, а також іншою допоміжною інформацією (Message, Number).

Кожен із класів виконує чітко визначену роль, що забезпечує модульність і масштабованість системи. Це дозволяє ефективно взаємодіяти з даними,

обробляти запити користувачів і виконувати задачі, пов'язані з рекомендаційними алгоритмами.

На рис. 3.2 наведено діаграму класів рівня користувацького інтерфейсу системи, яка відображає взаємодію між основними елементами програмного забезпечення та користувачем. Ця діаграма демонструє структуру компонентів, їхні властивості, методи, а також зв'язки між класами, які забезпечують функціонування графічного інтерфейсу системи.

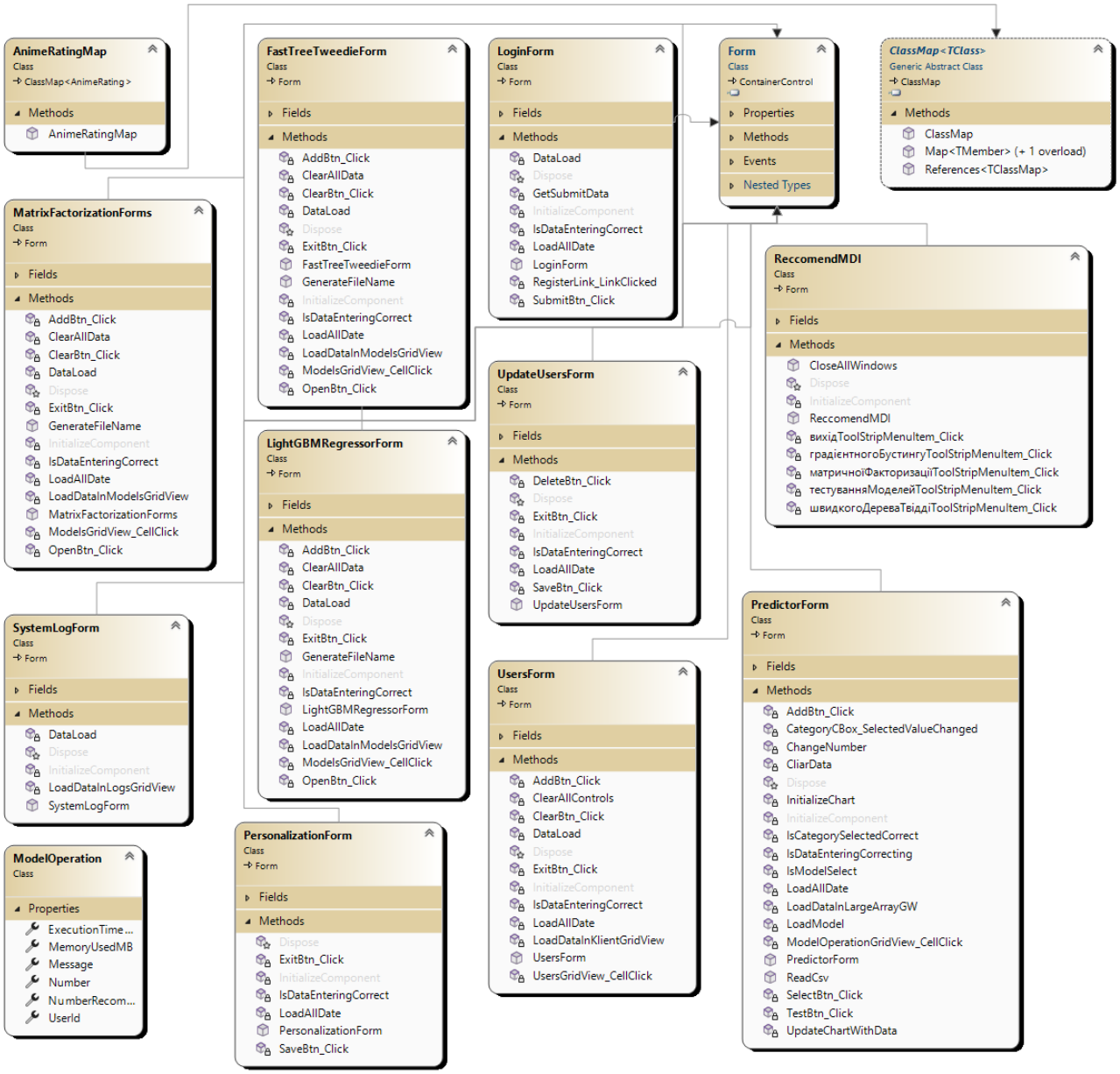

Рис. 3.2 – Діаграма класів рівня користувацького інтерфейсу

Нижче наведено детальний опис кожного з представлених класів:

– клас AnimeRatingMap відповідає за відображення карти рейтингу аніме. Містить метод AnimeRatingMap, що виконує основну логіку роботи з рейтингами для їхнього представлення у користувацькому інтерфейсі;

– клас MatrixFactorizationForm забезпечує інтерфейс для взаємодії користувача з алгоритмом матричної факторизації. Містить методи для додавання даних (AddBtn_Click), очищення форм (ClearAllBtn_Click), завантаження даних (LoadAllData) та відображення результатів у таблицях (MatrixFactorizationForm_Load);

– клас FastTreeTweedieForm відповідає за управління інтерфейсом для роботи з алгоритмом швидкого дерева Твідді. Включає методи для очищення даних, завантаження моделей і управління таблицями (AddBtn_Click, LoadAllData);

– клас LoginForm реалізує інтерфейс для авторизації користувача. Містить методи для отримання введених даних (GetSubmitData), перевірки правильності введення інформації (IsDataEnteringCorrect) та обробки подій авторизації (LoginForm_Load, SubmitBtn_Click);

– клас UpdateUserForm призначений для редагування даних користувачів. Містить методи для оновлення (UpdateUsersForm) та видалення користувачів (DeleteBtn_Click);

– клас LightGBMRegressorForm відповідає за інтерфейс для роботи з алгоритмом LightGBM. Включає функції для додавання, редагування даних та відображення результатів у таблицях (AddBtn_Click, LoadAllData, OpenBtn_Click);

– клас SystemLogForm використовується для відображення системних журналів. Методи класу забезпечують завантаження журналів (LoadDataLogsGridView) та взаємодію з таблицями логів;

– клас PersonalizationForm реалізує функції персоналізації користувацького інтерфейсу. Включає методи для збереження змін, очищення форм і завантаження даних (SaveBtn_Click, ClearAllBtn_Click);

– клас RecommendMDI є оловним контейнером для роботи з багатовіконним інтерфейсом. Забезпечує управління відкриттям та закриттям вікон для кожного алгоритму та функціонального модуля (CloseAllWindows, графікПопулярностіToolStripMenuItem_Click);

– клас PredictorForm Відповідає за прогнозування на основі моделей. Містить методи для обробки введених даних, відображення прогнозів та оновлення графіків (LoadDataAllRatingGridViewGW, UpdateChartWithData);

– клас UsersForm реалізує інтерфейс для роботи з інформацією про користувачів. Містить функції для додавання нових користувачів, оновлення та перевірки введених даних (AddBtn_Click, SaveBtn_Click);

– клас ModelOperation відображає основні операції, пов'язані з використанням моделей. Включає властивості для збереження часу виконання моделі (ExecutionTimeMS), кількості використаної пам'яті (MemoryUsedMB) та кількості рекомендованих елементів (NumberRecommends).

Представлені класи забезпечують багатофункціональний користувацький інтерфейс, дозволяючи інтерактивно працювати з даними, виконувати обчислення та аналізувати результати, що робить систему зручною та ефективною у використанні.

На рис. 3.3 представлено діаграму класів рівня бізнес-логіки системи, яка демонструє структуру компонентів, відповідальних за обробку даних і реалізацію основної логіки програми. Бізнес-логіка є центральним компонентом програмного забезпечення, що поєднує дані з їхньою обробкою, виконуючи ключові обчислення та прийняття рішень, необхідні для роботи системи. Вона виступає проміжним шаром між рівнем даних і користувацьким інтерфейсом, забезпечуючи ефективну обробку запитів і підтримку функціональності системи.

У цьому контексті класи бізнес-логіки виконують такі важливі завдання, як перевірка коректності даних, що надходять від користувачів або інших компонентів системи, управління доступом через ролі, шифрування та розшифрування інформації, а також взаємодія з категоріями для налаштування

структури та функціональності програми. Важливість бізнес-логіки полягає у забезпеченні стабільності, безпеки та відповідності системи вимогам користувачів.

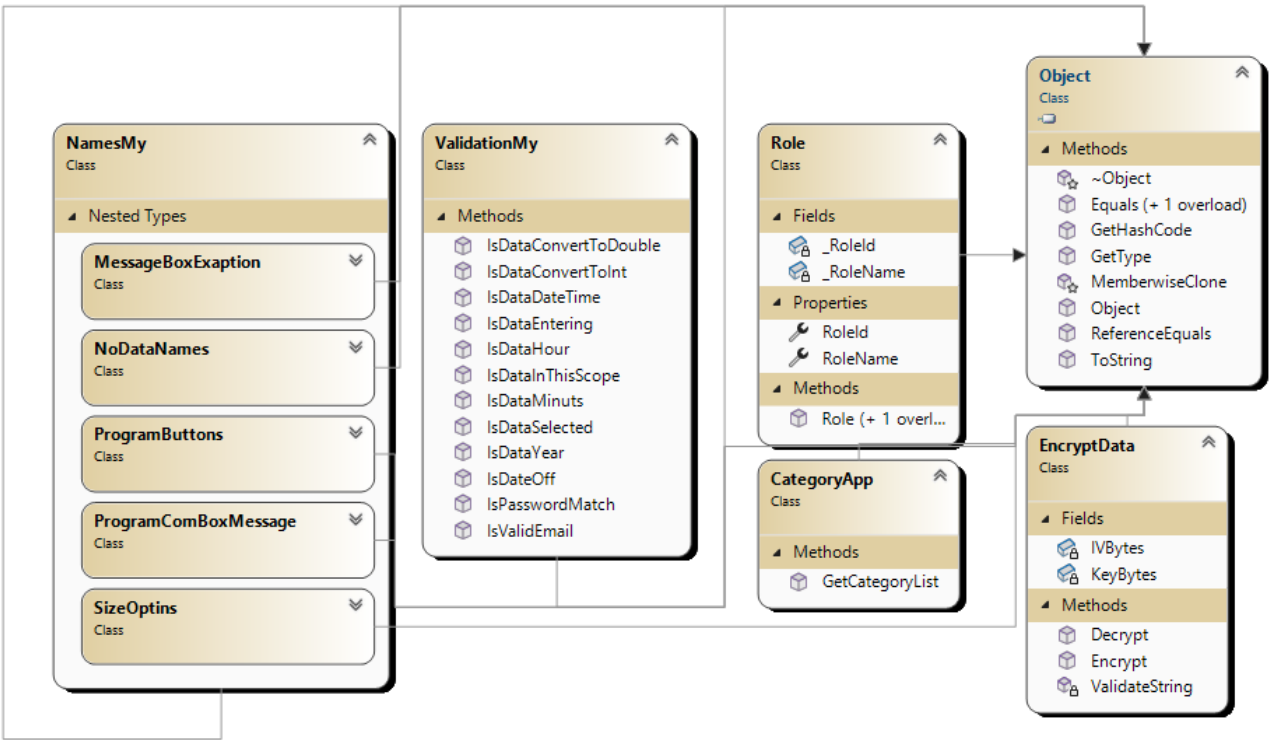

Рис. 3.3 – Діаграма класів рівня бізнес-логіки

Діаграма класів даного рівня складається із:

– клас NamesMy забезпечує структуру для роботи з загальними назвами та повідомленнями в програмі. Включає вкладені типи, такі як MessageBoxExaption (робота з виключеннями), NoDataNames (обробка відсутніх даних), ProgramButtons (управління кнопками програми), ProgramComBoxMessage (вибір повідомлень у списках) та SizeOptins (опції розмірів інтерфейсу);

– клас ValidationMy відповідає за перевірку коректності даних, що вводяться або обробляються системою. Містить методи для перевірки типів даних (IsDataConvertToDouble, IsDataConvertToInt), форматів дат (IsDataDateTime, IsDataYear), а також перевірку електронної пошти (IsValidEmail) та паролів (IsPasswordMatch). Забезпечує контроль якості введених даних;

– клас Role призначений для управління інформацією про ролі користувачів. Містить властивості, такі як RoleId (ідентифікатор ролі) і RoleName (назва ролі). Методи класу забезпечують доступ до інформації про ролі та їх обробку;

– клас CategoryApp відповідає за роботу з категоріями у програмі. Містить метод GetCategoryList, який дозволяє отримати список категорій, необхідних для роботи системи;

– клас EncryptData реалізує функціонал для шифрування даних. Включає поля для збереження байтів ключа (KeyBytes) і даних (IVBytes), а також методи для шифрування (Encrypt), розшифрування (Decrypt) і валідації рядків (ValidateString). Забезпечує безпеку збереження та передачі даних.

Класи рівня бізнес-логіки виконують критично важливі функції для забезпечення надійності системи, включаючи валідацію введених даних, безпеку інформації та управління доступом через ролі. Така структура сприяє модульності та спрощує подальший розвиток і підтримку програмного забезпечення.

3.2 Програмна реалізація обраних алгоритмів

Програмна реалізація обраних алгоритмів рекомендаційних систем є ключовим етапом створення ефективного програмного забезпечення, яке здатне вирішувати задачі персоналізації контенту. У цьому підрозділі розглядається реалізація алгоритмів, що базується на використанні мови програмування C# і технологій .NET, таких як ML.NET, що дозволяють забезпечити точність і продуктивність системи. Основна увага приділяється роботі з даними, які є фундаментальною частиною рекомендаційної системи. Завдяки чітко визначеним класам для роботи з даними та прогнозами система забезпечує ефективну інтеграцію алгоритмів і зручність у подальшій обробці результатів.

На рис. 3.4 представлено класи, які забезпечують функціональність для завантаження, зберігання та прогнозування даних, пов’язаних із взаємодіями користувачів та аніме. Класи структурують дані таким чином, щоб вони могли

бути використані як для навчання моделей, так і для отримання прогнозів. Нижче наведено детальний опис кожного з класів, їх функціонального призначення та ролі в системі.

```
// Основний клас для роботи з даними
20 references
public class AnimeRating {
    [LoadColumn(0)]
    [Name("user_id")]
    5 references
    public int UserId { get; set; }

    [LoadColumn(1)]
    [Name("anime_id")]
    8 references
    public int AnimeId { get; set; }

    [LoadColumn(2)]
    [Name("rating")]
    1 reference
    public float Rating { get; set; }
}
```

```
// Клас для прогнозу з додатковими деталями
0 references
public class AnimeRatingPredictionWithDetails {
    0 references
    public int UserId { get; set; }
    0 references
    public int AnimeId { get; set; }

    [ColumnName("Score")]
    0 references
    public float Score { get; set; }
}
```

Рис. 3.4 – Клас для завантаження, зберігання та прогнозування даних

Клас AnimeRating є основним для роботи з даними, які використовуються при навчанні моделей. Він містить властивості для ідентифікації користувача, ідентифікації аніме та рейтингу, який користувач призначив конкретному аніме. Атрибути [LoadColumn] забезпечують завантаження даних із певних стовпців у файл даних, а [Name] використовується для прив’язки до відповідних назв у наборах даних. Цей клас формує основу для роботи алгоритмів, що аналізують дані користувачів і їх вподобання.

Клас AnimeRatingPredictionWithDetails використовується для обробки прогнозів, отриманих від алгоритмів машинного навчання. Він містить властивості для ідентифікації користувача, аніме та прогнозованої оцінки (рейтингу), яка відображається у властивості Score. Атрибут [ColumnName] дозволяє налаштувати зв'язок між полями класу та результатами моделі машинного навчання. Цей клас забезпечує представлення результатів прогнозів і дозволяє використовувати їх для подальшого аналізу або генерації рекомендацій.

Обидва класи працюють у тісній взаємодії, забезпечуючи замкнутий цикл між підготовкою даних і аналізом результатів прогнозів. Завдяки чітко

визначеній структурі ці класи легко інтегруються в технологічний стек системи та сприяють реалізації алгоритмів машинного навчання з використанням технологій .NET.

На початковому етапі програмної реалізації було впроваджено метод матричної факторизації, який є одним із основних алгоритмів для рекомендаційних систем. Цей алгоритм дозволяє ефективно працювати з великими обсягами даних, де взаємодії між користувачами та елементами (наприклад, аніме) представлені у вигляді матриці. Впровадження алгоритму включало не лише реалізацію математичних операцій, але й створення інтерфейсу для завантаження даних, необхідних для його роботи.

У системі реалізовано функціонал завантаження файлів із даними за допомогою діалогового вікна (рис. 3.5). При натисканні кнопки викликається метод, який ініціює діалогове вікно для вибору файлу. Вікно налаштоване таким чином, щоб дозволити користувачеві вибирати файли у форматі CSV або будь-якому іншому форматі, якщо це необхідно. Діалогове вікно має фільтр для вибору відповідних файлів, що забезпечує зручність використання.

```
private void OpenBtn_Click(object sender, EventArgs e) {
    // Створення діалогового вікна для відкриття файлу
    OpenFileDialog openFileDialog = new OpenFileDialog();

    // Налаштування властивостей діалогового вікна
    openFileDialog.Filter = "CSV files (*.csv)|*.csv|All files (*.*)|*.*";
    openFileDialog.FilterIndex = 1;
    openFileDialog.RestoreDirectory = true;

    // Відображення діалогового вікна та обробка результату
    if (openFileDialog.ShowDialog() == DialogResult.OK) {
        _Path = openFileDialog.FileName;
        FileNameTBox.Text = _Path;
        RaportTBox.Text = "Завантаження даних...\r\n";
        Application.DoEvents();
```

Рис. 3.5 – Завантаження файлів із даними

Після вибору файлу система зберігає шлях до нього у внутрішній змінній _Path, яка дозволяє системі використовувати цей файл для подальшого аналізу. Шлях до файлу також виводиться в текстове поле інтерфейсу, щоб користувач міг переконатися у правильності вибору. Паралельно у вікно звітів виводиться повідомлення про початок завантаження даних, що забезпечує зворотний

зв'язок із користувачем та інформує про поточний статус роботи. Функція «Application.DoEvents» гарантує, що інтерфейс користувача залишається активним під час виконання задачі, дозволяючи користувачеві взаємодіяти з іншими елементами програми.

У коді на рис. 3.6 реалізовано ініціалізацію контексту для роботи з ML.NET, а також завантаження даних із зазначеного файлу. Спершу створюється об'єкт mlContext, який є базовим елементом для виконання всіх операцій, пов’язаних із машинним навчанням. Його створення з фіксованим значенням seed забезпечує відтворюваність результатів, дозволяючи отримувати однакові вихідні дані при кожному виконанні програми.

```
// Створення контексту ML
mlContext = new MLContext(seed: 0);

// Завантаження даних
dataView = mlContext.Data.LoadFromTextFile<AnimeRating>(_Path,
    hasHeader: true,
    separatorChar: ',');
```

Рис. 3.6 – Ініціалізацію контексту та завантаження даних

Після цього здійснюється завантаження даних із файлу, шлях до якого зберігається у змінній _Path. Для цього використовується метод LoadFromTextFile, який перетворює вхідний файл у формат, придатний для роботи з алгоритмами машинного навчання. Завантаження структуроване так, щоб відповідати класу AnimeRating, що дозволяє чітко визначити, які дані використовуються для аналізу. У процесі вказується, що файл має заголовок (hasHeader: true), а також зазначається символ-роздільник – кома (separatorChar: ','). Це забезпечує коректну обробку даних, незалежно від їхнього формату.

Результатом є об'єкт dataView, який є основною структурою для подальшого навчання моделей або оцінки даних. Такий підхід забезпечує гнучкість і універсальність у роботі з даними, полегшуючи подальшу обробку та інтеграцію алгоритмів машинного навчання у систему.

Код на рис. 3.7 виконує аналіз завантаженого набору даних, щоб отримати базову статистику про його структуру. За допомогою контексту mlContext дані у форматі dataView конвертуються у перелік об’єктів класу

AnimeRating. Це дозволяє здійснювати різноманітні операції, такі як підрахунок кількості унікальних значень у певних стовпцях або загальної кількості записів.

```
// Виведення інформації про дані
int userCount = mlContext.Data.CreateEnumerable<AnimeRating>(dataView,
  reuseRowObject: false).Select(x => x.UserId).Distinct().Count();
int animeCount = mlContext.Data.CreateEnumerable<AnimeRating>(dataView,
  reuseRowObject: false).Select(x => x.AnimeId).Distinct().Count();
long ratingCount = mlContext.Data.CreateEnumerable<AnimeRating>(dataView,
  reuseRowObject: false).LongCount();

RaportTBox.Text += ($"Кількість унікальних користувачів: {userCount}\r\n");
RaportTBox.Text += ($"Кількість унікальних аніме: {animeCount}\r\n");
RaportTBox.Text += ($"Кількість записів: {ratingCount}\r\n");
Application.DoEvents();
```

Рис. 3.7 – Аналіз завантаженого набору даних

Для визначення кількості унікальних користувачів код отримує значення поля UserId з усіх записів, видаляє повтори та рахує залишкові унікальні значення. Аналогічний підхід застосовується для підрахунку кількості унікальних аніме через ідентифікатор AnimeId. Загальна кількість записів у наборі даних визначається методом LongCount, який підраховує всі рядки у наборі без їхньої фільтрації.

Отримані результати додаються у текстове поле RaportTBox, забезпечуючи зворотний зв'язок із користувачем. У текстовому полі поступово відображається кількість унікальних користувачів, аніме та загальна кількість записів. Для забезпечення інтерактивності та відображення оновлень у реальному часі викликається функція Application.DoEvents(). Таким чином, цей код не лише аналізує структуру даних, але й інформує користувача про стан та основні характеристики завантаженого датасету.

У коді на рис. 3.8 реалізовано процес поділу набору даних на дві частини – тренувальну та тестову. Цей поділ є критично важливим етапом для забезпечення коректності оцінки алгоритмів машинного навчання, адже дозволяє перевірити, як модель справляється із завданням на нових, раніше невідомих даних.

```
// Розділяємо дані на тренувальний та тестовий набори
var trainTestData =
  mlContext.Data.TrainTestSplit(dataView, testFraction: 0.2);
var trainData = trainTestData.TrainSet;
var testData = trainTestData.TestSet;
```

Рис. 3.8 – Процес розподілу набору даних

Використовуючи метод TrainTestSplit із бібліотеки ML.NET, датасет, збережений у змінній dataView, розділяється на дві вибірки. Параметр testFraction задає частку тестових даних, у цьому випадку – 20%, що означає, що 80% даних будуть використані для навчання, а решта 20% для тестування. Результат зберігається у змінній trainTestData, яка містить два окремі набори: тренувальний (TrainSet) і тестовий (TestSet).

Тренувальні дані використовуються для побудови моделі, тобто для навчання алгоритму на вже відомих прикладах, а тестові – для оцінки її якості та перевірки здатності працювати з новими даними. Такий підхід дозволяє оцінити узагальнюючу здатність моделі та запобігає перенавчанню. Цей процес є стандартною практикою в машинному навчанні та необхідною умовою для отримання достовірних результатів під час тестування алгоритмів.

Код на рис. 3.9 реалізується створення конвеєра обробки даних і тренування моделі, яка використовує матричну факторизацію для побудови рекомендацій. Конвеєр обробки даних – це послідовність операцій, що застосовуються до вхідних даних для їх підготовки до навчання моделі. Цей підхід забезпечує автоматизацію та узгодженість процесу обробки даних.

```
var pipeline =
  mlContext.Transforms.Conversion.MapValueToKey(outputColumnName:
  "UserIdEncoded", inputColumnName: "UserId")
    .Append(mlContext.Transforms.Conversion.MapValueToKey(outputColumnName:
    "AnimeIdEncoded", inputColumnName: "AnimeId"))
    // Matrix Factorization Trainer
    .Append(mlContext.Recommendation().Trainers.MatrixFactorization(
        labelColumnName: "Rating",
        matrixColumnIndexColumnName: "UserIdEncoded",
        matrixRowIndexColumnName: "AnimeIdEncoded"
    ));
```

Рис. 3.9 – Створення конвеєра обробки даних

На початку конвеєр включає трансформацію даних, де категоріальні значення стовпців UserId і AnimeId перетворюються у числові ключі за допомогою методу MapValueToKey. Це необхідно для алгоритму матричної факторизації, який працює із числовими індексами, а не зі строковими чи іншими типами даних.

Наступним кроком у конвеєрі є додавання тренера для матричної факторизації. У цьому випадку використовується тренер, що входить до складу бібліотеки ML.NET, який налаштовується на використання стовпця Rating як цільової змінної, а UserIdEncoded і AnimeIdEncoded як індексів для матриці. Цей тренер дозволяє моделі визначити приховані залежності між користувачами та об’єктами (аніме), створюючи базу для точних персоналізованих рекомендацій. Результатом створення такого конвеєра є гнучка система, яка автоматично виконує всі необхідні перетворення даних і навчає модель у єдиному процесі.

Код на рис. 3.10 виконує навчання моделі та оцінку часу, необхідного для її навчання, що є важливим етапом у дослідженні її продуктивності. Для вимірювання швидкості використовується клас Stopwatch, який дозволяє точно фіксувати проміжки часу. На початку роботи запускається таймер, що починає відлік часу навчання.

```
// Дослідження швидкості навчання
Stopwatch trainingStopwatch = new Stopwatch();
trainingStopwatch.Start();
// Навчання моделі на тренувальному наборі
model = pipeline.Fit(trainData);
trainingStopwatch.Stop();
RaportTBox.Text += ($"Час тренування моделі: " +
  $"{trainingStopwatch.Elapsed.ToString("mm\\:ss\\.fff")}\r\n");
```

Рис. 3.10 – Навчання моделі та вимірювання часу навчання

Далі здійснюється навчання моделі на тренувальному наборі даних за допомогою створеного раніше конвеєра. Метод Fit виконує всі необхідні перетворення даних і адаптує модель до навчального набору, використовуючи алгоритм матричної факторизації. У цей момент модель аналізує взаємозв’язки

між користувачами, аніме та виставленими рейтингами, щоб визначити приховані закономірності.

Після завершення навчання таймер зупиняється, і обчислений час тренування відображається у форматі хвилин, секунд і мілісекунд. Результат додається до текстового поля RaportTBox, що надає користувачеві інформацію про тривалість цього процесу. Такий підхід дозволяє оцінити ефективність алгоритму та проаналізувати його придатність для використання у системах з великими обсягами даних.

У фрагменті коді на рис. 3.11 виконується вимірювання швидкості обчислень для оцінки моделі на тестовому наборі даних, що є важливим етапом аналізу її продуктивності та точності. Для цього використовується клас Stopwatch, який запускається на початку процесу, щоб зафіксувати час, необхідний для виконання оцінки.

```
// Оцінка моделі на тестовому наборі
var predictions = model.Transform(testData);
var metrics =
  mlContext.Regression.Evaluate(predictions, labelColumnName: "Rating");
evaluationStopwatch.Stop();
```

Рис. 3.11 – Вимірювання швидкості обчислень для оцінки моделі на тестовому наборі даних

На початку модель використовується для прогнозування результатів на тестовому наборі даних за допомогою методу Transform. Цей етап передбачає застосування навченої моделі до нових даних, щоб отримати прогнози, які потім порівнюються із фактичними значеннями у наборі. Далі метод Evaluate із модуля регресії ML.NET розраховує метрики, які визначають якість моделі. Для цього використовуються прогнозовані результати та значення рейтингу з тестового набору. Метрики, отримані в результаті, дозволяють оцінити, наскільки добре модель справляється зі своїм завданням, враховуючи цільову змінну Rating.

Після завершення обчислень таймер зупиняється, і зафіксований час дає змогу зрозуміти, наскільки швидко модель може виконувати свої прогнози. Такий аналіз є ключовим для визначення придатності моделі до реального

використання, особливо у сценаріях з великими обсягами даних або високими вимогами до швидкості роботи.

На рис. 3.12 навелено код, який відповідає за відображення результатів оцінки моделі, отриманих під час тестування на тестовому наборі даних. Перш за все, текстове поле RaportTBox оновлюється повідомленням про початок аналізу метрик, що дозволяє користувачеві отримати інформацію про якість роботи моделі.

```
// Виведення результатів оцінки моделі на тестовому наборі
RaportTBox.Text += ("Оцінка моделі на тестовому наборі:\r\n");
RaportTBox.Text += ($"RMSE: {metrics.RootMeanSquaredError:F2}\r\n");
RaportTBox.Text += ($"MAE: {metrics.MeanAbsoluteError:F2}\r\n");
RaportTBox.Text += ($"R-Squared: {metrics.RSquared:F2}\r\n");
RaportTBox.Text += ($"Час оцінки моделі: " +
  $"{evaluationStopwatch.Elapsed.ToString("mm\\:ss\\.fff")}\r\n");
```

Рис. 3.12 – Відображення результатів оцінки моделі

У полі результатів послідовно виводяться ключові метрики, які характеризують точність моделі. Значення RootMeanSquaredError (RMSE) показує середньоквадратичну похибку, яка відображає середній рівень відхилення прогнозів від реальних значень. MeanAbsoluteError (MAE) демонструє середню абсолютну похибку, що надає інформацію про точність у зрозумілому для інтерпретації форматі. Метрика R-Squared показує рівень пояснення варіації цільової змінної за допомогою моделі, і чим ближче це значення до 1, тим краще модель описує дані.

Окрім метрик, у текстове поле додається час, витрачений на оцінку моделі, що було зафіксовано за допомогою таймера evaluationStopwatch. Цей показник дозволяє оцінити ефективність моделі з точки зору швидкості роботи. Загалом, такий підхід забезпечує інформативний аналіз результатів тестування, дозволяючи зрозуміти як точність, так і продуктивність моделі.

Схожим чином, як і для матричної факторизації, було реалізовано код для алгоритмів швидкого дерева Твідді та градієнтного бустингу. Основна різниця полягає в структурі конвеєра, який формується відповідно до специфіки кожного з алгоритмів. Конвеєр обробки даних для швидкого дерева Твідді має

унікальну послідовність перетворень, що забезпечує оптимальну підготовку даних для роботи цього алгоритму.

У конвеєрі для швидкого дерева Твідді спочатку виконуються перетворення категоріальних значень полів UserId та AnimeId у числові ключі за допомогою методу MapValueToKey (рис. 3.13). Це дозволяє алгоритму обробляти ці дані як числові індекси. Далі відбувається зворотне перетворення ключів у числові значення через метод MapKeyToValue, після чого значення конвертуються у тип Single, який необхідний для роботи з функціями алгоритму.

```
var pipeline = mlContext.Transforms.Conversion.MapValueToKey(outputColumnName:
  "UserIdEncoded", inputColumnName: "UserId")
    .Append(mlContext.Transforms.Conversion.MapValueToKey(outputColumnName:
    "AnimeIdEncoded", inputColumnName: "AnimeId"))
    .Append(mlContext.Transforms.Conversion.MapKeyToValue("UserIdFloat",
    "UserIdEncoded"))
    .Append(mlContext.Transforms.Conversion.MapKeyToValue("AnimeIdFloat",
    "AnimeIdEncoded"))
    .Append(mlContext.Transforms.Conversion.ConvertType("UserIdFloat",
    outputKind: DataKind.Single))
    .Append(mlContext.Transforms.Conversion.ConvertType("AnimeIdFloat",
    outputKind: DataKind.Single))
    .Append(mlContext.Transforms.Concatenate("Features", "UserIdFloat",
    "AnimeIdFloat"))
    .Append(mlContext.Regression.Trainers.FastTreeTweedie(labelColumnName:
    "Rating", featureColumnName: "Features"));
```

Рис. 3.13 – Створення конвеєру обробки даних методу швидкого дерева Твідді

Після цих перетворень числові значення об'єднуються у єдину колонку Features за допомогою методу Concatenate, який створює необхідну структуру для навчання моделі. На останньому етапі додається тренер для алгоритму FastTreeTweedie, який налаштовується на використання стовпця Rating як цільової змінної та колонки Features як сукупності ознак. Цей тренер дозволяє моделі аналізувати та враховувати нелінійні залежності між характеристиками для побудови точних прогнозів.

Цей підхід забезпечує ефективну підготовку та навчання моделі, дозволяючи швидкому дереву Твідді оптимально працювати із завданнями прогнозування. Завдяки гнучкій структурі конвеєра система легко адаптується до різних типів даних, що робить її придатною для широкого спектра задач у рекомендаційних системах.

Подібно до реалізації конвеєра для швидкого дерева Твідді, для градієнтного бустингу побудовано окремий конвеєр обробки даних, який враховує специфіку цього алгоритму (рис. 3.14). Конвеєр починається з перетворення категоріальних значень UserId і AnimeId у числові ключі за допомогою методу MapValueToKey. Ця операція дозволяє працювати з даними у форматі, придатному для алгоритмів машинного навчання.

```
var pipeline = mlContext.Transforms.Conversion.MapValueToKey(outputColumnName:
  "UserIdEncoded", inputColumnName: "UserId")
    .Append(mlContext.Transforms.Conversion.MapValueToKey(outputColumnName:
    "AnimeIdEncoded", inputColumnName: "AnimeId"))
    .Append(mlContext.Transforms.Conversion.MapKeyToValue("UserIdFloat",
    "UserIdEncoded"))
    .Append(mlContext.Transforms.Conversion.MapKeyToValue("AnimeIdFloat",
    "AnimeIdEncoded"))
    .Append(mlContext.Transforms.Conversion.ConvertType("UserIdFloat",
    outputKind: DataKind.Single))
    .Append(mlContext.Transforms.Conversion.ConvertType("AnimeIdFloat",
    outputKind: DataKind.Single))
    .Append(mlContext.Transforms.Concatenate("Features", "UserIdFloat",
    "AnimeIdFloat"))
    .Append(mlContext.Regression.Trainers.LightGbm(labelColumnName:
    "Rating", featureColumnName: "Features"));
```

Рис. 3.14 – Створення конвеєру обробки даних методу градієнтного бустингу

Після цього виконуються зворотні перетворення за допомогою методу MapKeyToValue, що переводить ключі у числовий формат, який далі конвертується у тип Single за допомогою функції ConvertType. Це необхідно для створення єдиної структури даних, яка буде використовуватись у навчанні моделі.

Об’єднання числових значень у колонку Features здійснюється за допомогою методу Concatenate. Ця колонка стає основним вектором ознак, який використовує алгоритм для аналізу зв’язків між користувачами та аніме. На останньому етапі до конвеєра додається тренер LightGbm, який реалізує алгоритм градієнтного бустингу. Він працює з цільовою змінною Rating та використовує колонку Features як набір ознак.

Ця конфігурація конвеєра забезпечує навчання моделі на основі градієнтного бустингу, який ефективно виявляє складні взаємозв’язки у даних. Завдяки налаштуванню конвеєра система готова до роботи з великими

обсягами даних, забезпечуючи точність і продуктивність у завданнях прогнозування.

Для зберігання модлей реалізовано код, який наведено на рис. 3.15.

```
private void AddBtn_Click(object sender, EventArgs e) {
  if (IsDataEnteringCorrect()) {
    //Зберігання моделі
    string pathName = @"\teach\" + GenerateFileName() + ".zip";
    string localProj =
      System.IO.Path.GetDirectoryName(System.Reflection.Assembly.GetExecutingAssembly().Location);
    _ModelsProvider.InsertModels(ModelsNamesTBox.Text,
      Convert.ToInt32(CategoryCBox.SelectedValue), CategoryCBox.Text, pathName);
    mlContext.Model.Save(model, dataView.Schema, localProj + pathName);
    ClearAllData();
    _LogsProvider.InsertLogs(LoginForm.CurrentUser.UsersId,
      "Було навчено модель " +
      ModelsNamesTBox.Text, DateTime.Now);
    MessageBox.Show("Дані успішно збережено!");
  }
}
```

Рис. 3.15 – Зберігання натренованої моделі

Під час активації функції перевіряється правильність введених даних через метод «IsDataEnteringCorrect». Якщо перевірка успішна, відбувається формування унікального імені файлу для збереження моделі, використовуючи метод GenerateFileName, і визначення шляху до директорії, де буде зберігатися модель. Дані про нову модель, включаючи її ім'я, категорію та шлях до файлу, додаються у базу даних через метод _ModelsProvider.InsertModels. Після цього модель, разом зі схемою даних, зберігається у визначеній директорії за допомогою функції mlContext.Model.Save. Завершивши операцію збереження, система очищує введені дані, викликаючи метод ClearAllData, щоб підготувати інтерфейс до роботи з новими даними.

Для забезпечення аудиту подій у системі додається запис до журналу логів через метод _LogsProvider.InsertLogs. У цьому записі фіксується ідентифікатор користувача, який виконав операцію, назва моделі та час виконання дії. Наприкінці користувачу виводиться повідомлення через MessageBox.Show, яке підтверджує успішне завершення процесу збереження. Такий підхід забезпечує безпечне збереження результатів навчання моделей, документування операцій та інформування користувача.

Код на рис. 3.16 реалізує процес завантаження збереженої моделі, її інтеграції в систему та підготовки до виконання прогнозів. Спочатку визначається повний шлях до файлу моделі, об'єднуючи поточний шлях програми, отриманий через Application.StartupPath, з переданим аргументом FilePath. Цей підхід забезпечує гнучкість у розташуванні файлів моделі та їх завантаженні.

```
private void LoadModel(string FilePath) {
  string localProj = Application.StartupPath + FilePath;
  // Визначення DataViewSchema для конвеєра підготовки даних і навченої моделі
  DataViewSchema modelSchema;
  // Завантаження моделі
  ITransformer model = _Context.Model.Load(localProj, out modelSchema);
  // Використання моделі для прогнозування
  predictionEngine =
    _Context.Model.CreatePredictionEngine<AnimeRating, AnimeRatingPrediction>(model);
}
```

Рис. 3.16 – Завантаження моделі

Також створюється змінна типу DataViewSchema, яка використовується для опису структури даних, оброблених моделлю. Ця структура необхідна для коректної взаємодії з даними, що будуть оброблятися в подальшому.

Збережена модель завантажується за допомогою методу _Context.Model.Load, який повертає об'єкт типу ITransformer. Цей об'єкт містить всі трансформації та модель, які були створені під час навчання. Завдяки цьому завантажена модель може безпосередньо використовуватися для виконання прогнозів без додаткового налаштування.

Для прогнозування створюється об'єкт predictionEngine за допомогою методу _Context.Model.CreatePredictionEngine. Цей механізм дозволяє застосовувати модель до нових даних. Він створює зв'язок між класами AnimeRating (вхідні дані) та AnimeRatingPrediction (результати прогнозу), що забезпечує легкість і зручність у використанні моделі для генерації рекомендацій. Таким чином, код забезпечує інтеграцію моделі в систему та її готовність до виконання завдань прогнозування.

Тестування моделей є важливим етапом перевірки їхньої ефективності та продуктивності, який дозволяє оцінити точність, швидкість і обсяг

використаних ресурсів. У процесі тестування аналізуються результати роботи алгоритмів на нових даних, а також фіксуються ключові метрики для порівняння різних підходів.

У коді на рис. 3.17 реалізовано функціонал для запуску тестування моделей і збору даних про їх продуктивність. Спершу виконується ініціалізація графіка через метод InitializeChart, що готує візуалізацію для відображення результатів. Далі створюється об'єкт Stopwatch, який використовується для вимірювання часу виконання, та отримується початковий обсяг пам'яті за допомогою GC.GetTotalMemory(true). Це дозволяє відстежувати ресурси, які використовуються під час роботи моделей.

```
private void TestBtn_Click(object sender, EventArgs e) {
  InitializeChart(); // Ініціалізація графіка під час завантаження форми
                     // Створюємо об'єкт для вимірювання часу виконання
  var stopwatch = new System.Diagnostics.Stopwatch();
  // Отримуємо початковий обсяг пам'яті
  long initialMemory = GC.GetTotalMemory(true);
  // Проходимо через всі елементи _AllModelOperation
  foreach (var modelOperation in _AllModelOperation) {
    // Встановлюємо початкові значення часу та пам'яті
    stopwatch.Restart();
    long memoryBefore = GC.GetTotalMemory(true); // Пам'ять до виконання
```

Рис. 3.17 – Функціонал для запуску тестування моделей і збору даних про їх продуктивність

Тестування виконується для кожного елемента в колекції _AllModelOperation. Для кожної операції встановлюються початкові значення пам'яті й часу, що забезпечує точність вимірювань. Stopwatch перезапускається, а обсяг пам'яті до виконання операції фіксується за допомогою функції GC.GetTotalMemory(true). Таким чином, код забезпечує зручний механізм моніторингу продуктивності моделей, дозволяючи аналізувати та оптимізувати їхню роботу.

Код наведений на рис. 3.18 виконує генерацію рекомендацій для користувача на основі певної кількості найкращих прогнозів, які обчислюються моделлю. Спочатку фіксується обсяг пам'яті перед початком роботи через метод GC.GetTotalMemory(true), що дозволяє оцінити, скільки ресурсів було

використано під час виконання завдання. Це особливо корисно для аналізу продуктивності моделі.

```
var recommendations = new List<Tuple<int, float>>(); // Зберігаємо AnimeId та оцінку
// Використовуємо унікальні аніме зі списку _AllUniqueAnimes
for (int i = 0; i < modelOperation.NumberRecommendations && i < _AllUniqueAnimes.Count; i++) {
  var input = new AnimeRating {
    UserId = modelOperation.UserId,
    AnimeId = _AllUniqueAnimes[i].AnimeId // Використовуємо AnimeId з _AllUniqueAnimes
  };
  var prediction = predictionEngine.Predict(input);
  recommendations.Add(new Tuple<int, float>(input.AnimeId, prediction.Score));
}
// Вимірюємо час виконання
stopwatch.Stop();
```

Рис. 3.18 – Генерація рекомендацій для користувача

Для зберігання результатів створюється список recommendations, який містить пари значень: ідентифікатор аніме та передбачений рейтинг. Далі, використовуючи унікальні записи про аніме зі списку _AllUniqueAnimes, генерується прогноз для кожного з них. У циклі перевіряється, щоб кількість рекомендацій не перевищувала як максимальну кількість, вказану у modelOperation.NumberRecommendations, так і фактичну кількість доступних аніме. Для кожного елемента створюється об’єкт класу AnimeRating, який містить дані про користувача та ідентифікатор аніме. Використовуючи цей об'єкт, викликається метод Predict у predictionEngine, що обчислює прогнозовану оцінку для даного аніме. Після цього результат додається до списку рекомендацій у вигляді пари: ідентифікатор аніме та його прогнозований рейтинг.

По завершенні циклу таймер stopwatch зупиняється, фіксуючи час, витрачений на виконання операції. Це дозволяє оцінити, наскільки ефективно модель виконує прогнозування для заданої кількості рекомендацій. Код забезпечує не лише збереження результатів у зручному форматі, але й збір даних про використані ресурси, що є важливим для подальшого аналізу та оптимізації.

Код на рис. 3.19 завершує процес генерації рекомендацій, підраховує використані ресурси та відображає результати у текстовому полі для користувача. Спершу фіксується загальний час виконання операції у

мілісекундах, використовуючи значення таймера stopwatch.Elapsed.TotalMilliseconds, яке зберігається у властивості ExecutionTimeMilliseconds об’єкта modelOperation.

```
    modelOperation.ExecutionTimeMilliseconds = stopwatch.Elapsed.TotalMilliseconds;
    // Вимірюємо використану пам'ять
    long memoryAfter = GC.GetTotalMemory(false);
    modelOperation.MemoryUsedMB = (memoryAfter - memoryBefore) / (1024.0 * 1024.0); // Перетворюємо в MB
    RaportTBox.Text = "";
    // Виводимо результат в текстове поле
    RaportTBox.Text += ($"Операція номер: {modelOperation.Number}\r\n");
    RaportTBox.Text += ($"UserId: {modelOperation.UserId}\r\n");
    RaportTBox.Text += ($"Кількість рекомендацій: {modelOperation.NumberRecommendations}\r\n");
    RaportTBox.Text += ($"Час виконання: {modelOperation.ExecutionTimeMilliseconds:F2} мс\r\n");
    RaportTBox.Text += ($"Використана пам'ять: {modelOperation.MemoryUsedMB:F2} MB\r\n");
    RaportTBox.Text += "Рекомендації:\r\n";
    foreach (var recommendation in recommendations) // Виведемо топ-10 рекомендацій
    {
        RaportTBox.Text += ($"Аніме: {recommendation.Item1}, Оцінка: {recommendation.Item2:F2}\r\n");
    }
    RaportTBox.Text += "\r\n"; // Пустий рядок для розділення операцій
}
```

Рис. 3.19 – Завершення процесу генерації рекомендацій

Після цього визначається обсяг пам’яті, використаний під час виконання, шляхом обчислення різниці між обсягом пам’яті до (memoryBefore) та після операції (memoryAfter). Це значення перетворюється у мегабайти та записується у властивість MemoryUsedMB, що дозволяє оцінити ефективність використання пам’яті під час обчислень.

Текстове поле RaportTBox очищується, після чого в нього додається детальний звіт про виконання операції. Виводиться номер операції, ідентифікатор користувача (UserId), кількість згенерованих рекомендацій та основні метрики продуктивності, такі як час виконання та використана пам’ять. Це дозволяє користувачеві отримати чітке уявлення про ефективність обробки даних.

Далі виводиться список рекомендацій, де для кожного елемента відображається ідентифікатор аніме та його прогнозований рейтинг. Текст форматується для зручності читання, включаючи пустий рядок для розділення результатів операцій. Цей підхід забезпечує повноцінну візуалізацію результатів тестування, надаючи користувачеві всі необхідні дані для аналізу роботи моделі.

Метод «UpdateChartWithData», який наведено на рис. 3.20 відповідає за оновлення даних на графіку, забезпечуючи їх актуальність відповідно до останніх результатів роботи системи. Спочатку перевіряється, чи графік має серії даних для оновлення. Якщо такі серії існують, вони асоціюються із змінними: executionTimeSeries, що відповідає часу виконання операцій, та memoryUsedSeries, яка відображає використання пам'яті.

```
private void UpdateChartWithData() {
  // Переконуємося, що є серії для оновлення
  if (GraphicsCC.Series.Count > 0) {
    var executionTimeSeries = (LineSeries)GraphicsCC.Series[0];
    var memoryUsedSeries = (LineSeries)GraphicsCC.Series[1];

    // Очищуємо старі дані
    executionTimeSeries.Values.Clear();
    memoryUsedSeries.Values.Clear();

    // Додаємо нові дані
    foreach (var operation in _AllModelOperation) {
      executionTimeSeries.Values.Add(operation.ExecutionTimeMilliseconds);
      memoryUsedSeries.Values.Add(operation.MemoryUsedMB);
    }
  }
}
```

Рис. 3.20 – Оновлення даних на графіку

Для забезпечення точності відображення попередні дані з обох серій очищуються за допомогою методу Clear, щоб уникнути накладання нових значень на старі. Це дозволяє забезпечити візуальну чіткість графіка. Далі в циклі додаються нові дані для кожної операції, яка зберігається у колекції _AllModelOperation. Значення часу виконання (ExecutionTimeMilliseconds) додаються до серії executionTimeSeries, а обсяг використаної пам’яті (MemoryUsedMB) – до серії memoryUsedSeries. Це забезпечує відображення актуальних показників продуктивності на графіку.

Метод гарантує, що графік завжди містить лише релевантні дані, відображаючи інформацію у зручному для аналізу форматі. Такий підхід сприяє інтерактивності та зрозумілості інтерфейсу для користувача.

3.3 Тестування програмного забезпечення

Тестування програмного забезпечення є невід'ємною частиною процесу розробки, що дозволяє перевірити коректність роботи всіх функціональних компонентів системи. Основною метою тестування є виявлення помилок, оцінка продуктивності та забезпечення відповідності функціоналу заявленим вимогам. У рамках даного підрозділу розглядається тестування ключових форм та модулів, які забезпечують інтерактивність і зручність у використанні системи. Однією з таких форм є інтерфейс для тренування моделей, який, зокрема, використовує метод градієнтного бустингу.

На рис. 3.21 зображено форму, що використовується для тренування моделей, яка демонструє результати тестування програмного забезпечення. Ця форма дозволяє завантажувати датасети, обирати категорії моделей та тренувати їх за допомогою різних алгоритмів, таких як градієнтний бустинг. Основні елементи форми забезпечують доступ до базового функціоналу: вибір файлу для завантаження даних, введення назви моделі, вибір категорії та запуск процесу тренування.

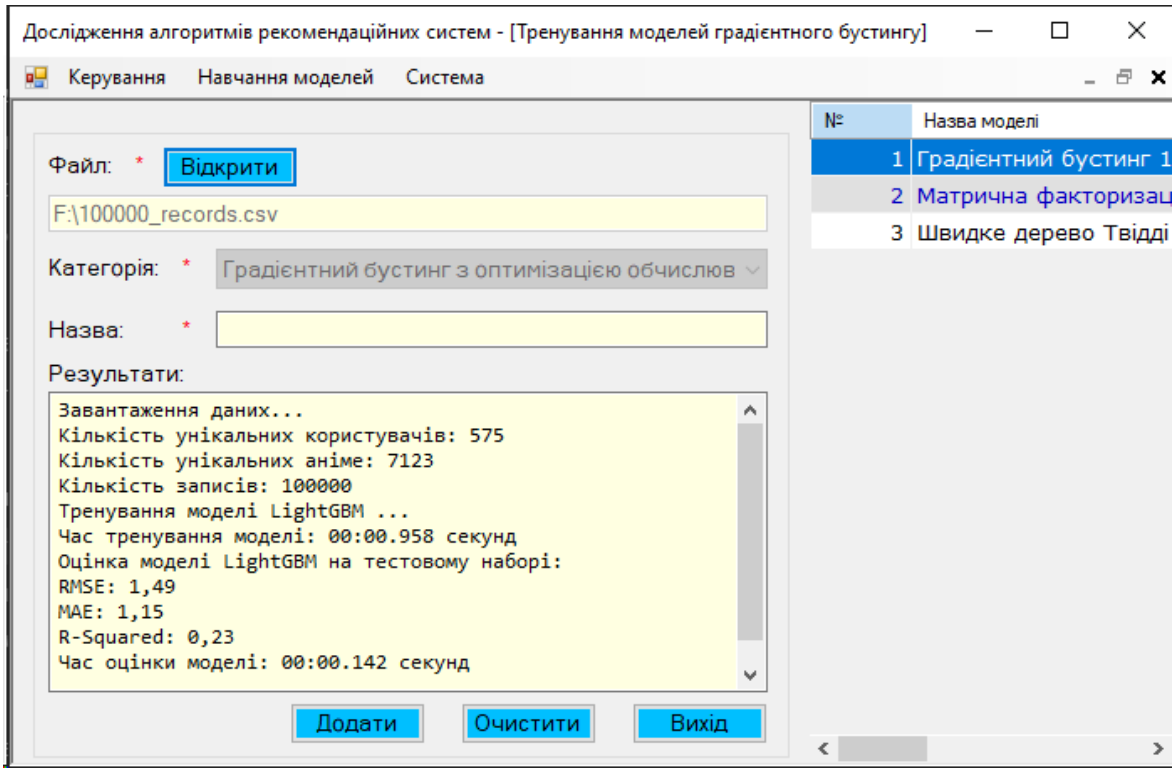

Рис. 3.21 – Тренування моделі із допомогою методу градієнтного бустингу

Результати тренування відображаються в текстовому полі «Результати». Тут надається детальна інформація про кількість унікальних користувачів та аніме у датасеті, загальну кількість записів, час, витрачений на тренування, а також основні метрики оцінки моделі, такі як RMSE, MAE та R-Squared. Окремо зазначено час, необхідний для оцінки моделі на тестовому наборі даних. Інтеграція результатів тестування у цю форму дозволяє користувачеві отримувати повний зворотний зв’язок про якість і продуктивність моделі, що є важливим для прийняття рішень щодо її використання.

На рис. 3.22 також відображено результати тестування роботи форми для тренування моделі, яка реалізує метод матричної факторизації. У цьому випадку була перевірена коректність функціонування основних елементів інтерфейсу, включаючи завантаження датасету, вибір категорії алгоритму та виведення результатів тренування. Особливу увагу було приділено перевірці точності підрахунку метрик моделі та відображенню часу, витраченого на навчання та оцінку.

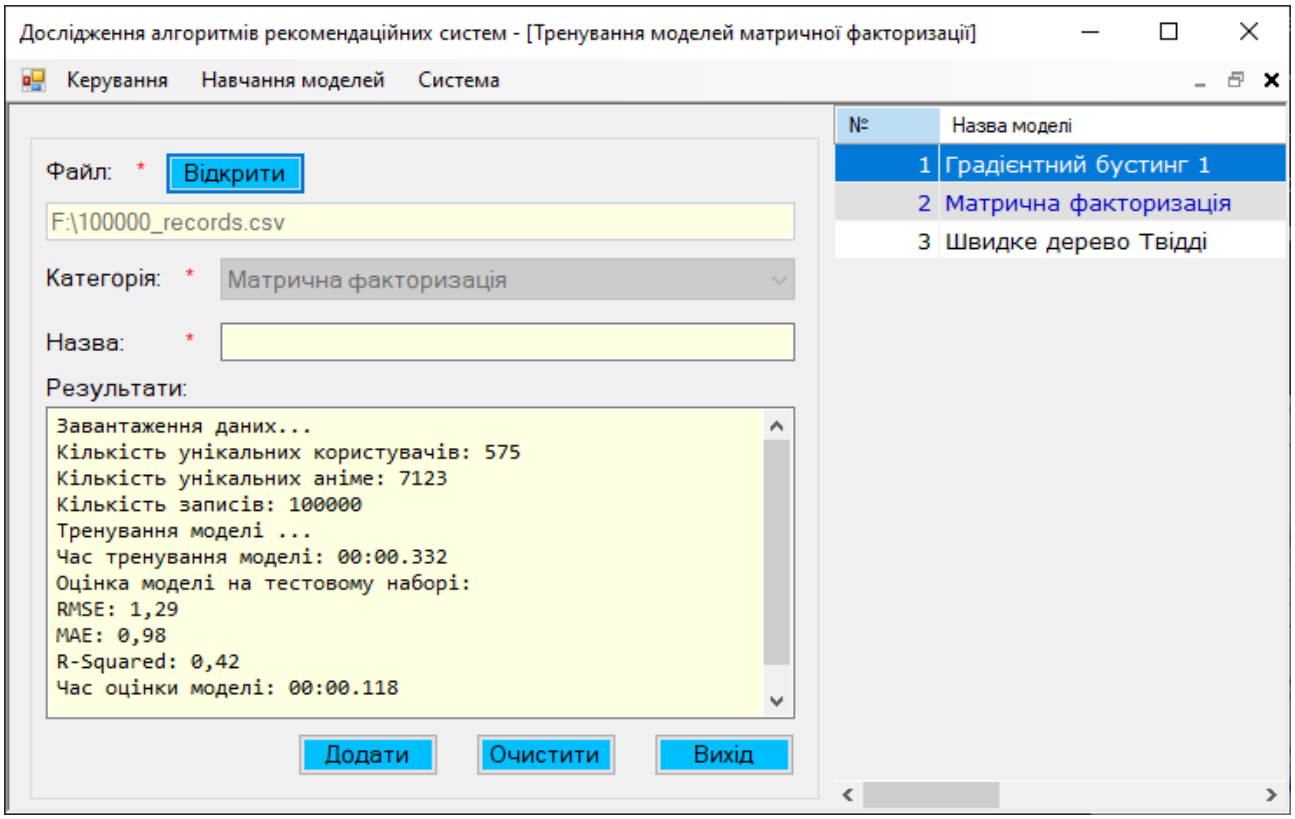

Рис. 3.22 – Тренування моделі із допомогою методу матричної факторизації

На рис. 3.23 також представлено результати тестування форми для тренування моделі із застосуванням алгоритму швидкого дерева Твідді. У рамках тестування перевірено коректність реалізації специфічних функцій, властивих цьому алгоритму, включаючи обробку особливостей даних, розрахунок метрик точності та часу виконання.

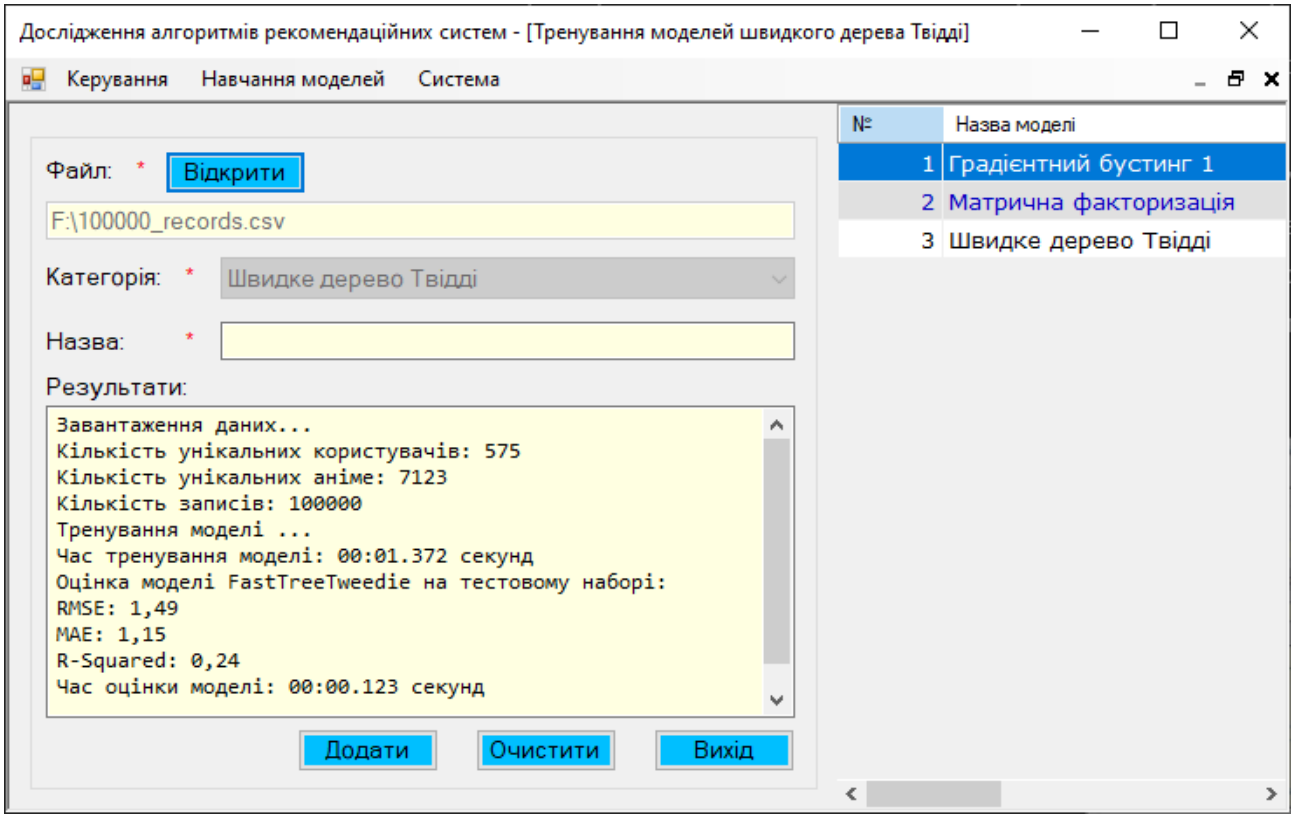

Рис. 3.23 – Тренування моделі із допомогою методу швидкого дерева Твідді

Особлива увага була приділена правильності роботи алгоритму з різними кількостями записів та унікальних значень у наборі даних. Усі етапи – від завантаження датасету до відображення результатів метрик RMSE, MAE та R-Squared – відпрацювали без помилок. Система також успішно обробила обсяг пам’яті, що використовується, та забезпечила коректне збереження результатів.

Додатково перевірено функцію введення та збереження назв моделей. Система коректно інтегрувала отримані моделі у список доступних, що підтверджує стабільність механізму взаємодії між інтерфейсом користувача та логікою збереження. Особливості роботи алгоритму швидкого дерева Твідді,

такі як обчислення часу тренування (1,372 секунди) та оцінки моделі (0,123 секунди), підтвердили високу продуктивність реалізації.

Тестування підтвердило, що форма коректно обробляє всі дані, незалежно від специфіки алгоритму. Це свідчить про високу універсальність інтерфейсу та надійність роботи системи навіть із більш складними моделями, такими як швидке дерево Твідді.

На наступному етапі тестування була перевірена форма перевірки моделей, яка відображена на рис. 3.24. Основним завданням цього етапу було забезпечити правильну роботу всіх функціональних елементів форми, що відповідають за вибір моделі, формування списків рекомендацій і проведення тестування.

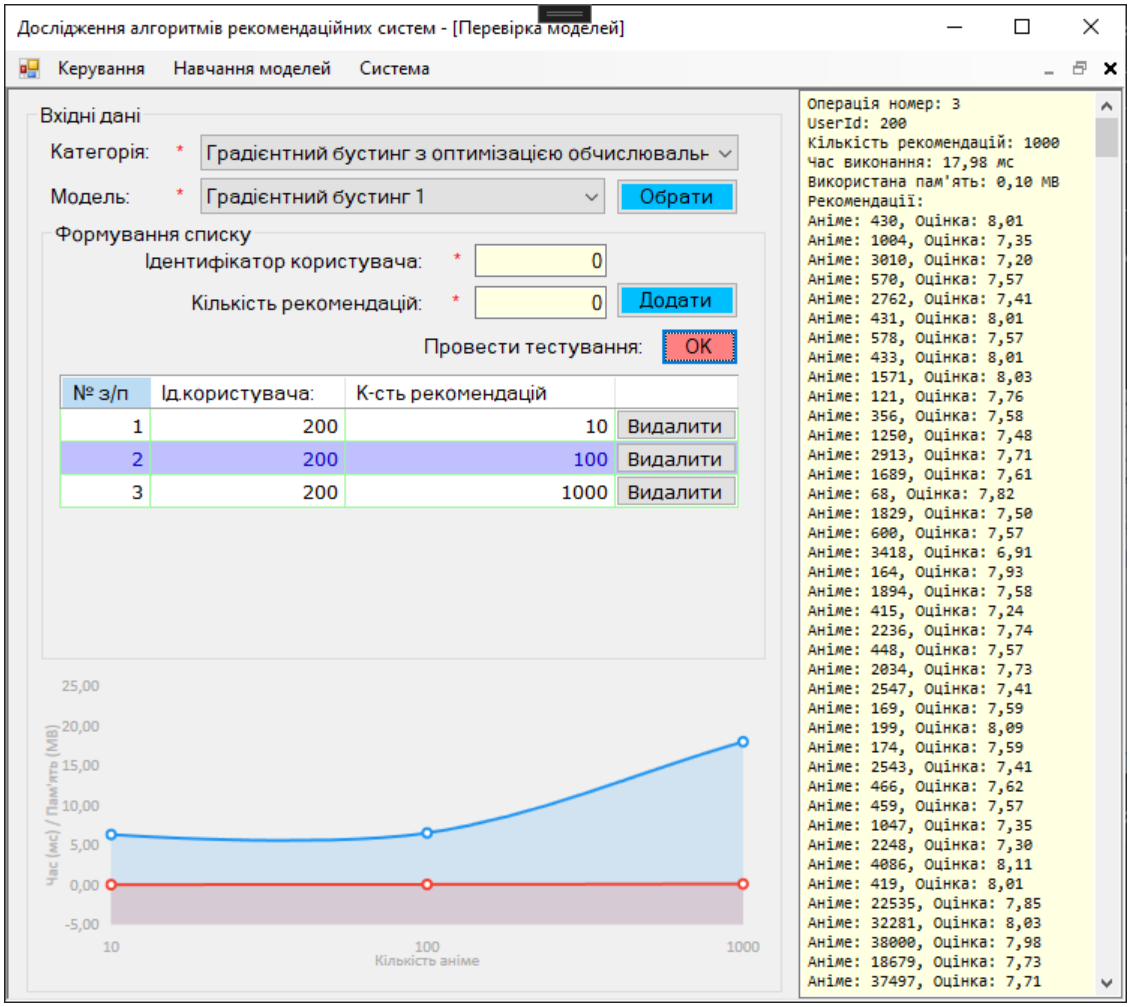

Рис. 3.24 – Форма для тестування моделей

У формі успішно протестовано функціонал вибору категорії алгоритму та відповідної моделі. Поля введення ідентифікатора користувача та кількості

рекомендацій коректно приймають дані, а кнопка "Додати" дозволяє формувати список завдань для тестування. Для кожного завдання у таблиці відображаються параметри: ідентифікатор користувача, кількість рекомендацій і можливість видалення запису. Система правильно обробляє всі внесені зміни та оновлює інтерфейс у реальному часі.

Тестування також підтвердило коректність роботи кнопки "ОК", яка запускає обчислення рекомендацій для зазначених завдань. У правій частині форми у реальному часі відображаються результати, включаючи номер операції, час виконання, використану пам’ять і детальний список рекомендацій із прогнозованими оцінками. Важливо зазначити, що система коректно обробляє різні сценарії, зокрема завдання із різною кількістю рекомендацій, що дозволяє тестувати продуктивність моделей у різних умовах.

Додатково було перевірено роботу графіка, який оновлюється під час виконання тестування. На графіку відображаються залежності часу виконання та використаної пам’яті від кількості рекомендацій. Це дозволяє користувачеві наочно аналізувати продуктивність алгоритмів та їхню ефективність.

Загалом тестування підтвердило, що форма працює стабільно та забезпечує зручний і функціональний інтерфейс для перевірки моделей. Усі елементи працюють коректно, а результати відображаються у зручному для аналізу вигляді, що дозволяє користувачеві приймати обґрунтовані рішення щодо якості та продуктивності алгоритмів.

Висновки до розділу 3

У рамках даного розділу було здійснено всебічну розробку та тестування програмного забезпечення для реалізації системи рекомендацій. Спочатку було обґрунтовано вибір трирівневої архітектури, яка забезпечує ефективну взаємодію між рівнями даних, бізнес-логіки та користувацького інтерфейсу. На основі цієї архітектури розроблено та описано діаграми класів для кожного рівня, що дозволило сформувати чітку структуру програмного забезпечення.

Детальний аналіз класів показав, як вони взаємодіють між собою для реалізації ключових функцій системи.

Програмна реалізація обраних алгоритмів включала приклади впровадження таких методів, як градієнтний бустинг, матрична факторизація та швидке дерево Твідді. Було розроблено конвеєри обробки даних, що забезпечують налаштування алгоритмів для роботи з конкретним набором даних. Завдяки цьому вдалося створити модулі, які дозволяють проводити навчання моделей, прогнозування та оцінку їхньої продуктивності. Наведені приклади коду підтверджують коректність реалізації математичних алгоритмів та інтеграцію з технологіями .NET.

На етапі тестування програмного забезпечення було перевірено форми для тренування моделей із використанням різних алгоритмів, а також форму для тестування моделей. Результати тестування підтвердили стабільність роботи всіх компонентів системи, зокрема правильність відображення результатів тренування, оцінки моделей та рекомендацій. Функціонал було протестовано на прикладі форм для тренування моделей за допомогою градієнтного бустингу, матричної факторизації та швидкого дерева Твідді. Також перевірено форму для тестування моделей, яка дозволяє генерувати рекомендації та візуалізувати продуктивність алгоритмів.

Отримані результати дозволяють перейти до наступного етапу – порівняння алгоритмів та формулювання рекомендацій щодо їх використання. Сформована архітектура, реалізовані алгоритми та результати тестування є базою для аналізу ефективності різних методів та обґрунтування їх застосування в конкретних умовах.

РОЗДІЛ 4 ПОРІВНЯННЯ АЛГОРИТМІВ ТА РЕКОМЕНДАЦІЇ ЩОДО ЇХ ВИКОРИСТАННЯ

4.1 Швидкість обробки даних у різних умовах

Ефективність алгоритмів рекомендаційних систем значною мірою залежить від їхньої здатності швидко обробляти великі обсяги даних та забезпечувати точність результатів. У цьому розділі представлено результати тренування та оцінки моделей, які використовувалися в експериментах із набором даних, що містить 100 000 записів, 575 унікальних користувачів і 7123 аніме. Основними алгоритмами для порівняння стали градієнтний бустинг, матрична факторизація та швидке дерево Твідді. Результати, наведені в табл. 4.1, дозволяють проаналізувати їхню продуктивність за ключовими показниками, включаючи швидкість навчання, точність та обчислювальні витрати.

Табл. 4.1 ілюструє, як кожен із алгоритмів справляється із завданнями рекомендацій за подібних умов.

Таблиця 4.1 – Результати навчання моделей на 100 000 записів

Характеристика	Градієнтний бустинг	Матрична факторизація	Швидке дерево Твідді
Кількість унікальних користувачів	575	575	575
Кількість унікальних аніме	7123	7123	7123
Кількість записів	100000	100000	100000
Час тренування моделі (секунд)	00:00.954	00:00.174	00:01.099
RMSE	1,49	1,28	1,49
MAE	1,15	0,98	1,15
R-Squared	0,23	0,42	0,24
Час оцінки моделі (секунд)	00:00.145	00:00.043	00:00.075

Наприклад, час тренування моделі для градієнтного бустингу становить 0.954 секунди, що є середнім значенням порівняно з іншими алгоритмами. Матрична факторизація показала найвищу швидкість навчання – 0.174 секунди,

значно перевершуючи конкурентів. Швидке дерево Твідді, хоч і продемонструвало найтриваліший час навчання (1.099 секунд), залишається конкурентоспроможним завдяки своїй універсальності.

На основі даних у табл. 4.1 видно, що матрична факторизація не лише найшвидша, але й забезпечує найкращі показники точності: RMSE 1.28 і R-Squared 0.42. Це робить її оптимальним вибором для систем, які потребують високої точності при мінімальних витратах часу. Градієнтний бустинг демонструє збалансовану продуктивність, забезпечуючи прийнятну швидкість навчання та точність. Швидке дерево Твідді, хоч і поступається швидкістю, підходить для сценаріїв, де потрібна більша адаптивність до особливостей даних, завдяки своїй стабільності.

Аналізуючи результати, наведені в табл. 4.2, можна зробити висновки про ефективність алгоритмів при збільшенні обсягів даних до 1 000 000 записів.

Таблиця 4.2 – Результати навчання моделей на 10 00 000 записів

Характеристика	Градієнтний бустинг	Матрична факторизація	Швидке дерево Твідді
Кількість унікальних користувачів	5616	5616	5616
Кількість унікальних аніме	12155	12155	12155
Кількість записів	1000000	1000000	1000000
Час тренування моделі (секунд)	00:04.089	00:01.383	00:08.972
RMSE	1,61	1,24	1,62
MAE	1,25	0,94	1,25
R-Squared	0,08	0,45	0,07

Матрична факторизація знову демонструє найкращі результати з точки зору швидкості навчання – лише 1.383 секунди, що суттєво перевершує час, необхідний для градієнтного бустингу та швидкого дерева Твідді. Також цей алгоритм забезпечує найточніші результати, досягаючи значення RMSE 1.24, MAE 0.94 та коефіцієнта детермінації (R-Squared) 0.45, що робить його ефективним для задач з високими вимогами до точності.

Градієнтний бустинг показує збалансовані результати, із часом тренування 4.089 секунди та середніми показниками точності: RMSE 1.61, MAE 1.25 і R-Squared 0.08. Цей алгоритм підходить для сценаріїв, де важлива стійкість роботи при збільшенні розмірів даних.

Швидке дерево Твідді демонструє найвищий час навчання – 8.972 секунди, що є його значним недоліком при роботі з великими обсягами даних. Крім того, його точність залишається на рівні градієнтного бустингу, з RMSE 1.62 та R-Squared 0.07. Проте час оцінки моделі – 0.767 секунди – є помірним і свідчить про достатню швидкість під час прогнозування.

На основі аналізу даних можна зробити висновок, що матрична факторизація залишається найкращим варіантом для завдань із великими обсягами даних, завдяки поєднанню високої швидкості тренування та точності. Градієнтний бустинг забезпечує прийнятну продуктивність, особливо у випадках, коли час тренування не є критичним. Швидке дерево Твідді може використовуватися у сценаріях, де висока швидкість тренування не є головним критерієм, а важлива стабільність алгоритму.

Аналізуючи результати, наведені в табл. 4.3, можна зробити висновки про ефективність алгоритмів при роботі з великими наборами даних обсягом 50 000 000 записів.

Таблиця 4.3 – Результати навчання моделей на 50 000 000 записів

Характеристика	Градієнтний бустинг	Матрична факторизація	Швидке дерево Твідді
Кількість унікальних користувачів	268973	268973	268973
Кількість унікальних аніме	16867	16867	16867
Кількість записів	50000000	50000000	50000000
Час тренування моделі (секунд)	247.626	71.881	541.314
RMSE	1,67	1,18	1,67
MAE	1,29	0,89	1,29
R-Squared	0,04	0,52	0,04
Час оцінки моделі (секунд)	41.374	19.051	54.643

Матрична факторизація знову демонструє найкращі результати за всіма ключовими параметрами. Час тренування моделі становить лише 71.881 секунди, що значно перевищує швидкість інших алгоритмів. Крім того, точність моделі, виражена через RMSE 1.18 і MAE 0.89, є найкращою серед порівнюваних алгоритмів. Значення коефіцієнта детермінації (R-Squared) 0.52 підтверджує високу здатність моделі до узагальнення.

Градієнтний бустинг демонструє середню швидкість тренування – 247.626 секунд, що є прийнятним результатом для таких великих наборів даних. Однак точність моделі є значно нижчою порівняно з матричною факторизацією: RMSE 1.67 і MAE 1.29, що свідчить про обмеження алгоритму при роботі з дуже великими обсягами даних. Час оцінки моделі, який становить 41.374 секунди, також є помірним, але поступається матричній факторизації.

Швидке дерево Твідді демонструє найтриваліший час тренування – 541.314 секунд, що є значним недоліком для задач із великими обсягами даних. Крім того, точність моделі на рівні RMSE 1.67 і R-Squared 0.04 аналогічна градієнтному бустингу, але значно поступається матричній факторизації. Час оцінки моделі також виявився найвищим серед усіх алгоритмів – 54.643 секунди, що знижує її привабливість для практичного використання.

Таким чином, матрична факторизація залишається найефективнішим алгоритмом, демонструючи відмінний баланс між швидкістю тренування та точністю результатів. Градієнтний бустинг може бути використаний у задачах із меншими вимогами до точності, але з помірним часом тренування. Швидке дерево Твідді, хоч і забезпечує стабільність, потребує значних обчислювальних ресурсів і демонструє низьку продуктивність у цьому сценарії.

Для візуалізації залежності часу навчання моделей від обсягу даних було створено графік, представлений на рис. 4.1. На ньому відображено, як змінюється тривалість тренування кожного з алгоритмів – градієнтного бустингу, матричної факторизації та швидкого дерева Твідді – зі збільшенням кількості записів у датасеті. Графік дозволяє чітко побачити різницю у

масштабованості кожного методу, що є ключовим аспектом для обробки великих наборів даних.

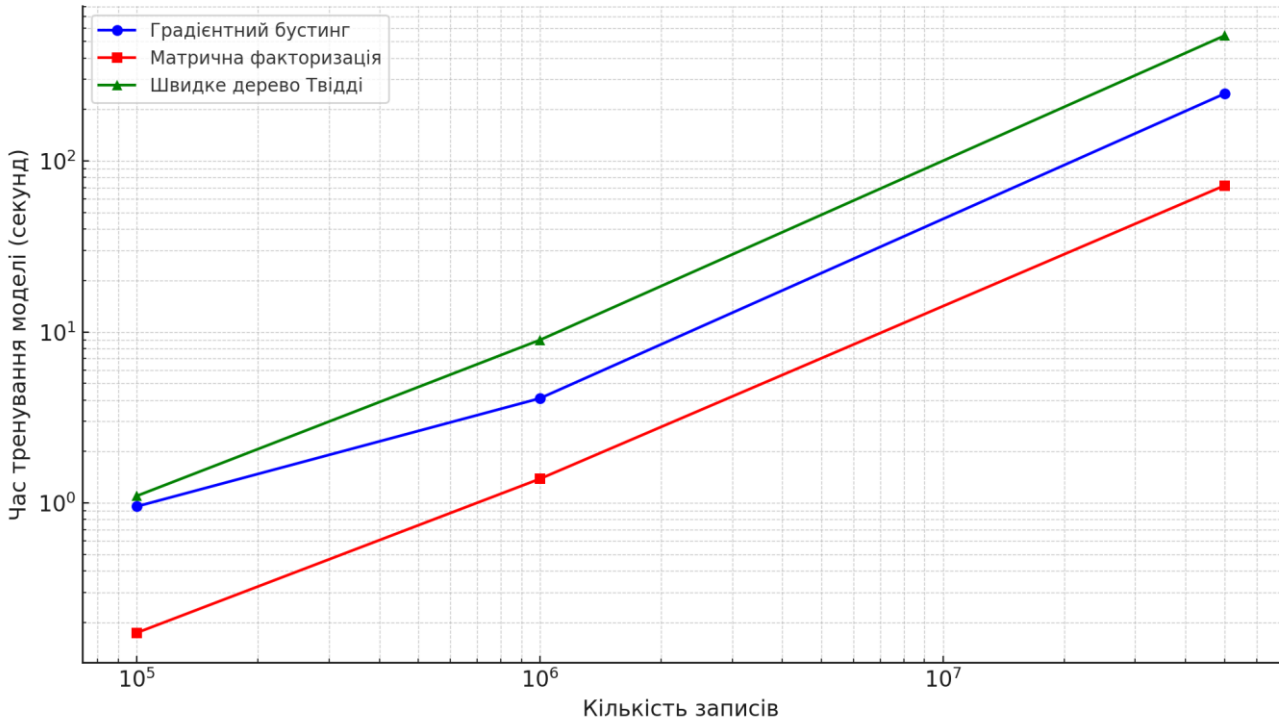

Рисунок 4.1 – Графік залежності часу навчання від кількості даних

Як видно з графіку, матрична факторизація демонструє найкращу продуктивність і стабільність у часі навчання, навіть при суттєвому зростанні кількості записів. Це робить її оптимальним вибором для задач, де важливою є висока швидкість тренування без значного збільшення витрат часу на обчислення.

Градієнтний бустинг, у свою чергу, має помірну залежність часу навчання від обсягу даних, зберігаючи стабільну швидкість при середніх розмірах датасетів. Він демонструє прийнятну масштабованість, яка може бути достатньою для багатьох сценаріїв, але поступається матричній факторизації у швидкості роботи.

Швидке дерево Твідді показує найбільшу залежність часу тренування від розміру набору даних. При великих обсягах записів час навчання значно зростає, що обмежує його ефективність у задачах, які вимагають швидкого виконання. Цей алгоритм може бути доцільним для застосування у вузьких сценаріях, де кількість даних є обмеженою.

Таким чином, графік наочно ілюструє переваги та недоліки кожного алгоритму у контексті їхньої масштабованості та продуктивності, що дозволяє приймати обґрунтовані рішення щодо їхнього використання у різних умовах.

4.2 Точність рекомендацій залежно від вхідних даних

У цьому підрозділі розглядається вплив обсягу вхідних даних на якість та швидкість формування рекомендацій, згенерованих натренованими моделями. Метою експерименту було оцінити, як різні алгоритми адаптуються до зміни розміру вхідного набору даних та як це позначається на їхній продуктивності. Було проведено серію тестів із різними обсягами даних, такими як 10, 100 і 1000 рекомендацій для одного користувача, що дозволяє наочно оцінити масштабованість та ефективність кожного з алгоритмів.

Результати генерації рекомендацій, представлені в табл. 4.4, охоплюють ключові характеристики, включаючи ідентифікатор користувача, кількість рекомендацій, споживану пам’ять і час, витрачений на виконання завдання. Ці параметри дозволяють порівняти роботу алгоритмів градієнтного бустингу, матричної факторизації та швидкого дерева Твідді за однакових умов. Проведений аналіз дає змогу оцінити реальну ефективність моделей у сценаріях з підвищеними вимогами до швидкості та точності.

Таблиця 4.4 – Результати надання рекомендацій

Характеристика	Градієнтний бустинг	Матрична факторизація	Швидке дерево Твідді
Ідентифікатор користувача	200	200	200
Кількість рекомендацій	10	10	10
Кількість виділеної пам’яті	0,02	0,01	0,03
Час генерації рекомендацій (мс.)	3,52	3,78	3,79

На основі даних таблиці можна зробити кілька висновків щодо поведінки алгоритмів. Градієнтний бустинг продемонстрував найшвидший час генерації рекомендацій – 3,52 мс, що свідчить про його ефективність у випадках із невеликим обсягом запитів. Матрична факторизація, хоча й витратила трохи

більше часу – 3,78 мс, забезпечила найнижче споживання пам’яті, що робить її оптимальним варіантом для сценаріїв із обмеженими ресурсами. Швидке дерево Твідді показало помірний результат із часом виконання 3,79 мс і найбільшим обсягом споживаної пам'яті (0,03 МБ), що вказує на потребу в оптимізації для сценаріїв, де важливі ресурсозатрати.

Для аналізу ефективності алгоритмів у сценаріях із середнім обсягом запитів було проведено тестування з формуванням 100 рекомендацій для одного користувача. Результати експерименту представлені в таблиці 4.5, яка дозволяє порівняти продуктивність алгоритмів за ключовими показниками: споживання пам’яті та час генерації рекомендацій.

Таблиця 4.5 – Результати надання рекомендацій

Характеристика	Градієнтний бустинг	Матрична факторизація	Швидке дерево Твідді
Ідентифікатор користувача	200	200	200
Кількість рекомендацій	100	100	100
Кількість виділеної пам’яті	0,04	0,02	0,04
Час генерації рекомендацій (мс.)	4,26	3,89	4,82

З аналізу даних видно, що матрична факторизація продовжує демонструвати найкращі результати серед представлених алгоритмів. Вона забезпечила найменший час генерації рекомендацій – 3,89 мс, що вказує на її високу ефективність у цьому сценарії. Крім того, цей алгоритм продемонстрував найнижчий рівень споживання пам’яті – 0,02 МБ, що робить його оптимальним для задач із середнім навантаженням.

Градієнтний бустинг показав час виконання 4,26 мс, що трохи перевищує показник матричної факторизації, але залишається прийнятним результатом. Споживання пам’яті також є на прийнятному рівні – 0,04 МБ, що свідчить про збалансованість цього алгоритму для задач, які не потребують максимальної оптимізації.

Швидке дерево Твідді, хоча й забезпечило прогнозування за 4,82 мс, продемонструвало найгірший час виконання серед трьох алгоритмів. Рівень

споживання пам’яті, аналогічний градієнтному бустингу (0,04 МБ), свідчить про стабільну роботу алгоритму, але із значними витратами ресурсів у порівнянні з матричною факторизацією.

Ці результати підтверджують, що матрична факторизація залишається найефективнішим варіантом у цьому сценарії, забезпечуючи високий рівень продуктивності при мінімальних витратах ресурсів. Градієнтний бустинг можна розглядати як альтернативу для випадків, де потрібна збалансована продуктивність, тоді як швидке дерево Твідді доцільно використовувати лише в умовах, де важлива стабільність алгоритму незалежно від часу виконання.

При тестуванні моделей із формуванням 1000 рекомендацій для одного користувача було отримано результати, представлені в табл. 4.6. Ці дані дозволяють оцінити ефективність алгоритмів у сценаріях із високим навантаженням, коли потрібно обробляти значні обсяги запитів. Основними критеріями аналізу залишаються час генерації рекомендацій і споживання пам’яті.

Таблиця 4.6 – Результати надання рекомендацій

Характеристика	Градієнтний бустинг	Матрична факторизація	Швидке дерево Твідді
Ідентифікатор користувача	200	200	200
Кількість рекомендацій	1000	1000	1000
Кількість виділеної пам’яті	0,11	0,11	0,11
Час генерації рекомендацій (мс.)	14,72	11,93	17,55

З отриманих даних видно, що матрична факторизація знову демонструє найкращі результати, завершуючи формування рекомендацій за 11,93 мс. Її продуктивність у цьому тесті підтверджує високу масштабованість алгоритму, що дозволяє ефективно працювати навіть із великими обсягами даних. Споживання пам’яті на рівні 0,11 МБ також є оптимальним і свідчить про стабільну роботу алгоритму.

Градієнтний бустинг потребував трохи більше часу для генерації рекомендацій – 14,72 мс. Хоча його продуктивність поступається матричній

факторизації, вона все ще залишається на прийнятному рівні для задач із високим навантаженням. Рівень споживання пам’яті також стабільний і дорівнює 0,11 МБ, що свідчить про збалансованість алгоритму.

Швидке дерево Твідді, з часом виконання 17,55 мс, виявилося найменш ефективним серед трьох алгоритмів у цьому тесті. Незважаючи на однакове зі спільниками споживання пам’яті, час генерації рекомендацій значно перевищує показники матричної факторизації та градієнтного бустингу, що робить його менш придатним для використання в умовах, де важлива швидкодія.

На основі проведеного аналізу можна зробити висновок, що матрична факторизація залишається найкращим вибором для роботи з великими обсягами запитів, забезпечуючи високу продуктивність та ефективність використання ресурсів. Градієнтний бустинг є прийнятним варіантом для задач, де потрібна стабільна швидкість роботи. Швидке дерево Твідді може використовуватися у випадках, коли точність алгоритму важливіша за його швидкодію.

Для кращого розуміння впливу кількості рекомендацій на швидкодію алгоритмів було створено графік, представлений на рис. 4.2.

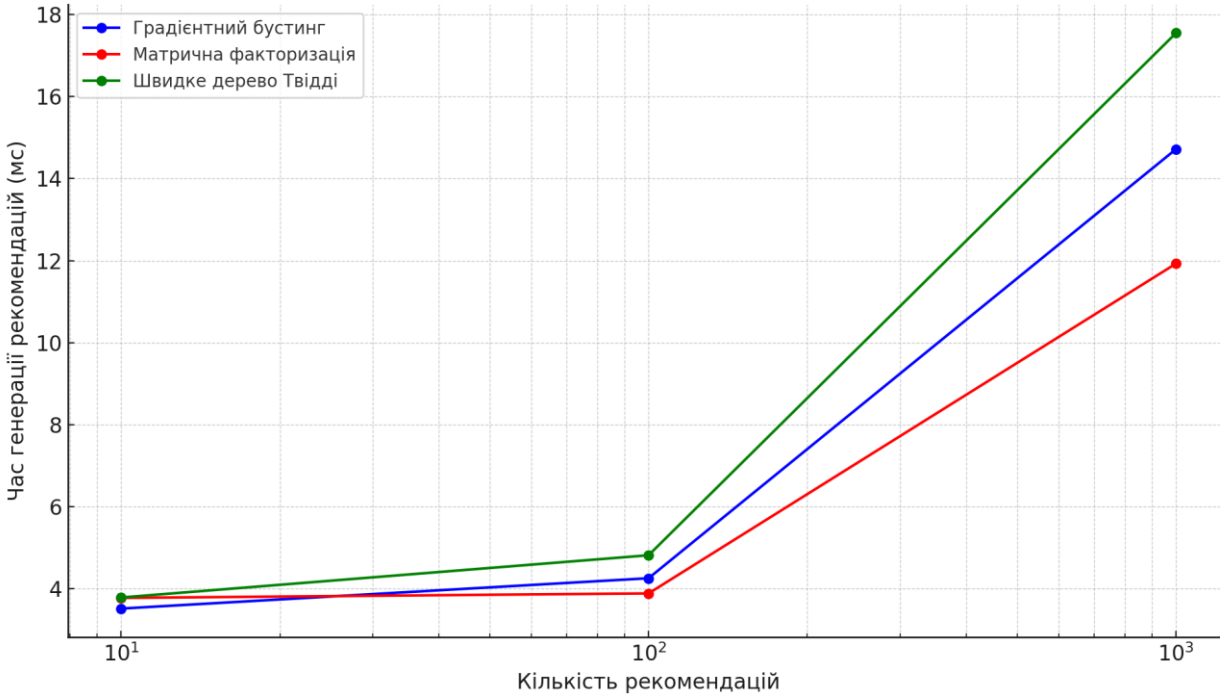

Рисунок 3.2 – Залежність часу генерації рекомендацій від кількості рекомендацій

Як видно з графіку, матрична факторизація демонструє найменший час генерації рекомендацій при всіх розглянутих обсягах даних. Цей алгоритм забезпечує стабільну продуктивність навіть при формуванні 1000 рекомендацій, що підтверджує його високу ефективність та адаптивність до великих обсягів запитів. Градієнтний бустинг займає проміжне положення, демонструючи помірне збільшення часу при збільшенні кількості рекомендацій. Це свідчить про його збалансованість, що робить його придатним для сценаріїв із середніми вимогами до швидкодії.

Швидке дерево Твідді показує найбільше зростання часу генерації рекомендацій із підвищенням кількості запитів. При формуванні 1000 рекомендацій його час виконання значно перевищує показники інших алгоритмів, що обмежує його використання у сценаріях, де потрібна висока швидкість обробки.

На рис. 4.3 представлено графік, що ілюструє залежність обсягу використаної пам'яті від кількості рекомендацій, сформованих за допомогою алгоритмів градієнтного бустингу, матричної факторизації та швидкого дерева Твідді.

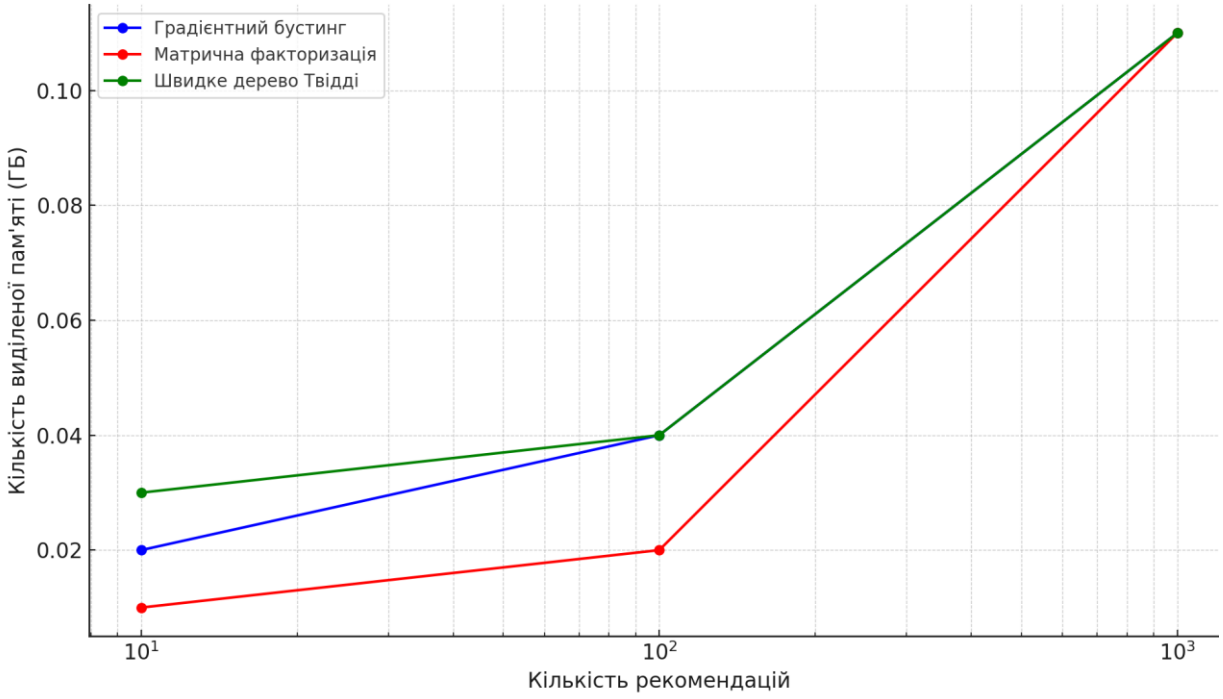

Рисунок 4.3 – Залежність кількості виділеної пам'яті від кількості рекомендацій для кожного алгоритму

Із графіку видно, що матрична факторизація демонструє найнижчі показники використання пам’яті при всіх тестованих кількостях рекомендацій. Це підтверджує її ефективність у задачах із обмеженими обчислювальними ресурсами. Навіть при обробці 1000 рекомендацій обсяг пам’яті залишається на прийнятному рівні.

Градієнтний бустинг має помірний рівень споживання пам'яті, який поступово зростає зі збільшенням кількості рекомендацій. Незважаючи на це, він залишається досить ефективним і придатним для використання в більшості сценаріїв.

Швидке дерево Твідді демонструє найвищі показники використання пам’яті при обробці великих обсягів запитів. Ця особливість може обмежувати його застосування у середовищах із жорсткими обмеженнями на обчислювальні ресурси.

Таким чином, матрична факторизація є найкращим вибором для систем, які мають обмежені ресурси пам’яті, тоді як градієнтний бустинг забезпечує збалансовану продуктивність. Швидке дерево Твідді підходить для сценаріїв, де доступні значні обчислювальні ресурси, але потрібна гнучкість у роботі з даними.

4.3 Вимоги до обчислювальних ресурсів і продуктивність на реальних даних

Аналіз вимог до обчислювальних ресурсів і продуктивності алгоритмів на реальних наборах даних є важливим етапом дослідження, який дозволяє оцінити, наскільки ефективно різні моделі справляються зі складністю обчислень у реальних умовах. Особливо це актуально для систем рекомендацій, які повинні забезпечувати швидку обробку великих обсягів даних у режимі реального часу. Вимоги до ресурсів, такі як обсяг споживаної пам’яті, швидкість обчислень і точність прогнозів, мають критичне значення при виборі алгоритму для конкретної задачі.

Для оцінки продуктивності були використані реальні набори даних із різною кількістю записів. Це дозволило порівняти поведінку кожного з алгоритмів у сценаріях з різними обсягами інформації. Розглянуті алгоритми, такі як градієнтний бустинг, матрична факторизація та швидке дерево Твідді, демонструють різні підходи до обробки даних, що впливає як на час тренування, так і на точність результатів. Основним критерієм аналізу було дослідження їхньої ефективності у залежності від розміру датасету, а також їхня здатність адаптуватися до збільшення обсягів вхідних даних.

У табл. 4.7 наведено результати навчання моделей на наборах даних із різною кількістю записів, що демонструє їхню здатність масштабуватися та ефективно використовувати обчислювальні ресурси.

Таблиця 4.7 – Результати навчання моделей при різних обсягах даних

Кількість записів	Час тренування (с)	RMSE	MAE	R-Squared
Градієнтний бустинг (100 000)	0.954	1.49	1.15	0.23
Градієнтний бустинг (1 000 000)	4.089	1.61	1.25	0.08
Градієнтний бустинг (50 000 000)	247.626	1.67	1.29	0.04
Матрична факторизація (100 000)	0.174	1.28	0.98	0.42
Матрична факторизація (1 000 000)	1.383	1.24	0.94	0.45
Матрична факторизація (50 000 000)	71.881	1.18	0.89	0.52
Швидке дерево Твідді (100 000)	1.099	1.49	1.15	0.24
Швидке дерево Твідді (1 000 000)	8.972	1.62	1.25	0.07
Швидке дерево Твідді (50 000 000)	541.314	1.67	1.29	0.04

Дані свідчать, що матрична факторизація забезпечує найшвидший час тренування навіть при значних обсягах даних, наприклад, для 50 мільйонів записів вона витрачає лише 71.881 секунди. Градієнтний бустинг демонструє помірну продуктивність, тоді як швидке дерево Твідді значно поступається за часом тренування. Щодо точності, матрична факторизація також випереджає інші алгоритми, показуючи найнижчі значення RMSE та MAE.

У табл. 4.8 наведено зміну часу генерації рекомендацій для кожного алгоритму при різній кількості рекомендацій.

Таблиця 4.8 – Час генерації рекомендацій при різній кількості рекомендацій

Кількість рекомендацій	Градієнтний бустинг (мс)	Матрична факторизація (мс)	Швидке дерево Твідді (мс)
10	3.52	3.78	3.79
100	4.26	3.89	4.82
1000	14.72	11.93	17.55

З результатів видно, що матрична факторизація демонструє найкращу продуктивність за швидкістю генерації рекомендацій, зберігаючи стабільність часу навіть при збільшенні кількості рекомендацій до 1000. Градієнтний бустинг займає проміжну позицію, тоді як швидке дерево Твідді потребує більше часу, особливо при великих обсягах рекомендацій.

У табл. 4.9 наведено, як змінюється споживання пам’яті при різній кількості рекомендацій.

Таблиця 4.9 – Споживання пам’яті при різній кількості рекомендацій

Кількість рекомендацій	Градієнтний бустинг (МБ)	Матрична факторизація (МБ)	Швидке дерево Твідді (МБ)
10	0.02	0.01	0.03
100	0.04	0.02	0.04
1000	0.11	0.11	0.11

Матрична факторизація демонструє найнижчі показники використання пам’яті для невеликих обсягів рекомендацій. Для 1000 рекомендацій обсяг споживаної пам’яті стає однаковим для всіх трьох алгоритмів, що свідчить про обмежений вплив збільшення обсягів рекомендацій на цей параметр.

Аналіз показав, що матрична факторизація є найкращим вибором для систем, які вимагають мінімального споживання ресурсів і швидкого виконання. Градієнтний бустинг є прийнятним для задач, де важлива точність, а

швидке дерево Твідді може використовуватись у сценаріях із помірними вимогами до продуктивності. Залежно від обсягу даних і кількості рекомендацій, вибір алгоритму має базуватися на конкретних вимогах до швидкодії, точності та ресурсів системи.

4.4 Практичні рекомендації щодо вибору методів для різних завдань

Вибір алгоритму для систем рекомендацій є важливим етапом, оскільки ефективність системи залежить від особливостей даних, обчислювальних ресурсів та вимог до точності й швидкості обробки. У рамках дослідження було проаналізовано три алгоритми: градієнтний бустинг, матрична факторизація та швидке дерево Твідді. Кожен із цих методів демонструє свої переваги у певних сценаріях використання, що дозволяє надати рекомендації щодо їх застосування залежно від специфіки завдань.

Градієнтний бустинг

Градієнтний бустинг виявився ефективним у задачах, де основна увага приділяється точності прогнозів. Завдяки здатності поступово вдосконалювати результати, цей алгоритм добре працює із даними, що мають складну структуру та велику кількість характеристик. Він є оптимальним для сценаріїв, де ресурси дозволяють використовувати методи з високими обчислювальними витратами, наприклад, для рекомендацій товарів у великих інтернет-магазинах або персоналізованого контенту на стрімінгових платформах. Градієнтний бустинг також підходить для систем, які працюють із відносно невеликими наборами даних, але вимагають високої точності.

Матрична факторизація

Матрична факторизація зарекомендувала себе як один із найбільш ресурсоефективних алгоритмів. Завдяки низьким вимогам до пам'яті та високій швидкості обробки, вона підходить для завдань, які вимагають швидкого формування рекомендацій у реальному часі. Матрична факторизація оптимальна для сценаріїв, де дані можуть бути представлені у вигляді матриці, наприклад, у системах рекомендацій фільмів, музики або електронних книг. Її

здатність до масштабування робить цей алгоритм ідеальним вибором для систем із великими обсягами даних, таких як онлайн-платформи з мільйонами користувачів та елементів контенту.

Швидке дерево Твідді

Швидке дерево Твідді є збалансованим алгоритмом, який поєднує в собі прийнятну швидкість і точність. Його можна рекомендувати для завдань, які потребують швидкого налаштування моделі під зміни у поведінці користувачів. Цей алгоритм добре підходить для інтерактивних систем, таких як мобільні додатки або платформи із середнім обсягом даних, де точність рекомендацій не є критичною, але важлива адаптивність та помірне використання ресурсів.

Комбінування алгоритмів

Для складних задач, які потребують поєднання точності, швидкості та адаптивності, доцільно використовувати комбіновані підходи. Наприклад, матрична факторизація може забезпечувати базові рекомендації для користувачів, тоді як градієнтний бустинг дозволить налаштовувати рекомендації з урахуванням поточної активності користувача. Швидке дерево Твідді може використовуватись як додатковий інструмент для швидкого реагування на зміни у поведінці.

Для задач із невеликим обсягом даних найдоцільніше використовувати градієнтний бустинг, оскільки він забезпечує високу точність навіть при малих вибірках. Для великих обсягів даних оптимальним вибором буде матрична факторизація, яка демонструє найкращі показники швидкості та економного використання пам'яті. Швидке дерево Твідді підходить для середніх обсягів даних, забезпечуючи збалансовані результати у продуктивності та точності.

Таким чином, вибір алгоритму повинен ґрунтуватися на специфічних вимогах завдання. Використання адаптованих або комбінованих рішень дозволяє максимально ефективно використовувати переваги кожного алгоритму, створюючи рекомендаційні системи, які відповідають вимогам точності, швидкодії та економічного використання ресурсів.

Висновки до розділу 4

У рамках даного розділу було здійснено детальний аналіз алгоритмів рекомендаційних систем, зокрема градієнтного бустингу, матричної факторизації та швидкого дерева Твідді, з метою оцінки їхньої швидкості, точності та вимог до обчислювальних ресурсів. Було проведено серію експериментів на наборах даних різного обсягу, включаючи 100 000, 1 000 000 та 50 000 000 записів, що дозволило оцінити здатність алгоритмів масштабуватися та їхню ефективність у різних умовах.

Проаналізовано час тренування моделей, результати прогнозів (RMSE, MAE, R-Squared) та швидкість генерації рекомендацій для користувачів. Матрична факторизація продемонструвала найкращі результати у швидкості тренування та використанні пам’яті, особливо на великих наборах даних, що робить її оптимальним вибором для завдань із великим обсягом даних та обмеженими ресурсами. Градієнтний бустинг забезпечив високу точність, особливо на малих наборах даних, однак вимагав більше часу для тренування на великих датасетах. Швидке дерево Твідді показало збалансовані результати, однак його ресурсомісткість є вищою порівняно з іншими алгоритмами, що обмежує його використання в умовах великих обсягів даних.

Досліджено вплив кількості рекомендацій на швидкість генерації та споживання пам’яті. Алгоритми демонструють різну адаптивність до зростання кількості рекомендацій, де матрична факторизація виявилась найбільш стабільною. На основі отриманих даних сформовано практичні рекомендації щодо вибору методів для різних завдань. Градієнтний бустинг рекомендується для сценаріїв із високими вимогами до точності, матрична факторизація – для завдань із великим обсягом даних, а швидке дерево Твідді – для задач, де потрібна швидка адаптація до змін у поведінці користувачів.

Отримані результати дозволяють обґрунтовано вибирати алгоритми залежно від специфіки задачі, що сприятиме підвищенню ефективності систем рекомендацій.

ЗАГАЛЬНИЙ ВИСНОВОК

Під час виконання даної роботи було досліджено, розроблено та проаналізовано алгоритми рекомендаційних систем з метою оцінки їх ефективності в різних умовах та формування рекомендацій щодо їх використання. Основною метою було вивчення математичних основ, реалізація та порівняння алгоритмів для створення систем, які здатні ефективно працювати з великими наборами даних та забезпечувати персоналізовані рекомендації.

У першому розділі була проведена ґрунтовна теоретична база рекомендаційних систем, що охоплює сутність цих систем, їх роль у сучасних технологіях та основні підходи до розробки. Висвітлено популярні моделі, такі як матрична факторизація, градієнтний бустинг та випадкові ліси, та показано їхнє застосування залежно від типу даних і задач.

Другий розділ висвітлює аналіз алгоритмів рекомендаційних систем, обґрунтування вибору тренувального набору даних і технологічного стеку. Було розглянуто математичні основи та переваги кожного алгоритму, обрано датасет на основі платформи MyAnimeList, що містить велику кількість записів та структурованих даних. Для реалізації системи було обрано мову програмування C# з використанням ML.NET та Microsoft Visual Studio, що забезпечило зручність розробки та високу продуктивність.

Третій розділ був присвячений розробці та тестуванню програмного забезпечення. Представлено трирівневу архітектуру системи, включаючи рівні даних, бізнес-логіки та користувацького інтерфейсу. Реалізовано основні алгоритми, такі як градієнтний бустинг, матрична факторизація та швидке дерево Твідді, а також проведено їх тестування з аналізом роботи кожного компонента системи. Тестування форм показало стабільність роботи програмного забезпечення та ефективність алгоритмів.

У четвертому розділі було здійснено порівняння алгоритмів за ключовими характеристиками: час тренування, точність прогнозів,

використання пам’яті та швидкість генерації рекомендацій. Матрична факторизація показала найкращі результати з точки зору швидкодії та точності, градієнтний бустинг – високу точність для великих наборів даних, а швидке дерево Твідді – збалансовані результати. Крім того, сформовано практичні рекомендації щодо вибору алгоритмів залежно від умов використання, а також можливості комбінування підходів для досягнення оптимальних результатів.

Проведена робота демонструє застосування алгоритмів рекомендаційних систем у реальних умовах, враховуючи обмеження ресурсів і вимоги до швидкості. Отримані результати створюють основу для подальшого вдосконалення та інтеграції цих алгоритмів у різноманітні системи, що потребують персоналізованого підходу до обробки даних та взаємодії з користувачами.

БІБЛІОГРАФІЧНИЙ СПИСОК

1. Нескородєва Т. В., Федоров Є. Є., Січко Т. В., Нескородєва А. Р. Експертні та рекомендаційні системи: навч. посіб. для здобувачів вищої освіти спеціальностей 122 «Комп'ютерні науки», 125 «Кібербезпека», 113 «Прикладна математика». Вінниця: ДонНУ імені Василя Стуса, 2023. – 224 с.

2. Aggarwal C. C. Recommender Systems: The Textbook. Cham: Springer, 2016. 498 p.

3. Falk K. Practical Recommender Systems. Shelter Island, New York: Manning Publications Co., 2019. 406 p.

4. Жеребцов О. М.; Нескородєва Т. В. Рекомендаційні системи для інтернет-магазину. Комп'ютерні технології обробки даних, 2022, –191с.

5. Застосування рекомендаційних системи в електронній комерції. URL: https://science.lpnu.ua/uk/sisn/vsi-vypusky/vypusk-15-2024/zastosuvannya-rekomendaciynyh-systemy-v-elektronniy-komerciyi?utm_source=chatgpt.com (дата звернення: 04.12.2024).

6. Коваль А. І. Метод створення рекомендаційних систем. Хмельниц. нац. ун-т. – Хмельницький, 2021. – 105 с.

7. Xiao Y. DFM-GCN: A Multi-Task Learning Recommendation Based on a Deep Graph Neural Network . ULR: https://www.mdpi.com/2227-7390/10/5/721 (дата звернення 04.12.2024).

8. Feng Xiaodong, et al. Social recommendation via deep neural network-based multi-task learning. Expert Systems with Applications, 2022, 206: 117755.

9. Clustering with Machine Learning – A Comprehensive Guide. URL: https://rocketloop.de/en/blog/clustering-machine-learning-comprehensive-guide/ (дата звернення: 04.12.2024).

10. Khan Fazlullah; Jabeen Qamar; Anjum Irum. A novel recommender system on structured data. International Journal of Interdisciplinary Research Centre (IJIRC) ISSN: 2455-2275(E) Volume II, Issue 5 May 2016. p 36.

11. Tarnowska Katarzyna, et al. Recommender System Based on Unstructured

Data. Recommender System for Improving Customer Loyalty, 2020, pp 87-111.

12. Pamir et al. Synthetic theft attacks and long short term memory-based preprocessing for electricity theft detection using gated recurrent unit. Energies, 2022, 15.8: 2778.

13. Huang Zan, et al. A graph-based recommender system for digital library. In: Proceedings of the 2nd ACM/IEEE-CS joint conference on Digital libraries. 2002. p. 65-73.

14. Niu Zhaoyang, et al. Recurrent attention unit: A new gated recurrent unit for long-term memory of important parts in sequential data. Neurocomputing, 2023, 517: 1-9.

15. Model-based Matrix Factorization Algorithm. ULR: https://medium.com/@xiaolancara/recommendation-for-beginners-model-based-matrix-factorization-algorithm-fbcb7279c292 (дата звернення: 04.12.2024).

16. Random Forest. URL: https://medium.com/@denizgunay/random-forest-af5bde5d7e1e (дата звернення: 04.12.2024).

17. Ajesh A.; Nair Jayashree; Jijin P. S. A random forest approach for rating-based recommender system. In: 2016 International conference on advances in computing, communications and informatics (ICACCI). IEEE, 2016. p. 1293-1297.

18. Gradient Boosting in ML. URL: https://www.geeksforgeeks.org/ml-gradient-boosting/ (дата звернення: 04.12.2024).

19. Beygelzimer Alina, et al. Online gradient boosting. Advances in neural information processing systems, 2015, p. 28.

20. Natekin Alexey; Knoll Alois. Gradient boosting machines, a tutorial. Frontiers in neurorobotics, 2013, 7: 21.

21. Natekin, A., & Knoll, A. Gradient boosting machines, a tutorial. Frontiers in neurorobotics. 2013. Vol. 7, No21, pp. 12..

22. Rendle, S., Krichene, W., Zhang, L., & Anderson, J. Neural collaborative filtering vs. matrix factorization revisited. In Proceedings of the 14th ACM Conference on Recommender Systems. 2020. pp. 240-248.

23. Valpione, S., Galvani, E., Tweedy, J., Mundra, P. A., Banyard, A.,

Middlehurst, P., ... & Marais, R. Immune awakening revealed by peripheral T cell dynamics after one cycle of immunotherapy. Nature cancer. 2020. Vol. 1, No2, pp. 210-221.

24. Anime Recommendation Database 2020. URL: https://www.kaggle.com/datasets/hernan4444/anime-recommendation-database-2020?select=animelist.csv (дата звернення: 04.12.2024).

25. Городенко В.П., Марценюк О.П. Програмування на мові Python. Теорія та практика: Навчальний посібник. - К.: Центр навчальної літератури, 2016. - 480 с.

26. Карапецький В. П. Побудова графічного контенту додатків з використанням JavaFX і Swing компонентів і даних, взятих із баз даних / В. П. Карапецький. – Науковий вісник НЛТУ. – 2015. – 418 с.

27. Безменов М.І., Безменова О.М., Калінін Д.В. Основи візуального програмування мовою C#: навч. посіб. для студентів навчально-наукового інституту комп’ютерних наук та інформаційних технологій. – Харків: ФОП Панов А. М., 2023. 648 с.

ДОДАТОК А

Технічно завдання

ЗАТВЕРДЖУЮ
Проректор Українського державного
університету науки і технологій
Анатолій РАДКЕВИЧ

«ДОСЛІДЖЕННЯ АЛГОРИТМІВ РЕКОМЕНДАЦІЙНИХ СИСТЕМ»
Технічне завдання
44165850.1351-01

Завідувач кафедри КІТ
_____Вадим ГОРЯЧКІН
Керівник розробки
_____Олександр ІВАНОВ
Виконавець
_____Максим ПЛЯШКО
Нормоконтролер
_____Світлана ВОЛКОВА

2024

ЗАТВЕРДЖЕНО
44165850.1351-01

«ДОСЛІДЖЕННЯ АЛГОРИТМІВ РЕКОМЕНДАЦІЙНИХ СИСТЕМ»

Технічне завдання

44165850.1351-01

Листів 15

2024

44165850.1351-01
ЗМІСТ

ВСТУП

У сучасному світі рекомендаційні системи стали ключовим інструментом для обробки та аналізу великих обсягів даних у багатьох галузях, включаючи електронну комерцію, медіасервіси, освітні платформи та соціальні мережі. Їхнє завдання полягає у персоналізації взаємодії користувачів із платформою, що дозволяє підвищити задоволеність користувачів і ефективність бізнес-процесів. З метою розробки та впровадження ефективних рекомендаційних систем існує велика кількість алгоритмів, кожен із яких має свої переваги, недоліки та специфіку використання.

У даному дослідженні увага зосереджена на порівнянні та аналізі трьох популярних алгоритмів рекомендаційних систем: градієнтного бустингу, матричної факторизації та швидкого дерева Твідді. Градієнтний бустинг є ансамблевим методом, який дозволяє підвищувати точність прогнозів завдяки послідовному покращенню моделей. Матрична факторизація є одним із найефективніших підходів до колаборативної фільтрації, що працює з великими масивами даних та зберігає продуктивність навіть у складних умовах. Швидке дерево Твідді поєднує в собі гнучкість та швидкість, що робить його привабливим для завдань із високими вимогами до обчислювальних ресурсів.

Дослідження охоплює основні аспекти роботи кожного алгоритму, їх переваги та недоліки, а також порівнює їх за показниками точності, швидкодії та ресурсозатратності. Особлива увага приділяється практичним аспектам впровадження, зокрема вибору технологічного стеку, до складу якого увійшли мова програмування C#, фреймворк машинного навчання ML.NET та система керування базами даних MS SQL Server.

Основною метою цього дослідження є визначення найефективнішого алгоритму для конкретних завдань, пов’язаних із рекомендаціями, а також розробка практичних рекомендацій для вибору та впровадження алгоритмів залежно від особливостей завдання.

44165850.1351-01

1 ПІДСТАВА ДЛЯ РОЗРОБКИ

Основою для розробки є наказ ректора Українського державного університету науки і технологій Радкевич А.В. «Про затвердження тем та призначення керівників дипломних проектів» №1196 ст від 05.12.2024 року.

Тема проекту: «Дослідження алгоритмів рекомендаційних систем».

Керівник дипломного проекту: Іванов О.П.

2 ПРИЗНАЧЕННЯ РОЗРОБКИ

Ця розробка спрямована на проведення дослідження та порівняння ефективності алгоритмів рекомендаційних систем, таких як градієнтний бустинг, матрична факторизація та швидке дерево Твідді. Головною метою є оцінка точності, швидкодії та ресурсозатратності цих алгоритмів для визначення їхньої придатності в системах персоналізованих рекомендацій.

Робота покликана виявити переваги та недоліки алгоритмів у різних аспектах, включаючи продуктивність на великих наборах даних, адаптивність до змін вподобань користувачів та оптимізацію обчислювальних ресурсів. Отримані результати допоможуть обрати найефективніший підхід для побудови рекомендаційних систем у контексті конкретних завдань та обмежень.

Окрім того, у розробці акцент зроблено на використанні технологічного стеку, що включає C#, ML.NET та MS SQL Server, для забезпечення високої продуктивності, масштабованості та інтеграції з сучасними платформами. Результати роботи можуть стати практичним посібником для розробників, які працюють над створенням або вдосконаленням систем рекомендацій у різних галузях.

2.1 Функціональне призначення розробки

Основні аспекти функціонального призначення розробки:

– проведення аналізу алгоритмів – головна функція розробки полягає у дослідженні та оцінці ефективності алгоритмів рекомендаційних систем, таких як градієнтний бустинг, матрична факторизація та швидке дерево Твідді. Це включає вивчення їхніх можливостей щодо точності прогнозів, швидкодії та використання обчислювальних ресурсів;

– порівняння продуктивності – розробка дозволить зіставити показники роботи алгоритмів за різних умов, включаючи обсяги даних, кількість

рекомендацій та сценарії використання. Це дасть змогу виявити переваги і недоліки кожного підходу та визначити їхню придатність для різних задач;

– інтеграція та тестування – розробка спрямована на реалізацію алгоритмів у середовищі C# з використанням ML.NET та MS SQL Server. Включає функції навчання, тестування моделей, генерацію рекомендацій, а також вимірювання швидкодії та споживання ресурсів у різних сценаріях;

– модуль авторизації та логування – система забезпечить авторизацію та реєстрацію користувачів, а також зберігання логів взаємодії для аналізу роботи системи;

– надання рекомендацій – створення можливості адаптивної генерації персоналізованих рекомендацій залежно від уподобань користувачів та їхньої активності.

Результати розробки допоможуть розробникам отримати обґрунтовані рекомендації щодо вибору алгоритмів та технологій для побудови ефективних рекомендаційних систем, а також забезпечать базу для подальших досліджень у цій галузі.

2.2 Експлуатаційне призначення розробки:

Основні аспекти експлуатаційного призначення розробки:

– оцінка та вибір алгоритмів рекомендаційних систем – результати дослідження сприятимуть розробникам у визначенні оптимального алгоритму для створення ефективних рекомендацій, враховуючи особливості завдань і обсяг даних;

– планування проектів – отримані дані щодо продуктивності, точності та ресурсозатратності алгоритмів допоможуть ефективно планувати структуру та функціональність системи рекомендацій;

– інтеграція та оптимізація роботи алгоритмів – розробка дозволяє адаптувати алгоритми до конкретних потреб проекту, оптимізувати їх для роботи з великими наборами даних та забезпечити стабільну роботу системи;

– підтримка масштабованих систем – завдяки інтеграції з C#, ML.NET та MS SQL Server, розробка забезпечує гнучкість і адаптивність у разі зміни вимог або збільшення обсягів даних;

– автоматизація та персоналізація – система сприятиме автоматичному генеруванню рекомендацій, що відповідають індивідуальним уподобанням користувачів, забезпечуючи їхню високу релевантність;

– документування процесів – експлуатаційне призначення передбачає створення детальної документації для подальшого розвитку системи, включаючи рекомендації щодо використання алгоритмів, налаштувань системи та оптимізації ресурсів.

Експлуатаційне призначення розробки полягає в ефективному впровадженні результатів дослідження для створення, вдосконалення та підтримки рекомендаційних систем, які враховують сучасні потреби користувачів і специфіку обробки великих обсягів даних.

3 ВИМОГИ ДО ПРОГРАМИ

3.1 Вимоги до функціональних характеристик

Основні вимоги до функціональних характеристик:

– навчання моделей – система повинна забезпечувати можливість ефективного навчання алгоритмів рекомендацій, таких як градієнтний бустинг, матрична факторизація та швидке дерево Твідді, з використанням тренувального набору даних;

– зберігання моделей – необхідно реалізувати функціонал для збереження натренованих моделей у форматі, що дозволяє їхнє подальше використання для прогнозування;

– тестування точності – система має підтримувати оцінку точності моделей на основі метрик, таких як RMSE, MAE та коефіцієнт детермінації R-Squared, для визначення ефективності алгоритмів;

– генерація рекомендацій – функціонал повинен забезпечувати автоматичне формування рекомендацій на основі прогнозів моделей для конкретних користувачів;

– оптимізація використання ресурсів – система повинна забезпечувати оптимальне використання пам’яті та обчислювальних ресурсів під час навчання та тестування моделей, а також під час генерації рекомендацій;

– вимірювання швидкості – необхідно реалізувати інструменти для вимірювання часу тренування моделей, оцінки точності та генерації рекомендацій у різних сценаріях;

– інтерфейс користувача – система повинна мати інтуїтивно зрозумілий інтерфейс для доступу до функцій навчання моделей, тестування, перегляду результатів та управління системою;

– авторизація та реєстрація користувачів – важливо впровадити модуль, що забезпечує реєстрацію нових користувачів, авторизацію та розмежування прав доступу до функцій системи;

– облік та логування подій – необхідно реалізувати функціонал для фіксації дій користувачів і операцій системи з метою аналізу роботи та забезпечення безпеки;

– робота з великими наборами даних – система повинна підтримувати роботу з об'ємними та складними наборами даних, такими як MyAnimeList, забезпечуючи стабільність і продуктивність;

– масштабованість – система повинна бути здатна адаптуватися до збільшення обсягу даних і кількості користувачів без значного зниження продуктивності;

– інтеграція з технологіями C# і ML.NET – необхідно забезпечити повну сумісність із C# та ML.NET для реалізації машинного навчання, а також з MS SQL Server для роботи з базами даних.

Ці вимоги до функціональних характеристик допоможуть створити ефективну та зручну у використанні систему для дослідження алгоритмів рекомендаційних систем, здатну забезпечувати точність, надійність та високу продуктивність у різних умовах.

Вхідні дані:

- дані про структуру документів – опис структури документів, які будуть зберігатися в базах даних, включаючи назви полів, типи даних та вкладені структури;
- обсяг даних – кількість та об'єм даних, які планується зберігати в базах даних, включаючи кількість записів та обсяг текстової і бінарної інформації;
- методи доступу до даних – опис методів доступу до даних, включаючи створення, зчитування, оновлення та видалення документів;
- сценарії використання – визначення конкретних сценаріїв використання, таких як додавання нових даних, пошук за певними умовами, оновлення та видалення записів;

– серверні налаштування – інформація про конфігурацію серверів, на яких розгорнуті бази даних, включаючи обсяг доступної пам'яті, кількість ядер процесора та інші параметри.

Вихідні дані:

– результати дослідження алгоритмів – звіт, що включає результати аналізу та порівняння алгоритмів рекомендаційних систем, таких як градієнтний бустинг, матрична факторизація та швидке дерево Твідді, з детальним описом їхніх переваг, недоліків і практичних аспектів використання;

– порівняльний аналіз продуктивності – інформація про продуктивність алгоритмів на різних наборах даних, включаючи показники точності (RMSE, MAE, R-Squared), швидкодії та використання пам'яті в різних сценаріях;

– рекомендації щодо вибору алгоритму – рекомендації, які алгоритми найкраще підходять для конкретних завдань, з урахуванням обсягів даних, потреб у точності та обчислювальних ресурсах;

– документація розробленого застосунку – опис функціональних можливостей віконного застосунку для навчання, тестування та зберігання моделей, а також для генерації рекомендацій;

– практичні висновки щодо використання технологій – рекомендації щодо застосування C#, ML.NET, MS SQL Server для реалізації та розгортання системи рекомендацій із зазначенням їхніх переваг у продуктивності та зручності інтеграції.

Вихідні дані:

– результати дослідження алгоритмів – звіт, що містить результати аналізу алгоритмів рекомендаційних систем (градієнтний бустинг, матрична факторизація, швидке дерево Твідді) з урахуванням їхніх характеристик, продуктивності та ефективності в різних сценаріях використання;

– рекомендації щодо вибору алгоритмів – інформація про те, який із досліджуваних алгоритмів найкраще підходить для певних типів завдань, з

детальним обґрунтуванням вибору та описом сильних і слабких сторін кожного підходу;

– план реалізації системи – документ, що включає етапи розробки та впровадження рекомендаційної системи з використанням обраного технологічного стеку (C#, ML.NET, MS SQL Server), а також алгоритмів, оптимальних для зазначених умов роботи;

– документація розробленого програмного забезпечення – опис функціональних можливостей віконного застосунку, зокрема механізмів навчання, тестування, зберігання моделей і генерації персоналізованих рекомендацій;

– висновки щодо практичного застосування – рекомендації для розробників і організацій стосовно впровадження розробленої системи в реальних умовах, включаючи сценарії її використання та потенційні перспективи розвитку.

3.2 Вимоги до надійності

Вимоги до надійності системи дослідження алгоритмів рекомендаційних систем визначаються наступними аспектами:

– забезпечення коректного повідомлення про результати навчання, тестування та використання моделей, зокрема відображення помилок у разі невдачі виконання операцій;

– створення резервної копії файлів моделі та налаштувань системи на зовнішньому носії для запобігання втраті даних у разі збоїв;

– наявність регулярного збереження тренувального набору даних та логів подій у надійному форматі, що дозволяє їх відновлення при необхідності.

3.3 Вимоги експлуатації

Система може експлуатуватись користувачами, які мають базові навички роботи з десктопними застосунками та ознайомлені з інструкціями, наданими в керівництві користувача. Для роботи з програмою необхідні мінімальні знання

у сфері алгоритмів машинного навчання, зокрема базове розуміння процесів навчання та оцінки моделей. Інтерфейс системи розроблено з урахуванням простоти використання, що робить її придатною для експлуатації як досвідченими розробниками, так і новачками у галузі рекомендаційних систем.

3.4 Вимоги до складу та параметрів технічних засобів

Система для дослідження алгоритмів рекомендаційних систем повинна функціонувати на пристроях із наступними технічними характеристиками:

– операційна система – комп’ютер має працювати під управлінням операційної системи, сумісної з технологічним стеком розробки, таким як Windows 10 або новіші версії, які забезпечують підтримку .NET Framework та ML.NET.

– діагональ монітора – для зручного відображення графічного інтерфейсу продукту монітор має мати діагональ не менше 21 дюйма, що забезпечує достатній огляд форми для тренування та тестування моделей.

– роздільна здатність монітора – для чіткого відображення елементів інтерфейсу необхідна мінімальна роздільна здатність Full HD (1920x1080). Бажано використовувати монітори з вищими показниками для покращення візуального комфорту.

– оперативна пам'ять (RAM) – мінімальна рекомендована кількість оперативної пам'яті становить 8 ГБ для ефективної роботи системи, навчання моделей і виконання обчислювально-інтенсивних завдань.

– вбудована пам'ять (жорсткий диск або SSD) – мінімальний обсяг внутрішньої пам’яті повинен становити 512 ГБ для зберігання програмного продукту, моделей машинного навчання та вхідних даних. Рекомендується використовувати SSD-накопичувачі для підвищення швидкодії.

– частота процесора – для швидкої обробки даних необхідний процесор із багатоядерною архітектурою та тактовою частотою не менше 3.0 ГГц. Рекомендується використовувати Intel Core i5 або еквіваленти AMD.

– графічний процесор (GPU) – якщо передбачено використання алгоритмів, які можуть бути прискорені за допомогою GPU, бажано мати дискретну графічну карту з підтримкою технологій CUDA або DirectML.

– порти та комунікації – пристрій має бути оснащений USB-портами для підключення зовнішніх носіїв, Ethernet-портом або модулем Wi-Fi для доступу до мережі та онлайн-ресурсів.

3.5 Вимоги до інформаційної та програмної сумісності

Програмні продукти розробляється для всіх видів операційних систем сімейства “Windows” починаючи від версії 7 та наступні версії.

4 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

До складу документації мають входити:

– текст програми;

– керівництво користувача для користування мобільним додатком.

Вся документація програмних додатків повинна задовольняти вимоги до програмної документації.

44165850.1351-01

5 СТАДІЇ ТА ЕТАПИ РОЗРОБКИ

Таблиця 5.1 – Стадії та етапи розробки

Стадія	Зміст	Строки виконання
Технічне завдання	Постановка задачі, аналіз літератури та визначення алгоритмів для дослідження рекомендаційних систем. Вибір тренувального набору даних та обґрунтування технологічного стеку. Формулювання вимог до технічних засобів та функціональних характеристик системи. Узгодження та затвердження технічного завдання.	04.09.23 –15.09.23
Робочий проект	Реалізація програмного забезпечення, що включає модулі для навчання, збереження та тестування моделей.	18.09.23 – 22.09.23
	Тестування програмного забезпечення, оцінка його точності, швидкодії та ресурсозатратності для різних сценаріїв.	25.09.23 – 27.09.23
	Розробка, узгодження та затвердження програмної документації.	28.09.23 – 29.09.23

6 ПОРЯДОК ПРИЙНЯТТЯ

Контроль за виконанням роботи здійснює керівник розробки доц. Іванов О. П.

7 ТЕХНІЧНО-ЕКОНОМІЧНІ ПОКАЗНИКИ

Показники та їх розрахунок у рамках даної роботи не наведені, оскільки розробка програмного забезпечення має навчальний характер і не передбачає комерційного використання.

ДОДАТОК Б

Технічне завдання

ЗАТВЕРДЖУЮ
Проректор Українського державного
університету науки і технологій
Анатолій РАДКЕВИЧ

«ДОСЛІДЖЕННЯ АЛГОРИТМІВ РЕКОМЕНДАЦІЙНИХ СИСТЕМ»
Текст програми
44165850.01223–01 12–01

Завідувач кафедри КІТ
_____Вадим ГОРЯЧКІН
Керівник розробки
_____Олександр ІВАНОВ
Виконавець
_____Максим ПЛЯШКО
Нормоконтролер
_____Світлана ВОЛКОВА

2024

ЗАТВЕРДЖЕНО

44165850.01223–01 12–01

ДОСЛІДЖЕННЯ АЛГОРИТМІВ РЕКОМЕНДАЦІЙНИХ СИСТЕМ

Текст програми
44165850.01223–01 12–01
Листів 17

2024

АНОТАЦІЯ

Документ «Дослідження алгоритмів рекомендаційних систем» входить до складу програмної документації для розробки системи дослідження та оцінки ефективності алгоритмів рекомендацій.

У даному документі описано особливості реалізації системи, яка включає навчання, тестування та порівняння алгоритмів градієнтного бустингу, матричної факторизації та швидкого дерева Твідді. Розробка виконана мовою C# із використанням фреймворку ML.NET у середовищі розробки Visual Studio 2022 з інтеграцією бази даних MS SQL Server.

ЗМІСТ

1 ТЕКСТ ПРОГРАМИ

1.1 Текст тренування моделей швидкого дерева Твідді FastTreeTweedieForm.cs

```
using Microsoft.ML.Data;
using Microsoft.ML;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using
MLPersonalizeWebApp.AppCode;
using
MLPersonalizeWebApp.Providers;
using
MLPersonalizeWebApp.Forms.System
s;

namespace
MLPersonalizeWebApp.Forms.Dictina
ry {
  public partial class
FastTreeTweedieForm : Form {
    private MLContext mlContext;
    private ITransformer model;
    private IDataView dataView;

    private string _Path = "";

    private int _selectedRowIndex = 0;
    private CategoryApp _CategoryApp
= new CategoryApp();
    private List<Category>
_CategoryList = new
List<Category>();
    private ValidationMy _Validation =
new ValidationMy();
    private ModelsProvider
_ModelsProvider = new
ModelsProvider();
    private List<Models> _ModelsList =
new List<Models>();
    private LogsProvider _LogsProvider
= new LogsProvider();
    private bool _IsModelTrain = false;

    public FastTreeTweedieForm() {
      InitializeComponent();
      LoadAllDate();
    }

    private void OpenBtn_Click(object
sender, EventArgs e) {
      // Створення діалогового вікна
для відкриття файлу
      OpenFileDialog openFileDialog =
new OpenFileDialog();

      openFileDialog.Filter = "CSV files
(*.csv)|*.csv|All files (*.*)|*.*";
      openFileDialog.FilterIndex = 1;
      openFileDialog.RestoreDirectory =
true;

      if (openFileDialog.ShowDialog()
== DialogResult.OK) {
```

```
        _Path =
openFileDialog.FileName;
        FileNameTBox.Text = _Path;
        RaportTBox.Text =
"Завантаження даних...\r\n";
        Application.DoEvents();

        // Створення контексту ML
        mlContext = new
MLContext(seed: 0);

        // Завантаження даних
        dataView =
mlContext.Data.LoadFromTextFile<A
nimeRating>(_Path,
            hasHeader: true,
            separatorChar: ',');

        // Підрахунок кількості
унікальних користувачів та аніме
        int userCount =
mlContext.Data.CreateEnumerable<An
imeRating>(dataView,
reuseRowObject: false)
            .Select(x =>
x.UserId).Distinct().Count();
        int animeCount =
mlContext.Data.CreateEnumerable<An
imeRating>(dataView,
reuseRowObject: false)
            .Select(x =>
x.AnimeId).Distinct().Count();
        long ratingCount =
mlContext.Data.CreateEnumerable<An
imeRating>(dataView,
reuseRowObject: false).LongCount();

        RaportTBox.Text +=
($"Кількість унікальних
користувачів: {userCount}\r\n");
        RaportTBox.Text +=
($"Кількість унікальних аніме:
{animeCount}\r\n");
        RaportTBox.Text +=
($"Кількість записів:
{ratingCount}\r\n");
        Application.DoEvents();

        // Розділяємо дані на
тренувальний та тестовий набори
        var trainTestData =
mlContext.Data.TrainTestSplit(dataVie
w, testFraction: 0.2);
        var trainData =
trainTestData.TrainSet;
        var testData =
trainTestData.TestSet;

        // Створюємо конвеєр обробки
даних та модель FastTreeTweedie
        var pipeline =
mlContext.Transforms.Conversion.Ma
pValueToKey(outputColumnName:
"UserIdEncoded", inputColumnName:
"UserId")

.Append(mlContext.Transforms.Conve
rsion.MapValueToKey(outputColumn
Name: "AnimeIdEncoded",
inputColumnName: "AnimeId"))

.Append(mlContext.Transforms.Conve
rsion.MapKeyToValue("UserIdFloat",
"UserIdEncoded"))

.Append(mlContext.Transforms.Conve
rsion.MapKeyToValue("AnimeIdFloat
", "AnimeIdEncoded"))

.Append(mlContext.Transforms.Conve
rsion.ConvertType("UserIdFloat",
outputKind: DataKind.Single))

.Append(mlContext.Transforms.Conve
rsion.ConvertType("AnimeIdFloat",
outputKind: DataKind.Single))
```

```
.Append(mlContext.Transforms.Concat
enate("Features", "UserIdFloat",
"AnimeIdFloat"))

.Append(mlContext.Regression.Trainer
s.FastTreeTweedie(labelColumnName:
"Rating", featureColumnName:
"Features"));

        RaportTBox.Text +=
"Тренування моделі ...\r\n";
        Application.DoEvents();

        // Вимірюємо час тренування
моделі
        var trainingTimer =
System.Diagnostics.Stopwatch.StartNe
w();

        // Навчання моделі на
тренувальному наборі
        model = pipeline.Fit(trainData);

        trainingTimer.Stop();
        RaportTBox.Text += ($"Час
тренування моделі:
{trainingTimer.Elapsed.ToString("mm\
\:ss\\.fff")} секунд\r\n");

        // Вимірюємо час оцінки моделі
        var evaluationTimer =
System.Diagnostics.Stopwatch.StartNe
w();

        // Оцінка моделі на тестовому
наборі
        var predictions =
model.Transform(testData);
        var metrics =
mlContext.Regression.Evaluate(predict
ions, labelColumnName: "Rating");
        evaluationTimer.Stop();

        // Виведення результатів оцінки
моделі
        RaportTBox.Text += "Оцінка
моделі FastTreeTweedie на тестовому
наборі:\r\n";
        RaportTBox.Text += ($"RMSE:
{metrics.RootMeanSquaredError:F2}\r
\n");
        RaportTBox.Text += ($"MAE:
{metrics.MeanAbsoluteError:F2}\r\n");
        RaportTBox.Text += ($"R-
Squared: {metrics.RSquared:F2}\r\n");
        RaportTBox.Text += ($"Час
оцінки моделі:
{evaluationTimer.Elapsed.ToString("m
m\\:ss\\.fff")} секунд\r\n");

        _IsModelTrain = true;
      }
    }


    private void AddBtn_Click(object
sender, EventArgs e) {
      if (IsDataEnteringCorrect()) {
        //Зберігання моделі
        string pathName = @"\teach\" +
GenerateFileName() + ".zip";
        string localProj =

System.IO.Path.GetDirectoryName(Sy
stem.Reflection.Assembly.GetExecutin
gAssembly().Location);

_ModelsProvider.InsertModels(Models
NamesTBox.Text,

Convert.ToInt32(CategoryCBox.Select
edValue), CategoryCBox.Text,
pathName);
```

```
        mlContext.Model.Save(model,
dataView.Schema, localProj +
pathName);
        ClearAllData();

_LogsProvider.InsertLogs(LoginForm.
CurrentUser.UsersId,
          "Було навчено модель " +
          ModelsNamesTBox.Text,
DateTime.Now);
        MessageBox.Show("Дані
успішно збережено!");
      }
    }

    private void ClearBtn_Click(object
sender, EventArgs e) {
      ClearAllData();
    }

    private void ExitBtn_Click(object
sender, EventArgs e) {
      this.Close();
    }

    private void
ModelsGridView_CellClick(object
sender, DataGridViewCellEventArgs
e) {
      if (e.ColumnIndex == 5 &&
ModelsGridView[0,
e.RowIndex].Value.ToString() !=
_ModelsList[0].Message) {
        if (MessageBox.Show("Ви дійсно
хочете видалити цей елемент?",
"Видалити",
MessageBoxButtons.YesNo) ==
DialogResult.Yes) {

_ModelsProvider.DeleteModelsByMod
elsId(Convert.ToInt32(ModelsGridVie
w[0, e.RowIndex].Value.ToString()));
          DataLoad();
        }
      }
    }

    public string GenerateFileName() {
      DateTime now = DateTime.Now;
      string fileName =
string.Format("{0}_{1}_{2}_{3}_{4}
_{5}",
        now.Year, now.Month, now.Day,
now.Hour, now.Minute, now.Second);

      return fileName;
    }

    private void ClearAllData() {
      _IsModelTrain = false;
      ModelsNamesTBox.Text =
String.Empty;
      RaportTBox.Text = String.Empty;
      DataLoad();
    }
    private bool IsDataEnteringCorrect()
{
      bool isCorrect = true;
      if (!_IsModelTrain) {
        MessageBox.Show("Неможливо
зберегти дані. \r\nЩе не навчено
модель!", "Увага!");
        isCorrect = false;
      }
      if
(Convert.ToInt32(CategoryCBox.Selec
tedValue) > 0) {
        CategoryValidationLbl.Text =
NamesMy.ProgramButtons.RequiredV
alidation;
      } else {
        CategoryValidationLbl.Text =
NamesMy.ProgramButtons.ErrorValid
ation;
        isCorrect = false;
      }
```

```
    if
(_Validation.IsDataEntering(ModelsNa
mesTBox.Text)) {
      ModelsNamesValidationLbl.Text
=
NamesMy.ProgramButtons.RequiredV
alidation;
    } else {
      ModelsNamesValidationLbl.Text
=
NamesMy.ProgramButtons.ErrorValid
ation;
      isCorrect = false;
    }
    return isCorrect;
  }

  private void LoadAllDate() {
    _CategoryList =
_CategoryApp.GetCategoryList();
    CategoryCBox.DataSource =
_CategoryList;
    CategoryCBox.ValueMember =
"CategoryId";
    CategoryCBox.DisplayMember =
"CategoryName";
    CategoryCBox.SelectedIndex = 2;
    CategoryCBox.Enabled = false;
    DataLoad();
  }

  private void DataLoad() {
    int firstRowIndex = 0;
    if
(ModelsGridView.FirstDisplayedScroll
ingRowIndex > 0) {
      firstRowIndex =
ModelsGridView.FirstDisplayedScrolli
ngRowIndex;
    }
    try {
      _ModelsList =
_ModelsProvider.GetAllModels();
LoadDataInModelsGridView(_Models
List);
      if (_selectedRowIndex ==
ModelsGridView.Rows.Count) {
        _selectedRowIndex =
ModelsGridView.Rows.Count - 1;
      }
      if (_selectedRowIndex >= 0) {

ModelsGridView.FirstDisplayedScrolli
ngRowIndex = firstRowIndex;

ModelsGridView.Rows[_selectedRowI
ndex].Selected = true;
      }
    } catch (Exception ex) {

MessageBox.Show(ex.ToString());
    }
  }

  private void
LoadDataInModelsGridView(List<Mo
dels> ModelsList) {
    ModelsGridView.DataSource =
null;
    ModelsGridView.Columns.Clear();

ModelsGridView.AutoGenerateColum
ns = false;

ModelsGridView.RowHeadersVisible
= false;

    ModelsGridView.DataSource =
ModelsList;

    if (ModelsList.Count > 0) {
      if (ModelsList[0].Message ==
NamesMy.NoDataNames.NoDataInMo
dels) {
```

```
        DataGridViewColumn
messageColumn = new
DataGridViewTextBoxColumn();

messageColumn.DataPropertyName =
"Message";
        messageColumn.Width =
ModelsGridView.Width -
NamesMy.SizeOptins.MinusSizePanel;

ModelsGridView.Columns.Add(messa
geColumn);
      } else {
        DataGridViewColumn
DetailIdColumn = new
DataGridViewTextBoxColumn();

DetailIdColumn.DataPropertyName =
"ModelsId";

ModelsGridView.Columns.Add(DetailI
dColumn);

ModelsGridView.Columns[0].Visible
= false;

        DataGridViewColumn
numberColumn = new
DataGridViewTextBoxColumn();
        numberColumn.HeaderText =
"№ ";

numberColumn.DataPropertyName =
"Number";

numberColumn.DefaultCellStyle.Align
ment =
DataGridViewContentAlignment.Midd
leRight;
        numberColumn.Width =
NamesMy.SizeOptins.NumberSize;

ModelsGridView.Columns.Add(numbe
rColumn);

        DataGridViewColumn
ModelsNamesColumn = new
DataGridViewTextBoxColumn();

ModelsNamesColumn.HeaderText =
"Назва моделі";

ModelsNamesColumn.DataPropertyNa
me = "ModelsNames";
        ModelsNamesColumn.Width =
250;

ModelsGridView.Columns.Add(Model
sNamesColumn);

        DataGridViewColumn
CategoryNameColumn = new
DataGridViewTextBoxColumn();

CategoryNameColumn.HeaderText =
"Категорія";

CategoryNameColumn.DataPropertyN
ame = "CategoryName";
        CategoryNameColumn.Width =
150;

ModelsGridView.Columns.Add(Categ
oryNameColumn);

        DataGridViewColumn
ModelsFileModelColumn = new
DataGridViewTextBoxColumn();

ModelsFileModelColumn.HeaderText
= "Файл";

ModelsFileModelColumn.DataPropert
yName = "ModelsFileModel";
```

```
        ModelsFileModelColumn.Width
= 200;

ModelsGridView.Columns.Add(Model
sFileModelColumn);

        DataGridViewButtonColumn
IsResidesBtn = new
DataGridViewButtonColumn();
        IsResidesBtn.HeaderText =
"Видалити";
        IsResidesBtn.Text =
"Видалити";

IsResidesBtn.UseColumnTextForButto
nValue = true;
        IsResidesBtn.ToolTipText =
"Видалити";
        IsResidesBtn.Width =
NamesMy.SizeOptins.DeleteBtnSize;

ModelsGridView.Columns.Add(IsResi
desBtn);

      }
      for (int i = 0; i <
ModelsGridView.Columns.Count; i++)
{

ModelsGridView.Columns[i].HeaderC
ell.Style.BackColor = Color.LightGray;
      }
    }
  }


 }
}
```

1.2 Текст тренування моделей градієнтного бустингу LightGBMRegressorForm.cs

```
using Microsoft.ML;
using Microsoft.ML.Data;
using
MLPersonalizeWebApp.AppCode;
using
MLPersonalizeWebApp.Providers;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using
MLPersonalizeWebApp.Forms.System
s;

namespace
MLPersonalizeWebApp.Forms.Dictina
ry {
  public partial class
LightGBMRegressorForm : Form {
    private MLContext mlContext;
    private ITransformer model;
    private IDataView dataView;
```

```
    private string _Path = "";

    private int _selectedRowIndex = 0;
    private CategoryApp _CategoryApp
= new CategoryApp();
    private List<Category>
_CategoryList = new
List<Category>();
    private ValidationMy _Validation =
new ValidationMy();
    private ModelsProvider
_ModelsProvider = new
ModelsProvider();
    private List<Models> _ModelsList =
new List<Models>();
    private LogsProvider _LogsProvider
= new LogsProvider();
    private bool _IsModelTrain = false;



    public LightGBMRegressorForm() {
      InitializeComponent();
      LoadAllDate();
    }

    private void OpenBtn_Click(object
sender, EventArgs e) {
      // Створення діалогового вікна
для відкриття файлу
      OpenFileDialog openFileDialog =
new OpenFileDialog();

      openFileDialog.Filter = "CSV files
(*.csv)|*.csv|All files (*.*)|*.*";
      openFileDialog.FilterIndex = 1;
      openFileDialog.RestoreDirectory =
true;

      if (openFileDialog.ShowDialog()
== DialogResult.OK) {
        _Path =
openFileDialog.FileName;
        FileNameTBox.Text = _Path;
        RaportTBox.Text =
"Завантаження даних...\r\n";
        Application.DoEvents();

        // Створення контексту ML
        mlContext = new
MLContext(seed: 0);

        // Завантаження даних
        dataView =
mlContext.Data.LoadFromTextFile<A
nimeRating>(_Path,
            hasHeader: true,
            separatorChar: ',');

        // Підрахунок кількості
унікальних користувачів та аніме
        int userCount =
mlContext.Data.CreateEnumerable<An
imeRating>(dataView,
reuseRowObject: false)
            .Select(x =>
x.UserId).Distinct().Count();
        int animeCount =
mlContext.Data.CreateEnumerable<An
imeRating>(dataView,
reuseRowObject: false)
            .Select(x =>
x.AnimeId).Distinct().Count();
        long ratingCount =
mlContext.Data.CreateEnumerable<An
imeRating>(dataView,
reuseRowObject: false).LongCount();

        RaportTBox.Text +=
($"Кількість унікальних
користувачів: {userCount}\r\n");
        RaportTBox.Text +=
($"Кількість унікальних аніме:
{animeCount}\r\n");
        RaportTBox.Text +=
($"Кількість записів:
{ratingCount}\r\n");
```

```
        Application.DoEvents();

        // Розділяємо дані на тренувальний та тестовий набори
        var trainTestData = mlContext.Data.TrainTestSplit(dataView, testFraction: 0.2);
        var trainData = trainTestData.TrainSet;
        var testData = trainTestData.TestSet;

        // Створюємо конвеєр обробки даних та модель LightGBM
        var pipeline = mlContext.Transforms.Conversion.MapValueToKey(outputColumnName: "UserIdEncoded", inputColumnName: "UserId")

.Append(mlContext.Transforms.Conversion.MapValueToKey(outputColumnName: "AnimeIdEncoded", inputColumnName: "AnimeId"))

.Append(mlContext.Transforms.Conversion.MapKeyToValue("UserIdFloat", "UserIdEncoded"))

.Append(mlContext.Transforms.Conversion.MapKeyToValue("AnimeIdFloat", "AnimeIdEncoded"))

.Append(mlContext.Transforms.Conversion.ConvertType("UserIdFloat", outputKind: DataKind.Single))

.Append(mlContext.Transforms.Conversion.ConvertType("AnimeIdFloat", outputKind: DataKind.Single))

.Append(mlContext.Transforms.Concatenate("Features", "UserIdFloat", "AnimeIdFloat"))
            // Додаємо LightGBM Regressor

.Append(mlContext.Regression.Trainers.LightGbm(labelColumnName: "Rating", featureColumnName: "Features"));

        RaportTBox.Text += "Тренування моделі LightGBM ...\r\n";
        Application.DoEvents();

        // Вимірюємо час тренування моделі
        var trainingTimer = System.Diagnostics.Stopwatch.StartNew();

        // Навчання моделі на тренувальному наборі
        model = pipeline.Fit(trainData);

        trainingTimer.Stop();
        RaportTBox.Text += ($"Час тренування моделі: {trainingTimer.Elapsed.ToString("mm\:ss\\.fff")} секунд\r\n");
        Application.DoEvents();

        // Вимірюємо час оцінки моделі
        var evaluationTimer = System.Diagnostics.Stopwatch.StartNew();

        // Оцінка моделі на тестовому наборі
        var predictions = model.Transform(testData);
```

```
        var metrics =
mlContext.Regression.Evaluate(predict
ions, labelColumnName: "Rating");

        evaluationTimer.Stop();
        RaportTBox.Text += ("Оцінка
моделі LightGBM на тестовому
наборі:\r\n");
        RaportTBox.Text += ($"RMSE:
{metrics.RootMeanSquaredError:F2}\r
\n");
        RaportTBox.Text += ($"MAE:
{metrics.MeanAbsoluteError:F2}\r\n");
        RaportTBox.Text += ($"R-
Squared: {metrics.RSquared:F2}\r\n");
        RaportTBox.Text += ($"Час
оцінки моделі:
{evaluationTimer.Elapsed.ToString("m
m\\:ss\\.fff")} секунд\r\n");

        _IsModelTrain = true;
      }
    }


    private void AddBtn_Click(object
sender, EventArgs e) {
      if (IsDataEnteringCorrect()) {
        //Зберігання моделі
        string pathName = @"\teach\" +
GenerateFileName() + ".zip";
        string localProj =

System.IO.Path.GetDirectoryName(Sy
stem.Reflection.Assembly.GetExecutin
gAssembly().Location);

_ModelsProvider.InsertModels(Models
NamesTBox.Text,

Convert.ToInt32(CategoryCBox.Select
edValue), CategoryCBox.Text,
pathName);
        mlContext.Model.Save(model,
dataView.Schema, localProj +
pathName);
        ClearAllData();

_LogsProvider.InsertLogs(LoginForm.
CurrentUser.UsersId,
          "Було навчено модель " +
          ModelsNamesTBox.Text,
DateTime.Now);
        MessageBox.Show("Дані
успішно збережено!");
      }
    }

    private void ClearBtn_Click(object
sender, EventArgs e) {
      ClearAllData();
    }

    private void ExitBtn_Click(object
sender, EventArgs e) {
      this.Close();
    }

    private void
ModelsGridView_CellClick(object
sender, DataGridViewCellEventArgs
e) {
      if (e.ColumnIndex == 5 &&
ModelsGridView[0,
e.RowIndex].Value.ToString() !=
_ModelsList[0].Message) {
        if (MessageBox.Show("Ви дійсно
хочете видалити цей елемент?",
"Видалити",
MessageBoxButtons.YesNo) ==
DialogResult.Yes) {

_ModelsProvider.DeleteModelsByMod
elsId(Convert.ToInt32(ModelsGridVie
w[0, e.RowIndex].Value.ToString()));
          DataLoad();
```

```
      }
     }
    }

    public string GenerateFileName() {
     DateTime now = DateTime.Now;
     string fileName =
string.Format("{0}_{1}_{2}_{3}_{4}
_{5}",
       now.Year, now.Month, now.Day,
now.Hour, now.Minute, now.Second);

     return fileName;
    }

    private void ClearAllData() {
     _IsModelTrain = false;
     ModelsNamesTBox.Text =
String.Empty;
     RaportTBox.Text = String.Empty;
     DataLoad();
    }
    private bool IsDataEnteringCorrect()
{
     bool isCorrect = true;
     if (!_IsModelTrain) {
      MessageBox.Show("Неможливо
зберегти дані. \r\nЩе не навчено
модель!", "Увага!");
      isCorrect = false;
     }
     if
(Convert.ToInt32(CategoryCBox.Selec
tedValue) > 0) {
      CategoryValidationLbl.Text =
NamesMy.ProgramButtons.RequiredV
alidation;
     } else {
      CategoryValidationLbl.Text =
NamesMy.ProgramButtons.ErrorValid
ation;
      isCorrect = false;
     }
     if
(_Validation.IsDataEntering(ModelsNa
mesTBox.Text)) {
      ModelsNamesValidationLbl.Text
=
NamesMy.ProgramButtons.RequiredV
alidation;
     } else {
      ModelsNamesValidationLbl.Text
=
NamesMy.ProgramButtons.ErrorValid
ation;
      isCorrect = false;
     }
     return isCorrect;
    }

    private void LoadAllDate() {
     _CategoryList =
_CategoryApp.GetCategoryList();
     CategoryCBox.DataSource =
_CategoryList;
     CategoryCBox.ValueMember =
"CategoryId";
     CategoryCBox.DisplayMember =
"CategoryName";
     CategoryCBox.SelectedIndex = 0;
     CategoryCBox.Enabled = false;
     DataLoad();
    }

    private void DataLoad() {
     int firstRowIndex = 0;
     if
(ModelsGridView.FirstDisplayedScroll
ingRowIndex > 0) {
      firstRowIndex =
ModelsGridView.FirstDisplayedScrolli
ngRowIndex;
     }
     try {
      _ModelsList =
_ModelsProvider.GetAllModels();
```

```
LoadDataInModelsGridView(_Models
List);
      if (_selectedRowIndex ==
ModelsGridView.Rows.Count) {
        _selectedRowIndex =
ModelsGridView.Rows.Count - 1;
      }
      if (_selectedRowIndex >= 0) {

ModelsGridView.FirstDisplayedScrolli
ngRowIndex = firstRowIndex;

ModelsGridView.Rows[_selectedRowI
ndex].Selected = true;
      }
    } catch (Exception ex) {

MessageBox.Show(ex.ToString());
    }
  }

  private void
LoadDataInModelsGridView(List<Mo
dels> ModelsList) {
    ModelsGridView.DataSource =
null;
    ModelsGridView.Columns.Clear();

ModelsGridView.AutoGenerateColum
ns = false;

ModelsGridView.RowHeadersVisible
= false;

    ModelsGridView.DataSource =
ModelsList;

    if (ModelsList.Count > 0) {
      if (ModelsList[0].Message ==
NamesMy.NoDataNames.NoDataInMo
dels) {
        DataGridViewColumn
messageColumn = new
DataGridViewTextBoxColumn();

messageColumn.DataPropertyName =
"Message";
        messageColumn.Width =
ModelsGridView.Width -
NamesMy.SizeOptins.MinusSizePanel;

ModelsGridView.Columns.Add(messa
geColumn);
      } else {
        DataGridViewColumn
DetailIdColumn = new
DataGridViewTextBoxColumn();

DetailIdColumn.DataPropertyName =
"ModelsId";

ModelsGridView.Columns.Add(DetailI
dColumn);

ModelsGridView.Columns[0].Visible
= false;

        DataGridViewColumn
numberColumn = new
DataGridViewTextBoxColumn();
        numberColumn.HeaderText =
"№ ";

numberColumn.DataPropertyName =
"Number";

numberColumn.DefaultCellStyle.Align
ment =
DataGridViewContentAlignment.Midd
leRight;
        numberColumn.Width =
NamesMy.SizeOptins.NumberSize;
```

```
ModelsGridView.Columns.Add(numberColumn);

        DataGridViewColumn ModelsNamesColumn = new DataGridViewTextBoxColumn();

ModelsNamesColumn.HeaderText = "Назва моделі";

ModelsNamesColumn.DataPropertyName = "ModelsNames";
        ModelsNamesColumn.Width = 250;

ModelsGridView.Columns.Add(ModelsNamesColumn);

        DataGridViewColumn CategoryNameColumn = new DataGridViewTextBoxColumn();

CategoryNameColumn.HeaderText = "Категорія";

CategoryNameColumn.DataPropertyName = "CategoryName";
        CategoryNameColumn.Width = 150;

ModelsGridView.Columns.Add(CategoryNameColumn);

        DataGridViewColumn ModelsFileModelColumn = new DataGridViewTextBoxColumn();

ModelsFileModelColumn.HeaderText = "Файл";

ModelsFileModelColumn.DataPropertyName = "ModelsFileModel";
        ModelsFileModelColumn.Width = 200;

ModelsGridView.Columns.Add(ModelsFileModelColumn);

        DataGridViewButtonColumn IsResidesBtn = new DataGridViewButtonColumn();
        IsResidesBtn.HeaderText = "Видалити";
        IsResidesBtn.Text = "Видалити";

IsResidesBtn.UseColumnTextForButtonValue = true;
        IsResidesBtn.ToolTipText = "Видалити";
        IsResidesBtn.Width = NamesMy.SizeOptins.DeleteBtnSize;

ModelsGridView.Columns.Add(IsResidesBtn);

       }
       for (int i = 0; i < ModelsGridView.Columns.Count; i++)
{

ModelsGridView.Columns[i].HeaderCell.Style.BackColor = Color.LightGray;
       }
     }
   }


  }
}
```

1.3 Текст тренування моделей матричної факторизації MatrixFactorizationForms.cs

```
using Microsoft.ML;
using MLPersonalizeWebApp.AppCode;
using MLPersonalizeWebApp.Providers;
using System;
using System.Collections.Generic;
using Microsoft.ML.Trainers;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using Microsoft.ML.Data;
using System.Diagnostics;
using MLPersonalizeWebApp.Forms.Systems;

namespace MLPersonalizeWebApp.Forms.Dictinary {
  public partial class MatrixFactorizationForms : Form {
    private MLContext mlContext;
    private ITransformer model;
    private IDataView dataView;

    private string _Path = "";

    private int _selectedRowIndex = 0;
    private CategoryApp _CategoryApp = new CategoryApp();
    private List<Category> _CategoryList = new List<Category>();
    private ValidationMy _Validation = new ValidationMy();
    private ModelsProvider _ModelsProvider = new ModelsProvider();
    private List<Models> _ModelsList = new List<Models>();
    private LogsProvider _LogsProvider = new LogsProvider();
    private bool _IsModelTrain = false;

    public MatrixFactorizationForms() {
      InitializeComponent();
      LoadAllDate();
    }

    private void OpenBtn_Click(object sender, EventArgs e) {
      // Створення діалогового вікна для відкриття файлу
      OpenFileDialog openFileDialog = new OpenFileDialog();

      // Налаштування властивостей діалогового вікна
      openFileDialog.Filter = "CSV files (*.csv)|*.csv|All files (*.*)|*.*";
      openFileDialog.FilterIndex = 1;
      openFileDialog.RestoreDirectory = true;

      // Відображення діалогового вікна та обробка результату
```

```
    if (openFileDialog.ShowDialog()
== DialogResult.OK) {
        _Path =
openFileDialog.FileName;
        FileNameTBox.Text = _Path;
        RaportTBox.Text =
"Завантаження даних...\r\n";
        Application.DoEvents();

        // Створення контексту ML
        mlContext = new
MLContext(seed: 0);

        // Завантаження даних
        dataView =
mlContext.Data.LoadFromTextFile<A
nimeRating>(_Path,
            hasHeader: true,
            separatorChar: ',');

        // Виведення інформації про
дані
        int userCount =
mlContext.Data.CreateEnumerable<An
imeRating>(dataView,
reuseRowObject: false).Select(x =>
x.UserId).Distinct().Count();
        int animeCount =
mlContext.Data.CreateEnumerable<An
imeRating>(dataView,
reuseRowObject: false).Select(x =>
x.AnimeId).Distinct().Count();
        long ratingCount =
mlContext.Data.CreateEnumerable<An
imeRating>(dataView,
reuseRowObject: false).LongCount();

        RaportTBox.Text +=
($"Кількість унікальних
користувачів: {userCount}\r\n");
        RaportTBox.Text +=
($"Кількість унікальних аніме:
{animeCount}\r\n");
        RaportTBox.Text +=
($"Кількість записів:
{ratingCount}\r\n");
        Application.DoEvents();

        // Розділяємо дані на
тренувальний та тестовий набори
        var trainTestData =
mlContext.Data.TrainTestSplit(dataVie
w, testFraction: 0.2);
        var trainData =
trainTestData.TrainSet;
        var testData =
trainTestData.TestSet;

        // Створюємо конвеєр обробки
даних та модель
        var pipeline =
mlContext.Transforms.Conversion.Ma
pValueToKey(outputColumnName:
"UserIdEncoded", inputColumnName:
"UserId")

.Append(mlContext.Transforms.Conve
rsion.MapValueToKey(outputColumn
Name: "AnimeIdEncoded",
inputColumnName: "AnimeId"))
            // Matrix Factorization Trainer

.Append(mlContext.Recommendation()
.Trainers.MatrixFactorization(
                labelColumnName: "Rating",

matrixColumnIndexColumnName:
"UserIdEncoded",

matrixRowIndexColumnName:
"AnimeIdEncoded"
            ));

        RaportTBox.Text +=
"Тренування моделі ...\r\n";
        Application.DoEvents();
```

```
        // Дослідження швидкості навчання
        Stopwatch trainingStopwatch = new Stopwatch();
        trainingStopwatch.Start();

        // Навчання моделі на тренувальному наборі
        model = pipeline.Fit(trainData);

        trainingStopwatch.Stop();
        RaportTBox.Text += ($"Час тренування моделі: {trainingStopwatch.Elapsed.ToString("mm\\:ss\\.fff")}\r\n");

        // Дослідження швидкості обчислень
        Stopwatch evaluationStopwatch = new Stopwatch();
        evaluationStopwatch.Start();

        // Оцінка моделі на тестовому наборі
        var predictions = model.Transform(testData);
        var metrics = mlContext.Regression.Evaluate(predictions, labelColumnName: "Rating");

        evaluationStopwatch.Stop();

        // Виведення результатів оцінки моделі на тестовому наборі
        RaportTBox.Text += ("Оцінка моделі на тестовому наборі:\r\n");
        RaportTBox.Text += ($"RMSE: {metrics.RootMeanSquaredError:F2}\r\n");
        RaportTBox.Text += ($"MAE: {metrics.MeanAbsoluteError:F2}\r\n");
        RaportTBox.Text += ($"R-Squared: {metrics.RSquared:F2}\r\n");
        RaportTBox.Text += ($"Час оцінки моделі: {evaluationStopwatch.Elapsed.ToString("mm\\:ss\\.fff")}\r\n");

        _IsModelTrain = true;
      }
    }



    private void AddBtn_Click(object sender, EventArgs e) {
      if (IsDataEnteringCorrect()) {
        //Зберігання моделі
        string pathName = @"\teach\" + GenerateFileName() + ".zip";
        string localProj =

System.IO.Path.GetDirectoryName(System.Reflection.Assembly.GetExecutingAssembly().Location);

_ModelsProvider.InsertModels(ModelsNamesTBox.Text,

Convert.ToInt32(CategoryCBox.SelectedValue), CategoryCBox.Text, pathName);
        mlContext.Model.Save(model, dataView.Schema, localProj + pathName);
        ClearAllData();

_LogsProvider.InsertLogs(LoginForm.CurrentUser.UsersId,
          "Було навчено модель " + ModelsNamesTBox.Text, DateTime.Now);
        MessageBox.Show("Дані успішно збережено!");
```

```
    }
  }

  private void ClearBtn_Click(object
sender, EventArgs e) {
    ClearAllData();
  }

  private void ExitBtn_Click(object
sender, EventArgs e) {
    this.Close();
  }

  private void
ModelsGridView_CellClick(object
sender, DataGridViewCellEventArgs
e) {
    if (e.ColumnIndex == 5 &&
ModelsGridView[0,
e.RowIndex].Value.ToString() !=
_ModelsList[0].Message) {
      if (MessageBox.Show("Ви дійсно
хочете видалити цей елемент?",
"Видалити",
MessageBoxButtons.YesNo) ==
DialogResult.Yes) {

_ModelsProvider.DeleteModelsByMod
elsId(Convert.ToInt32(ModelsGridVie
w[0, e.RowIndex].Value.ToString()));
        DataLoad();
      }
    }
  }



  public string GenerateFileName() {
    DateTime now = DateTime.Now;
    string fileName =
string.Format("{0}_{1}_{2}_{3}_{4}
_{5}",
      now.Year, now.Month, now.Day,
now.Hour, now.Minute, now.Second);
    return fileName;
  }

  private void ClearAllData() {
    _IsModelTrain = false;
    ModelsNamesTBox.Text =
String.Empty;
    RaportTBox.Text = String.Empty;
    DataLoad();
  }
  private bool IsDataEnteringCorrect()
{
    bool isCorrect = true;
    if (!_IsModelTrain) {
      MessageBox.Show("Неможливо
зберегти дані. \r\nЩе не навчено
модель!", "Увага!");
      isCorrect = false;
    }
    if
(Convert.ToInt32(CategoryCBox.Selec
tedValue) > 0) {
      CategoryValidationLbl.Text =
NamesMy.ProgramButtons.RequiredV
alidation;
    } else {
      CategoryValidationLbl.Text =
NamesMy.ProgramButtons.ErrorValid
ation;
      isCorrect = false;
    }
    if
(_Validation.IsDataEntering(ModelsNa
mesTBox.Text)) {
      ModelsNamesValidationLbl.Text
=
NamesMy.ProgramButtons.RequiredV
alidation;
    } else {
      ModelsNamesValidationLbl.Text
=
```

```
NamesMy.ProgramButtons.ErrorValid
ation;
       isCorrect = false;
     }
     return isCorrect;
   }

   private void LoadAllDate() {
     _CategoryList =
_CategoryApp.GetCategoryList();
     CategoryCBox.DataSource =
_CategoryList;
     CategoryCBox.ValueMember =
"CategoryId";
     CategoryCBox.DisplayMember =
"CategoryName";
     CategoryCBox.SelectedIndex = 1;
     CategoryCBox.Enabled = false;
     DataLoad();
   }

   private void DataLoad() {
     int firstRowIndex = 0;
     if
(ModelsGridView.FirstDisplayedScroll
ingRowIndex > 0) {
       firstRowIndex =
ModelsGridView.FirstDisplayedScrolli
ngRowIndex;
     }
     try {
       _ModelsList =
_ModelsProvider.GetAllModels();

LoadDataInModelsGridView(_Models
List);
       if (_selectedRowIndex ==
ModelsGridView.Rows.Count) {
         _selectedRowIndex =
ModelsGridView.Rows.Count - 1;
       }
       if (_selectedRowIndex >= 0) {

ModelsGridView.FirstDisplayedScrolli
ngRowIndex = firstRowIndex;

ModelsGridView.Rows[_selectedRowI
ndex].Selected = true;
       }
     } catch (Exception ex) {

MessageBox.Show(ex.ToString());
     }
   }

   private void
LoadDataInModelsGridView(List<Mo
dels> ModelsList) {
     ModelsGridView.DataSource =
null;
     ModelsGridView.Columns.Clear();

ModelsGridView.AutoGenerateColum
ns = false;

ModelsGridView.RowHeadersVisible
= false;

     ModelsGridView.DataSource =
ModelsList;

     if (ModelsList.Count > 0) {
       if (ModelsList[0].Message ==
NamesMy.NoDataNames.NoDataInMo
dels) {
         DataGridViewColumn
messageColumn = new
DataGridViewTextBoxColumn();

messageColumn.DataPropertyName =
"Message";
         messageColumn.Width =
ModelsGridView.Width -
NamesMy.SizeOptins.MinusSizePanel;
```

```
ModelsGridView.Columns.Add(messa
geColumn);
        } else {
          DataGridViewColumn
DetailIdColumn = new
DataGridViewTextBoxColumn();

DetailIdColumn.DataPropertyName =
"ModelsId";

ModelsGridView.Columns.Add(DetailI
dColumn);

ModelsGridView.Columns[0].Visible
= false;

        DataGridViewColumn
numberColumn = new
DataGridViewTextBoxColumn();
        numberColumn.HeaderText =
"№ ";

numberColumn.DataPropertyName =
"Number";

numberColumn.DefaultCellStyle.Align
ment =
DataGridViewContentAlignment.Midd
leRight;
        numberColumn.Width =
NamesMy.SizeOptins.NumberSize;

ModelsGridView.Columns.Add(numbe
rColumn);

        DataGridViewColumn
ModelsNamesColumn = new
DataGridViewTextBoxColumn();

ModelsNamesColumn.HeaderText =
"Назва моделі";

ModelsNamesColumn.DataPropertyNa
me = "ModelsNames";
        ModelsNamesColumn.Width =
250;

ModelsGridView.Columns.Add(Model
sNamesColumn);

        DataGridViewColumn
CategoryNameColumn = new
DataGridViewTextBoxColumn();

CategoryNameColumn.HeaderText =
"Категорія";

CategoryNameColumn.DataPropertyN
ame = "CategoryName";
        CategoryNameColumn.Width =
150;

ModelsGridView.Columns.Add(Categ
oryNameColumn);

        DataGridViewColumn
ModelsFileModelColumn = new
DataGridViewTextBoxColumn();

ModelsFileModelColumn.HeaderText
= "Файл";

ModelsFileModelColumn.DataPropert
yName = "ModelsFileModel";
        ModelsFileModelColumn.Width
= 200;

ModelsGridView.Columns.Add(Model
sFileModelColumn);

        DataGridViewButtonColumn
IsResidesBtn = new
DataGridViewButtonColumn();
```

```
        IsResidesBtn.HeaderText =
"Видалити";
        IsResidesBtn.Text =
"Видалити";

IsResidesBtn.UseColumnTextForButto
nValue = true;
        IsResidesBtn.ToolTipText =
"Видалити";
        IsResidesBtn.Width =
NamesMy.SizeOptins.DeleteBtnSize;

ModelsGridView.Columns.Add(IsResi
desBtn);
```

```
        }
        for (int i = 0; i <
ModelsGridView.Columns.Count; i++)
{

ModelsGridView.Columns[i].HeaderC
ell.Style.BackColor = Color.LightGray;
        }
      }
    }

  }
}
```

```
}
```

ДОДАТОК В

Технічно завдання

ЗАТВЕРДЖУЮ
Проректор Українського державного
університету науки і технологій
Анатолій РАДКЕВИЧ

«ДОСЛІДЖЕННЯ АЛГОРИТМІВ РЕКОМЕНДАЦІЙНИХ СИСТЕМ»
Керівництво користувача
44165850.01351 – 01 І3 01

Завідувач кафедри КІТ
_____Вадим ГОРЯЧКІН
Керівник розробки
_____Олександр ІВАНОВ
Виконавець
_____Максим ПЛЯШКО
Нормоконтролер
_____Світлана ВОЛКОВА

2024

ЗАТВЕРДЖЕНО
44165850.01351-01 ІЗ 01

«ДОСЛІДЖЕННЯ АЛГОРИТМІВ РЕКОМЕНДАЦІЙНИХ СИСТЕМ»

Керівництво користувача

44165850.01351 – 01 ІЗ 01

Листів 9

2024

44165850.01351-01 І3 01

АНОТАЦІЯ

Документ 12345678.01234 – 01 І3 01 «Дослідження алгоритмів рекомендаційних систем. Керівництво користувача».

Програма розроблена мовою C# із використанням фреймворку ML.NET та бази даних MS SQL Server у програмному середовищі Visual Studio 2022.

ЗМІСТ

1 ВВЕДЕННЯ

Програмний засіб “Дослідження алгоритмів рекомендаційних систем” призначений для аналізу та оцінки ефективності сучасних алгоритмів рекомендацій, таких як градієнтний бустинг, матрична факторизація та швидке дерево Твідді.

Головною метою розробки є надання інструментарію для тренування, тестування та порівняння моделей, а також збирання показників точності, швидкодії та ресурсозатратності, на основі яких проводиться дослідження. Система дозволяє аналізувати продуктивність алгоритмів у різних сценаріях використання та обирати оптимальні підходи для конкретних завдань.

Цей програмний продукт призначений для науковців, розробників і аналітиків, які досліджують алгоритми рекомендаційних систем або працюють над їх впровадженням у практичних проектах. Інтуїтивний інтерфейс забезпечує легкість використання без необхідності глибокого технічного досвіду, а також дозволяє адаптувати систему для нових задач і даних.

Програмний засіб не потребує ознайомлення з додатковою експлуатаційною документацією і може використовуватися будь-яким користувачем із базовими знаннями у сфері інформаційних технологій.

2 ПРИЗНАЧЕННЯ ТА УМОВИ ЗАСТОСУВАННЯ

Функціональним призначенням розробленого програмного засобу є дослідження алгоритмів рекомендаційних систем із метою вибору найбільш ефективних підходів для задач персоналізації контенту та аналізу їхньої продуктивності в різних умовах.

Експлуатаційне призначення додатку “Дослідження алгоритмів рекомендаційних систем” – проведення тренування, тестування та порівняння алгоритмів, таких як градієнтний бустинг, матрична факторизація та швидке дерево Твідді. Програма дозволяє оцінювати точність рекомендацій, швидкість роботи та використання пам’яті при різних обсягах даних та кількості запитів.

Для забезпечення коректної роботи програми необхідно дотримуватися таких умов: наявність стабільного інтернет-з’єднання для завантаження необхідних бібліотек та компонентів, а також використання сучасного обладнання з відповідними характеристиками.

Програмний засіб розрахований на використання на пристроях, що відповідають таким мінімальним вимогам:

- процесор: тактова частота не менше 3.0 ГГц;
- оперативна пам’ять (RAM): не менше 8 ГБ;
- вбудована пам’ять (SSD): щонайменше 256 ГБ вільного місця;
- операційна система: Windows 10 або вище;
- роздільна здатність монітора: не нижче Full HD (1920x1080).

Ці умови забезпечують стабільне функціонування програмного забезпечення та зручність у користуванні навіть при роботі з великими обсягами даних.

3 ПІДГОТОВКА ДО РОБОТИ

Для початку роботи з програмним засобом “Дослідження алгоритмів рекомендаційних систем” необхідно виконати кілька кроків. Спершу необхідно завантажити та встановити програму, переконавшись, що на пристрої вже встановлено середовище виконання .NET Framework версії 4.7.2 або вище. Також слід переконатися, що наявна операційна система відповідає мінімальним вимогам, зокрема Windows 10 або новіша.

Перед запуском програми користувач повинен переконатися, що всі необхідні бібліотеки ML.NET та компоненти для роботи з базою даних MS SQL Server успішно встановлені. Конфігураційні файли програми, що зберігають дані підключення до бази даних та параметри роботи алгоритмів, мають бути поміщені до тієї ж директорії, що й виконуваний файл програми.

Для забезпечення коректної роботи програми слід переконатися, що підключення до інтернету стабільне, оскільки це може знадобитися для завантаження додаткових компонентів або оновлення програмного забезпечення. Дотримання цих умов гарантує успішну підготовку програми до роботи.

4 ОПИС ОПЕРАЦІЙ

Після встановлення та запуску програмного забезпечення “Дослідження алгоритмів рекомендаційних систем” на формі додатку представлені елементи для роботи з даними, управління моделями та авторизації користувачів:

– кнопка «Відкрити» – надає можливість вибору датасету для подальшого навчання чи тестування моделей. Після натискання відкривається діалогове вікно, де користувач може вказати шлях до потрібного файлу.

– кнопка «Додати» – виконує збереження тренованої моделі до бази даних MS SQL Server. Перед додаванням користувач повинен заповнити всі необхідні поля, такі як назва моделі, параметри та метадані.

– кнопка «Очистити» – забезпечує очищення полів форми від введених чи відображених даних, дозволяючи підготувати форму для введення нової інформації.

– кнопка «Вихід» – завершує роботу форми та закриває її, повертаючи користувача до попереднього інтерфейсу або виходу з програми.

– кнопка «Обрати» – дозволяє вибрати один із запропонованих алгоритмів для подальшого тестування. Користувач може переглядати доступні опції, після чого алгоритм активується для роботи.

– кнопка «Підтвердити» – використовується для авторизації користувача. Після введення облікових даних програма перевіряє їх у базі даних, підтверджуючи доступ до системи.

– кнопка «Реєстрація» – надає можливість новим користувачам зареєструватися. Відкривається форма, де вводяться персональні дані та створюється обліковий запис.

– кнопка «Зберегти» – забезпечує оновлення даних, внесених користувачем. Це дозволяє коригувати інформацію в базі даних, зберігаючи її актуальність.

– кнопка «Видалити» – виконує операцію видалення вибраних записів з бази даних. Після натискання програма відображає повідомлення для підтвердження дії.

Інтерфейс програми побудований таким чином, щоб користувач міг швидко та ефективно виконувати всі необхідні операції. Завдяки інтуїтивно зрозумілому дизайну, використання програми не вимагає значного досвіду чи спеціальної підготовки.

5 АВАРІЙНІ СИТУАЦІЇ

Якщо десктопний додаток запускається на пристрої, який не відповідає мінімальним системним вимогам, програма виведе повідомлення про помилку і завершить роботу.

У разі відсутності з’єднання з базою даних або неможливості доступу до обраного датасету, додаток повідомить користувача про помилку і запропонує перевірити підключення до мережі або правильність вказаного шляху до файлу.

Якщо під час роботи програми виникає некоректна поведінка, наприклад, зависання інтерфейсу чи збої в обробці даних, користувачеві рекомендується закрити додаток і перезапустити його.

У випадку помилок, пов'язаних з авторизацією або реєстрацією, додаток надасть інформацію про причину збою, таку як некоректний ввід даних, і запропонує виправити її для продовження роботи.

При виявленні інших критичних помилок програма автоматично створить журнал подій із детальним описом помилки, що дозволить розробникам провести аналіз і усунути проблему в майбутніх оновленнях.

6 РЕКОМЕНДАЦІЇ ЩОДО ЗАСТОСУВАННЯ

Після встановлення десктопного додатка на пристрій користувачеві необхідно запустити його через виконуваний файл, розташований у вказаній директорії.

Після запуску програми відкриється головне вікно, яке містить усі основні функціональні елементи для роботи з додатком, такі як вибір датасету, тренування моделей, тестування алгоритмів, а також управління користувачами через модулі авторизації та реєстрації.

Користувачам рекомендується перед початком роботи ознайомитися з інтерфейсом програми та перевірити коректність вказаного шляху до датасету. Для забезпечення ефективної роботи слід використовувати пристрої, що відповідають системним вимогам програми.

Після завершення роботи з додатком його можна закрити за допомогою відповідної кнопки «Вихід» у меню програми або стандартним методом закриття віконного додатка.