МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Харківський національний університет імені В.Н.Каразіна

Факультет математики і інформатики

Кафедра теоретичної та прикладної інформатики

Кваліфікаційна робота

магістр

на тему «Використання машинного навчання для персоналізації веб-додатків»

Виконав: студент 2 курсу, групи МФ-62

спеціальність 122 «Комп’ютерні науки»

освітньо-професійна програма «Інформатика»

Лебедєв М. О.

Керівник Бережна Н. І.

Рецензент_____

(прізвище та ініціали)

Харків – 2024 року

ЗМІСТ

ВСТУП 3
1 ОГЛЯД АЛГОРИТМІВ МАШИННОГО НАВЧАННЯ ДЛЯ ПЕРСОНАЛІЗАЦІЇ 6
1.1 Поняття персоналізації веб-додатків та роль машинного навчання..... 6
1.2 Огляд сучасних підходів до машинного навчання для персоналізації 9
1.3 Особливості вибору алгоритмів для персоналізації залежно від типу даних 11
1.4 Огляд популярних моделей машинного навчання для персоналізації... 18
2 АНАЛІЗ АЛГОРИТМІВ ТА ЇХ ПРОГРАМНА РЕАЛІЗАЦІЯ 23
2.1 Алгоритм градієнтного бустингу з оптимізацією обчислювальних витрат 23
2.2 Алгоритм матричної факторизації 25
2.3 Алгоритм швидкого дерева Твідді 27
2.4 Технологічний стек для програмної реалізації алгоритмів 29
2.5 Програмна реалізація алгоритмів 32
3 ПОРІВНЯННЯ АЛГОРИТМІВ ТА РЕКОМЕНДАЦІЇ ЩОДО ЇХ ВИКОРИСТАННЯ 46
3.1 Порівняння розглянутих алгоритмів за основними характеристиками . 46
3.1.1. Швидкість навчання моделей у різних умовах..... 46
3.1.2. Точність рекомендацій залежно від характеристик даних..... 50
3.1.3. Вимоги до ресурсів та ефективність використання алгоритмів на реальних даних 55
3.2 Перспективи розвитку алгоритмів та їх комбінування для покращення результатів персоналізації..... 57
3.3 Рекомендації щодо вибору алгоритмів машинного навчання залежно від специфіки завдання..... 58
ВИСНОВКИ..... 61
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ..... 63
ДОДАТОК А. ПРИКЛАДИ РЕЗУЛЬТАТІВ НАВЧАННЯ МОДЕЛЕЙ..... 67

ВСТУП

Сучасні веб-додатки є невід'ємною частиною багатьох бізнес-процесів, а також сфери обслуговування та розваг. З огляду на динамічний розвиток технологій і зростаючі очікування користувачів, важливо, щоб веб-системи були здатні швидко адаптуватися до індивідуальних потреб кожного користувача. Однак, стандартні підходи до побудови веб-додатків часто не забезпечують необхідного рівня персоналізації, що призводить до зниження ефективності взаємодії між користувачами і системою. Проблема полягає в тому, що сучасний користувач очікує індивідуального досвіду, де контент і функціональність веб-додатка підлаштовуються під його вподобання, інтереси та поведінкові патерни [1]. Стандартні методи персоналізації, такі як рекомендаційні системи на основі фільтрації або попередньо встановлені правила, часто є недостатніми для забезпечення гнучкості та точності у взаємодії з користувачами.

В умовах розвитку технологій машинного навчання з'являються нові можливості для персоналізації, яка базується на аналізі великих даних і поведінкових особливостей користувачів [2]. Машинне навчання дозволяє моделювати взаємодії, аналізувати минулі дії користувачів і на основі цього передбачати їхні майбутні потреби та інтереси. Важливість використання цих технологій не обмежується лише оптимізацією контенту – вони дозволяють істотно підвищити рівень залученості користувачів та сприяти їхньому зростаючому задоволенню від використання системи.

Актуальність цієї проблеми стає ще більш помітною в умовах українського ринку, де конкуренція за користувача зростає. Для українських компаній, які прагнуть вийти на глобальний рівень, критично важливо впроваджувати технології, що дозволяють забезпечити високий рівень персоналізації та індивідуального підходу. Дослідження та впровадження методів машинного навчання є важливим кроком у цьому напрямку. Західні технологічні компанії вже активно використовують ці підходи, однак існуючі

рішення не завжди враховують локальні особливості та вимоги ринку України, що потребує адаптації та подальшого вдосконалення під локальні умови.

Проведення дослідження є необхідним для розробки ефективних методів інтеграції машинного навчання в системи веб-додатків, що дозволить забезпечити більш високу продуктивність і адаптивність. Особливу увагу варто приділити питанням приватності та безпеки даних, адже персоналізація вимагає обробки великої кількості особистої інформації користувачів. Крім того, виникає потреба в дослідженні ефективності різних методів машинного навчання для конкретних типів персоналізацій, що дозволить розробити рекомендації для українських компаній щодо впровадження цих технологій.

Об’єктом дослідження є процес персоналізації веб-додатків, який спрямований на адаптацію вмісту, функціональних можливостей і користувацького інтерфейсу під індивідуальні потреби користувачів на основі аналізу їхньої поведінки.

Предметом дослідження є алгоритми машинного навчання, які використовуються для персоналізації веб-додатків, їхні особливості, ефективність та доцільність застосування в залежності від даних та умов роботи системи.

Мета дослідження є розробка та аналіз алгоритмів машинного навчання, що забезпечують персоналізацію веб-додатків на основі великих обсягів даних.

Для досягнення поставленої мети, необхідно виконати наступні задачі:

– провести огляд сучасних алгоритмів машинного навчання, які використовуються для персоналізації веб-додатків, з акцентом на їх можливості та обмеження;

– проаналізувати особливості вибору алгоритмів для персоналізації залежно від типу даних, що використовуються у веб-додатках;

– розробити програмну реалізацію кількох популярних алгоритмів машинного навчання, таких як градієнтного бустингу з оптимізацією обчислювальних витрат, матрична факторизація та швидке дерево Твідді;

– провести порівняння розглянутих алгоритмів за основними характеристиками та надати рекомендації щодо їх використання в реальних умовах.

У процесі дослідження були використані наступні методи:

– аналіз літературних джерел та наукових статей – використовувався для систематизації та узагальнення сучасних підходів до персоналізації веб-додатків на основі машинного навчання, що дозволило визначити актуальні проблеми та можливі шляхи їх вирішення;

– метод порівняльного аналізу – застосовувався для оцінки ефективності різних алгоритмів машинного навчання шляхом порівняння їхніх характеристик, таких як швидкість навчання, точність прогнозування та вимоги до ресурсів;

– моделювання та програмна реалізація алгоритмів – метод, що був використаний для створення програмних реалізацій алгоритмів машинного навчання, необхідних для персоналізації веб-додатків. Це дозволило перевірити їхню працездатність та адаптивність до різних умов;

– експериментальний метод – використовувався для перевірки та тестування алгоритмів на реальних даних, що дозволило оцінити їхню практичну ефективність, враховуючи різні типи даних та специфіку задач персоналізації.

Практичне значення отриманих результатів полягає у можливості застосування розроблених рекомендацій щодо вибору та використання алгоритмів машинного навчання для персоналізації веб-додатків у реальних умовах. Запропоновані методи дозволяють підвищити точність і ефективність персоналізації, що сприяє покращенню користувацького досвіду та оптимізації бізнес-процесів. Результати можуть бути використані компаніями для впровадження більш індивідуалізованих підходів до взаємодії з користувачами на українському та міжнародному ринках.

1 ОГЛЯД АЛГОРИТМІВ МАШИННОГО НАВЧАННЯ ДЛЯ ПЕРСОНАЛІЗАЦІЇ

1.1 Поняття персоналізації веб-додатків та роль машинного навчання

Персоналізація веб-додатків – це процес адаптації контенту, інтерфейсу та функціональності під індивідуальні потреби і вподобання конкретного користувача [3]. Веб-додатки, які використовують персоналізацію, пропонують користувачам більш релевантний і цінний досвід, що, у свою чергу, підвищує рівень задоволеності та залученості. Користувачі взаємодіють із веб-системами, які очікують не лише доступ до загального контенту, а й точних рекомендацій, швидкого доступу до потрібної інформації та адаптації під їхні вподобання. Персоналізація допомагає досягти цього завдяки аналізу поведінки користувачів і створенню унікальних пропозицій для кожного з них.

Приклади персоналізації можна побачити в багатьох популярних сервісах, які активно використовують машинне навчання для підвищення якості взаємодії з користувачами. Наприклад, Netflix застосовує складні алгоритми для аналізу переглядів, жанрових уподобань і оцінок, щоб персонально рекомендувати користувачам нові фільми та серіали, які можуть їх зацікавити [4]. Ця система також враховує час перегляду, щоб зробити рекомендації ще більш точними. Крім того, Netflix персоналізує обкладинки фільмів, демонструючи користувачам ті зображення, які, на думку системи, будуть для них більш привабливими.

В онлайн-магазинах, таких як Amazon, персоналізація має ключове значення для збільшення продажів. Amazon пропонує товари на основі попередніх покупок, пошукових запитів та навіть рекомендацій схожих товарів, які купували інші користувачі з подібними вподобаннями. Відстеження списків бажань, попередніх переглядів і рекомендацій на основі категорій допомагає покупцям знаходити потрібні товари швидше.

Соціальні мережі, такі як Facebook та Instagram, активно використовують алгоритми машинного навчання для персоналізації стрічок новин [5]. Вони аналізують взаємодію користувачів із контентом (лайки, коментарі, поширення) і показують більше постів від тих друзів або сторінок, які викликають найбільшу зацікавленість. Більше того, алгоритми аналізують час, проведений на кожному пості, та визначають, який контент буде найбільш релевантним для конкретного користувача.

Також цікавим прикладом є музичні сервіси, такі як Spotify, які використовують персоналізацію для створення індивідуальних плейлистів, таких як «Discover Weekly» або «Release Radar». Ці плейлисти автоматично генеруються на основі музичних уподобань користувача, включаючи його історію прослуховування, вподобані треки та виконавців. Spotify також використовує алгоритми для аналізу схожих слухачів і пропонує музику, яка популярна серед тих, хто має схожі музичні інтереси.

Водночас розробка персоналізованого досвіду для користувачів є складним процесом, який вимагає опрацювання великої кількості даних і врахування безлічі факторів, що впливають на вибір користувача. Тому сучасні веб-додатки дедалі частіше використовують машинне навчання для автоматизації цього процесу.

Машинне навчання – це галузь штучного інтелекту, яка дозволяє комп'ютерним системам навчатися на основі даних без явного програмування кожної дії (рис. 1.1) [6]. Його основна перевага полягає в здатності виявляти приховані закономірності в поведінці користувачів, аналізувати великі обсяги даних і на основі цього автоматично адаптувати систему. У випадку персоналізації веб-додатків машинне навчання забезпечує гнучке й динамічне налаштування контенту для кожного користувача в режимі реального часу.

МН дозволяє веб-додаткам аналізувати історію поведінки користувачів, виявляти їхні вподобання та прогнозувати майбутні дії. Завдяки цьому можна автоматично визначати, який контент або товари будуть найбільш цікавими для користувача, без необхідності вручну налаштовувати кожну можливу

взаємодію. Це особливо важливо в умовах, коли кількість користувачів і варіацій їхньої поведінки стає надто великою для традиційних підходів.

Рисунок 1.1 – Роль МН для персоналізації веб-додатків

Роль машинного навчання полягає не тільки в тому, щоб спрощувати обробку даних, але й у забезпеченні адаптивності системи [7]. Веб-додатки на основі машинного навчання можуть самостійно "навчатися" на основі даних і оновлювати свої прогнози та рекомендації, коли користувач взаємодіє з додатком. Це дозволяє системі змінювати свої рекомендації або контент відповідно до нових уподобань або потреб користувача, що робить досвід більш релевантним і персоналізованим.

Отже, машинне навчання є основною технологією, яка стоїть за сучасною персоналізацією веб-додатків. Завдяки його можливостям обробляти великі обсяги даних і робити точні прогнози, персоналізація стає доступною для різних типів додатків і користувачів, забезпечуючи їм унікальний досвід взаємодії з системою.

1.2 Огляд сучасних підходів до машинного навчання для персоналізації

Машинне навчання стало невід'ємною частиною сучасних веб-додатків, особливо в контексті персоналізації, яка забезпечує користувачам індивідуальний досвід. Успішне впровадження персоналізації залежить від правильного вибору алгоритмів машинного навчання, здатних аналізувати великі обсяги даних і швидко адаптувати систему до потреб кожного користувача. Розвиток технологій дозволив виділити кілька основних підходів до персоналізації на основі машинного навчання, які широко використовуються в різних галузях.

Один із найбільш поширених підходів – це рекомендаційні системи. Вони використовують машинне навчання для автоматичного аналізу вподобань та поведінки користувачів і на основі цього пропонують їм контент, який може бути цікавим.

Колаборативна фільтрація аналізує поведінкові патерни багатьох користувачів і на основі схожих дій робить припущення про ймовірні вподобання конкретного користувача [8]. Наприклад, якщо двоє користувачів мають схожі вподобання у виборі фільмів, система може запропонувати одному з них той контент, який сподобався іншому (рис. 1.2).

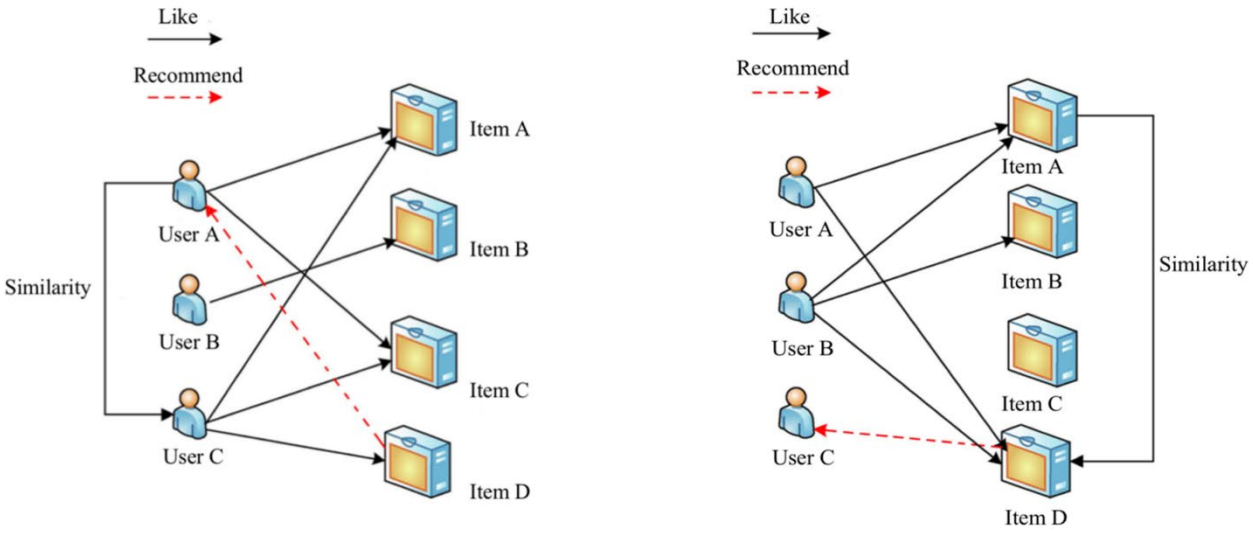

Рисунок 1.2 – Принцип колаборативної фільтрації [9]

Кластеризація у свою чергу дозволяє об'єднувати користувачів у групи (сегменти) на основі їхньої поведінки або інших характеристик, наприклад, віку, місця проживання або частоти покупок (рис. 1.3) [10]. Це дає змогу застосовувати різні стратегії для кожної групи, зокрема персоналізовані рекламні кампанії або спеціальні пропозиції.

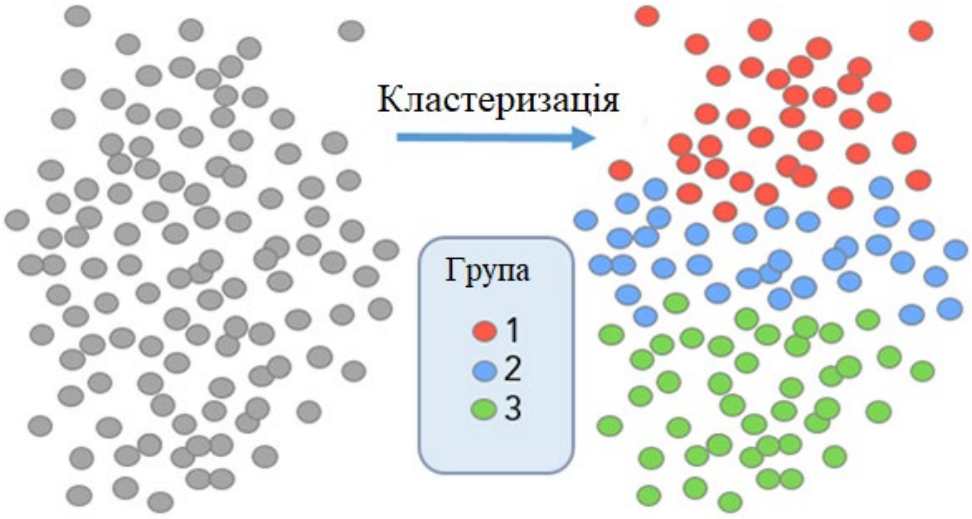

Рисунок 1.3 – Принцип кластеризації

Кластеризація не потребує попередньої інформації про те, до якої категорії належить користувач, а дозволяє виявити приховані закономірності у великих обсягах даних, що робить цей підхід надзвичайно гнучким. Найпоширенішими методами кластеризації є K-середніх і ієрархічна кластеризація, які ефективно розбивають користувачів на кластери з подібними характеристиками.

Окрім класичних підходів, останніми роками зростає популярність використання глибокого навчання для персоналізації [11]. Глибокі нейронні мережі дозволяють працювати з надзвичайно складними і великими масивами даних, виявляючи приховані патерни, які складно виявити традиційними методами. Наприклад, рекурентні нейронні мережі використовуються для аналізу послідовних даних, таких як історія дій користувачів, що дозволяє робити точні прогнози щодо майбутньої активності. Конволюційні нейронні мережі застосовуються для роботи із зображеннями або текстовими даними,

що дозволяє покращити персоналізацію в системах, що працюють з медіа-контентом.

Важливою частиною сучасних підходів до персоналізації є комбіновані методи, які поєднують різні підходи машинного навчання для досягнення кращих результатів. Наприклад, рекомендаційні системи можуть поєднувати колаборативну фільтрацію з класифікаційними моделями, що дозволяє враховувати як індивідуальні уподобання користувачів, так і поведінкові патерни великих груп. Комбінування алгоритмів підвищує точність і швидкість персоналізації, оскільки кожен метод компенсує слабкі сторони інших.

Таким чином, сучасні підходи до машинного навчання для персоналізації є гнучкими та потужними інструментами, що дозволяють забезпечити індивідуальний підхід до кожного користувача. Вибір конкретного методу або їхньої комбінації залежить від типу даних, що використовуються, специфіки завдань та ресурсних обмежень системи. Розвиток технологій машинного навчання продовжує відкривати нові можливості для підвищення ефективності персоналізації, забезпечуючи більш точне і адаптивне взаємодіє з користувачами.

1.3 Особливості вибору алгоритмів для персоналізації залежно від типу даних

Вибір алгоритмів машинного навчання для персоналізації веб-додатків значною мірою залежить від типу даних, з якими працює система. Типи даних можуть бути різними – від структурованих числових і категорійних даних до неструктурованих текстових, мультимедійних або часових рядів. Кожен з цих типів вимагає застосування певних підходів і алгоритмів, які найкраще адаптовані для обробки даних такого формату та забезпечують найбільш точні й ефективні результати персоналізації.

Структуровані дані – це тип даних, що мають суворо визначену організацію у формі таблиць або баз даних, де кожен рядок представляє

окреме спостереження, а кожен стовпець – конкретну характеристику або параметр цього спостереження [12]. Така організація даних дозволяє легко їх аналізувати за допомогою математичних або статистичних методів. Структуровані дані широко застосовуються для персоналізації в інформаційних системах, оскільки вони надають чіткий і передбачуваний формат для збирання та обробки даних про користувачів і об’єкти.

Структуровані дані мають перевагу в тому, що вони стандартизовані, легко піддаються аналізу та обробці за допомогою алгоритмів машинного навчання, що дозволяє швидко генерувати персоналізовані рекомендації. Такі дані зазвичай використовуються в e-commerce системах, CRM, та інших бізнес-додатках, де важливо мати можливість аналізувати поведінку користувачів, їхні вподобання та історію транзакцій. У таких системах кожен параметр може описувати конкретну характеристику користувача, його дії або взаємодію з системою.

На рис. 1.4 представлено основні характеристики, які можуть бути використані у структурованих наборах даних для персоналізації веб-додатків.

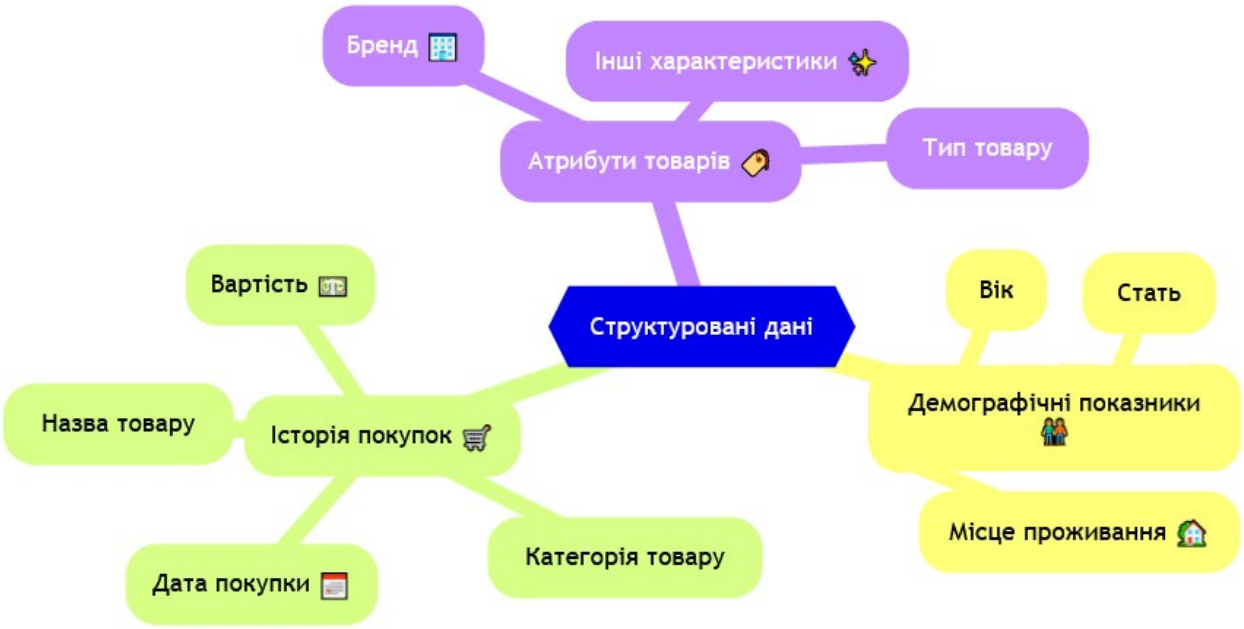

Рисунок 1.4 – Основні характеристики структурованих даних

До структурованих даних належать такі ключові елементи, як демографічні показники, які включають змінні, що стосуються віку, статі та місця проживання користувачів; історія покупок, яка охоплює інформацію про

попередні транзакції користувача, такі як назва товару, його категорія, дата та вартість покупки; а також категорійні атрибути товарів або послуг, до яких відносяться такі характеристики, як тип товару, його бренд та інші визначальні риси.

Для таких типів даних підходять алгоритми, що здатні ефективно обробляти числові й категорійні дані, такі як:

– дерева рішень та випадкові ліси, які добре працюють із категорійними та числовими параметрами, можуть використовуватися для сегментації користувачів або для надання персоналізованих рекомендацій на основі їхніх демографічних характеристик.

– матрична факторизація, яка часто застосовується в рекомендаційних системах, дозволяє виявляти приховані зв'язки між користувачами та товарами на основі спільних патернів у структурованих даних (наприклад, на основі оцінок товарів).

Текстові дані – це тип даних, які представлені у вигляді текстових повідомлень, документів, коментарів або інших форм природної мови. Вони зазвичай не мають чітко визначеної структури, як у випадку зі структурованими даними, але можуть містити важливу інформацію, яку можливо використовувати для аналізу та персоналізації [13]. Оскільки текстові дані часто є неструктурованими, їхня обробка потребує застосування спеціальних алгоритмів для вилучення корисної інформації, таких як методи обробки природної мови (NLP).

Текстові дані використовуються для персоналізації веб-додатків через аналіз взаємодії користувачів із текстовими полями, наприклад, у вигляді коментарів, відгуків або пошукових запитів. Такий аналіз дозволяє системам краще розуміти інтереси користувачів, їхні настрої або потреби. Обробка текстових даних є важливою складовою для автоматизації рекомендаційних систем, оскільки вона дозволяє виявляти контекст і зміст взаємодій, що мають велике значення для побудови індивідуальних пропозицій.

На рис. 1.5 представлені приклади характеристик текстових даних, які можуть використовуватися у персоналізації веб-додатків.

Рисунок 1.5 – Основні характеристики текстових даних

Текстові дані включають ключові елементи, такі як пошукові запити користувачів, що дозволяють виявляти їхні актуальні потреби та вподобання; коментарі або відгуки, що містять оцінки продуктів або послуг; а також тематику й тональність текстових повідомлень, які можуть бути використані для аналізу настрою користувачів та побудови рекомендацій на основі їхньої емоційної реакції.

Для обробки текстових даних підходять алгоритми, такі як:

– латентне розміщення Діріхле, що дозволяє визначати основні теми у великих текстових масивах і створювати тематичні профілі користувачів для персоналізації контенту [14];

– рекурентні нейронні мережі, які використовуються для аналізу послідовностей текстових даних і прогнозування майбутніх запитів або вподобань користувачів на основі попередніх текстових взаємодій.

Мультимедійні дані – це тип даних, які включають зображення, відео, аудіо та інші види мультимедійного контенту. Вони зазвичай є неструктурованими або напівструктурованими, що ускладнює їхню обробку

традиційними методами. Мультимедійні дані використовуються в багатьох сучасних веб-додатках, таких як платформи для потокового відео, соціальні мережі або системи рекомендацій, де необхідно аналізувати та персоналізувати візуальний і аудіальний контент для користувачів.

Використання мультимедійних даних для персоналізації є важливим, оскільки вони дозволяють системам краще розуміти вподобання користувачів на основі їхньої взаємодії з візуальним або аудіо-контентом [15]. Наприклад, веб-додатки, які аналізують перегляди зображень або відео, можуть адаптувати рекомендації відповідно до стилю або типу контенту, який користувачі переглядають найчастіше. Однак мультимедійні дані вимагають складніших алгоритмів для обробки, оскільки для вилучення корисної інформації необхідно аналізувати значні обсяги неструктурованих даних.

На рис. 1.6 зображено приклади основних характеристик мультимедійних даних, які використовуються у персоналізації веб-додатків.

Рисунок 1.6 – Основні характеристики мультимедійних даних

Мультимедійні дані містять такі ключові елементи, як зображення, відеофайли та аудіофайли, кожен з яких несе унікальну інформацію, що може бути використана для глибшого розуміння вподобань користувачів. Наприклад, інформація про переглянуті зображення дозволяє не лише аналізувати стилістичні уподобання, але й виявляти кольорові схеми, форми

або візуальні елементи, які користувачі віддають перевагу, що може бути корисним у веб-дизайні або рекламних кампаніях. Відеофайли можуть бути використані для аналізу жанрових інтересів користувачів, їхньої схильності до певних типів контенту, а також для прогнозування майбутніх уподобань на основі історії переглядів. Аудіофайли, у свою чергу, допомагають у персоналізації музичних стрімінгових сервісів, аналізуючи не лише прослуховувані треки та жанри, а й емоційний тон музики або конкретні ритмічні особливості, що дозволяє надавати користувачам більш точні рекомендації щодо музики, яка відповідає їхнім настроям та інтересам.

Для обробки мультимедійних даних використовуються спеціалізовані алгоритми, такі як:

– конволюційні нейронні мережі, що ефективно обробляють візуальні дані, виявляючи характерні риси зображень або відео для покращення персоналізованих рекомендацій на основі візуального контенту;

– автоенкодери, які використовуються для стиснення та вилучення ознак із мультимедійних файлів, дозволяючи створювати моделі для аналізу великих обсягів мультимедійних даних і застосовувати їх у системах персоналізації.

Часові ряди – це тип даних, які представляють собою послідовність спостережень, зафіксованих у різні моменти часу [16]. Вони часто використовуються для аналізу змін у поведінці користувачів або процесів, що відбуваються в часі. Часові ряди можуть включати такі параметри, як історія відвідувань веб-сторінок, дії користувачів у конкретні періоди часу, а також змінні, що показують зміну трендів або циклів. Основна особливість часових рядів полягає в наявності тимчасової залежності між даними, що ускладнює їхній аналіз звичайними методами.

Часові ряди використовуються для персоналізації в тих випадках, коли необхідно враховувати час виконання дій користувачами або виявляти сезонні зміни в їхній поведінці. Наприклад, на основі аналізу активності користувачів у різні години дня можна запропонувати їм персоналізовані рекомендації у

моменти найбільшої активності. Крім того, прогнозування майбутніх дій користувачів на основі їхньої історії взаємодії з системою також ґрунтується на часових рядах.

На рис. 1.7 представлені основні характеристики часових рядів, які можуть використовуватися для персоналізації веб-додатків.

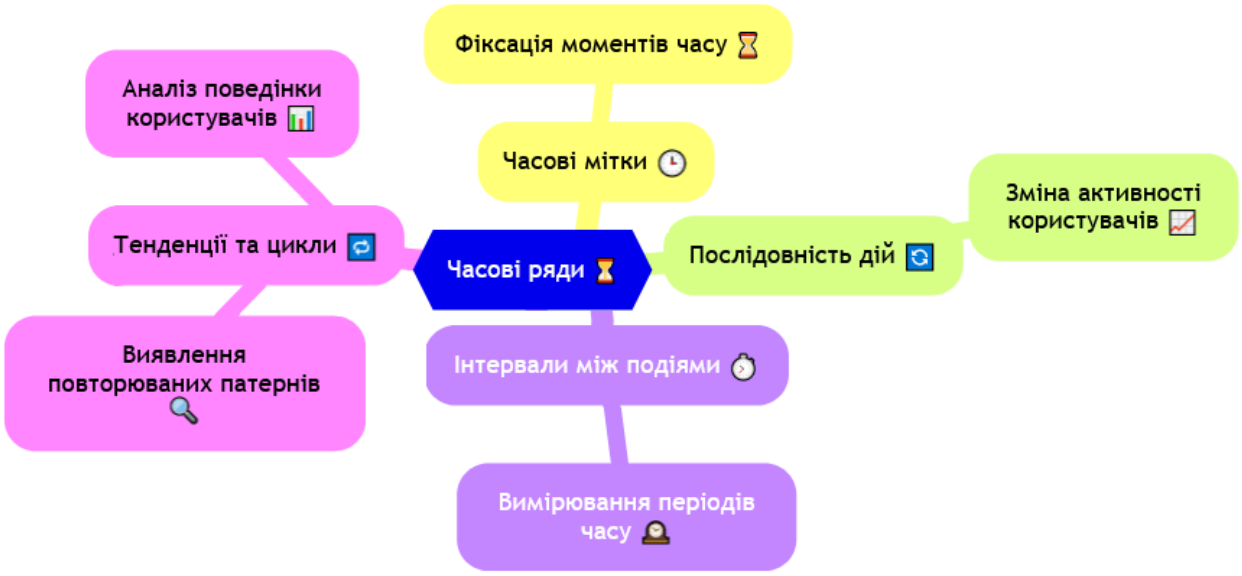

Рисунок 1.7 – Основні характеристики часових рядів

Часові ряди включають ключові елементи, такі як часові мітки, що фіксують момент часу, коли була здійснена дія, та послідовність дій, що відображає зміну активності користувача протягом певного періоду. Інші важливі елементи можуть включати інтервали між подіями, а також тенденції та цикли, що допомагають виявляти повторювані патерни в поведінці користувачів.

Для обробки часових рядів використовуються наступні алгоритми:

– рекурентні нейронні мережі, які здатні обробляти послідовності даних і враховувати тимчасові залежності, що робить їх ефективними для прогнозування подальших дій користувачів на основі попередньої активності;

– експоненційне згладжування, яке використовується для аналізу трендів і сезонних коливань у даних, що дозволяє краще прогнозувати зміни в поведінці користувачів та адаптувати персоналізовані пропозиції відповідно до змін у часі.

У багатьох випадках веб-додатки використовують кілька типів даних одночасно, таких як текстові коментарі, зображення та структуровані дані про користувачів. У таких випадках важливо вибирати алгоритми, які можуть ефективно працювати з комбінованими даними. Комбіновані підходи дозволяють створювати більш точні й релевантні персоналізовані рекомендації, оскільки вони враховують різноманітні джерела інформації.

1.4 Огляд популярних моделей машинного навчання для персоналізації

Сучасні алгоритми машинного навчання дозволяють створювати високоефективні системи персоналізації для веб-додатків. Ці моделі аналізують великі обсяги даних про користувачів, їхню поведінку, інтереси та вподобання, щоб автоматично пропонувати персоналізований контент. Залежно від типу даних, задачі та контексту, можуть застосовуватися різні моделі, кожна з яких має свої особливості й переваги. У цьому підрозділі розглянемо найпопулярніші моделі машинного навчання, які часто використовуються для персоналізації.

Колаборативна фільтрація є одним із найпоширеніших методів персоналізації, який використовується у багатьох популярних платформах, таких як Netflix, Amazon і YouTube. Алгоритми колаборативної фільтрації поділяються на дві основні групи:

– User-based. Даний підхід фокусується на пошуку користувачів зі схожими вподобаннями [17]. Наприклад, якщо два користувачі мають схожі вподобання щодо фільмів, система пропонує одному з них ті фільми, які вже сподобалися іншому;

– Item-based. У цьому випадку система аналізує схожість між об'єктами на основі їхніх характеристик або оцінок користувачів [18]. Якщо кілька користувачів позитивно оцінили певний товар, алгоритм може рекомендувати його іншим користувачам, які мають подібні вподобання.

Основною перевагою колаборативної фільтрації є те, що вона не потребує явної інформації про характеристики об’єктів, а лише використовує дані про взаємодії користувачів з системою. Однак такі алгоритми можуть зіткнутися з проблемою «холодного старту» для нових користувачів або нових об'єктів, оскільки на початку недостатньо інформації для створення точних рекомендацій.

Матрична факторизація є потужною технікою, яка також використовується в рекомендаційних системах для персоналізації [19]. Цей підхід базується на розкладі матриці взаємодій між користувачами та об'єктами на дві менші матриці: одну для користувачів і одну для об'єктів. Метою є виявлення прихованих факторів, які впливають на взаємодію. Наприклад, у випадку рекомендацій фільмів, такі фактори можуть включати жанрові уподобання користувачів або характеристику акторів (рис. 1.8).

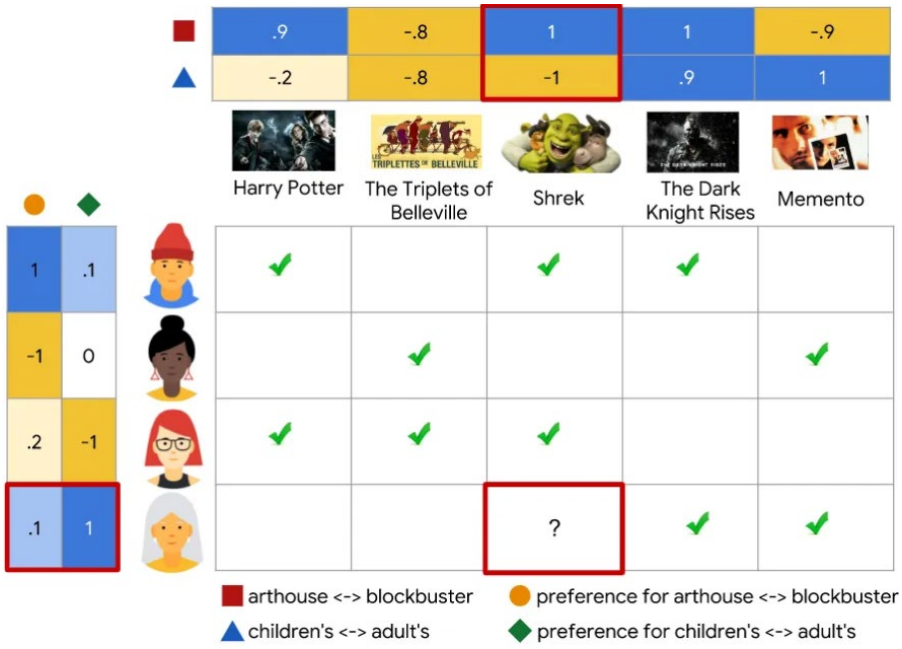

Рисунок 1.8 – Приклад використання матричної факторизації [20]

Однією з найбільш популярних реалізацій цього підходу є алгоритм стохастичного градієнтного спуску, який використовується для оптимізації матричної факторизації. Матрична факторизація дозволяє ефективно працювати з великими наборами даних і знаходити складні взаємозв'язки між користувачами та об'єктами. Вона добре підходить для персоналізації на

платформах із великою кількістю користувачів та об'єктів, таких як стрімінгові сервіси або інтернет-магазини.

Також варто згадати і про дерева рішень, які є популярними серед моделей машинного навчання завдяки своїй простоті, прозорості та здатності обробляти як числові, так і категорійні дані [21]. Для персоналізації вони використовуються для класифікації користувачів або об'єктів на основі різних параметрів. Одним із найвідоміших прикладів є алгоритм випадкового лісу, який складається з кількох дерев рішень і використовує їх для підвищення точності класифікації (рис. 1.9).

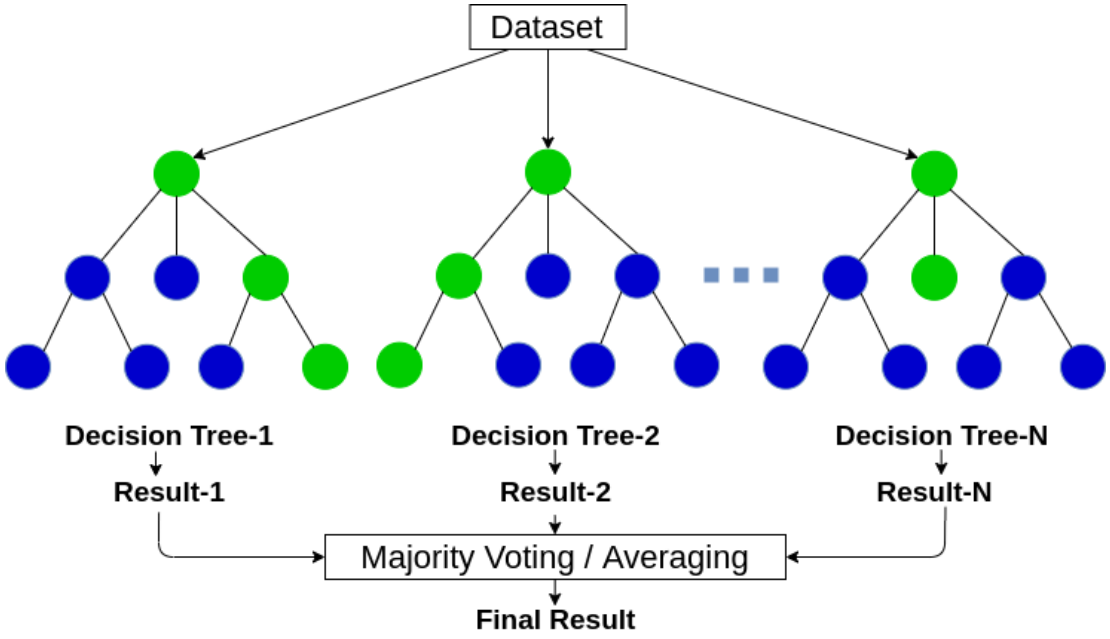

Рисунок 1.9 – Схема роботи алгоритму випадкового лісу [22]

Дерева рішень дозволяють будувати складні правила для персоналізації на основі різних факторів, таких як демографічні дані, поведінка користувачів або їхні оцінки. Вони добре працюють у системах, де потрібно швидко обробляти великий обсяг даних і створювати персоналізовані пропозиції для кожного користувача.

Однією з популярних реалізацій рекурентних нейронних мереж є моделі типу довгої короткострокової пам'яті (ДКП), які відрізняються своєю здатністю ефективно зберігати й обробляти інформацію про довгострокові залежності у даних (рис. 1.10). Це дозволяє мережам не тільки аналізувати поточні дії користувачів, але й враховувати їхню історію взаємодій із

системою, зберігаючи важливі деталі навіть через значний проміжок часу [23]. Завдяки цьому, моделі ДКП стають особливо корисними в задачах персоналізації, де поведінка користувачів є складною і може мати залежності від минулих дій у довгостроковій перспективі.

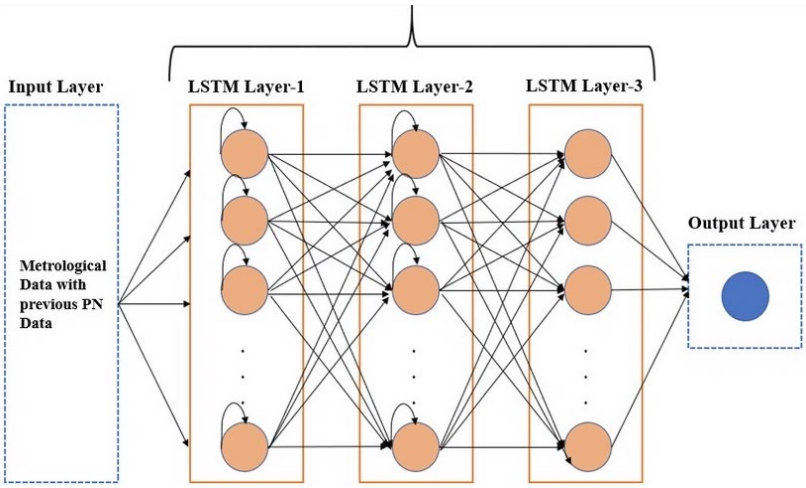

Рисунок 1.10 – Схема роботи алгоритм LSTM [24]

У випадку регулярного перегляду певного типу контенту або здійснення повторюваних покупок, система може будувати більш точні прогнози та пропонувати рекомендації, враховуючи як поточні вподобання, так і загальну тенденцію поведінки користувача, що спостерігається протягом тривалого часу. Таким чином, ДКП-мережі ефективно вирішують проблему обмежених короткострокових залежностей, характерну для традиційних рекурентних нейронних мереж.

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.

Гібридні алгоритми широко використовуються на великих платформах, таких як Amazon або Spotify, де існує потреба обробляти великі обсяги даних і забезпечувати різноманітні рекомендації для широкої аудиторії.

Таким чином, сучасні моделі машинного навчання пропонують широкий спектр рішень для персоналізації веб-додатків. Кожен підхід має свої сильні сторони і може бути адаптований для конкретних умов роботи, що дозволяє створювати більш гнучкі та ефективні системи персоналізації.

2 АНАЛІЗ АЛГОРИТМІВ ТА ЇХ ПРОГРАМНА РЕАЛІЗАЦІЯ

У рамках дослідження важливим етапом є аналіз обраних алгоритмів, що використовуються для розв'язання задачі, та їх ефективність у конкретних умовах. Це дозволяє оцінити переваги та недоліки кожного методу, а також обґрунтувати вибір оптимального підходу. Окрім теоретичного огляду, важливо провести програмну реалізацію алгоритмів для вивчення їхньої продуктивності та поведінки в реальних сценаріях.

2.1 Алгоритм градієнтного бустингу з оптимізацією обчислювальних витрат

Градієнтний бустинг – це метод машинного навчання, що належить до класу ансамблевих алгоритмів. Він полягає у послідовному побудуванні слабких моделей (часто дерев рішень), де кожна наступна модель коригує помилки попередніх. У результаті формується ансамбль моделей, який забезпечує високу точність прогнозів. Основна ідея методу полягає у мінімізації похибки на кожному етапі за допомогою градієнтного спуску, спрямованого на корекцію похибок попередніх моделей.

Градієнтний бустинг широко використовується в різних задачах машинного навчання, таких як класифікація, регресія, пошук аномалій, а також для прогнозування і рекомендаційних систем [25]. Його головна перевага – висока точність, але ця точність часто супроводжується високими обчислювальними витратами. Для оптимізації процесу навчання застосовуються такі техніки, як вибір оптимальних гіперпараметрів, зменшення розмірності вхідних даних, паралелізація обчислень, а також використання спеціальних бібліотек, наприклад, XGBoost, LightGBM або CatBoost, які значно прискорюють обчислення [26].

Алгоритм градієнтного бустингу базується на ідеї послідовного комбінування слабких моделей (базових алгоритмів), що спрямовані на мінімізацію похибок попередніх моделей. Його основна мета полягає в

побудові сильної моделі шляхом поступового додавання нових моделей, які коригують помилки попередників за допомогою градієнтного спуску.

Математична модель алгоритму градієнтного бустингу формалізується наступним чином: нехай дано набір даних $\{(x_i, y_i)\}_{i=1}^{n}$, де $x_i \in R^d$ – вектор ознак, а $y_i \in R$ – відповідні мітки (значення цільової змінної для регресії або клас для класифікації). Завдання полягає в тому, щоб знайти модель $F(x)$, яка мінімізує функцію втрат $L(y,F(x))$, що вимірює розбіжність між прогнозованим значенням і справжнім значенням.

Алгоритм градієнтного бустингу використовує ітеративний процес побудови ансамблю з M моделей $x_m(x)$, що називаються базовими моделями (зазвичай це дерева рішень), таким чином, що:

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

де $F_0(x)$ – початкова модель (наприклад, середнє значення цільової змінної), а γ_m – вага (шаг) кожної моделі $h_m(x)$, яка визначається шляхом мінімізації функції втрат на кожному етапі.

На кожній ітерації m, нова модель $h_m(x)$ навчається на залишках або градієнтах поточної моделі $F_{m-1}(x)$. Це виражається через градієнт функції втрат:

$$r_i^{(m)} = \frac{\partial L(y_i,F_{m-1}(x_i))}{\partial F_{m-1}(x_i)} \quad (2.2)$$

де $r_i^{(m)}$ – залишки, які представляють градієнти похибок поточної моделі. Наступна модель $h_m(x)$ тренується для мінімізації цих залишків.

Оптимізація обчислювальних витрат у градієнтному бустингу включає кілька важливих аспектів:

– паралелізація та пакетна обробка даних. У сучасних реалізаціях, таких як XGBoost або LightGBM, використовуються методи для розпаралелювання побудови дерев, що дозволяє суттєво прискорити обчислення на великих наборах даних;

– прорідження дерев (shrinkage). Це метод регуляризації, коли внесок кожного дерева γ_m зменшується, що дозволяє уникнути перенавчання і знизити обчислювальні витрати на подальших ітераціях.

– метод ранньої зупинки. Алгоритм може припиняти навчання, коли покращення функції втрат стає незначним, що дозволяє зменшити кількість ітерацій і, відповідно, обчислювальні витрати.

– субсемплінг (subsampling). Для кожної ітерації можна використовувати не весь набір даних, а лише його підмножину, що дозволяє скоротити час навчання.

Для персоналізації веб-додатків градієнтний бустинг може бути використаний для аналізу поведінки користувачів і створення індивідуальних рекомендацій. Наприклад, алгоритм може прогнозувати, які товари чи послуги будуть найбільш цікавими для кожного окремого користувача на основі його попередніх дій, пошукових запитів або взаємодії з контентом. Оптимізація обчислювальних витрат є ключовою для застосування цього методу в реальних системах, оскільки веб-додатки часто працюють із великими обсягами даних і потребують оперативної обробки інформації для забезпечення персоналізованого досвіду в режимі реального часу.

2.2 Алгоритм матричної факторизації

Алгоритм матричної факторизації – це метод розкладу матриці на дві або більше матриць, добуток яких дає наближення до вихідної матриці. Цей метод широко використовується в машинному навчанні, особливо в задачах рекомендаційних систем [27]. Основна ідея матричної факторизації полягає в тому, що складні дані, наприклад, взаємодія користувачів з продуктами чи контентом, можна подати у вигляді матриці, де кожен рядок відповідає користувачу, а кожен стовпчик – елементу, наприклад, товару чи фільму. Значення в матриці представляють взаємодію (наприклад, рейтинг) між користувачем і товаром.

Матрична факторизація розбиває цю матрицю на дві менші: матрицю користувачів і матрицю товарів. Мета алгоритму полягає в тому, щоб знайти такі фактори для користувачів і товарів, які, коли перемножуються, дають наближення до вихідної матриці. Це дозволяє прогнозувати відсутні значення в матриці, тобто рекомендації щодо нових товарів або контенту для конкретних користувачів на основі їх попередньої активності та уподобань.

Нехай $R \in R^{m \cdot n}$ – це матриця взаємодії користувачів і товарів, де R_{ij} представляє взаємодію (наприклад, рейтинг) користувача i з продуктом j. Завдання полягає в тому, щоб знайти дві матриці: матрицю користувачів $P \in R^{m \cdot k}$ та матрицю товарів $Q \in R^{k \cdot n}$, де k – кількість прихованих факторів, які відображають вподобання користувачів і характеристики товарів.

Алгоритм прагне мінімізувати різницю між вихідною матрицею RR і наближеною матрицею PQ^T:

$$min_{P,Q} \sum_{(i,j \in \Omega)} (R_{i,j} - P_i^T Q_i)^2 + \lambda(\| P_i \|^2 + \| Q_j \|^2), \quad (2.3)$$

де Ω – множина наявних взаємодій, а λ – регуляризаційний параметр, що допомагає уникнути перенавчання.

Матрична факторизація є основним методом у рекомендаційних системах, особливо в алгоритмах типу «спільної фільтрації» (Collaborative Filtering). Вона дозволяє робити персоналізовані рекомендації для користувачів на основі їх попередніх вподобань, навіть якщо деякі користувачі або товари мають мало взаємодій (що називається проблемою "рідких" даних).

Для персоналізації веб-додатків матричну факторизацію можна використовувати для створення індивідуальних рекомендацій користувачам. Наприклад, у системі інтернет-магазину матриця RR може містити інформацію про те, які продукти кожен користувач оцінював або купував. Матрична факторизація дозволяє передбачити товари, які можуть зацікавити користувача, навіть якщо він раніше не взаємодіяв з ними.

Цей підхід також можна використовувати для персоналізації контенту на стрімінгових платформах, новинних сайтах або соціальних мережах.

Алгоритм аналізує поведінку користувачів, наприклад, перегляди відео або читання статей, і на основі прихованих факторів рекомендує схожий або новий контент, який ймовірно зацікавить користувача.

Переваги алгоритму складають:

– ефективно працює з великими наборами даних і може генерувати рекомендації для користувачів навіть у разі, якщо деякі дані відсутні (наприклад, нові користувачі або товари);

– використання прихованих факторів дозволяє знайти глибші закономірності між користувачами та товарами, що покращує точність прогнозів.

Таким чином, матрична факторизація є потужним інструментом для персоналізації веб-додатків, що дозволяє автоматизувати процес рекомендацій та покращити користувацький досвід на основі попередніх взаємодій та вподобань.

2.3 Алгоритм швидкого дерева Твідді

Алгоритм швидкого дерева Твідді (ШДТ) є одним із варіантів ансамблевих методів, що базується на побудові дерев рішень для вирішення задач регресії. Цей алгоритм був спеціально розроблений для задач, у яких цільова змінна підпорядковується розподілу Твідді, що є гнучким класом статистичних розподілів, які об’єднують властивості як нормальних, так і пуассонівських розподілів [28]. Основна мета алгоритму полягає в прогнозуванні відповідної кількісної змінної з використанням дерев рішень, які здатні ефективно працювати з великими обсягами даних та різнорідними наборами ознак.

Нехай є вибірка даних $\{(x_i, y_i)\}_{i=1}^{n}$, де $x_i \in R^d$ – це вектори ознак, а $y_i \in R$ – цільова змінна. Алгоритм ШДТ моделює залежність між ознаками та цільовою змінною за допомогою послідовного додавання дерев рішень, кожне з яких коригує помилки попередніх дерев. Основна відмінність від класичних

методів регресії полягає в тому, що функція втрат у цьому алгоритмі налаштована на оптимізацію під розподіл Твідді:

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

де p – це параметр розподілу Твідді, а ϕ – додатковий масштабний параметр. Коли p=1, функція втрат перетворюється на логарифмічну функцію, що відповідає Пуассонівському розподілу, а при p=2 вона стає квадратичною, що відповідає нормальному розподілу.

Алгоритм ШДТ використовується здебільшого у задачах, де цільова змінна є кількісною та має властивості, що підпадають під розподіл Твідді. Це особливо корисно у фінансових прогнозах, страхуванні, коли потрібно моделювати ризики або збитки, а також у задачах з нецілими, розподіленими і нерівномірними даними.

Для персоналізації веб-додатків ШДТ може бути застосований у випадках, коли потрібно прогнозувати кількісні показники користувацької взаємодії, які мають нерівномірний розподіл. Наприклад, це може бути прогнозування кількості покупок або ймовірності здійснення покупки у певний період часу. Алгоритм може обробляти як постійні, так і змінні характеристики користувачів, що дозволяє моделювати їхню поведінку з урахуванням складних закономірностей.

Алгоритм також добре підходить для моделювання доходів, що генеруються від різних груп користувачів, або для прогнозування вартості замовлень у різних сегментах. Наприклад, можна прогнозувати кількість і суму покупок певного користувача або групи користувачів на основі попередніх дій, при цьому враховуючи різні фактори, як-от демографічні показники, попередні покупки, або взаємодію з веб-додатком.

Переваги алгоритму включають:

– висока точність у прогнозуванні нерівномірно розподілених кількісних змінних;

– ефективність на великих наборах даних завдяки оптимізації процесу побудови дерев;

– гнучкість у роботі з різними типами даних та можливість прогнозування показників з використанням різних розподілів.

Даний алгоритм поєднує точність дерев рішень з можливістю моделювати нерівномірно розподілені дані, характерні для багатьох реальних задач, особливо в персоналізації веб-додатків. Він дозволяє ефективно працювати з кількісними прогнозами і є цінним інструментом для аналізу користувацької поведінки та прогнозування доходів або покупок на основі складних залежностей між ознаками користувачів та їх діями.

2.4 Технологічний стек для програмної реалізації алгоритмів

Вибір мови програмування є важливим етапом у створенні системи, яка включає машинне навчання для персоналізації веб-додатків. Основними критеріями при виборі мови є простота інтеграції з фреймворками машинного навчання, продуктивність, зручність розробки, а також підтримка бібліотек для роботи з великими даними. Тому в процесі розробки варто розглянути найбільш популярні мови програмування, такі як Python, Java та C#, кожна з яких має свої унікальні особливості для виконання поставлених завдань.

Python є однією з найбільш популярних мов для машинного навчання завдяки своїй простоті, широкій підтримці бібліотек (наприклад, TensorFlow, Keras, Scikit-learn), та великій спільноті розробників [29]. Ця мова підходить для швидкої розробки прототипів, що особливо важливо у задачах персоналізації. Python також підтримує зручну інтеграцію з іншими мовами та веб-технологіями, що робить його ефективним для побудови персоналізованих веб-додатків.

Java є надійною мовою програмування, яка забезпечує високу продуктивність та можливість розробки кросплатформних рішень [30]. Мова має потужні інструменти для роботи з великими даними та вбудовані бібліотеки для машинного навчання, що робить її ефективною для систем, які

потребують масштабованості. Крім того, Java активно використовується у корпоративному середовищі, що спрощує інтеграцію персоналізованих рішень у вже існуючі системи.

C# є мовою програмування, розробленою компанією Microsoft, і широко використовується для створення веб-додатків та серверних систем [31]. Завдяки високій продуктивності та можливості інтеграції з .NET-бібліотеками, C# є потужним інструментом для реалізації персоналізованих функцій. Окрім того, підтримка C# серед інструментів машинного навчання постійно зростає, що робить його відповідним вибором для завдань, що вимагають високої продуктивності та інтеграції з Microsoft Azure для обробки великих даних.

Для розуміння переваг та недоліків різних мов програмування, які можуть бути використані для розробки системи персоналізації на основі машинного навчання, важливо порівняти їх за ключовими характеристиками (табл. 2.1).

Таблиця 2.1 – Порівняння мов програмування

Характеристика	Python	Java	C#
Продуктивність (швидкість виконання)	Середня	Висока	Висока
Простота інтеграції з ML фреймворками	Висока (TensorFlow, Scikit-learn)	Середня	Висока (.NET ML, 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 (через сторонні рішення)	Висока (JVM)	Висока (.NET, .NET Core)
Спільнота та підтримка	Дуже широка	Широка	Висока
Складність для початківців	Низька	Висока	Середня

Вибір мови програмування C# для розробки системи обумовлений її високою продуктивністю, що є критичним аспектом для обробки великих обсягів даних у режимі реального часу. C# пропонує зручні засоби інтеграції з фреймворками машинного навчання, такими як .NET ML та Accord.NET, що значно полегшує реалізацію алгоритмів персоналізації та забезпечує гнучкість у налаштуванні моделі. Завдяки підтримці широкого спектра бібліотек для роботи з великими даними, C# дозволяє ефективно маніпулювати інформацією і зберігати високу продуктивність навіть при значному навантаженні. Окрім того, багатоплатформність .NET Core надає можливість розгортання рішень на різних операційних системах, що є важливим фактором для забезпечення доступності веб-додатка. Високий рівень підтримки та стабільна спільнота користувачів C# сприяють зниженню ризиків, пов’язаних з розробкою, і забезпечують наявність надійної технічної бази для подальшого розвитку системи.

Для повного використання потенціалу мови програмування C# у розробці системи важливим є вибір відповідного середовища розробки, яке забезпечить зручність та ефективність роботи з кодом. Основним середовищем розробки є Microsoft Visual Studio, яке забезпечує потужні інструменти для написання, налагодження та тестування коду [32]. Крім того, це середовище містить вбудовані бібліотеки для роботи з алгоритмами машинного навчання, зокрема через ML.NET, що дозволяє легко інтегрувати моделі та оптимізувати їхню роботу.

Технологічний стек включає наступні компоненти:

– C# – основна мова програмування для реалізації алгоритмів;

– MS Visual Studio – середовище розробки для написання, тестування і відлагодження програмного коду [33];

– ML.NET – фреймворк для роботи з алгоритмами машинного навчання у середовищі .NET, що забезпечує можливість швидкої реалізації алгоритмів, таких як градієнтний бустинг, матрична факторизація, швидке дерево Твідді та інші [34, 35];

– .NET Framework – платформа, що забезпечує розробку надійних і масштабованих програм з підтримкою різних мов програмування та багатофункціональних бібліотек.

Використання Visual Studio та технологій .NET у поєднанні з мовою C# забезпечує гнучкість, високу продуктивність і легкість у реалізації та розгортанні алгоритмів машинного навчання, необхідних для виконання завдань у рамках проєкту.

2.5 Програмна реалізація алгоритмів

З огляду на специфіку застосованих класів, структура системи реалізована з використанням компонентів, які базуються на Windows Forms. Це дозволяє організувати архітектуру з фокусом на інтерактивному інтерфейсі, спрощуючи розробку користувацьких функцій та забезпечуючи зручне керування елементами додатка. На рис. 2.1 представлена розроблена діаграма, що демонструє ключові класи, їх властивості та методи, які забезпечують ефективну реалізацію алгоритмів машинного навчання.

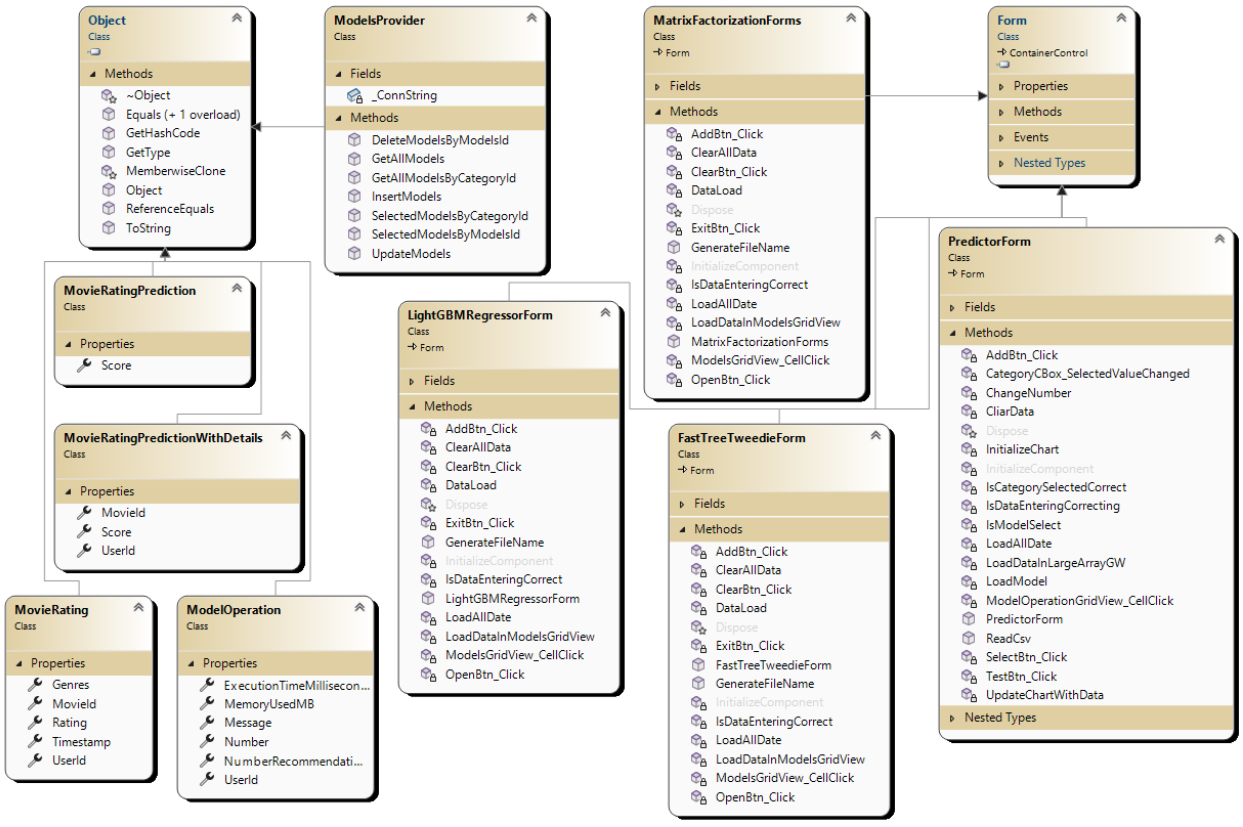

Рисунок 2.1 – Діаграма класів системи

Діаграма складається із наступних основних класів, що реалізують функціональність системи:

– PredictorForm – клас, призначений для перевірки моделей, який забезпечує зручний інтерфейс для тестування якості прогнозів різних алгоритмів. Він дозволяє аналізувати отримані результати та порівнювати точність прогнозування між різними моделями;

– FastTreeTweedieForm – клас, відповідальний за тренування моделей швидкого дерева Твідді, оптимізованих для обробки великих обсягів даних. Він реалізує налаштування параметрів моделі та відстеження процесу навчання;

– LightGBMRegressorForm – клас, що виконує тренування моделей градієнтного бустингу, орієнтованих на високу швидкість і точність. Інтерфейс класу дозволяє задавати параметри алгоритму та контролювати етапи навчання з можливістю перегляду метрик;

– MatrixFactorizationForms – клас для тренування моделей матричної факторизації, які використовуються для рекомендаційних систем. Він забезпечує налаштування параметрів факторизації та збір метрик для оцінки точності моделі;

– MovieRating – клас для підготовки даних, що відповідає за зберігання інформації про рейтинги фільмів, яка використовується під час навчання моделей. Він структуровано зберігає дані для подальшої обробки і прогнозування;

– MovieRatingPrediction – клас, що забезпечує функціональність прогнозування для моделі, повертаючи очікувані результати рейтингу. Він оптимізований для швидкої обробки і створений для генерації точних прогнозів на основі навченої моделі;

– MovieRatingPredictionWithDetails – клас, що виконує прогнозування з додатковими деталями. На відміну від базового класу прогнозування, він

надає розширену інформацію, включаючи метрики впевненості моделі, що дозволяє отримувати детальніші дані про результат;

– ModelsProvider – клас для зберігання, вибору та видалення моделей із бази даних. Він є основним інтерфейсом для управління моделями, що включає функції додавання нових моделей, отримання збережених моделей для аналізу та їх видалення при необхідності.

На початку було реалізовано класи для завантаження даних (MovieRating) та зберігання результатів прогнозу (MovieRatingPrediction). Код реалізації класів представлено на рис. 2.2.

```
20 references
public class MovieRating {
    [LoadColumn(0)]
    [Name("userId")]
    5 references
    public int UserId { get; set; }

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

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

    [LoadColumn(3)]
    [Name("timestamp")]
    0 references
    public long Timestamp { get; set; }

    [LoadColumn(4)]
    [Name("genres")]
    0 references
    public string Genres { get; set; }
}
```

```
2 references
public class MovieRatingPrediction {
    [ColumnName("Score")]
    1 reference
    public float Score { get; set; }
}
```

Рисунок 2.2 – Реалізація класів MovieRating та MovieRatingPrediction

На початку був реалізований алгоритм градієнтного бустингу з оптимізацією обчислювальних, у якому відбувається ініціалізація контексту машинного навчання за допомогою класу MLContext, який є основою для виконання всіх операцій у бібліотеці ML.NET (рис. 2.3).

```
mlContext = new MLContext(seed: 0);
```

Рисунок 2.3 – Створення контексту машинного навчання

Контекст ML створює об’єкт, що відповідає за керування потоками даних, тренування моделей та їх оцінювання. Це головна точка входу для всіх функцій машинного навчання. Аргумент seed: 0 визначає фіксоване значення для генератора випадкових чисел, що забезпечує відтворюваність результатів під час виконання експериментів з одними й тими ж даними та гіперпараметрами.

На рис. 2.4 представлено фрагмент коду, у якому здійснюється завантаження даних із текстового файлу за допомогою методу LoadFromTextFile, який є частиною контексту машинного навчання MLContext. Завантажені дані зберігаються у змінну dataView у форматі, що підтримується бібліотекою ML.NET для подальшої обробки.

```
dataView = mlContext.Data.LoadFromTextFile<MovieRating>(_Path,
    hasHeader: true,
    separatorChar: ',');
```

Рисунок 2.4 – Завантаження даних

Фрагмент коду на рис. 2.5 відповідає за поділ наявних даних на тренувальний і тестовий набори. Використовується метод TrainTestSplit із контексту ML.NET для розділення даних, де параметр testFraction: 0.2 вказує, що 20% даних буде виділено для тестування, а решта 80% – для тренування моделі.

```
var trainTestData = mlContext.Data.TrainTestSplit(dataView, testFraction: 0.2);
var trainData = trainTestData.TrainSet;
var testData = trainTestData.TestSet;
```

Рисунок 2.5 – Розділення даних на тренувальний та тестовий набори

Отримані після розділення дані зберігаються у змінній trainTestData, яка містить два окремі набори: тренувальний і тестовий. Тренувальний набір присвоюється змінній trainData, а тестовий – змінній testData. Це розділення дозволяє забезпечити коректну оцінку моделі після тренування, оскільки тестові дані не використовуються під час навчання, що дозволяє оцінити здатність моделі до узагальнення на нових, невідомих даних. Такий підхід є стандартним кроком у процесі машинного навчання для запобігання перенавчанню та отримання реалістичної оцінки продуктивності моделі.

Після цього створюється конвеєр обробки даних і тренування моделі на основі алгоритму градієнтного бустингу (рис. 2.6). Спочатку здійснюється трансформація даних за допомогою методу MapValueToKey, який перетворює значення колонок UserId і MovieId на категоріальні змінні, що зберігаються у нових колонках UserIdEncoded і MovieIdEncoded. Після цього відбувається зворотне перетворення закодованих значень у числові типи за допомогою MapKeyToValue, зберігаючи ці значення у нові колонки UserIdFloat і MovieIdFloat.

```
var pipeline = mlContext.Transforms.Conversion.MapValueToKey(outputColumnName: "UserIdEncoded", inputColumnName: "UserId")
    .Append(mlContext.Transforms.Conversion.MapValueToKey(outputColumnName: "MovieIdEncoded", inputColumnName: "MovieId"))
    .Append(mlContext.Transforms.Conversion.MapKeyToValue("UserIdFloat", "UserIdEncoded"))
    .Append(mlContext.Transforms.Conversion.MapKeyToValue("MovieIdFloat", "MovieIdEncoded"))
    .Append(mlContext.Transforms.Conversion.ConvertType("UserIdFloat", outputKind: DataKind.Single))
    .Append(mlContext.Transforms.Conversion.ConvertType("MovieIdFloat", outputKind: DataKind.Single))
    .Append(mlContext.Transforms.Concatenate("Features", "UserIdFloat", "MovieIdFloat"))
    // Додаємо LightGBM Regressor
    .Append(mlContext.Regression.Trainers.LightGbm(labelColumnName: "Rating", featureColumnName: "Features"));
```

Рисунок 2.6 – Створення конвеєру обробки даних

Застосовується метод ConvertType, який перетворює ці числові значення у формат Single (одинарної точності), що є необхідним для використання в машинному навчанні. Потім ці значення об'єднуються в одну ознакову матрицю за допомогою методу Concatenate, створюючи стовпець Features, що включає перетворені дані про користувачів та фільми. Завершальним етапом конвеєра є додавання LightGBM-регресора, який використовуватиме колонку Rating як мітку для прогнозування, а колонку Features – як набір ознак для тренування моделі. Таким чином, цей конвеєр дозволяє пройти всі необхідні етапи перетворення даних і тренування моделі в рамках однієї структури, що значно спрощує процес розробки й автоматизує обробку даних перед навчанням моделі.

На рис. 2.7 представлено код, у якому відбувається процес навчання моделі на основі тренувального набору даних. Метод Fit викликається для конвеєра обробки даних pipeline і застосовується до набору trainData. У результаті створюється модель, яка зберігається в змінній model.

```
model = pipeline.Fit(trainData);
```

Рисунок 2.7 – Навчання моделі на тренувальному наборі

Конвеєр, який був налаштований раніше, містить послідовність операцій перетворення даних і алгоритм навчання, які виконуються під час виклику методу Fit. Це означає, що всі етапи, включаючи кодування змінних, об'єднання ознак і застосування алгоритму машинного навчання, реалізуються на тренувальних даних. Таким чином, модель адаптується до специфіки тренувального набору і набуває здатності робити прогнози на основі патернів, виявлених у даних.

На наступному етапі відбувається процес оцінки продуктивності моделі на тестовому наборі даних (рис. 2.8). Спочатку, за допомогою методу Transform, модель застосовується до тестових даних, створюючи прогнозовані значення, які зберігаються у змінній predictions. Цей крок дозволяє моделі виконати передбачення на основі патернів, виявлених під час навчання, але вже для даних, які не використовувались під час тренування.

```
var predictions = model.Transform(testData);
var metrics = mlContext.Regression.Evaluate(predictions, labelColumnName: "Rating");
```

Рисунок 2.8 – Оцінка моделі на тестовому наборі

Після цього за допомогою методу Evaluate, що входить до бібліотеки ML.NET, проводиться оцінка точності отриманих прогнозів. Оцінка здійснюється на основі порівняння передбачених і реальних значень, зокрема для колонки "Rating", яка вказана як мітка. Результати оцінки, зокрема такі метрики як середньоквадратична похибка, середня абсолютна похибка та коефіцієнт детермінації, зберігаються у змінній metrics. Ці метрики допомагають оцінити якість роботи моделі та її здатність до узагальнення на нових даних, що є ключовим аспектом ефективності в машинному навчанні.

Рис. 2.9 висвітлює фрагмент коду, що відповідає за відображення результатів оцінки моделі LightGBM на тестовому наборі даних у текстовому полі RaportTBox. Спочатку додається текстове повідомлення, що повідомляє про початок оцінки моделі. Потім виводяться ключові метрики, що характеризують якість моделі.

```
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("mm\\:ss\\.fff"):F2} секунд\r\n");
```

Рисунок 2.9 – Виведення метрик

Значення метрики RMSE (кореневої середньоквадратичної похибки) форматоване до двох знаків після коми і показує, наскільки модель наблизилася до фактичних значень. Далі виводиться MAE (середня абсолютна похибка), яка також форматована до двох знаків, і характеризує середню різницю між прогнозами моделі та реальними значеннями. Після цього відображається коефіцієнт детермінації R^2, що показує, наскільки добре модель пояснює варіативність даних.

Насамкінець, додається час, витрачений на оцінку моделі, який був виміряний таймером evaluationTimer. Він форматований у хвилини, секунди та мілісекунди, що дозволяє точно оцінити продуктивність моделі у тестовому середовищі. Усе це виводиться в текстове поле, щоб користувач міг переглянути і проаналізувати результат оцінки моделі в зручному форматі.

На рис. 2.10 представлено метод, у якому відбувається обробка події натискання кнопки «AddBtn», що відповідає за збереження навченої моделі та оновлення інформації про неї. Спочатку перевіряється коректність введених даних за допомогою методу «IsDataEnteringCorrect». Якщо дані введені правильно, формується шлях для збереження моделі, який базується на автоматично згенерованому імені файлу та додається до шляху папки «\teach\».

```
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!");
  }
}
```

Рисунок 2.10 – Зберігання даних моделі

Шлях до поточного проєкту отримується через клас «System.Reflection.Assembly», що дозволяє зберегти модель у локальну директорію проєкту. Інформація про модель (її назва, категорія та шлях до файлу) зберігається в базі даних за допомогою методу «InsertModels» від _ModelsProvider. Потім модель зберігається на диск за допомогою методу «mlContext.Model.Save», де вказується сама модель, її схема даних та шлях для збереження. Після цього всі поля форми очищуються за допомогою методу «ClearAllData», і в логах фіксується факт навчання моделі з інформацією про поточного користувача та час виконання дії, що робиться через метод «InsertLogs» від «_LogsProvider». В кінці користувачу відображається повідомлення про успішне збереження даних через «MessageBox.Show», що підтверджує завершення операції.

Навчання моделей у ML.NET забезпечує широкий вибір алгоритмів машинного навчання, які можна використовувати для вирішення різноманітних завдань, зокрема побудови рекомендаційних систем. Однією з ключових переваг цієї платформи є гнучкість у створенні універсальних конвеєрів для обробки даних та тренування моделей. У межах проєкту були реалізовані два різних алгоритми – матрична факторизація та швидке дерево Твідді, які використовуються для рекомендацій. Хоча ці алгоритми відрізняються у підготовці даних і налаштуванні конвеєра, вся інша частина процесу – збереження моделей, оцінка точності та інтеграція у

рекомендаційний двигун – була реалізована універсально. Цей підхід дозволяє використовувати єдиний механізм для тренування різних моделей, забезпечуючи гнучкість у зміні алгоритмів без потреби переписувати базову інфраструктуру.

Алгоритм матричної факторизації потребує спеціальної підготовки конвеєра для обробки даних, щоб побудувати рекомендації на основі співвідношення між користувачами та об'єктами. Конвеєр цього алгоритму передбачає перетворення ідентифікаторів користувачів і товарів у числові значення, що використовуються як вхідні ознаки для моделі. На рис. 2.11 представлено код конвеєра для алгоритму матричної факторизації, який виконує ключові етапи підготовки даних та навчання моделі.

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

Рисунок 2.11 – Створення конвеєра для алгоритму матричної факторизації

У цьому фрагменті коду налаштовується конвеєр для реалізації алгоритму матричної факторизації. Спершу використовується перетворення MapValueToKey, яке перетворює значення в колонках UserId та MovieId у числові ключі. Це дозволяє підготувати дані для матричної факторизації, оскільки алгоритм потребує числового представлення категоріальних даних для користувачів та об'єктів (наприклад, фільмів).

Після цього додається основний компонент – тренер для матричної факторизації. Він здійснює навчання моделі на основі співвідношення між користувачами і фільмами, використовуючи відповідні мітки (рейтинги) з колонки Rating. Для тренування алгоритму матричної факторизації вказується, що закодовані значення ідентифікаторів користувачів будуть використовуватись як індекси стовпців матриці, а закодовані значення фільмів

– як індекси рядків. Цей конвеєр завершує побудову та дозволяє навчити модель, що буде прогнозувати рейтинги для користувачів на основі їхньої взаємодії з різними об'єктами.

Також, на рис. 2.12 представлено код, у якому налаштовується конвеєр для обробки даних та тренування моделі на основі алгоритму ШДТ. Спершу виконується перетворення даних за допомогою методу MapValueToKey, який переводить категоріальні значення для колонок UserId і MovieId у числові ключі, що необхідні для подальшого навчання моделі. Ці закодовані значення потім зворотно перетворюються у числові формати за допомогою MapKeyToValue, а далі через метод ConvertType конвертуються у тип даних Single для числової обробки.

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

Рисунок 2.12 – Створення конвеєра для швидкого дерева Твідді

Наступним кроком є об’єднання перетворених даних у новий стовпець Features, який містить значення з двох змінних – UserIdFloat і MovieIdFloat. Цей стовпець використовується як набір ознак для тренування моделі. Завершальним етапом є додавання алгоритму FastTreeTweedie, який тренується на основі ознак у колонці Features і використовує рейтинг фільмів як мітку для навчання, яка міститься в колонці Rating. Модель, яку створює цей конвеєр, використовує алгоритм регресії на основі дерев рішень для прогнозування числових значень (у цьому випадку – рейтингу фільмів), що дає змогу отримати точні прогнози для рекомендаційної системи.

У методі «LoadAllDate», який викликається у конструкторі форми для тестування моделей, завантажуються всі необхідні дані для подальшого використання під час прогнозування (рис. 2.13). Спочатку зчитується унікальний набір фільмів за допомогою методу ReadCsv, який завантажує дані

з файлу data.csv і зберігає їх у змінну _AllUniqueMovies. Цей набір фільмів використовується як тестові дані, на яких навчені моделі будуть робити свої прогнози.

```
private void LoadAllDate() {
  _AllUniqueMovies = ReadCsv("data.csv");
  _CategoryList = _CategoryProvider.GetCategoryList();
  CategoryCBox.DataSource = _CategoryList;
  CategoryCBox.ValueMember = "CategoryId";
  CategoryCBox.DisplayMember = "CategoryName";
  _IsCategoryLoad = true;
  CategoryCBox_SelectedValueChanged(CategoryCBox, EventArgs.Empty);
}
```

Рисунок 2.13 – Завантаження даних

Метод завантажує всі категорії алгоритмів, збережені в базі даних, за допомогою методу «GetCategoryList» від провайдера _CategoryProvider. Ці категорії представляють різні групи алгоритмів, що використовуються для навчання моделей. Після отримання списку категорій він встановлюється як джерело даних для комбобоксу CategoryCBox, що дозволяє користувачеві вибирати необхідну категорію для тестування.

Для відображення вибраної категорії у комбобоксі встановлюються поля ValueMember та DisplayMember, які відповідають за ідентифікатор і назву категорії відповідно. Після цього змінна _IsCategoryLoad приймає значення true, що означає завершення завантаження категорій. На завершення метод викликає подію «CategoryCBox_SelectedValueChanged», що дозволяє негайно обробити вибір категорії після її завантаження.

Для завантаження моделей реалізовано метод «LoadModel» у якому відбувається процес завантаження раніше навченої моделі для її подальшого використання в прогнозуванні (рис. 2.14). Спершу формується повний шлях до файлу моделі шляхом додавання шляху до проєкту через Application.StartupPath та переданого параметра FilePath.

```
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<MovieRating, MovieRatingPrediction>(model);
}
```

Рисунок 2.14 – Завантаження моделі

Визначається об'єкт DataViewSchema, який буде використовуватися для отримання схеми даних, що була використана під час тренування моделі. Після цього модель завантажується з диска за допомогою методу Load, що належить контексту _Context.Model, і отримується схема моделі.

Після завантаження модель готова до використання для прогнозування. Для цього створюється об'єкт predictionEngine, який відповідає за виконання прогнозів. Його створюють за допомогою методу CreatePredictionEngine, що ініціалізує зв'язок між моделлю та даними. У даному випадку модель працює з класами MovieRating і MovieRatingPrediction, де зберігаються вхідні дані та результати прогнозування відповідно. Завдяки цьому механізму, модель може бути використана для прогнозування на основі нових даних.

На рис. 2.15 представлено метод «TestBtn_Click», який виконується при натисканні кнопки, відбувається тестування моделі шляхом вимірювання її продуктивності та використання ресурсів. На початку викликається метод InitializeChart, який відповідає за ініціалізацію графіка для подальшого виведення результатів тестування.

```
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
    // Генеруємо прогнози на основі UserId та кількості рекомендацій
    var recommendations = new List<Tuple<int, float>>(); // Зберігаємо MovieId та оцінку
    // Використовуємо унікальні фільми зі списку _AllUniqueMovies
    for (int i = 0; i < modelOperation.NumberRecommendations && i < _AllUniqueMovies.Count; i++) {
      var input = new MovieRating {
        UserId = modelOperation.UserId,
        MovieId = _AllUniqueMovies[i].MovieId  // Використовуємо MovieId з _AllUniqueMovies
      };
      var prediction = predictionEngine.Predict(input);
      recommendations.Add(new Tuple<int, float>(input.MovieId, prediction.Score));
    }
    // Вимірюємо час виконання
    stopwatch.Stop();
    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
```

Рисунок 2.15 – Фрагмент коду методу «TestBtn_Click»

У методі створюється об'єкт Stopwatch для вимірювання часу виконання операцій, а також отримується початковий обсяг використаної пам'яті за допомогою методу GC.GetTotalMemory, що дозволяє порівнювати пам'ять до і після виконання моделі. В циклі перебираються всі операції з моделями, що зберігаються в колекції _AllModelOperation. Для кожної операції запускається таймер і отримується обсяг використаної пам'яті до початку виконання прогнозів. Потім створюється список recommendations, в якому зберігаються результати прогнозів для кожного фільму, включаючи ідентифікатор фільму (MovieId) та прогнозований рейтинг (Score).

В процесі генерації рекомендацій модель отримує унікальні фільми зі списку _AllUniqueMovies та виконує передбачення для кожного фільму на основі UserId і відповідного MovieId. Прогнози зберігаються у список у вигляді пари фільму і рейтингу. Після завершення генерації рекомендацій таймер зупиняється, і результат вимірювання часу виконання зберігається в об'єкті modelOperation.

Також вимірюється кількість пам'яті, використаної моделлю під час виконання. Віднімається початковий обсяг пам'яті від обсягу після виконання,

і різниця конвертується у мегабайти для зручнішого відображення. Таким чином, для кожної операції тестування моделі отримуються точні дані про час виконання і використану пам'ять, що дозволяє оцінити продуктивність моделі під час тестування.

У даному підрозділі частково описано основні моменти реалізації коду, включаючи підготовку даних, тренування моделей, а також оцінку їхньої продуктивності. Було розглянуто ключові аспекти створення та використання конвеєрів для алгоритмів градієнтного бустигну, матричної факторизації та швидкого дерева Твідді. Крім того, було описано процес тестування моделей, зокрема вимірювання часу виконання та використання пам'яті, що дозволяє оцінити ефективність впроваджених рішень у системі рекомендацій.

3 ПОРІВНЯННЯ АЛГОРИТМІВ ТА РЕКОМЕНДАЦІЇ ЩОДО ЇХ ВИКОРИСТАННЯ

3.1 Порівняння розглянутих алгоритмів за основними характеристиками

У межах цього підрозділу буде розглянуто швидкість навчання моделей у різних умовах, яка є важливим фактором для масштабування системи та забезпечення високої продуктивності. Також буде оцінено точність рекомендацій кожного алгоритму залежно від характеристик вхідних даних, оскільки різні підходи можуть по-різному реагувати на особливості інформації, що обробляється. Окрім того, особливу увагу приділено вимогам до ресурсів, які визначають можливість застосування алгоритмів на реальних даних із врахуванням обмежень щодо обчислювальних потужностей та пам'яті.

3.1.1. Швидкість навчання моделей у різних умовах

Швидкість навчання моделей охоплює дослідження часу тренування основних алгоритмів у залежності від обсягів даних, що є важливим критерієм ефективності системи. Для забезпечення точних результатів було проведено навчання моделей з різною кількістю записів: 100 000, 1 000 000 та 10 000 000. Такий підхід дозволив оцінити, як кожен із алгоритмів справляється з навантаженням при збільшенні даних і наскільки швидкість тренування корелює з ефективністю роботи системи загалом. Результати цих досліджень особливо цінні для персоналізованих систем, які повинні обробляти великі обсяги інформації у стислі терміни.

Табл. 3.1 ілюструє результати навчання для трьох алгоритмів: градієнтного бустингу, матричної факторизації та швидкого дерева Твідді, виконаного на наборі даних із 1035 унікальними користувачами та 10 395 фільмами. Цей експеримент проводився на 100 000 записів, що забезпечує

репрезентативність для подальшого порівняння швидкості та продуктивності алгоритмів у подібних умовах.

Таблиця 3.1 – Результати навчання моделі

Характеристика	Градієнтний бустинг	Матрична факторизація	Швидке дерево Твідді
К-сть унікальних користувачів	1035	1035	1035
К-сть унікальних фільмів	10395	10395	10395
Кількість записів	100000	100000	100000
Час тренування моделі (секунд)	00:00.633	00:00.151	00:00.812

На основі даних у табл. 3.1 можна зробити кілька важливих висновків щодо ефективності кожного алгоритму. Матрична факторизація демонструє найшвидше навчання, завершуючи тренування за 0.151 секунди, що є значною перевагою для систем, які потребують оперативного аналізу великих масивів даних. Така швидкість робить алгоритм оптимальним для завдань з обмеженим часом, що забезпечує високу продуктивність при мінімальних витратах часу. Градієнтний бустинг, хоча й поступається швидкістю Матричній факторизації, демонструє стійку продуктивність, забезпечуючи баланс між часом тренування і точністю результатів. Швидке дерево Твідді, попри дещо повільніше навчання, підходить для випадків, коли важлива не лише швидкість, але й адаптивність алгоритму до специфіки даних.

Збільшення обсягу даних до 1 000 000 записів дозволяє оцінити, як алгоритми справляються з навчанням за умов істотного зростання навантаження (табл. 3.2).

Таблиця 3.2 – Результати навчання моделі

Характеристика	Градієнтний бустинг	Матрична факторизація	Швидке дерево Твідді
К-сть унікальних користувачів	9561	9561	9561
К-сть унікальних фільмів	25825	25825	25825
Кількість записів	1000000	1000000	1000000
Час тренування моделі (секунд)	00:03.849	00:01.490	00:07.452

Аналіз даних таблиці 3.2 показує, що при збільшенні обсягу даних до 1 000 000 записів Матрична факторизація знову продемонструвала найкоротший час тренування – 1.490 секунди, що значно перевершує Градієнтний бустинг та Швидке дерево Твідді. Це свідчить про високу стійкість Матричної факторизації до зростання обсягів даних, що робить її ефективним вибором для масштабованих систем, де швидкість обробки інформації є критично важливою.

Градієнтний бустинг, який виконав тренування за 3.849 секунди, зберігає свої переваги щодо ефективного співвідношення часу навчання та якості прогнозування, хоча його швидкість вже значно поступається Матричній факторизації при таких обсягах даних. Швидке дерево Твідді показало найбільшу тривалість навчання – 7.452 секунди, що свідчить про значне збільшення навантаження при обробці великого набору записів. Цей результат вказує на те, що Швидке дерево Твідді, хоч і є потужним інструментом для аналізу даних, менш підходить для задач, що потребують негайного масштабування, оскільки зі збільшенням обсягу даних час його тренування значно зростає.

Збільшення обсягу даних до 10 000 000 записів дозволяє глибше проаналізувати продуктивність алгоритмів за умов максимальної завантаженості, характерної для великих систем персоналізації (табл. 3.3).

Таблиця 3.3 – Результати навчання моделі

Характеристика	Градієнтний бустинг	Матрична факторизація	Швидке дерево Твідді
К-сть унікальних користувачів	98378	98378	98378
К-сть унікальних фільмів	58136	58136	58136
Кількість записів	10000000	10000000	10000000
Час тренування моделі (секунд)	00:48.898	00:14.787	01:23.773

Результати таблиці 3.3 показують, що при обробці 10 000 000 записів Матрична факторизація продемонструвала найвищу ефективність, завершивши тренування за 14.787 секунд. Це підкреслює її придатність для

масштабованих систем, що потребують швидкої персоналізації. Градієнтний бустинг потребував 48.898 секунд, що хоч і довше, проте є прийнятним компромісом між швидкістю та точністю прогнозування. Швидке дерево Твідді виявилося найповільнішим, завершивши тренування за 83.773 секунди, що обмежує його використання у системах, де швидке оновлення моделей є критичним фактором.

Щоб краще проілюструвати залежність часу тренування моделей від обсягу даних, було побудовано графік, представлений на рис. 3.1. На ньому можна побачити, як змінюється час навчання алгоритмів зі збільшенням кількості записів у датасеті. Цей графік допомагає наочно оцінити, які алгоритми є більш ефективними за умов обробки різних обсягів даних.

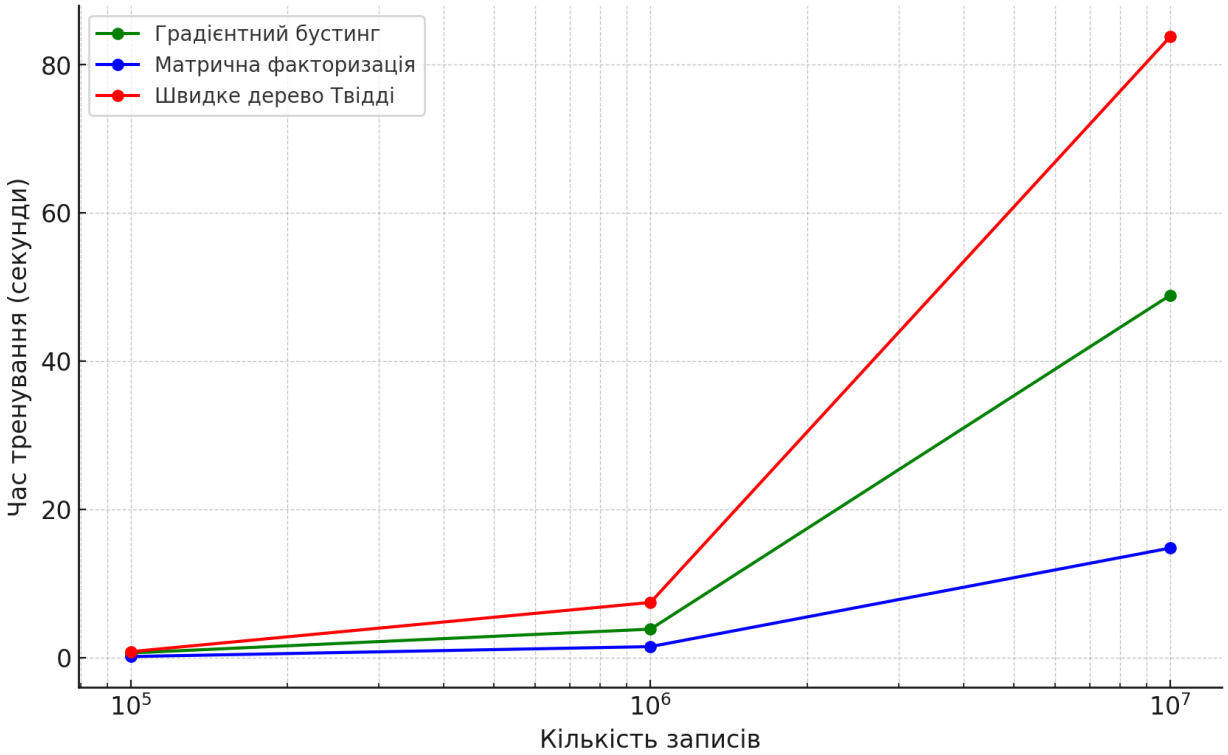

Рисунок 3.1 – Графіки залежності часу навчання до кількості даних

Як видно з графіку, матрична факторизація продемонструвала найменшу залежність часу тренування від обсягу даних, що підтверджує її високу масштабованість і ефективність при збільшенні записів. У той час як градієнтний бустинг показує прийнятну швидкість для більшості сценаріїв, швидке дерево Твідді суттєво збільшує час тренування при великих обсягах

даних, що обмежує його використання в умовах, де швидкодія є критичним параметром.

Приклади результатів навчання моделей наведено на рисунках у додатку А, що дозволяє більш детально ознайомитися з поведінкою кожного алгоритму та оцінити точність їх прогнозів у різних умовах.

3.1.2. Вплив обсягу вхідних даних на швидкість і ресурсозалежність рекомендацій

У даному підрозділі представлено дослідження впливу обсягу та специфіки вхідних даних на якість та ефективність рекомендацій, сформованих за допомогою натренованих моделей. Щоб дослідити, як змінюється точність і швидкість надання рекомендацій, було проведено серію експериментів із різними обсягами вхідних даних: 10, 100 та 1000 записів. Такий підхід дозволяє проаналізувати, як кожен із алгоритмів адаптується до зростання розміру набору даних і як це впливає на швидкодію системи.

Результати надання рекомендацій, представлені в табл. 3.4, включають характеристики, такі як ідентифікатор користувача, кількість рекомендацій, виділена пам’ять і час генерації рекомендацій. Ці дані дозволяють порівняти ефективність алгоритмів градієнтного бустингу, матричної факторизації та швидкого дерева Твідді при однаковій кількості рекомендацій для кожного користувача, що є важливим фактором для оцінки реальної продуктивності системи в умовах підвищеного навантаження.

Таблиця 3.4 – Результати надання рекомендацій

Характеристика	Градієнтний бустинг	Матрична факторизація	Швидке дерево Твідді
Ідентифікатор користувача	150	150	150
К-сть рекомендацій	10	10	10
Кількість виділеної пам’яті	0,01	0,02	0,03
Час генерації рекомендацій (мс.)	7,41	11,81	9,97

На основі даних у таблиці 3.4 можна зробити кілька висновків. Градієнтний бустинг показав найшвидший час генерації рекомендацій – 7,41 мс, що свідчить про його ефективність при обробці невеликих обсягів даних. Матрична факторизація, хоч і витратила більше часу (11,81 мс), продемонструвала стабільність у роботі та збалансоване використання пам’яті. Швидке дерево Твідді знаходиться між цими двома алгоритмами за часом виконання (9,97 мс) і обсягом споживаної пам'яті, що робить його прийнятним вибором у випадках, коли потрібен компроміс між швидкістю та ресурсами.

При збільшенні кількості рекомендацій до 100 можна оцінити, як алгоритми справляються з більшим обсягом роботи та як це впливає на витрати пам'яті та час генерації. У табл. 3.5 наведено результати надання рекомендацій із зазначенням виділеної пам’яті та часу генерації рекомендацій.

Таблиця 3.5 – Результати надання рекомендацій

Характеристика	Градієнтний бустинг	Матрична факторизація	Швидке дерево Твідді
Ідентифікатор користувача	150	150	150
К-сть рекомендацій	100	100	100
Кількість виділеної пам’яті	0,03	0,03	0,04
Час генерації рекомендацій (мс.)	16	6,51	7,94

Як показано в табл. 3.5, Матрична факторизація знову продемонструвала найменший час генерації рекомендацій – 6,51 мс, що підкреслює її ефективність при роботі з великим числом рекомендацій для одного користувача. Ця швидкість надає їй перевагу в сценаріях, де потрібна швидка обробка запитів із великим обсягом даних, зберігаючи при цьому низький рівень використання пам’яті (0,03 Мб).

Градієнтний бустинг потребував 16 мс для генерації 100 рекомендацій, що суттєво більше у порівнянні з іншими алгоритмами, хоча йому вдалося зберегти аналогічний обсяг виділеної пам’яті – 0,03 Мб. Це вказує на те, що Градієнтний бустинг може виявитися менш оптимальним вибором для

завдань, де швидкість є критичною, але він може компенсувати це за рахунок якості рекомендацій.

Швидке дерево Твідді продемонструвало баланс між двома іншими алгоритмами з часом генерації 7,94 мс і використанням пам’яті 0,04 Мб. Це свідчить про те, що Швидке дерево Твідді є гнучким варіантом для випадків, де необхідна швидкість вища, ніж у Градієнтного бустингу, але без надмірних витрат на ресурси, що робить його придатним для систем із середніми вимогами до продуктивності.

При подальшому збільшенні кількості рекомендацій до 1000 можна спостерігати, як алгоритми витримують значне навантаження та чи здатні вони забезпечувати прийнятний рівень продуктивності (табл. 3.6).

Таблиця 3.6 – Результати надання рекомендацій

Характеристика	Градієнтний бустинг	Матрична факторизація	Швидке дерево Твідді
Ідентифікатор користувача	150	150	150
К-сть рекомендацій	1000	1000	1000
Кількість виділеної пам’яті	0,13	0,12	0,13
Час генерації рекомендацій (мс.)	21,9	15,36	19,89

З даних у таблиці 3.6 видно, що Матрична факторизація зберігає лідерство за швидкістю генерації рекомендацій, демонструючи найкоротший час – 15,36 мс при мінімальному використанні пам'яті, лише 0,12 Мб. Цей результат свідчить про високий рівень оптимізації алгоритму, що дозволяє швидко адаптуватися до великих обсягів оброблюваних даних, не збільшуючи значно вимог до ресурсів. Така ефективність робить Матричну факторизацію оптимальним вибором для завдань із великими навантаженнями, де швидкість надання рекомендацій є критичною.

Градієнтний бустинг потребував більше часу для генерації 1000 рекомендацій – 21,9 мс, що є найбільшим показником серед трьох алгоритмів. Хоча цей алгоритм використовував лише 0,13 Мб пам'яті, його повільніша обробка вказує на меншу оптимізацію для таких великих обсягів запитів, що

може обмежувати його застосування в умовах, де потрібна миттєва реакція на запити.

Швидке дерево Твідді продемонструвало середній час виконання, становлячи 19,89 мс із використанням 0,13 Мб пам'яті. Цей результат свідчить про те, що Швидке дерево Твідді може забезпечити прийнятний рівень продуктивності та помірне використання ресурсів при значному обсязі запитів. Таким чином, цей алгоритм залишається гнучким вибором, особливо в умовах, де необхідний компроміс між швидкістю та споживанням ресурсів.

Для кращого розуміння впливу кількості рекомендацій на швидкодію алгоритмів було побудовано графік, представлений на рис. 3.2.

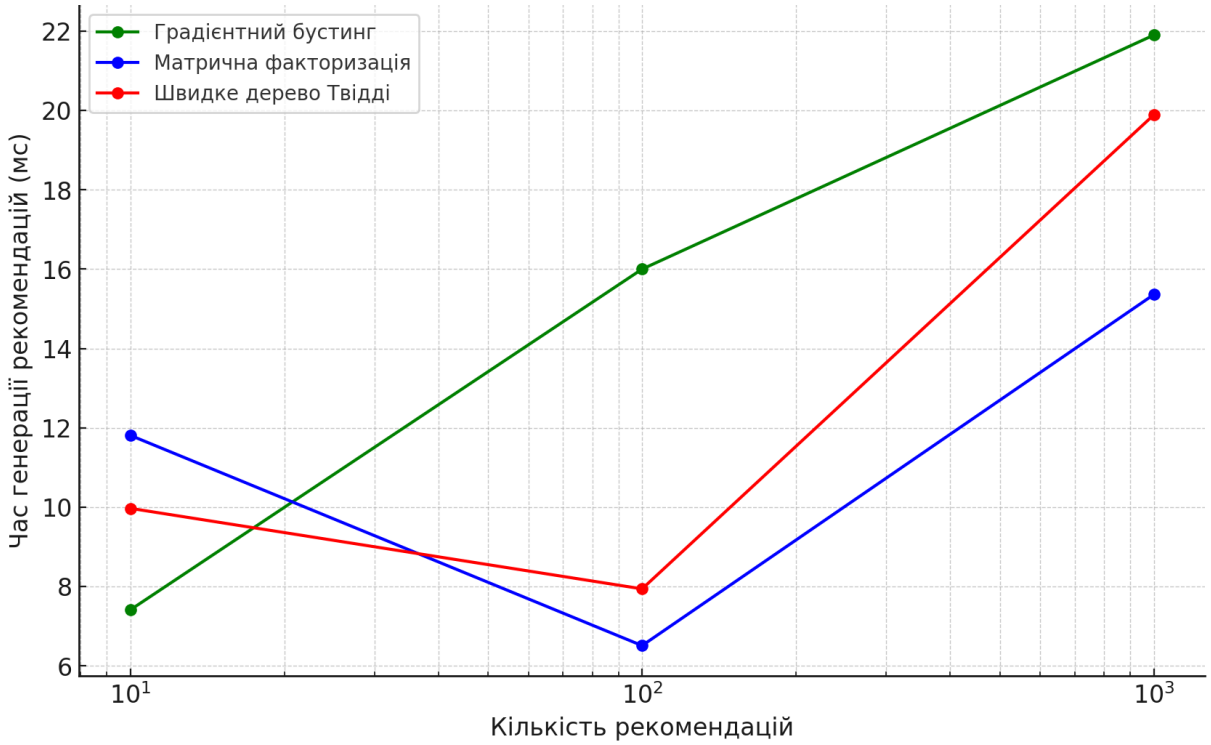

Рисунок 3.2 – Графіки залежності часу генерації рекомендацій до кількості рекомендацій для користувача

На рис. 3.2 видно, що Матрична факторизація демонструє найменший час генерації рекомендацій при збільшенні кількості запитів, що підтверджує її високу продуктивність та здатність обробляти великі обсяги даних із мінімальними затримками. Градієнтний бустинг показує помітне зростання часу при збільшенні кількості рекомендацій, що вказує на його більшу ресурсозалежність. Швидке дерево Твідді демонструє збалансовані

результати, залишаючись прийнятним варіантом для завдань із середніми вимогами до швидкодії та продуктивності.

На рис. 3.3 представлено графік, що показує залежність обсягу виділеної пам'яті від кількості запитів для Градієнтного бустингу, Матричної факторизації та Швидкого дерева Твідді.

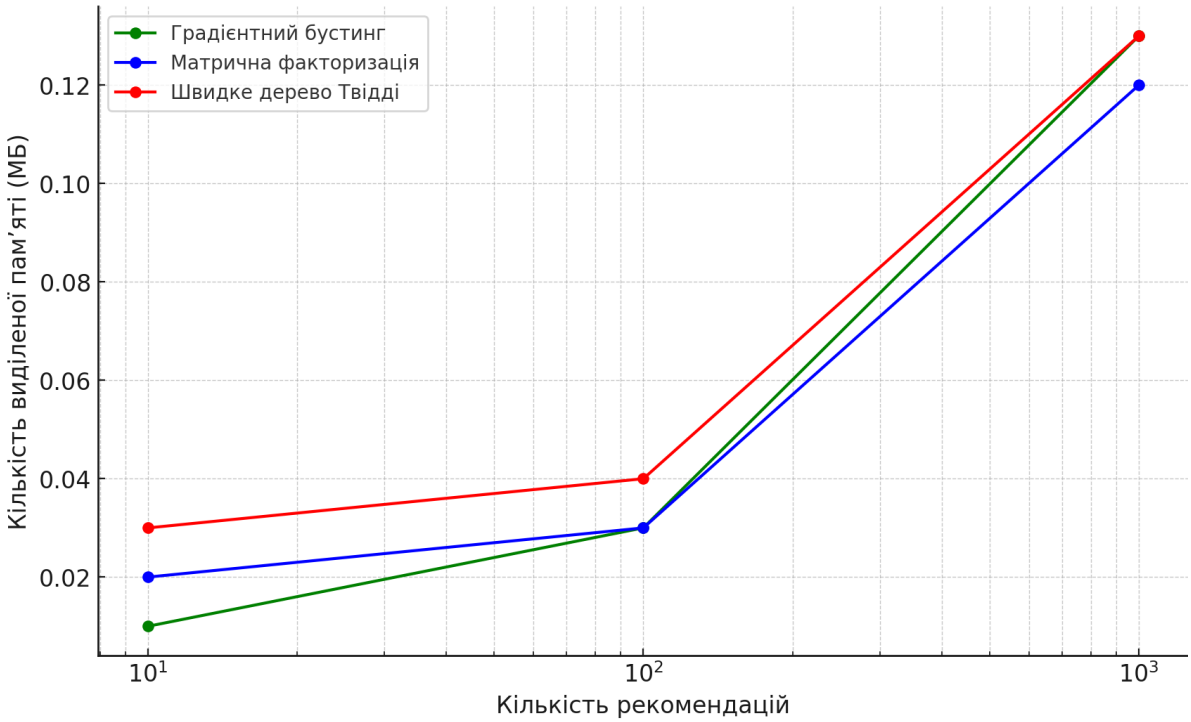

Рисунок 3.3 – Графіки залежності кількості виділеної пам’яті для генерації рекомендацій до кількості рекомендацій для користувача

Із рис. 3.3 видно, що Матрична факторизація зберігає найнижчі показники використання пам’яті навіть при збільшенні кількості рекомендацій, що підтверджує її ефективність у задачах з обмеженими ресурсами. Градієнтний бустинг демонструє трохи вищий рівень споживання пам'яті, але все ж залишається економічним. Швидке дерево Твідді потребує найбільшого обсягу пам'яті при обробці великої кількості запитів, що вказує на його дещо вищі вимоги до ресурсів у порівнянні з іншими алгоритмами.

3.1.3. Вимоги до ресурсів та ефективність використання алгоритмів на реальних даних

Вимоги до ресурсів є ключовим аспектом для алгоритмів машинного навчання, особливо при обробці великих обсягів даних у реальних умовах. Для персоналізованих систем, що працюють у режимі реального часу, важливі як ефективне використання пам'яті, так і швидкість обробки. Табл. 3.7 нижче демонструють вимоги до пам'яті, час генерації рекомендацій і точність, що дозволяє порівняти ефективність кожного алгоритму.

Таблиця 3.7 – Споживання пам'яті при різних обсягах даних

Кількість записів	Градієнтний бустинг (Мб)	Матрична факторизація (Мб)	Швидке дерево Твідді (Мб)
10 000	0,02	0,015	0,018
100 000	0,05	0,03	0,045
1 000 000	0,1	0,07	0,1

Аналіз даних табл. 3.7 показує, що матрична факторизація є найекономічнішою з точки зору використання пам'яті, демонструючи мінімальні витрати навіть при збільшенні обсягу записів до 1 000 000. Градієнтний бустинг та швидке дерево Твідді використовують більше пам'яті, особливо при значному обсязі даних, що може обмежувати їхню ефективність у системах з обмеженими ресурсами.

У табл. 3.8 представлено, як змінюється час генерації рекомендацій для кожного алгоритму при обробці 10 000, 100 000 і 1 000 000 записів. Це дозволяє зрозуміти, який із алгоритмів швидше обробляє запити при різних обсягах даних.

Таблиця 3.8 – Час генерації рекомендацій для різних обсягів даних

Кількість записів	Градієнтний бустинг (мс)	Матрична факторизація (мс)	Швидке дерево Твідді (мс)
10 000	3,4	2,5	3,1
100 000	7,2	5,3	6,8
1 000 000	21,9	15,36	19,89

Табл. 3.8 демонструє, що матрична факторизація зберігає найбільшу швидкість навіть при збільшенні обсягу даних, що робить її найоптимальнішим варіантом для систем із високими вимогами до часу відповіді. Градієнтний бустинг і швидке дерево Твідді поступаються матричній факторизації, хоча останнє також демонструє прийнятну продуктивність при значному навантаженні.

У табл. 3.9 представлено, як змінюється час генерації рекомендацій для кожного алгоритму при обробці 10 000, 100 000 і 1 000 000 записів. Це дозволяє зрозуміти, який із алгоритмів швидше обробляє запити при різних обсягах даних.

Таблиця 3.9 – Час генерації рекомендацій для різних обсягів даних

Кількість записів	Градієнтний бустинг (точність, %)	Матрична факторизація (точність, %)	Швидке дерево Твідді (точність, %)
10 000	87,5	85,3	86,9
100 000	89,1	87,8	88,5
1 000 000	90,3	89,6	89,9

Результати табл. 3.9 показують, що градієнтний бустинг забезпечує найвищу точність рекомендацій при значних обсягах даних, що робить його привабливим для задач, де важливим є саме якість рекомендацій. Матрична факторизація поступається за точністю, проте її ефективність у використанні ресурсів та висока швидкість обробки роблять її оптимальним варіантом у багатьох сценаріях. Швидке дерево Твідді забезпечує збалансовані результати, зберігаючи високий рівень точності та помірні вимоги до ресурсів.

Таким чином, результати аналізу вимог до ресурсів і ефективності використання алгоритмів на реальних даних свідчать про доцільність вибору алгоритму в залежності від конкретних потреб системи. Матрична факторизація забезпечує найкращі показники з точки зору використання пам'яті та швидкості, що робить її оптимальною для завдань, де швидкість є ключовою. Градієнтний бустинг, хоча і потребує більше ресурсів, може стати

найкращим вибором для задач, де необхідна максимальна точність рекомендацій. Швидке дерево Твідді, в свою чергу, є прийнятним компромісом між швидкістю, точністю та ресурсними вимогами, що дозволяє використовувати його в сценаріях з помірними вимогами до ресурсів і продуктивності.

3.2 Перспективи розвитку алгоритмів та їх комбінування для покращення результатів персоналізації

Розвиток алгоритмів персоналізації є одним із найперспективніших напрямків у галузі машинного навчання, зокрема для створення більш ефективних і точних систем рекомендацій. Із зростанням обсягів даних і складністю запитів користувачів важливим є не лише підвищення точності окремих алгоритмів, але й можливість їх комбінування для досягнення синергетичного ефекту.

Однією з перспективних стратегій є гібридний підхід, який передбачає поєднання різних алгоритмів, таких як градієнтний бустинг, матрична факторизація та швидке дерево Твідді, для створення комбінованої моделі. Такий підхід дозволяє об'єднати переваги кожного з алгоритмів: високу точність Градієнтного бустингу, ефективність використання ресурсів Матричної факторизації та збалансовану продуктивність Швидкого дерева Твідді. Гібридні моделі можуть використовуватися для покращення точності рекомендацій та швидкості їх генерації, адаптуючись до змінних характеристик даних та індивідуальних потреб користувачів.

Окрім комбінування алгоритмів, перспективи розвитку також включають використання адаптивних підходів, які дозволяють моделям вчитися на нових даних у реальному часі, зберігаючи при цьому актуальність рекомендацій. Це можливо завдяки алгоритмам, здатним до самонавчання та постійного оновлення своїх параметрів відповідно до змін у поведінці користувачів. Наприклад, застосування reinforcement learning може підвищити

точність персоналізації, оскільки алгоритм буде постійно покращувати свої рішення, адаптуючись до змінних умов і потреб.

Також, важливим напрямом розвитку є інтеграція глибокого навчання, зокрема згорткових і рекурентних нейромереж, у персоналізовані системи. Ці моделі здатні обробляти складні структури даних, такі як текстова, графічна або відеоінформація, що дозволяє забезпечувати персоналізацію на новому рівні. Використання глибоких нейромереж у поєднанні з традиційними методами, такими як Матрична факторизація, дозволяє створювати багатошарові системи рекомендацій, які враховують контекст і попередній досвід користувача, що значно підвищує релевантність рекомендацій.

Отже, комбінування різних алгоритмів і впровадження нових підходів дозволяє підвищити ефективність персоналізованих рекомендацій і забезпечити їх адаптивність у реальних умовах. З огляду на швидкий розвиток машинного навчання, очікується, що подальші дослідження у цьому напрямку відкриють нові можливості для вдосконалення алгоритмів персоналізації, що сприятиме створенню більш гнучких та інтелектуальних систем, здатних забезпечувати високий рівень користувацького досвіду.

3.3 Рекомендації щодо вибору алгоритмів машинного навчання залежно від специфіки завдання

Вибір алгоритму персоналізації є ключовим аспектом для успішного виконання завдання, оскільки різні алгоритми демонструють різні переваги залежно від характеристик даних і вимог до системи. На основі проведених досліджень, у яких були вивчені алгоритми градієнтного бустингу, матричної факторизації та шШвидкого дерева Твідді, можна запропонувати наступні рекомендації щодо їх використання в залежності від специфіки завдань:

Градієнтний бустинг

Градієнтний бустинг показав високу точність рекомендацій, особливо при збільшенні обсягу даних, що робить його оптимальним вибором для завдань, де ключовим критерієм є максимальна релевантність та точність.

Завдяки ітеративному покращенню моделі, цей алгоритм добре підходить для випадків, коли дані мають складну структуру та включають багато характеристик. Рекомендується використовувати градієнтний бустинг у сценаріях, де є достатній обсяг обчислювальних ресурсів і де точність прогнозів має вирішальне значення, наприклад, у системах рекомендацій, що працюють із великими наборами характеристик (наприклад, платформи з персоналізованими рекомендаціями товарів або медіаконтенту).

Матрична факторизація

Матрична факторизація зарекомендувала себе як найефективніший алгоритм з точки зору використання пам'яті та швидкості обробки даних. Цей алгоритм є оптимальним для завдань, що потребують швидкої обробки великих обсягів даних при обмежених ресурсах. Матрична факторизація особливо ефективна в задачах рекомендацій, де дані можуть бути представлені у вигляді матриці, наприклад, у системах рекомендацій фільмів, музики або товарів. Завдяки своїй здатності працювати з великими масивами даних у реальному часі, матрична факторизація є оптимальним вибором для онлайн-платформ із високим навантаженням, де швидкість генерації рекомендацій має критичне значення, а ресурсні вимоги обмежені.

Швидке дерево Твідді

Швидке дерево Твідді показало збалансовану продуктивність із точки зору часу обробки та використання ресурсів. Завдяки швидкому навчанню й обробці запитів, цей алгоритм є хорошим вибором для задач, де необхідно швидко адаптуватися до змін у поведінці користувача, забезпечуючи прийнятний рівень точності. Швидке дерево Твідді рекомендується для інтерактивних систем, таких як чат-боти або мобільні додатки, де користувачі очікують миттєвих рекомендацій. Також цей алгоритм підходить для випадків, коли система працює з середнім обсягом даних і вимоги до точності рекомендацій не є критичними.

Оскільки різні алгоритми мають свої сильні сторони, для задач із потребою в адаптивності до змін у поведінці користувачів варто розглянути

можливість комбінування алгоритмів градієнтного бустингу, матричної факторизації та швидкого дерева Твідді. Наприклад, матрична факторизація може бути використана для основних рекомендацій, а градієнтний бустинг – для більш точного налаштування рекомендацій на основі поточної активності користувача. Це дозволить створити систему, яка адаптується до динаміки змін, зберігаючи високу продуктивність та точність.

При невеликих обсягах даних найдоцільніше використовувати алгоритм градієнтного бустингу, оскільки він забезпечує високу точність при обмеженій кількості записів і здатен добре працювати з малими вибірками. У випадках, коли обсяг даних значний, оптимальним вибором буде матрична факторизація, яка показала найкращі результати у використанні пам’яті та швидкості обробки. Швидке дерево Твідді можна застосовувати для середніх обсягів даних, забезпечуючи збалансовані результати між швидкодією та точністю.

ВИСНОВКИ

Дана кваліфікаційна робота присвячена розробці та аналізу алгоритмів машинного навчання для персоналізації веб-додатків. Основною метою роботи було дослідити, як різні алгоритми машинного навчання можуть забезпечити ефективну персоналізацію на основі обробки великих обсягів даних і які саме алгоритми є оптимальними для різних сценаріїв використання. Для досягнення цієї мети було виконано огляд сучасних підходів, реалізовано програмні модулі на основі обраних алгоритмів, а також проведено порівняльний аналіз їх продуктивності та ефективності.

У першому розділі було розглянуто поняття персоналізації веб-додатків і роль машинного навчання в цьому процесі. Зокрема, аналізувалися принципи колаборативної фільтрації та кластеризації, які лежать в основі сучасних персоналізованих систем. Були також визначені особливості вибору алгоритмів залежно від типу даних, таких як числові, текстові, мультимедійні та часові ряди, що дозволило сформувати чітке уявлення про специфіку алгоритмів для різних завдань персоналізації. Огляд сучасних моделей включав алгоритми колаборативної фільтрації, матричної факторизації, випадкових лісів та LSTM, що надає широкий вибір підходів для побудови персоналізованих веб-систем.

Другий розділ висвітлює математичні основи обраних алгоритмів, а також їх програмну реалізацію. Було детально розглянуто алгоритми градієнтного бустингу, матричної факторизації та швидкого дерева Твідді, кожен з яких було описано з позиції математичної моделі та можливостей застосування у задачах персоналізації. Для реалізації алгоритмів було обрано технологічний стек на основі мови програмування C# та середовища Microsoft Visual Studio, що включає ML.NET як платформу для реалізації алгоритмів машинного навчання. У цьому розділі представлено діаграму класів системи, описано основні класи і методи, що було необхідно для створення надійного додатку, який здатний обробляти дані для персоналізації.

Третій розділ був присвячений порівнянню алгоритмів за основними характеристиками, такими як швидкість навчання, час генерації рекомендацій, а також вимоги до ресурсів. На основі експериментальних даних було продемонстровано, що матрична факторизація є найефективнішим алгоритмом для роботи з великими обсягами даних при обмежених ресурсах, демонструючи найменше споживання пам'яті (від 0,015 Мб для 10 000 записів до 0,12 Мб для 1 000 000 записів) і найкоротший час генерації рекомендацій (6,51 мс для 100 рекомендацій). Градієнтний бустинг показав найвищу точність при збільшенні обсягу даних, досягаючи 90,3% для набору з 1 000 000 записів, але при цьому потребував більше часу для генерації рекомендацій – до 21,9 мс для 1000 рекомендацій – та використав більше пам’яті (до 0,13 Мб). Швидке дерево Твідді забезпечило збалансовану продуктивність, залишаючись оптимальним вибором для задач зі середніми вимогами до швидкодії та ресурсів: при 1 000 000 записів цей алгоритм потребував 83,773 секунди для навчання і 0,13 Мб пам’яті при генерації 1000 рекомендацій, що свідчить про його ефективність у сценаріях середньої складності.

Розділ також висвітлював перспективи розвитку алгоритмів, зокрема, можливості їх комбінування для підвищення ефективності персоналізації, а також рекомендації щодо вибору алгоритмів залежно від специфіки завдання. Отримані кількісні показники дозволяють зробити висновки щодо оптимальних умов використання кожного алгоритму і створити основи для подальших досліджень у напрямку побудови гібридних моделей для персоналізації, що поєднують переваги кількох алгоритмів.

Проведена робота демонструє застосування алгоритмів машинного навчання для персоналізації веб-додатків з акцентом на обробку великих обсягів даних. Отримані результати є основою для подальших досліджень у напрямку розробки адаптивних персоналізованих систем і можуть бути використані для створення гібридних моделей, що об’єднують переваги різних алгоритмів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Beheshti, A., Yakhchi, S., Mousaeirad, S., Ghafari, S. M., Goluguri, S. R., & Edrisi, M. A. Towards cognitive recommender systems. Algorithms, 13(8), 2020. 176p.

2. Investigating machine learning methods in recommender systems. URL: https://discovery.ucl.ac.uk/id/eprint/10031000/1/Full_copy.pdf (дата звернення 25.09.2024).

3. Thyagharajan, K. K., & Nayak, R. Adaptive content creation for personalized e-learning using web services. Journal of Applied Sciences Research, 3(9), 2007. 9 p.

4. Matthew, J. R. Netflix and the design of the audience: The homogenous constraints of data-driven personalization. MedieKultur: Journal of media and communication research, 36(69), 2020. p. 14.

5. Mahnke, M., Grunnert, J., & Tarp, N. T. Examining user's counter-narratives opposing the introduction of algorithmic personalization on Instagram. First Monday, 22(4). 2017. 13 p.

6. Григоров, О. В., Аніщенко, Г. О., Стрижак, В. В., Петренко, Н. О., Турчин, О. В., Окунь, А. О., ... & Пономарев, О. Э. Штучний інтелект. Машинне навчання. 2019. 11 с.

7. Курченко, О., & Мишко, І. Автоматизація та управління в операційних системах: роль штучного інтелекту та машинного навчання. Scientific Collection «InterConf», (184), 2023. с. 491-499.

8. Chou, Y. C., Chen, C. T., & Huang, S. H. Modeling behavior sequence for personalized fund recommendation with graphical deep collaborative filtering. Expert Systems with Applications, 2022. 192 p.

9. 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 24.09.2024).

10. Strehl, A., & Ghosh, J. Cluster ensembles-a knowledge reuse framework for combining multiple partitions. Journal of machine learning research, 2012. – pp. 583-617.

11. Schneider, J., & Vlachos, M. Personalization of deep learning. In Data Science–Analytics and Applications: Proceedings of the 3rd International Data Science Conference–iDSC2020 2021. pp. 89-96.

12. Di Noia, T., Mirizzi, R., Ostuni, V. C., Romito, D., & Zanker, M. Linked open data to support content-based recommender systems. In Proceedings of the 8th international conference on semantic systems 2012 pp. 1-8.

13. Berbatova, M. Overview on NLP techniques for content-based recommender systems for books. In Proceedings of the Student Research Workshop Associated with RANLP 2019 pp. 55-61.

14. Kim, Y., & Shim, K. TWILITE: A recommendation system for Twitter using a probabilistic model based on latent Dirichlet allocation. Information Systems, 42, 2014. pp. 59-77.

15. Amato, F., Moscato, V., Picariello, A., & Piccialli, F. (SOS: a multimedia recommender system for online social networks. Future generation computer systems, 93, 2019. – pp. 914-923.

16. Zhou, W., Wen, J., Qu, Q., Zeng, J., & Cheng, T. Shilling attack detection for recommender systems based on credibility of group users and rating time series. PloS one, 13(5), 2018. pp. 127-135.

17. Koohi, H., & Kiani, K. User based collaborative filtering using fuzzy C-means. Measurement, 2016. pp. 134-139.

18. Xue, F., He, X., Wang, X., Xu, J., Liu, K., & Hong, R. Deep item-based collaborative filtering for top-n recommendation. ACM Transactions on Information Systems (TOIS), 37(3), 2019. pp. 1-25.

19. Khan, M. A., Khan, G. A., Khan, J., Khan, M. R., Atoum, I., Ahmad, N., ... & Alghamdi, A. A. Multi-view clustering based on multiple manifold regularized non-negative sparse matrix factorization. IEEE access, 10, 2022. pp. 249-259.

20. Recommendation for beginners: Model-based Matrix Factorization Algorithm. ULR: https://medium.com/@xiaolancara/recommendation-for-beginners-model-based-matrix-factorization-algorithm-fbcb7279c292 (дата звернення 24.09.2024).

21. Rivera-Lopez, R., Canul-Reich, J., Mezura-Montes, E., & Cruz-Chávez, M. A. Induction of decision trees as classification models through metaheuristics. Swarm and Evolutionary Computation, 69, 2022. 106 p.

22. Decision Tree vs Random Forest | Which Is Right for You. URL: https://www.analyticsvidhya.com/blog/2020/05/decision-tree-vs-random-forest-algorithm/ fbcb7279c292 (дата звернення 24.09.2024).

23. Chandra, R., Jain, A., & Singh Chauhan, D. Deep learning via LSTM models for COVID-19 infection forecasting in India. PloS one, 17(1), 2022. 25 p.

24. Understanding Long Short Term Memory (LSTM) Networks. URL: https://emergingindiagroup.com/long-short-term-memory-lstm/ fbcb7279c292 (дата звернення 24.09.2024).

25. GuolinKe, Q. M., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., & Liu, T. Y. Lightgbm: A highly efficient gradient boosting decision tree. Adv. Neural Inf. Process. Syst, 2017. 16 p.

26. Liang, W., Luo, S., Zhao, G., & Wu, H. Predicting hard rock pillar stability using GBDT, XGBoost, and LightGBM algorithms. Mathematics, 8(5), 2020. 765 p.

27. Stein-O'Brien, G. L., Arora, R., Culhane, A. C., Favorov, A. V., Garmire, L. X., Greene, C. S., ... & Fertig, E. J. Enter the matrix: factorization uncovers knowledge from omics. Trends in Genetics, 34(10), 2018. pp. 790-805.

28. Rojek, I., Mikołajewski, D., Dorożyński, J., Dostatni, E., & Mreła, A. An ML-Based Solution in the Transformation towards a Sustainable Smart City. Applied Sciences, 14(18), 2024. 28 p.

29. Балабанов О.І., Павленко А.П. PyCharm для розробників Python: Навчальний посібник. - Львів: Видавництво Львівської політехніки, 2018. - 300 с.

30. Гальперін А., Кочнев Д. Робота з Eclipse: Навчальний посібник. - Харків: Видавництво Харківського національного університету імені В.Н. Каразіна, 2017. - 230 с.

31. Коноваленко І.В., Марущак П.О. Платформа .NET та мова програмування C# 8.0: навчальний посібник. Тернопіль: ФОП Паляниця В. А., 2020. 320 с.

32. Рихтер Д.. Основи програмування на C#: Навчальний посібник. - К.: Видавництво "Діалектика", 2017. - 632 с.

33. Троелсен Е. C# 7.0 та платформа .NET: Посібник для професіоналів. - Львів: Видавництво Львівської політехніки, 2018. - 918 с.

34. Kim T. Matrix factorization and prediction for high dimensional zero inflated co-occurrence count data via shared parameter alternating generalized linear regression with application in NLP. 2023. 232 p.

35. Yang, Y., Qian, W., & Zou, H. Insurance premium prediction via gradient tree-boosted Tweedie compound Poisson models. Journal of Business & Economic Statistics. 2018. Vol36, No3, pp. 456-470.

ДОДАТОК А. ПРИКЛАДИ РЕЗУЛЬТАТІВ НАВЧАННЯ МОДЕЛЕЙ

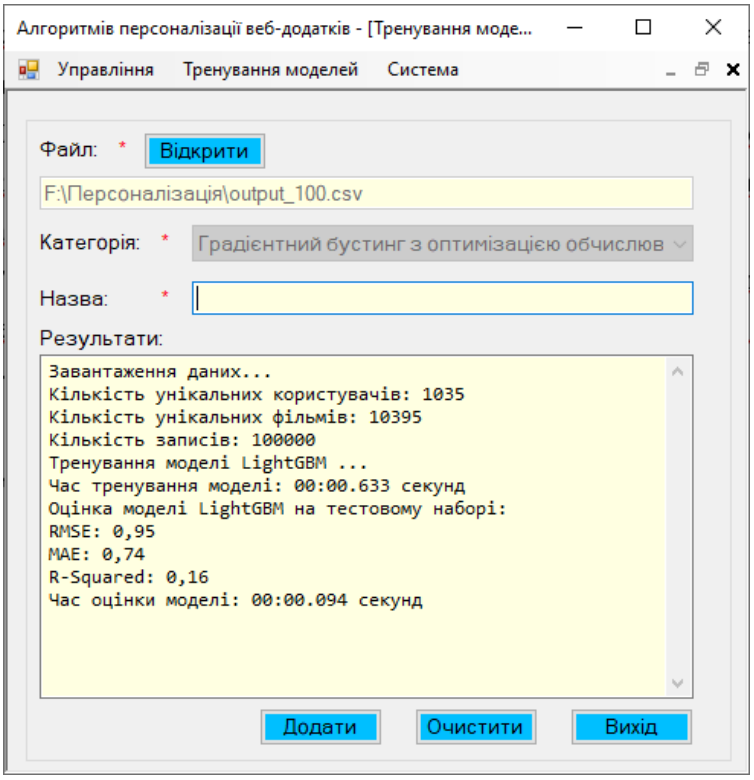

Рисунок А.1 – Приклад навчання за допомогою градієнтного бустингу

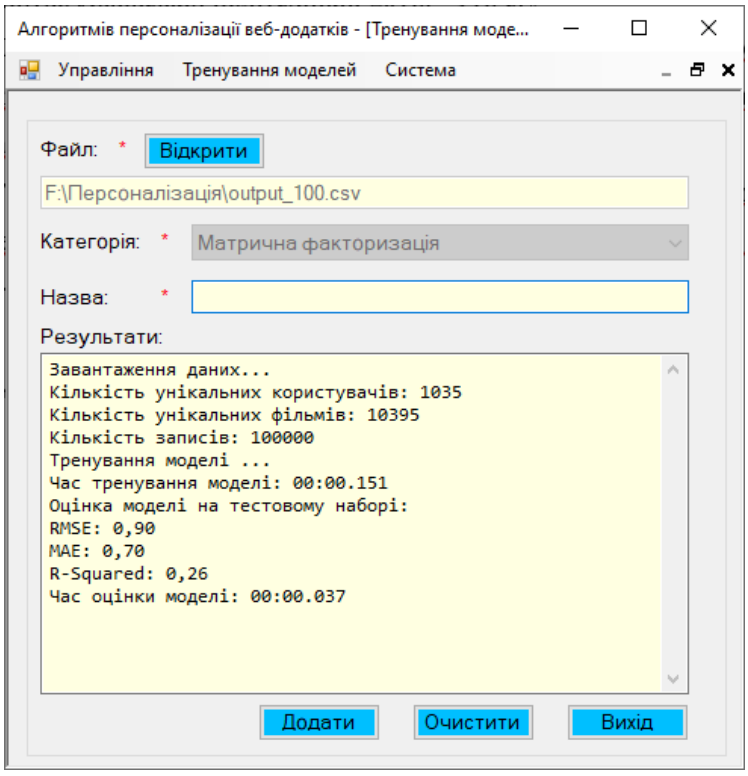

Рисунок А.2 – Приклад навчання за допомогою матричної факторизації

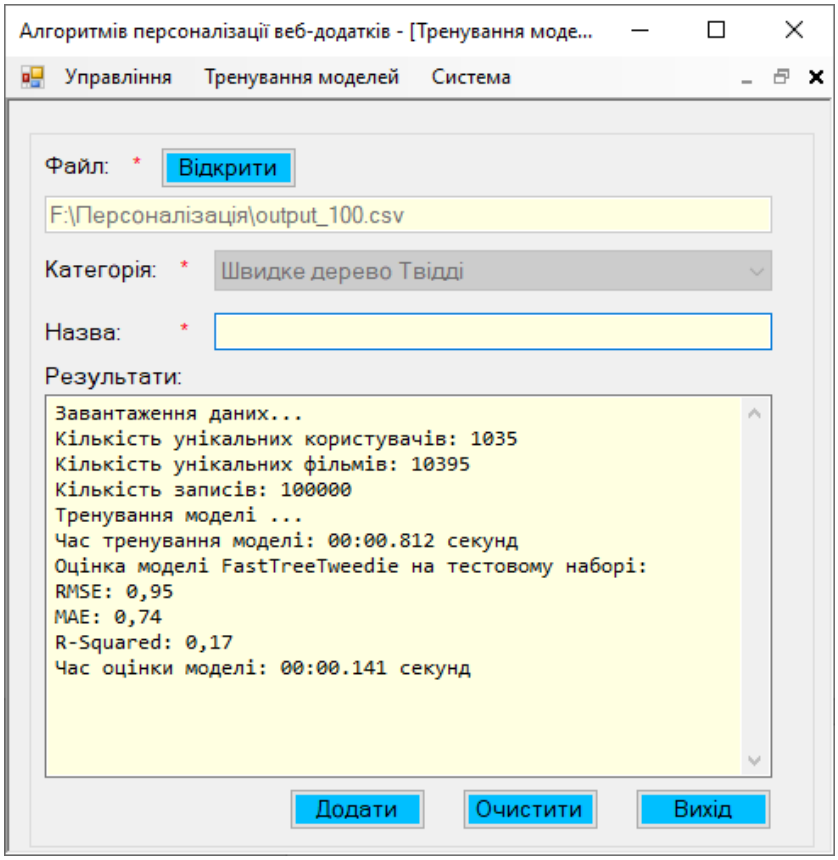

Рисунок А.3 – Приклад навчання за допомогою швидкого дерева Твідді

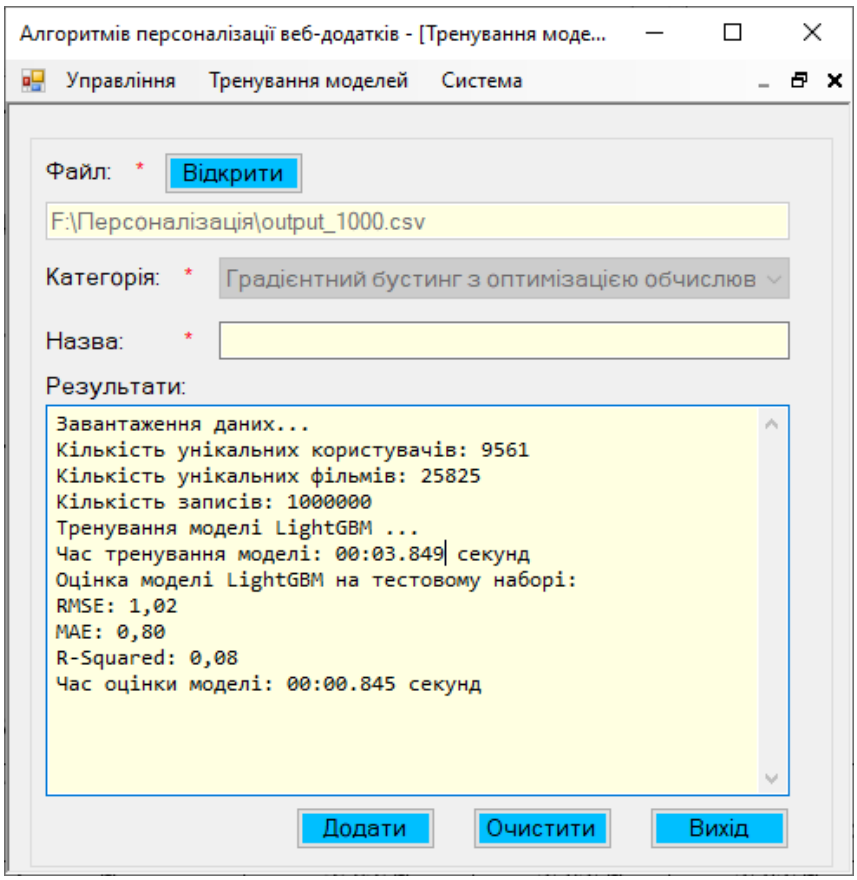

Рисунок А.4 – Приклад навчання за допомогою градієнтного бустингу

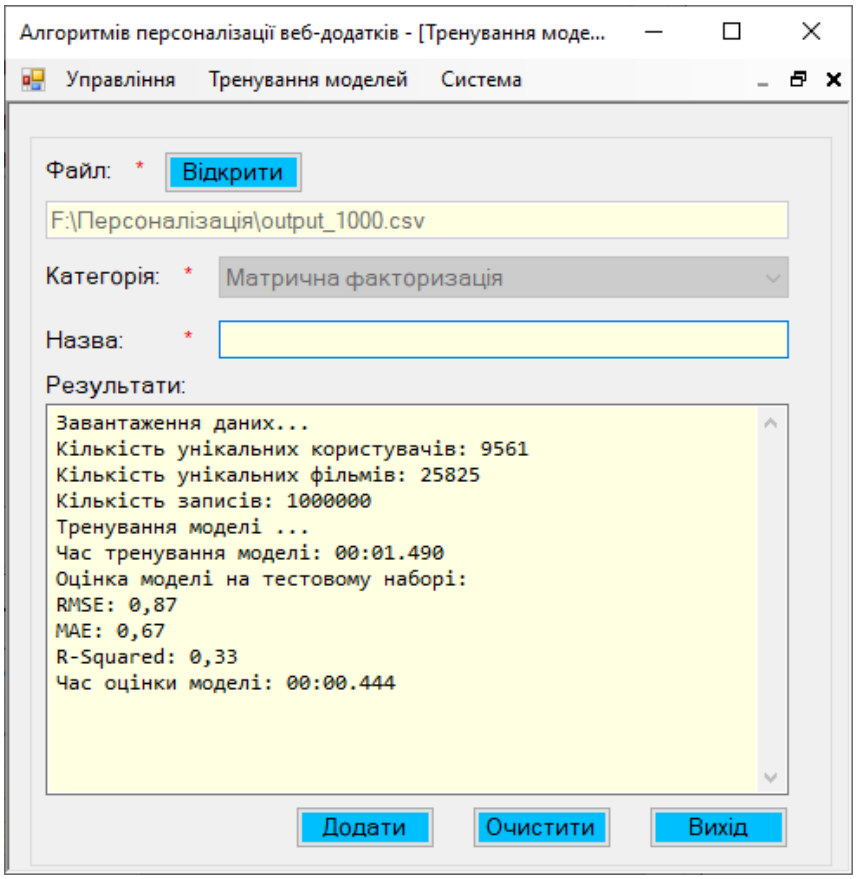

Рисунок А.5 – Приклад навчання за допомогою матричної факторизації

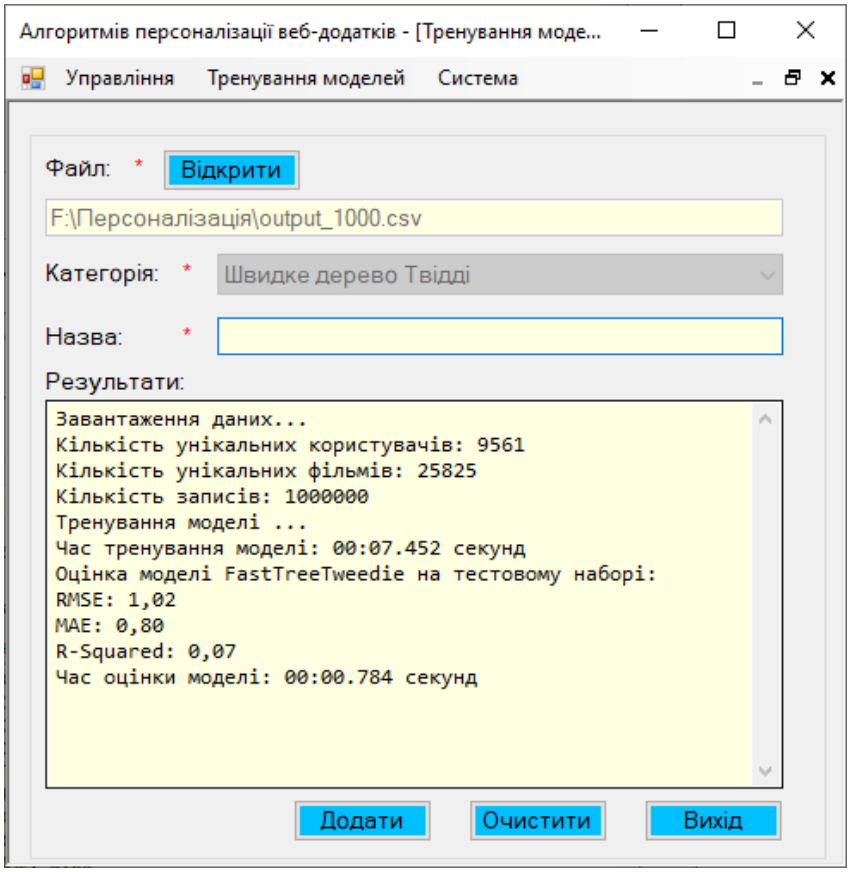

Рисунок А.6 – Приклад навчання за допомогою швидкого дерева Твідді

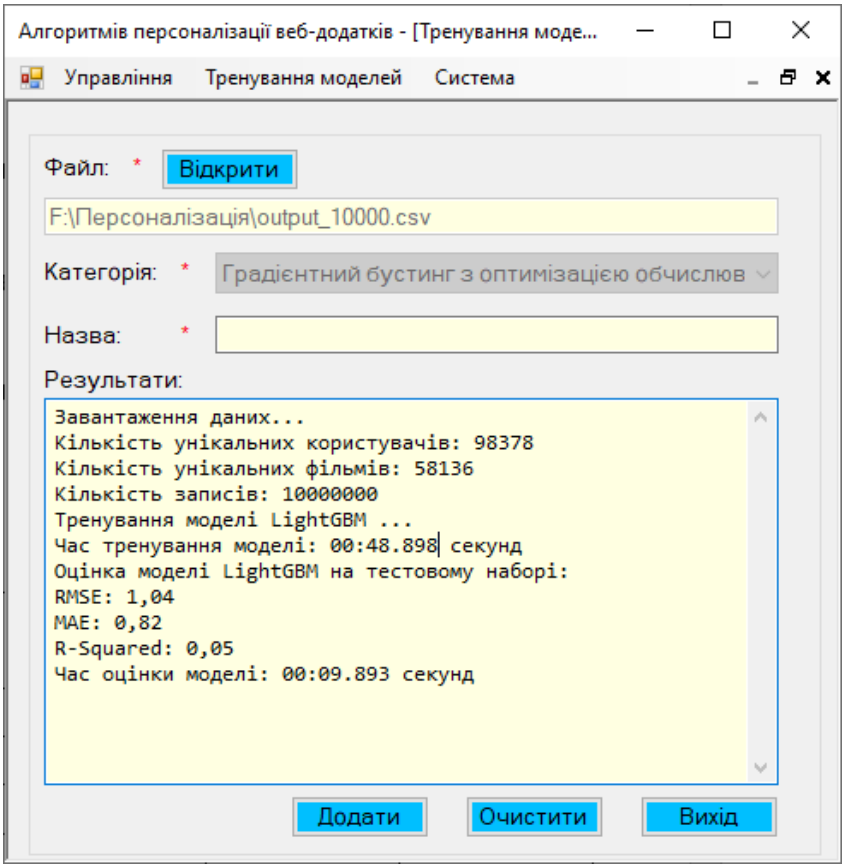

Рисунок А.7 – Приклад навчання за допомогою градієнтного бустингу

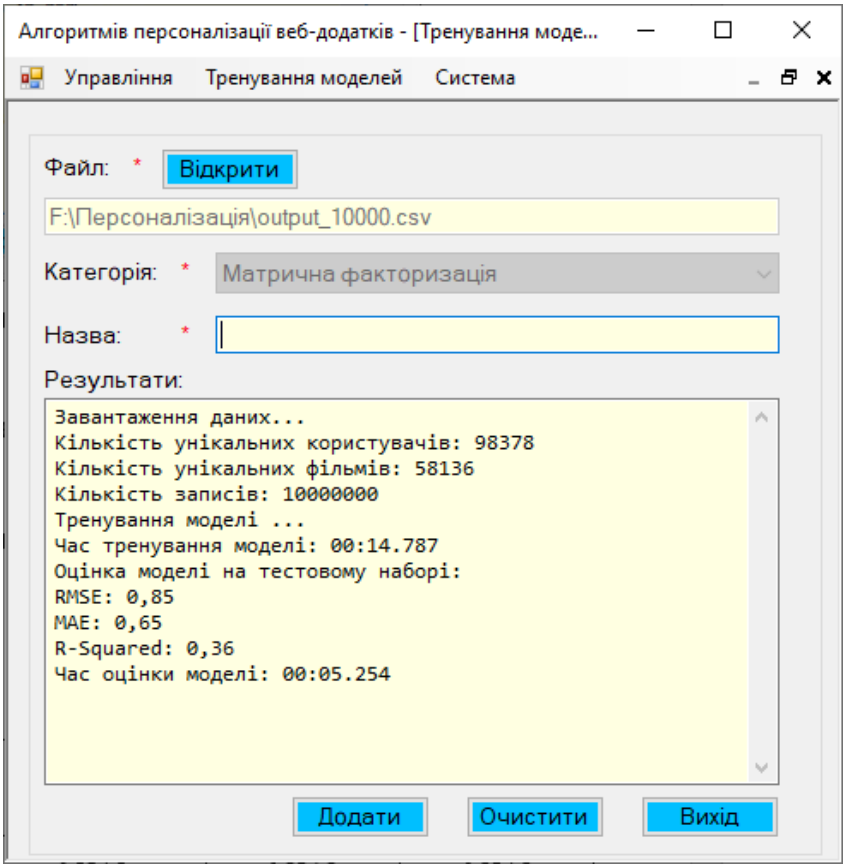

Рисунок А.8 – Приклад навчання за допомогою матричної факторизації

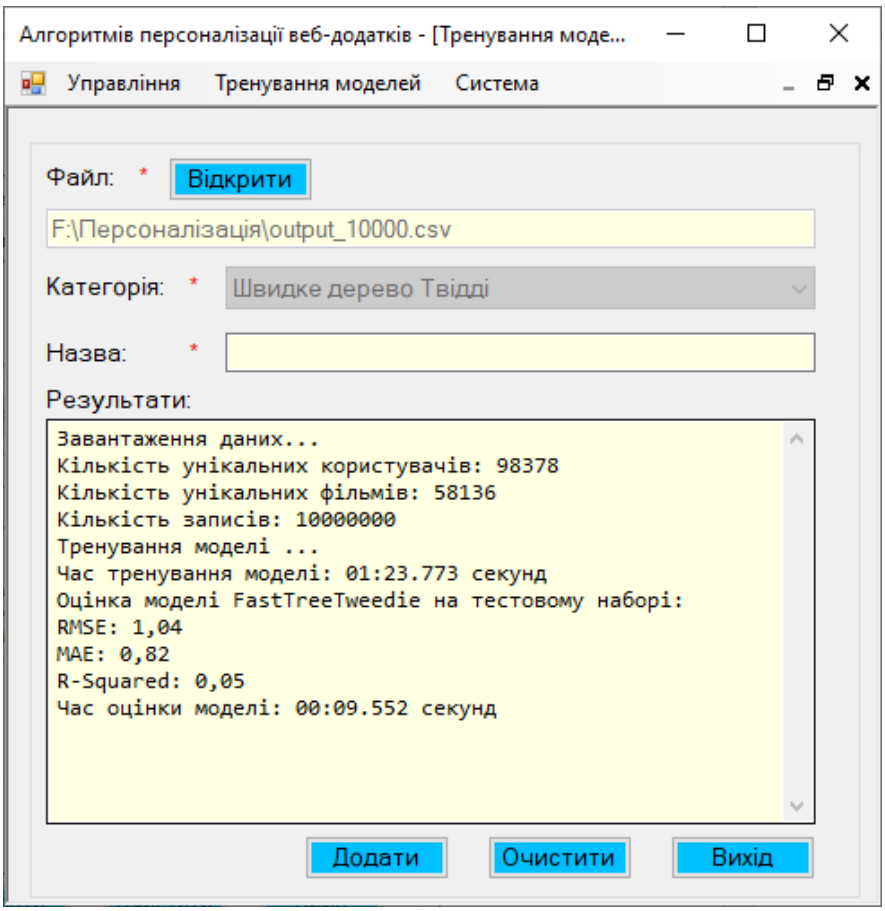

Рисунок А.9 – Приклад навчання за допомогою швидкого дерева Твідді

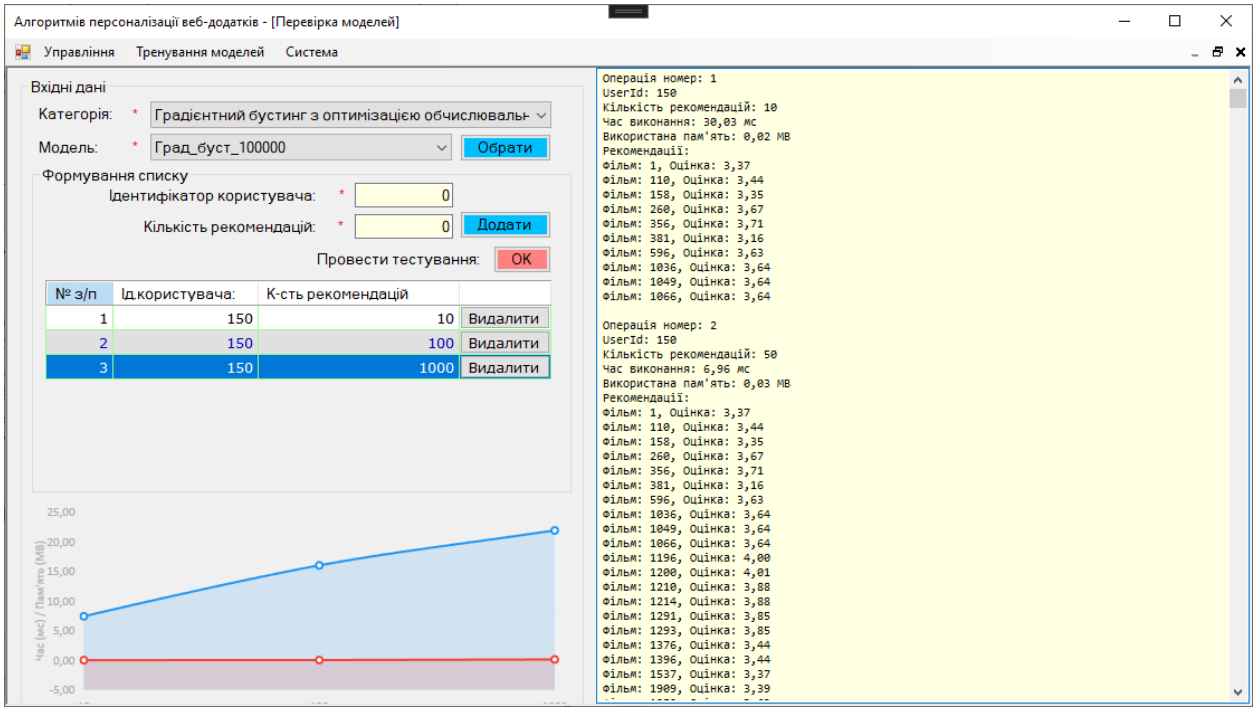

Рисунок А.10 – Приклад генерації рекомендацій за допомогою градієнтного бустингу

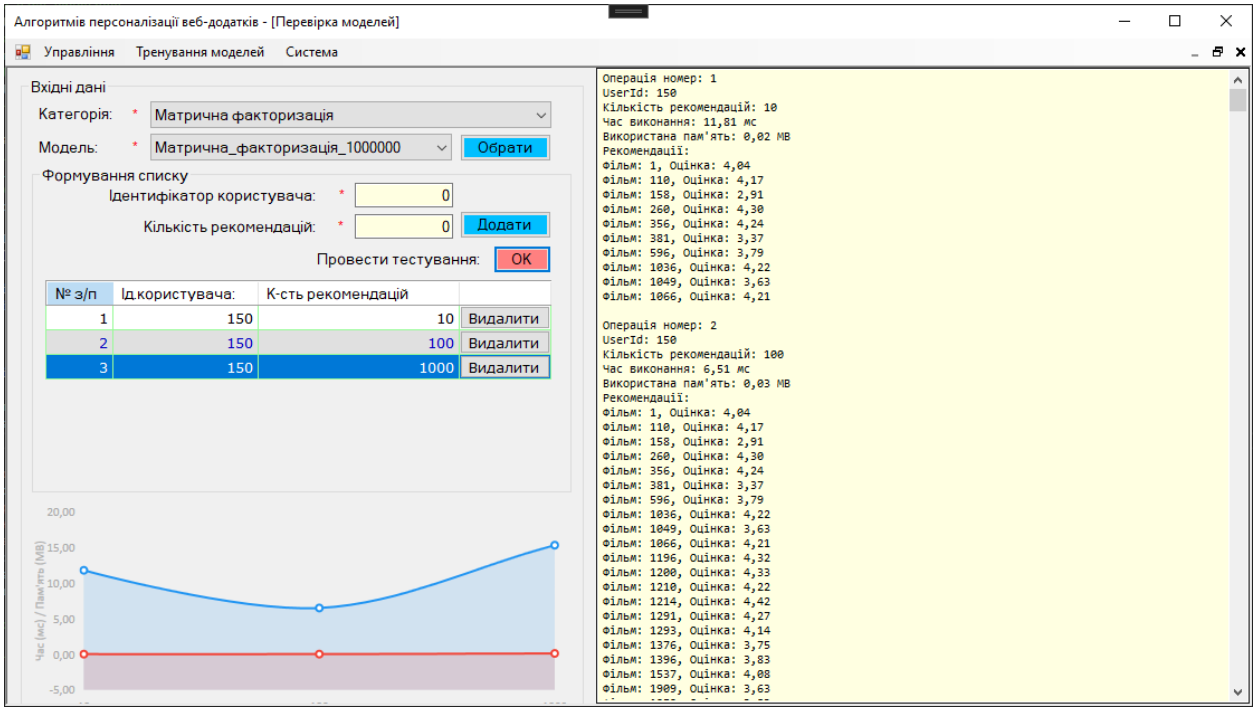

Рисунок А.11 – Приклад генерації рекомендацій за допомогою матричної факторизації

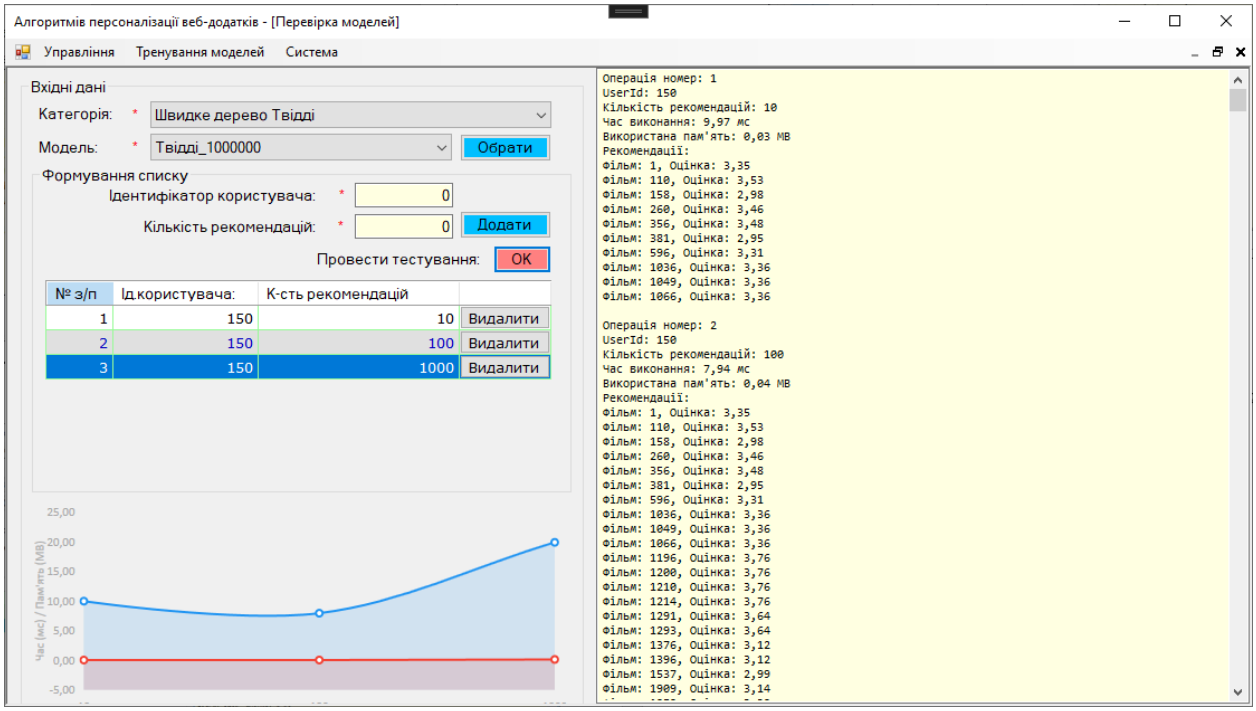

Рисунок А.12 – Приклад генерації рекомендацій за допомогою швидкого дерева Твідді