НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут атомної та теплової енергетики

Кафедра цифрових технологій в енергетиці

«До захисту допущено»
Завідувач кафедри
_____ Наталія АУШЕВА
“___”_____2026 р.

Дипломна робота
на здобуття ступеня бакалавр

За освітньою програмою “Цифрові технології в енергетиці”

Спеціальності 122 “Комп’ютерні науки”

на тему: “Система рекомендацій фільмів з поясненнями на основі методів штучного інтелекту”

Виконав: студент 4 курсу, групи ТР-21
Коваленко Костянтин Анатолійович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник асистент Владислав Тітов _____
(посада, науковий ступінь, вчене звання, ім’я, ПРІЗВИЩЕ) (підпис)

Рецензент доцент каф. ТАЕ, к.т.н., Артур РАЧИНСЬКИЙ _____
(посада, науковий ступінь, вчене звання, ім’я, ПРІЗВИЩЕ) (підпис)

Н.контроль ст.викладач Ольга БЕСПАЛА _____
(посада, ім’я, ПРІЗВИЩЕ) (підпис)

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

Київ - 2026

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ

Кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

Рівень вищої освіти – перший (бакалаврський)
спеціальність 122 “Комп’ютерні науки”
Освітньо-професійна програма “Цифрові технології в енергетиці”

ЗАТВЕРДЖУЮ
Завідувач кафедри ЦТЕ
_____ Наталія АУШЕВА
(підпис)
“___” _____2026 р.

ЗАВДАННЯ

на дипломну роботу студенту

Коваленко Костянтину Анатолійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи “ Система рекомендацій фільмів з поясненнями на основі методів штучного інтелекту”
Науковий керівник Тітов Владислав Миколайович
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від “28” травня 2026 року № 1992-с

2. Термін подання студентом роботи 04.06.2026

3. Вихідні дані до роботи мова програмування Python, бібліотеки Streamlit, pandas, NumPy, scikit-learn, середовище розробки Visual Studio Code, набір даних про фільми, користувачів та їхні оцінки

4. Перелік питань, які потрібно розробити _____

1) провести аналіз існуючих систем рекомендацій фільмів;
2) визначити основні функції системи рекомендацій фільмів з поясненнями;

3) обґрунтувати вибір інструментів, алгоритмів та технологій реалізації системи;
4) розробити архітектуру програмного забезпечення та структуру даних;
5) реалізувати алгоритми контентної, колаборативної та гібридної фільтрації;
6) реалізувати механізм пояснення сформованих рекомендацій;
7) створити веб-інтерфейс користувача засобами Streamlit;
8) представити процес використання системи та результати формування рекомендацій.
9) Дата видачі завдання 15.09.2025

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Вибір теми роботи	12.04.26	виконано
2.	Аналіз методів та засобів розв’язання задачі	13.04-17.04.26	виконано
3.	Розробка архітектури та загальної структури системи	18.04-22.04.26	виконано
4.	Розробка окремих підсистем	22.04-27.04.26	виконано
5.	Програмна реалізація системи	28.04-05.05.26	виконано
6.	Оформлення пояснювальної записки	06.05-13.05.26	виконано
7.	Захист програмного забезпечення	14.05.26	виконано
8.	Передзахист	04.06.26	виконано
9.	Захист	17.06.26	виконано

Студент _____ Костянтин КОВАЛЕНКО
(підпис) (прізвище та ініціали)

Керівник _____ Владислав ТІТОВ
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Мета роботи – розробка програмного забезпечення системи рекомендацій фільмів з поясненнями на основі методів штучного інтелекту.

Методи та засоби: контентна фільтрація на основі TF-IDF та косинусної подібності, колаборативна фільтрація на основі матриці «користувач — фільм», гібридне ранжування рекомендацій, SVD-бейзлайн з урахуванням зміщень користувачів і фільмів, персоналізований SVD з пошуком сусідів у латентному просторі, мова програмування Python, бібліотеки pandas, NumPy, scikit-learn, scipy, joblib, фреймворк Streamlit, середовище розробки Visual Studio Code.

Результат – програмна система персоналізованого підбору фільмів, яка формує рекомендації на основі контентних, колаборативних і рейтингових ознак, а також надає користувачеві пояснення причин появи рекомендованих фільмів у підсумковому списку.

Ключові слова: РЕКОМЕНДАЦІЙНА СИСТЕМА, ШТУЧНИЙ ІНТЕЛЕКТ, МАШИННЕ НАВЧАННЯ, TF-IDF, КОЛАБОРАТИВНА ФІЛЬТРАЦІЯ, SVD, STREAMLIT, ПОЯСНЮВАНІ РЕКОМЕНДАЦІЇ.

ABSTRACT

The purpose of the work is to develop software for a movie recommendation system with explanations based on artificial intelligence methods.

Methods and tools: content-based filtering using TF-IDF and cosine similarity, collaborative filtering based on the “user–movie” matrix, hybrid recommendation ranking, SVD baseline with user and movie bias adjustment, personalized SVD with neighbor search in latent space, Python programming language, pandas, NumPy, scikit-learn, scipy, joblib libraries, Streamlit framework, Visual Studio Code development environment.

The result is a software system for personalized movie selection that generates recommendations based on content, collaborative, and rating features, and also provides the user with explanations of why the recommended movies appear in the final list.

Keywords: RECOMMENDATION SYSTEM, ARTIFICIAL INTELLIGENCE, MACHINE LEARNING, TF-IDF, COLLABORATIVE FILTERING, SVD, STREAMLIT, EXPLAINABLE RECOMMENDATIONS.

ЗМІСТ

ВСТУП

У сучасному інформаційному середовищі користувачі стикаються з постійним зростанням кількості цифрового контенту. Особливо помітною ця проблема є у сфері фільмів і серіалів, де користувачеві часто складно самостійно обрати релевантний фільм серед великої кількості доступних варіантів. У таких умовах рекомендаційні системи виконують важливу роль, оскільки вони допомагають скоротити час пошуку, враховують індивідуальні вподобання користувача та формують персоналізований перелік фільмів. Проте звичайної рекомендації часто недостатньо: користувачеві важливо розуміти, чому система запропонувала саме цей фільм, які його характеристики були враховані та як рекомендація пов’язана з попередніми оцінками.

Актуальність теми роботи зумовлена необхідністю створення програмних систем, які не лише формують персоналізовані рекомендації, а й пояснюють їх користувачеві. У сфері кіноконтенту пояснюваність має особливе значення, оскільки вибір фільму часто залежить не лише від жанру або рейтингу, а й від поєднання сюжету, акторського складу, стилю, популярності та попереднього досвіду перегляду. Якщо система просто показує список фільмів без пояснень, користувач не завжди довіряє результату. Якщо ж рекомендація супроводжується зрозумілим поясненням, користувач може оцінити логіку роботи системи та прийняти більш обґрунтоване рішення.

Метою роботи є розробка програмної системи рекомендацій фільмів з поясненнями на основі методів штучного інтелекту. Для досягнення цієї мети у роботі використано гібридний підхід, який поєднує контентну фільтрацію, колаборативну фільтрацію, SVD-бейзлайн, персоналізований SVD та механізми пояснення результатів. Такий підхід дозволяє враховувати як змістові характеристики фільмів, так і поведінкові закономірності користувачів.

Для досягнення поставленої мети необхідно розв’язати такі завдання: провести аналіз підходів, методів та алгоритмів побудови рекомендаційних

систем фільмів; проаналізувати програмні засоби для реалізації системи рекомендацій фільмів з поясненнями; розробити програмне забезпечення рекомендаційної системи з використанням гібридного підходу; реалізувати веб-інтерфейс користувача для взаємодії із системою; реалізувати механізм пояснення сформованих рекомендацій; провести тестування розробленого програмного забезпечення та продемонструвати результати його роботи.

Об’єктом роботи є процес формування персоналізованих рекомендацій фільмів. Предметом роботи є методи, алгоритми та програмні засоби побудови пояснюваної рекомендаційної системи, що поєднує контентні, колаборативні та латентні фактори.

Практична перевірка результатів виконувалася шляхом запуску розробленого веб-застосунку, тестування сценаріїв роботи з демо-профілями, додавання оцінок, формування рекомендацій і перегляду пояснень до них. У межах практичної демонстрації було перевірено роботу системи для користувачів із наявною історією оцінювання та для нового користувача, який перебуває у сценарії cold-start.

Робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків. У першому розділі розглянуто постановку задачі та предметну область рекомендаційних систем для фільмів. У другому розділі описано методи й алгоритми формування пояснюваних рекомендацій. У третьому розділі наведено програмну реалізацію системи. У четвертому розділі продемонстровано роботу веб-застосунку, сценарії використання та результати формування рекомендацій.

1 ПОСТАНОВКА ЗАДАЧІ ТА АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

У першому розділі розглядається проблематика створення системи рекомендацій фільмів з поясненнями та виконується постановка прикладної задачі, яка розв’язується у дипломній роботі. Наведено характеристику предметної області рекомендаційних систем для фільмів, описано основні типи вхідних даних, які використовуються для персоналізації, зокрема контентні ознаки фільмів, оцінки користувачів та поведінкові сигнали. Окремо визначено вимоги до програмної системи, яка повинна формувати персоналізований перелік фільмів і надавати користувачеві пояснення причин появи рекомендацій. Також у розділі виконано огляд методів побудови рекомендацій, зокрема контентної фільтрації, колаборативної фільтрації, гібридного підходу та SVD-моделей. Завершальна частина розділу присвячена аналізу програмних засобів, які використовуються для реалізації системи, зокрема Python, pandas, NumPy, scikit-learn, scipy, joblib та Streamlit.

1.1. Постановка прикладної задачі рекомендаційної системи фільмів з поясненнями

Рекомендаційні системи (RS) використовують різні інформаційні сигнали, щоб допомогти користувачам знаходити релевантний контент у великих каталогах (заповнення інформаційного пробілу). У сфері фільмів та медіа такі системи зазвичай працюють із двома основними типами даних: 1) контентними ознаками фільмів (жанр, актори, опис, ключові слова тощо) і 2) оцінками або переглядами користувачів. Контентно-орієнтовані фільтраційні методи рекомендують схожі за ознаками фільми до тих, які вже подобаються користувачу. Натомість колаборативні методи аналізують матрицю «користувач–

фільм» і рекомендують те, що сподобалося подібним користувачам. Використання гібридних підходів дозволяє враховувати обидва джерела інформації.

Галузь кіно-стримінгу стрімко розвивається, і якісна рекомендаційна система є важливим активом сервісу. Наприклад, Netflix відзначає, що його система рекомендацій має вирішальне значення для залучення і збереження користувачів. Проте масивні дані і багатоманітність смаків користувачів ускладнюють прості підходи. Тому в академічній літературі та на практиці застосовують методи машинного навчання для персоналізації. Зокрема, для фільмів характерне використання багатої метаданих (сюжет, акторський склад, рейтинги IMDb тощо), а також отримання зворотного зв’язку від користувачів (вподобання, рейтинги). Такі дані об’єднуються у комплексних моделях рекомендацій [14].

Завдання полягає у створенні програмної системи, що генерує персоналізовані рекомендації фільмів з поясненнями. Система має опрацьовувати вхідні дані – базу фільмів з їх ознаками (жанри, ключові слова, опис), інформацію про користувачів (і їхні оцінки чи вподобання) – та навчати моделі рекомендацій. Вимоги: модель має видавати перелік N фільмів для кожного користувача, які користувачеві, ймовірно, сподобаються, а також формувати коротке пояснення (наприклад, «Рекомендується тому що…») на основі характеристик фільмів або поведінки схожих користувачів.

Програмна система повинна мати модульну архітектуру: окремі компоненти для підготовки даних, навчання моделей, генерації рекомендацій та формування пояснень, а також інтерфейс для взаємодії з користувачем. Важливі вимоги – масштабованість та можливість адаптації: при зміні бази фільмів чи нових даних про користувачів система повинна швидко перевчатися. Інтерфейс повинен забезпечувати ввід інформації (наприклад, вибір існуючого користувача чи введення нового) та вивід результатів. Також система повинна підтримувати сценарії cold-start (новий користувач чи фільм) і warm-start.

Оцінювання якості рекомендацій проводиться через відкладені (offline) метрики: Precision@N, Recall@N тощо. Успішність системи оцінюється також за якістю пояснень – наскільки користувач розуміє і довіряє результатам. Таким чином, досягти потрібно балансу між точністю рекомендацій та їх прозорістю.

1.2. Огляд методів побудови персоналізованих і пояснюваних рекомендацій

Загалом існують два головні підходи до рекомендації контенту: контентна фільтрація і колаборативна фільтрація. Контентна фільтрація представляє кожний фільм як вектор ознак (наприклад, TF-IDF вектор опису) і оцінює схожість між фільмами. «Рекомендації подібних до тих, які користувач уже подивився» – саме таку стратегію описано у рекомендаційних системах контентного типу. Найпоширеніша міра схожості – косинусна подібність між векторами ознак. З іншого боку, колаборативна фільтрація оперує матрицею оцінок користувачів: кожен користувач представляється вектором оцінок фільмів. За аналогією до контентного підходу, в колаборативному аналізі можна розглядати «користувачів-сусідів» (user-based CF) або «фільми-сусідів» (item-based CF). Наприклад, якщо двоє користувачів дають схожі оцінки, то фільми, улюблені одним, варто запропонувати іншому. Такий підхід відкриває шлях до «сюрпризних» рекомендацій, оскільки можна рекомендувати те, що не обов’язково має спільні ознаки з вподобаннями користувача, але подобалося схожим користувачам.

Окрім простих методів, часто застосовують матричні факторизації. Вони розкладають матрицю «користувач–фільм» на дві низькорозмірні матриці (матриці прихованих факторів користувачів та фільмів). Кожен користувач і фільм отримує латентний вектор ознак. Такі латентні фактори моделюють «приховані характеристики», наприклад уявні параметри жанру чи атмосфери

фільму. Основна роль латентних факторів – узгодження переваг користувача з атрибутами фільму. Наприклад, якщо вектор користувача має велике значення за «любовні сюжетні лінії», система буде рекомендувати фільми з аналогічною спрямованістю, навіть якщо вони не мають спільних явних тегів. Матриці факторизації, зокрема SVD (сингулярний розклад) з обліком зсувів (bias), широко використовуються в сучасних системах. Таким чином, гібридні рішення можуть поєднувати контентні та факторизаційні підходи, підвищуючи якість рекомендацій [1],[3],[4].

Обґрунтування рекомендацій (explanations) стає дедалі важливішим аспектом створення RS, оскільки простий список пропонованих фільмів часто залишає користувача в невіданні про причинність вибору. Згідно з Lee та Jung, для наративних творів (фільмів) питання «чому ця рекомендація» є критичним, тому потрібно виявляти характерні елементи сюжету чи стилю. Наприклад, система Netflix демонструє пояснення у вигляді «Оскільки ви переглянули Madam Secretary, вам може сподобатися …». У науковій літературі розрізняють пояснення, засновані на контенті (пояснення через схожі характеристики: жанр, актори, сюжет) та колаборативні (пояснення через дії схожих користувачів). Такі пояснення не лише покращують сприйняття рекомендацій, але й підвищують довіру користувача.

Загально кажучи, пояснювані системи намагаються врахувати когнітивні критерії користувача: що саме впливає на його вподобання і чому запропонований фільм задовольнить ці критерії. Наприклад, сюжетно-орієнтовані методи пояснення аналізують мережу персонажів або характеристик фільму, щоб знайти спільні риси з вподобаними користувачем стрічками. У нашій роботі також реалізовано простий механізм пояснень: система може виділяти ключові жанри або відомих акторів, що перетинаються з побажаннями користувача, або зазначати схожих за смаком користувачів [15].

У межах роботи було обрано гібридний підхід до побудови рекомендаційної системи, оскільки жоден окремий метод не забезпечує повного

розв’язання задачі персоналізованого підбору фільмів з поясненнями. Контентна фільтрація добре працює з описовими ознаками фільмів, такими як жанри, ключові слова, акторський склад, режисер і короткий опис сюжету. Вона дозволяє знаходити фільми, схожі на ті, які користувач уже оцінив позитивно. Проте її обмеженням є те, що рекомендації можуть бути надто схожими між собою та не завжди враховують колективний досвід інших користувачів.

Колаборативна фільтрація, навпаки, дозволяє враховувати закономірності в оцінках багатьох користувачів. Якщо користувачі мають схожу історію оцінювання, система може рекомендувати фільми, які сподобалися користувачам зі схожими смаками. Такий підхід дає змогу знаходити неочевидні рекомендації, які не завжди мають прямий збіг за жанрами або ключовими словами. Водночас колаборативна фільтрація залежить від кількості оцінок, тому для нових користувачів або маловідомих фільмів її ефективність знижується.

SVD-модель використовується для виявлення прихованих закономірностей у матриці оцінок. Вона дозволяє подати користувачів і фільми у латентному просторі, де схожість визначається не лише прямими оцінками, а й узагальненими факторами. У роботі реалізовано Classic SVD як базову факторизаційну модель та Personalized SVD, який додатково враховує близькість рекомендованих фільмів до вже вподобаних користувачем стрічок.

Таким чином, гібридний підхід дозволяє поєднати переваги кількох методів. Контентна складова забезпечує змістову відповідність рекомендацій, колаборативна складова враховує поведінку схожих користувачів, SVD-бейзлайн дає узагальнений прогноз очікуваної оцінки, а персоналізований SVD посилює зв’язок рекомендацій із конкретними вподобаннями користувача. Саме тому для розробленої системи було обрано не один окремий алгоритм, а комбіновану модель, яка дозволяє працювати як у сценарії cold-start, так і в сценарії warm-start.

1.3. Аналіз програмних засобів реалізації системи

Для реалізації системи рекомендацій фільмів з поясненнями було обрано мову програмування Python, оскільки вона має розвинену екосистему бібліотек для обробки даних, машинного навчання, побудови рекомендаційних моделей та створення інтерактивних веб-застосунків. Python дозволяє поєднати в одному програмному середовищі підготовку даних, навчання моделей, збереження артефактів, формування рекомендацій і відображення результатів у користувацькому інтерфейсі.

Для обробки табличних даних у роботі використано бібліотеку pandas. Вона застосовується для завантаження наборів даних про фільми, оцінки користувачів, ключові слова, акторський склад і додаткові метадані. За допомогою pandas виконуються фільтрація, об’єднання таблиць, очищення пропущених значень, узгодження ідентифікаторів фільмів та формування структур, придатних для подальшого навчання моделей. Бібліотека NumPy використовується для числових операцій, роботи з масивами та виконання допоміжних обчислень під час підготовки рекомендаційних ознак.

Для реалізації методів машинного навчання використано бібліотеку scikit-learn. У межах контентної фільтрації застосовано TfidfVectorizer, який перетворює текстові описи фільмів, жанри, ключові слова, акторів і режисерів у числове TF-IDF-представлення. Це дозволяє порівнювати фільми між собою за змістовими ознаками. Для обчислення подібності між векторними представленнями використовується косинусна подібність, яка дає змогу визначати, наскільки близькими є фільми за описом та іншими контентними характеристиками.

Для реалізації SVD-бейзлайну в роботі використано TruncatedSVD з бібліотеки scikit-learn. Цей підхід застосовується для побудови латентних представлень користувачів і фільмів на основі матриці оцінок. Завдяки цьому система може враховувати не лише явні характеристики фільмів, а й приховані

закономірності у поведінці користувачів. Для роботи з розрідженими матрицями використано бібліотеку scipy, оскільки матриця «користувач — фільм» містить значну кількість порожніх значень і потребує ефективного збереження та обробки.

Для збереження навчених моделей, матриць подібності, векторизаторів та інших артефактів використано joblib. Це дозволяє після етапу навчання не виконувати повторну підготовку моделей під час кожного запуску веб-застосунку, а завантажувати вже готові артефакти. Такий підхід підвищує швидкодію системи та робить її використання зручнішим для демонстрації.

Веб-інтерфейс користувача реалізовано за допомогою фреймворку Streamlit. Його використання є доцільним для цієї роботи, оскільки Streamlit дозволяє швидко створювати інтерактивні застосунки мовою Python без розробки окремої складної фронтенд-частини. У системі Streamlit використовується для вибору демонстраційного профілю, додавання та зміни оцінок фільмів, запуску генерації рекомендацій, порівняння різних моделей і перегляду пояснень до результатів.

Отже, обрані програмні засоби повністю відповідають задачам роботи. pandas і NumPy забезпечують підготовку та обробку даних, scikit-learn використовується для побудови TF-IDF-моделі, обчислення подібності та реалізації SVD-підходу, scipy забезпечує ефективну роботу з розрідженими матрицями, joblib використовується для збереження артефактів, а Streamlit забезпечує створення зручного веб-інтерфейсу. Таке поєднання інструментів дозволило реалізувати повний цикл роботи рекомендаційної системи: від підготовки даних і навчання моделей до формування пояснюваних рекомендацій у веб-застосунку [16].

2 МЕТОДИ ТА АЛГОРИТМИ ФОРМУВАННЯ ПОЯСНЮВАНИХ РЕКОМЕНДАЦІЙ

У другому розділі описуються методи та алгоритми, які використовуються для формування пояснюваних рекомендацій фільмів. Розглянуто етап підготовки вхідних даних про фільми, користувачів та їхні оцінки, а також формування контентних ознак на основі жанрів, ключових слів, описів, акторського складу та інших характеристик фільмів. Окрему увагу приділено побудові контентної моделі на основі TF-IDF та косинусної подібності, колаборативній фільтрації на основі матриці «користувач — фільм» і гібридному ранжуванню рекомендацій. Також у розділі розглядаються сценарії cold-start та warm-start, SVD-бейзлайн з урахуванням зміщень користувачів і фільмів, персоналізований SVD з пошуком сусідів у латентному просторі та принципи формування пояснень до результатів рекомендацій. Завершальна частина розділу присвячена метрикам оцінювання якості top-N рекомендацій, узгодженню алгоритмів у межах єдиного рекомендаційного процесу та вибору ваг гібридної моделі.

2.1. Підготовка вхідних даних про фільми, користувачів та оцінки

На початку необхідно зібрати і привести до єдиного формату дані про фільми, користувачів і їх рейтинги чи вподобання. У проєкті використано набір фільмів з описами (жанрами, ключовими словами, коротким описом сюжету) та матрицю оцінок користувачів (наприклад, дані MovieLens). Дані очищаються від пропусків: непотрібні або некоректні значення видаляються чи заповнюються. Текстові поля обробляються стандартно: видалення стоп-слів, перетворення на нижній регістр, токенізація, лематизація (за потреби) тощо.

Далі будується матриця «користувач–фільм», де кожна клітинка містить оцінку (або 1/0 для вподобання) даного користувача фільму. Цю матрицю використовують для колаборативних моделей. Окремо зберігаються контентні ознаки фільмів: наприклад, у вигляді текстової колонки «опис» або множини бінарних ознак жанрів. Для текстових описів підготуємо TF-IDF вектори (див. розділ 2.3), а для категоріальних (жанри, актори) – аналогічні бінарні чи частотні вектори.

На цьому етапі важливо поділити дані на навчальну та тестову вибірки для валідації моделей. Найчастіше це робиться по користувачах: залишаємо останні рейтинги кожного користувача для тесту і тренуємо моделі на залишених даних. Цей розподіл дозволяє перевірити, наскільки добре алгоритм прогнозує вподобання “загублених” фільмів для вже знайомих користувачів. Під час тренування ми також готуємо артефакти: наприклад, збережені слова для TF-IDF, матриці латентних факторів SVD тощо.

2.2. Формування контентних ознак фільмів

Контентні моделі побудовані на основі описових ознак фільмів. Систематично зібрані такі характеристики: жанри (драматичний, комедія, фантастика і т.д.), ключові слова/теги, опис сюжету, додаткові характеристики (режисер, актори), рейтинги з IMDb тощо. Якщо в описі фільму є текст (сюжетний опис або огляд), його можна перетворити в векторну форму.

У нашій системі основною контентною ознакою є TF-IDF-ваги слів опису фільму. Структуруюча інформація (жанри, актори) додається або через окремі матриці ознак (наприклад, багатоваріантна бінарна матриця для жанрів) або шляхом включення цих даних в об’єднаний текстовий опис (наприклад, додаємо назви жанрів до текстового опису перед побудовою TF-IDF). Таке поєднання дозволяє моделі врахувати як семантичний зміст, так і категорійні ознаки.

Також проводиться нормалізація даних: числові ознаки (наприклад, тривалість фільму чи середня оцінка) нормалізуються на стандартний діапазон. Це дозволяє поєднувати різнотипні ознаки у гібридних моделях. Після формування векторів ознак кожен фільм матиме свій представник у багатовимірному просторі контентних параметрів, придатний для розрахунку подібностей і подальшого ранжування рекомендацій [7],[8].

2.3. Контентна модель на основі TF-IDF та косинусної подібності

Для побудови контентної частини рекомендаційної системи у роботі використано текстове представлення фільмів. У межах підготовки даних для кожного фільму формується об’єднаний опис, до якого входять жанри, ключові слова, акторський склад, режисер, назва колекції, короткий опис сюжету та назва фільму. Такий підхід дає змогу представити фільм не лише через один текстовий опис, а через сукупність ознак, які краще характеризують його зміст і жанрову належність. У програмній реалізації ці ознаки об’єднуються в єдине текстове поле, після чого передаються до моделі векторизації.

Для перетворення текстових даних у числову форму використано підхід TF-IDF. Його призначення полягає в тому, щоб оцінити важливість окремих слів або словосполучень у межах опису конкретного фільму з урахуванням їх поширеності в усьому наборі фільмів. Якщо певне слово часто трапляється в описі одного фільму, але рідко використовується в описах інших, воно отримує більшу вагу і стає більш значущим для пошуку подібних фільмів. Якщо ж слово є загальним і зустрічається майже всюди, його вплив зменшується, оскільки воно не допомагає відрізнити один фільм від іншого. Саме тому TF-IDF добре підходить для задач, де потрібно визначити змістову близькість текстових описів.

У реалізації системи використано TfidfVectorizer із бібліотеки scikit-learn. Для зменшення кількості неінформативних ознак застосовано англомовні стоп-слова, обмеження максимальної кількості ознак і врахування не лише окремих слів, а й коротких словосполучень. Це дозволяє краще зберігати зміст фільмів, оскільки окремі слова не завжди передають повний контекст, тоді як словосполучення можуть точніше відображати жанр, тему або характер сюжету. Після навчання векторизатора кожен фільм отримує числове представлення, яке може бути використане для порівняння з іншими фільмами.

Після формування TF-IDF-представлень система обчислює подібність між фільмами за допомогою косинусної міри. У текстовому вигляді її можна пояснити як спосіб оцінити, наскільки близькими є напрями двох векторних описів. Якщо два фільми мають схожі жанри, ключові слова, акторів або сюжетні характеристики, їхні векторні представлення будуть близькими, а значення подібності буде високим. Якщо ж фільми істотно відрізняються за змістом, їхня подібність буде низькою. У результаті система може визначити, які фільми найбільш близькі до тих, що вже були високо оцінені користувачем.

Контентна модель використовується не лише для самостійного пошуку схожих фільмів, а й як одна зі складових гібридного ранжування. Якщо користувач уже поставив високі оцінки певним фільмам, система може сформувати його контентний профіль через ці фільми та знайти інші стрічки з подібними ознаками. Такий підхід особливо важливий для ситуацій, коли про користувача ще немає достатньої історії оцінювання. У цьому випадку контентна модель дозволяє надати рекомендації навіть без повноцінного колаборативного сигналу [16].

2.4. Колаборативна фільтрація на основі матриці «користувач — фільм»

Колаборативна фільтрація в роботі використовується для врахування поведінкових закономірностей користувачів. На відміну від контентного підходу, який аналізує властивості фільмів, колаборативний підхід спирається на оцінки, поставлені користувачами. Основна ідея полягає в тому, що користувачі зі схожою історією оцінювання можуть мати подібні смаки, тому фільми, які сподобалися одним користувачам, можуть бути рекомендовані іншим користувачам зі схожим профілем.

У програмній реалізації формується матриця, у якій рядки відповідають користувачам, а стовпці — фільмам. Значення в цій матриці є оцінками, які користувачі поставили відповідним фільмам. Оскільки кожен користувач оцінює лише невелику частину всього каталогу, така матриця є розрідженою. Це означає, що більшість комірок не містить значень, адже для більшості пар «користувач — фільм» оцінка невідома. Завдання колаборативної моделі полягає в тому, щоб використати наявні оцінки для пошуку схожих користувачів і прогнозування потенційно цікавих фільмів.

У реалізації системи використано підхід із центруванням оцінок користувачів. Це потрібно для того, щоб зменшити вплив індивідуального стилю оцінювання. Наприклад, один користувач може майже всім фільмам ставити високі оцінки, а інший — оцінювати суворіше. Якщо не враховувати такі особливості, система може неправильно визначити схожість між користувачами. Центрування дозволяє порівнювати не абсолютні оцінки, а відхилення від типового рівня оцінювання конкретного користувача.

Після цього між користувачами обчислюється косинусна подібність. Якщо два користувачі схоже оцінювали одні й ті самі фільми, вони вважаються близькими за вподобаннями. Для активного користувача система знаходить групу найближчих користувачів і враховує фільми, які вони оцінили високо. Таким

чином формується колаборативний сигнал, який потім використовується в гібридній моделі. Перевага цього підходу полягає в тому, що він дозволяє знаходити неочевидні рекомендації, які не завжди можна отримати лише через жанри або текстовий опис фільму.

2.5. Гібридне ранжування рекомендацій

Гібридне ранжування є центральним елементом розробленої системи, оскільки воно поєднує кілька джерел інформації про релевантність фільму. У роботі враховуються чотири основні компоненти: колаборативний сигнал, контентна подібність, популярність фільму та середня оцінка TMDB. Кожна з цих складових описує рекомендацію з іншого боку. Колаборативний сигнал показує, наскільки фільм відповідає поведінці схожих користувачів. Контентна складова показує, наскільки фільм близький за змістом до вже вподобаних фільмів. Популярність відображає загальну зацікавленість аудиторії, а TMDB-рейтинг дає додаткову оцінку якості фільму на основі зовнішніх метаданих.

Перед об’єднанням усі компоненти приводяться до спільного діапазону значень. Це потрібно тому, що різні показники мають різну природу: оцінки, популярність і подібність не можна коректно порівнювати без нормалізації. Після нормалізації кожен компонент отримує власну вагу, а підсумковий бал формується як узгоджене поєднання цих складових. У текстовому вигляді це можна описати так: система оцінює кожен фільм за кількома критеріями, визначає важливість кожного критерію та формує загальний рейтинг кандидата.

У проєкті передбачено автоматичний і ручний режими роботи з вагами. Автоматичний режим використовує ваги, підібрані під час навчання та оцінювання моделей. У збереженому звіті навчання найкращою комбінацією для гібридної моделі є співвідношення, за якого найбільший вплив має колаборативна складова, далі враховується контентна подібність, а популярність і TMDB-

рейтинг використовуються як допоміжні фактори. Ручний режим дозволяє змінити фокус рекомендацій через готові пресети: орієнтацію на схожих користувачів, схожий контент, популярні фільми або високий рейтинг.

Гібридний підхід дає змогу компенсувати обмеження окремих моделей. Контентна модель краще працює тоді, коли мало даних про користувача, але вона може звужувати рекомендації до дуже схожих фільмів. Колаборативна модель краще враховує колективний досвід, але потребує достатньої кількості оцінок. Додавання популярності й рейтингу допомагає стабілізувати результати, особливо для нових профілів. Завдяки такому поєднанню система може працювати як у режимі cold-start, так і в режимі warm-start.

2.6. Адаптація ваг моделі для сценаріїв cold-start та warm-start

Сценарії cold-start і warm-start вимагають гнучкої адаптації моделей. Cold-start (новий користувач) означає, що система має мало або зовсім немає попередньої інформації про даного користувача (немає оцінок). У такому випадку основна увага має бути приділена контентним сигналам: система більше спирається на жанрові чи описові характеристики фільму, а також може використовувати демографічні або поведінкові дані (якщо вони є). Наприклад, якщо користувач поділився своїми жанровими вподобаннями при реєстрації, система дає їм більше ваги.

У сценарії warm-start (існуючий користувач) є достатньо даних – рейтингів чи історії переглядів. Тоді колаборативна модель може брати гору: точніші передбачення можуть забезпечувати SVD-модель і колаборативна складова, які враховують сформовану історію оцінювання користувача, що враховують смак користувача, і контент стає допоміжним. Фактично, можна налаштовувати вагові коефіцієнти по мірі накопичення даних: спочатку контентна частка висока, потім поступово її зменшують, збільшуючи вагу колаборативної.

Схема адаптації у нашій системі наступна: при нових користувачах або якщо користувач має лише одну або кілька оцінок, система збільшує вагу контентної моделі. Коли ж користувач накопичує достатню статистику (warm-start), більшість рекомендацій ґрунтується на колаборативній складовій. Такий динамічний підхід дозволяє пристосовуватися до різних рівнів досвіду користувача.

2.7. SVD-бейзлайн з урахуванням зміщень користувачів і фільмів

Окрім гібридної моделі, у роботі реалізовано окремий SVD-бейзлайн, який використовується для порівняння результатів і демонстрації факторизаційного підходу. У фактичній реалізації застосовано TruncatedSVD з бібліотеки scikit-learn, а сама модель будується не безпосередньо на початкових оцінках, а на залишкових значеннях після врахування базових зміщень. Такий підхід дозволяє відокремити загальні закономірності оцінювання від прихованих факторів, які описують індивідуальні вподобання користувачів і властивості фільмів.

SVD-бейзлайн враховує кілька складових. Спочатку визначається загальний середній рівень оцінок у навчальних даних. Потім враховується індивідуальна схильність користувача ставити оцінки вище або нижче середнього рівня. Окремо враховується зміщення фільму, тобто його загальна тенденція отримувати вищі або нижчі оцінки від користувачів. Лише після цього модель аналізує залишкову частину оцінок і навчає латентні фактори, які відображають приховані закономірності у взаємодії користувачів і фільмів.

Такий підхід важливий, оскільки не всі високі оцінки пояснюються персональним смаком. Наприклад, деякі фільми загалом мають високу популярність і стабільно отримують високі оцінки від багатьох користувачів. Так само деякі користувачі можуть бути більш схильними ставити високі або низькі

оцінки незалежно від конкретного фільму. Якщо ці ефекти не врахувати, модель може переоцінювати популярні фільми або неправильно інтерпретувати стиль оцінювання користувача. Використання зміщень дозволяє зробити прогноз більш збалансованим.

У результаті Classic SVD формує прогноз очікуваної оцінки для кожного фільму-кандидата. У веб-інтерфейсі ця модель виводиться окремою колонкою, що дає змогу порівняти її поведінку з гібридною моделлю. На практиці Classic SVD часто рекомендує фільми з високою загальною оцінковою стабільністю, оскільки значну роль у прогнозі можуть відігравати зміщення фільмів. Саме тому в роботі додатково реалізовано Personalized SVD, який посилює персоналізацію через пошук близьких фільмів у латентному просторі.

2.8. Персоналізований SVD з пошуком сусідів у латентному просторі

Персоналізований SVD розширює базову факторизаційну модель і робить її ближчою до задачі top-N рекомендацій. Якщо Classic SVD насамперед прогнозує очікувану оцінку фільму, то Personalized SVD додатково враховує, наскільки фільм-кандидат пов’язаний із тими фільмами, які користувач уже оцінив високо. У реалізації для цього використовуються латентні представлення фільмів, отримані після навчання SVD-моделі.

Після навчання факторизаційної моделі кожен фільм має приховане числове представлення. У цьому просторі фільми, які мають схожі закономірності оцінювання користувачами, розташовуються ближче один до одного. Якщо користувач високо оцінив певний фільм, система може знайти інші фільми, які є близькими до нього у латентному просторі. Такі фільми формують пул кандидатів для персоналізованої SVD-моделі. Це дозволяє рекомендаціям

спиратися не лише на загальний прогноз рейтингу, а й на конкретні фільми, які вже сподобалися користувачу.

У підсумковому ранжуванні Personalized SVD враховує кілька сигналів: подібність до найближчого вподобаного фільму, приховану відповідність між профілем користувача і фільмом, прогнозовану оцінку та зміщення фільму. Найбільший вплив має саме подібність до вже вподобаного фільму, оскільки цей компонент робить рекомендацію більш персональною. Якщо ж користувач ще не має фільмів з високими оцінками, модель тимчасово використовує результати Classic SVD як резервний варіант.

Такий механізм особливо корисний для демонстрації переходу від загальних рекомендацій до персоналізованих. Коли новий користувач ще не має історії, система не може повноцінно знайти латентних сусідів. Після додавання кількох високих оцінок Personalized SVD починає використовувати їх як опорні точки. Завдяки цьому користувач може побачити, що рекомендації змінюються не випадково, а відповідно до внесених ним оцінок.

2.9. Формування пояснень до результатів рекомендацій

Для підвищення зрозумілості рекомендацій в систему інтегровано модуль пояснень. Ідея – після визначення топ-N фільмів пояснити, чому вони були обрані. Застосовані кілька стратегій пояснення:

Контентне пояснення: виділення ключових рис фільму, які збігаються з уподобаннями користувача. Наприклад, якщо користувач любить фільми певного жанру (драми з сильним сюжетом), у поясненні зазначаємо, що рекомендований фільм має аналогічний жанр та сюжети. Референтний приклад: Netflix часто використовує підписи виду «Тому що ви дивилися X».

Колаборативне пояснення: вказівка на дії схожих користувачів. Наприклад, «Цей фільм подивилися і високо оцінили користувачі з подібними смаками

(такими, що дивилися Y, Z)». Таким чином пояснення відносить рекомендацію до групи схожих спільнот.

Гібридні пояснення: поєднання контентних і соціальних аргументів, наприклад: «Рекомендується, оскільки це драма з акторами, яких ви любите, та її також радять користувачі схожі на вас».

Наші пояснення формуються на основі результатів аналізу фільму і користувача. Завдяки цьому користувач розуміє зв’язок між своїми вподобаннями і запропонованою стрічкою. Наприклад, система може вивести: «Система рекомендує цей фільм, тому що він в тому ж жанрі (екшен, пригоди) і має акторів, яких ви оцінювали раніше». Такі пояснення підтверджують раніше сформульовані тези про роль контентних і колаборативних підходів.

2.10. Метрики оцінювання якості top-N рекомендацій

Для оцінювання якості рекомендацій у роботі використано підхід holdout evaluation. Його суть полягає в тому, що для частини користувачів один із високо оцінених фільмів тимчасово приховується з історії оцінювання. Після цього система формує список рекомендацій, а якість перевіряється за тим, чи зміг алгоритм повернути прихований фільм у верхню частину списку. Такий підхід дає змогу перевірити, наскільки добре модель відтворює реальні вподобання користувачів на основі неповної інформації.

Основними метриками є HitRate, Recall, Precision, NDCG і Coverage. HitRate показує частку користувачів, для яких прихований релевантний фільм потрапив до списку рекомендацій. Recall у цьому сценарії має близьке значення до HitRate, оскільки для кожного користувача приховується один релевантний фільм. Precision показує, яка частка позицій у сформованому списку виявилася релевантною. NDCG додатково враховує позицію знайденого фільму: якщо релевантний фільм розташований ближче до початку списку, оцінка буде вищою.

Coverage показує, яку частину каталогу система фактично рекомендує різним користувачам.

У звіті навчання для гібридної моделі оцінювання виконано на вибірці з сорока користувачів. Найкраща комбінація ваг показала вищий результат за NDCG у порівнянні з окремим SVD-бейзлайном. Це свідчить про те, що поєднання колаборативного сигналу, контентної подібності, популярності та рейтингу дозволяє сформувати більш збалансований список рекомендацій. Окремий SVD-бейзлайн також оцінювався за top-N метриками та додатково за похибками прогнозування рейтингу, що дозволило порівняти якість ранжування і точність числового прогнозу [17].

2.11. Узгодження алгоритмів у межах єдиного рекомендаційного процесу

У межах розробленої системи окремі алгоритми рекомендацій не використовуються ізольовано, а об’єднуються в єдиний процес формування результату. Це важливо, оскільки кожен метод має власне призначення і по-різному реагує на кількість доступної інформації про користувача. Контентна модель краще працює тоді, коли потрібно врахувати зміст фільмів: жанри, опис, ключові слова, акторів, режисера та належність до певної колекції. Колаборативна модель є ефективною тоді, коли вже накопичено достатньо оцінок і можна знаходити закономірності в поведінці користувачів. SVD-модель дозволяє узагальнити ці закономірності та виявити приховані зв’язки між користувачами й фільмами.

Загальний процес роботи системи можна описати як послідовність кількох етапів. Спочатку система отримує інформацію про користувача: обраний демо-профіль, історичні оцінки, оцінки поточної сесії та параметри генерації рекомендацій. Далі визначається, скільки інформації вже доступно для

персоналізації. Якщо користувач новий або має дуже мало оцінок, система більше спирається на контентні ознаки та допоміжні загальні фактори. Якщо ж користувач уже оцінив достатню кількість фільмів, система активніше використовує колаборативні сигнали та результати SVD-моделей.

Після цього формується набір кандидатів, тобто фільмів, які потенційно можуть бути рекомендовані користувачу. Із цього набору виключаються фільми, які вже були оцінені або враховані в межах поточного профілю. Для кожного кандидата розраховується кілька типів оцінок: контентна релевантність, колаборативна релевантність, популярність і рейтингова характеристика. Такі компоненти не замінюють одна одну, а доповнюють загальну оцінку фільму. Наприклад, фільм може бути не найпопулярнішим у загальному каталозі, але мати високу подібність до вже вподобаних користувачем стрічок. Або навпаки, фільм може бути популярним і мати високий середній рейтинг, але слабко відповідати індивідуальному профілю користувача.

Гібридне ранжування дозволяє збалансувати ці фактори. У підсумковий список потрапляють не просто випадково популярні фільми, а ті, що мають достатньо високий сумарний показник з урахуванням профілю користувача. Саме тому система може працювати в різних сценаріях: для нового користувача, для користувача з кількома оцінками та для користувача з уже сформованою історією вподобань. Такий підхід також робить рекомендації більш стабільними, оскільки помилки одного алгоритму частково компенсуються сигналами інших компонентів.

Окреме значення має пояснюваність результатів. Після формування списку рекомендацій система не обмежується лише виведенням назв фільмів. Для кожної рекомендації формується текстове пояснення, яке показує, які саме фактори вплинули на появу фільму в добірці. Це може бути подібність за контентними ознаками, близькість до вже оцінених фільмів, високий прогноз моделі або внесок популярності. Завдяки цьому користувач бачить не лише кінцевий результат, а й логіку, за якою система прийшла до цього результату.

2.12. Обмеження використаних методів та способи їх компенсації

Незважаючи на переваги обраного підходу, кожен із використаних методів має певні обмеження. Контентна фільтрація добре пояснює рекомендації через ознаки фільмів, але може звужувати результати. Якщо користувач високо оцінює фільми певного жанру або конкретної серії, контентна модель буде схильна рекомендувати переважно схожі стрічки. Це зручно для точного підбору за інтересами, але може зменшити різноманітність рекомендацій і не завжди відкриває користувачу нові типи фільмів. Колаборативна фільтрація має інше обмеження: вона залежить від кількості оцінок. Якщо користувач новий, система ще не має достатньо інформації про його смаки. Якщо фільм рідко оцінювався, модель також не може надійно визначити його місце в структурі вподобань інших користувачів. Саме тому колаборативний підхід слабше працює в умовах холодного старту. Однак після накопичення оцінок він стає дуже корисним, оскільки дозволяє знаходити неочевидні зв’язки між користувачами й фільмами. SVD-модель також має свої особливості. Вона здатна виявляти приховані закономірності, але її результати складніше пояснити безпосередньо. Латентні фактори не завжди можна прямо назвати жанром, актором або конкретною характеристикою фільму. Через це SVD доцільно поєднувати з більш зрозумілими компонентами, наприклад із контентною моделлю та текстовими поясненнями. У такому випадку SVD підвищує якість ранжування, а контентні ознаки допомагають зробити результат зрозумілим для користувача. Для компенсації цих обмежень у роботі використовується гібридний підхід. Він дозволяє змінювати вагу окремих компонентів залежно від ситуації. Якщо користувач має мало оцінок, більшу роль відіграють контентні ознаки, популярність і рейтинг. Якщо оцінок достатньо, посилюється роль колаборативної моделі. Такий механізм не усуває всі недоліки окремих методів, але робить систему більш адаптивною та придатною для практичного

використання. Важливим обмеженням також є якість вхідних даних. Якщо описи фільмів неповні, жанри вказані неточно або оцінки користувачів нерівномірні, це впливає на якість рекомендацій. Тому підготовка даних є не менш важливою, ніж вибір алгоритму. Перед навчанням моделей потрібно очистити дані, узгодити ідентифікатори фільмів, прибрати некоректні значення та сформувати єдину структуру, з якою можуть працювати всі компоненти системи. Використання MovieLens-подібних даних є доцільним для такої задачі, оскільки вони містять реальні оцінки користувачів і дозволяють перевіряти рекомендаційні алгоритми в умовах, наближених до практичних досліджень. Таким чином, обрані методи доцільно розглядати не як окремі незалежні рішення, а як частини єдиного механізму. Контентна фільтрація забезпечує зрозумілість і роботу з ознаками фільмів, колаборативна фільтрація враховує досвід інших користувачів, SVD узагальнює приховані закономірності, а гібридне ранжування поєднує ці результати в одному списку рекомендацій.

2.13. Оцінювання результатів навчання та вибір ваг гібридної моделі

У розробленій системі важливим етапом є не лише навчання окремих моделей, а й оцінювання їхньої якості. Для цього у проєкті передбачено створення окремого звіту навчання, який зберігається у файлі training_report.json. Такий звіт виконує кілька функцій: фіксує параметри навчання, кількість оброблених даних, характеристики побудованих моделей, результати автоматичного підбору ваг і метрики якості. Завдяки цьому процес навчання є відтворюваним, а результати можна аналізувати після завершення роботи скрипта.

У процесі навчання використовувався набір ratings_small.csv. Після підготовки даних було сформовано каталог із 9082 фільмів, 99810 оцінок, 671

користувачем і 9025 фільмами, які мають хоча б одну оцінку. Такі дані достатні для демонстрації повного циклу роботи рекомендаційної системи, оскільки вони містять як метадані фільмів, так і взаємодії користувачів із фільмами. Використання MovieLens-подібних даних є доцільним для цієї задачі, оскільки такі набори широко застосовуються в дослідженнях рекомендаційних систем і дають можливість перевіряти алгоритми на реальних оцінках користувачів.

Оцінювання гібридної моделі виконувалося за top-N логікою. Для частини користувачів один фільм з високою оцінкою тимчасово вилучався з історії, після чого система формувала список рекомендацій. Якщо прихований фільм потрапляв у top-10, це вважалося успішним результатом. Така перевірка імітує реальну ситуацію, коли система повинна на основі неповної історії користувача передбачити, який фільм йому може сподобатися. Для оцінювання використовувались HitRate, Recall, Precision, NDCG і Coverage.

Окрему увагу приділено автоматичному підбору ваг гібридної моделі. У системі перевіряються різні комбінації ваг для collaborative, content, popularity і rating. Найкраща комбінація обирається насамперед за NDCG, потім за HitRate, а після цього за Coverage. Такий порядок вибору є логічним, оскільки для рекомендаційної системи важливо не лише повернути релевантний фільм у списку, а й розмістити його якомога вище. Саме тому NDCG є важливою метрикою для задачі ранжування, де позиція елемента має практичне значення.

За результатами навчання найкращою для гібридної моделі стала комбінація, у якій колаборативна складова має вагу 0,55, контентна складова — 0,25, популярність — 0,10, а рейтинг TMDB — 0,10. Такий результат показує, що для наявного набору даних найбільш значущим є поведінковий сигнал від схожих користувачів. Водночас контентна складова залишається важливою, оскільки вона забезпечує змістову відповідність фільмів профілю користувача. Популярність і рейтинг виконують допоміжну роль, стабілізуючи рекомендації та запобігаючи потраплянню до списку випадкових маловідомих кандидатів.

Гібридна модель у проведеному оцінюванні показала HitRate@10 на рівні 0,125, Recall@10 на рівні 0,125, Precision@10 на рівні 0,0125, NDCG@10 на рівні 0,072766 та Coverage@10 на рівні 0,026976. Classic SVD у цьому ж сценарії показав HitRate@10 на рівні 0,1, Recall@10 на рівні 0,1, Precision@10 на рівні 0,01, NDCG@10 на рівні 0,041005 та Coverage@10 на рівні 0,007267. Це свідчить про те, що гібридний підхід у межах цієї реалізації є більш збалансованим для top-N рекомендацій, ніж окремий SVD-бейзлайн. Особливо показовою є різниця за NDCG, оскільки вона відображає якість розташування релевантного фільму у списку.

Крім top-N метрик, для SVD-бейзлайну додатково оцінювалася точність прогнозування рейтингу через RMSE і MAE. Значення RMSE становило 0,748274, а MAE — 0,649344. Ці метрики показують середній рівень відхилення прогнозованих оцінок від фактичних. Проте важливо розуміти, що якість числового прогнозу рейтингу не завжди повністю відповідає якості top-N рекомендацій. У практичній рекомендаційній системі користувач бачить не всі прогнозовані оцінки, а лише верхню частину списку, тому ранжувальні метрики є більш показовими для оцінки кінцевого результату.

Таким чином, використання training_report.json дозволило не лише зберегти результати навчання, а й обґрунтувати вибір ваг гібридної моделі. У роботі було показано, що автоматичний підбір ваг допомагає адаптувати систему до конкретного набору даних, а порівняння з SVD-бейзлайном підтверджує доцільність використання гібридного підходу. Після додавання цього підрозділу поточні висновки до розділу потрібно перейменувати на «2.14 Висновки до розділу».

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

У третьому розділі розглядається програмна реалізація системи рекомендацій фільмів з поясненнями. Описано загальну архітектуру програмного комплексу, структуру проєкту та призначення основних модулів, які забезпечують підготовку даних, навчання моделей, генерацію рекомендацій, формування пояснень і роботу веб-інтерфейсу. Розділ містить опис моделі даних, початкових CSV-файлів, підготовлених таблиць та збережених артефактів навчання. Окремо розглянуто реалізацію pipeline підготовки даних і навчання моделей, використання TF-IDF, cosine similarity, TruncatedSVD, матриці «користувач — фільм» і збереження результатів за допомогою joblib. Також наведено опис логіки генерації рекомендацій, механізмів пояснення результатів, організації профілів користувачів, обробки поточної сесії та автоматичного підбору ваг гібридної моделі. Розділ демонструє, як теоретичні методи, описані раніше, реалізовано у вигляді цілісного програмного комплексу.

3.1. Загальна архітектура програмного комплексу

Загальна архітектура системи побудована за модульним принципом. У програмному комплексі виділено кілька основних складових блоків, які відповідають за підготовку даних, навчання моделей, ранжування рекомендацій, формування пояснень та взаємодію з користувачем. Блок підготовки даних забезпечує завантаження та обробку вхідних таблиць, а також формування матриці «користувач — фільм» і векторів TF-IDF. Блок навчання моделей відповідає за тренування моделей колаборативної фільтрації, зокрема SVD-бейзлайну, а також за побудову контентної моделі через обчислення TF-IDF-словника.

Блок ранжування рекомендацій виконує обчислення прогнозованих оцінок для невідомих пар «користувач — фільм» і визначає порядок відображення рекомендованих фільмів. Блок формування пояснень створює текстове обґрунтування для кожної рекомендації на основі аналізу ознак фільмів, поведінки користувача та внеску окремих компонентів моделі. Інтерфейс користувача реалізовано як модуль веб-застосунку на основі Streamlit, який забезпечує діалог із кінцевим користувачем, дозволяє обирати профіль, додавати оцінки, запускати генерацію рекомендацій і переглядати пояснення до отриманих результатів.

Взаємодія між компонентами організована таким чином, що дані та навчені артефакти зберігаються у файловій системі. Окремі модулі звертаються до підготовлених артефактів, зокрема до файлів із серіалізованими моделями та матрицями подібності. Веб-інтерфейс отримує результат від моделі через функцію рекомендування, після чого відображає користувачеві список фільмів і відповідні пояснення.

Така система дозволяє окремо оновлювати кожен блок, наприклад перевчати модель на нових даних без зміни інтерфейсу користувача. Для організації коду застосовано MVC-подібну структуру, у якій логіка отримання рекомендацій та пояснень відокремлена від представлення результатів у вигляді веб-сторінки.

3.2. Структура проєкту та призначення основних модулів

У проєкті реалізовано модульну структуру, у якій окремі файли відповідають за підготовку даних, навчання моделей, формування рекомендацій, збереження артефактів і запуск веб-інтерфейсу. Такий підхід спрощує супровід програмного забезпечення, оскільки кожна частина системи виконує окрему функцію та може бути змінена без повного переписування всієї системи.

Основним файлом веб-застосунку є app.py. Він відповідає за запуск інтерфейсу користувача, реалізованого за допомогою Streamlit, завантаження підготовлених моделей і даних, обробку дій користувача та відображення результатів рекомендацій. У цьому файлі реалізовано вибір демонстраційного профілю, роботу з оцінками фільмів, налаштування ваг гібридної моделі, порівняння результатів Hybrid model, Classic SVD та Personalized SVD, а також формування пояснень до рекомендованих фільмів.

Файл train.py є точкою запуску процесу навчання моделей. Його призначення полягає в тому, щоб викликати основний навчальний pipeline, реалізований у модулі src/train_models.py. Завдяки цьому користувач може запустити повну підготовку даних і навчання моделей однією командою, не звертаючись окремо до кожного внутрішнього модуля.

Модуль src/train_models.py є центральним модулем підготовки даних і навчання моделей. У ньому реалізовано завантаження початкових CSV-файлів із каталогу data, зокрема movies_metadata.csv, credits.csv, keywords.csv, ratings_small.csv та links_small.csv. У цьому модулі виконується очищення даних, узгодження ідентифікаторів фільмів, обробка жанрів, ключових слів, акторського складу, режисерів, колекцій фільмів і коротких описів. На основі цих даних формується об’єднане текстове поле content, яке надалі використовується для побудови контентної моделі.

У src/train_models.py також реалізовано навчання контентної моделі. Для цього використовується TfidfVectorizer з бібліотеки scikit-learn, який перетворює об’єднані текстові описи фільмів у TF-IDF-представлення. Після цього між фільмами обчислюється косинусна подібність, що дозволяє знаходити фільми зі схожими жанрами, ключовими словами, акторами, режисерами та описами. Результатом роботи цього етапу є збереження TF-IDF-векторизатора, таблиці підготовлених фільмів і матриці контентної подібності.

Для колаборативної частини в src/train_models.py формується матриця «користувач — фільм». У ній рядки відповідають користувачам, стовпці —

фільмам, а значення — оцінкам користувачів. Для коректнішого порівняння профілів використовується центрування оцінок користувачів, після чого обчислюється косинусна подібність між користувачами. Це дозволяє знаходити користувачів зі схожими вподобаннями та використовувати їхні оцінки як колаборативний сигнал під час формування рекомендацій.

У цьому ж модулі реалізовано SVD-бейзлайн на основі TruncatedSVD з бібліотеки scikit-learn. Модель враховує глобальний середній рейтинг, зміщення користувачів, зміщення фільмів і латентні фактори. Такий підхід дозволяє отримувати прогноз очікуваної оцінки фільму користувачем і використовувати його як окремий рекомендаційний режим Classic SVD. Також на основі латентних представлень фільмів реалізовано Personalized SVD, який шукає фільми, близькі до вже високо оцінених користувачем стрічок.

Модуль src/prepare_data.py призначений для окремого запуску етапу підготовки даних. Він викликає функцію load_and_prepare_data з модуля src/train_models.py, після чого створює файли processed_movies.csv та processed_ratings.csv у каталозі data. Цей модуль може використовуватися тоді, коли потрібно оновити лише підготовлені таблиці без повного повторного навчання всіх моделей.

Модуль src/content_recommender.py відповідає за окреме навчання контентної моделі. Він завантажує processed_movies.csv, будує TF-IDF-представлення фільмів, обчислює матрицю контентної подібності та зберігає відповідні артефакти в каталозі models. Такий модуль є корисним для перевірки або повторного створення лише контентної частини системи.

Модуль src/collaborative_recommender.py призначений для окремого навчання колаборативної моделі. Він завантажує processed_ratings.csv, формує матрицю «користувач — фільм», обчислює подібність між користувачами та зберігає user_movie_matrix.pkl і user_similarity.pkl. Ці артефакти надалі використовуються під час формування рекомендацій на основі поведінки схожих користувачів.

Модуль src/hybrid_recommender.py забезпечує окремий запуск гібридного рекомендування. Він завантажує підготовлені артефакти з каталогу models, зчитує оброблені рейтинги з data/processed_ratings.csv і формує top-N рекомендації для заданого користувача. У цьому модулі поєднуються колаборативний сигнал, контентна подібність, популярність і рейтингова характеристика фільму.

Модуль src/recommender.py реалізує базову контентну рекомендацію за назвою фільму. Він завантажує movies.pkl і content_similarity.pkl, знаходить фільм за частиною назви та повертає список найбільш схожих фільмів. Цей модуль використовується як спрощений приклад роботи контентного підходу, коли рекомендація формується не за профілем користувача, а за схожістю до конкретного фільму.

Каталог data містить початкові та підготовлені дані. До початкових файлів належать movies_metadata.csv, credits.csv, keywords.csv, ratings.csv, ratings_small.csv, links.csv та links_small.csv. Після підготовки створюються processed_movies.csv і processed_ratings.csv, які використовуються під час навчання моделей і запуску веб-застосунку. Окремо в data/posters зберігаються постери фільмів, що використовуються для візуального відображення рекомендацій в інтерфейсі.

Каталог models містить збережені артефакти навчання. До них належать movies.pkl, tfidf.pkl, content_similarity.pkl, user_movie_matrix.pkl, user_similarity.pkl, svd_baseline.pkl і training_report.json. Файли у форматі pkl зберігаються за допомогою joblib і використовуються для швидкого завантаження моделей без повторного навчання. Файл training_report.json містить інформацію про параметри навчання, результати оцінювання, підібрані ваги гібридної моделі та характеристики використаних даних.

Файл requirements.txt містить перелік основних бібліотек, необхідних для запуску проєкту: pandas, NumPy, scikit-learn, scipy, joblib і Streamlit. Файл TRAINING.md містить інструкції щодо запуску процесу навчання, опис

створюваних артефактів, параметри командного запуску, автоматичний підбір ваг і пояснення метрик оцінювання якості рекомендацій.

Отже, структура проєкту побудована таким чином, щоб відокремити підготовку даних, навчання моделей, генерацію рекомендацій і роботу веб-інтерфейсу. Основним сценарієм навчання є запуск train.py, а основним сценарієм взаємодії користувача із системою є запуск app.py через Streamlit. Збереження моделей і проміжних результатів у каталозі models дозволяє повторно використовувати навчені артефакти та забезпечує стабільну роботу рекомендаційної системи.

3.3. Модель даних і збережені артефакти навчання

Дані про фільми зберігаються у файлах movies_metadata.csv, credits.csv, keywords.csv та links_small.csv. У них містяться ідентифікатори фільмів, назви, жанри, короткі описи, популярність, середні оцінки, ключові слова, акторський склад, режисери та зв’язки між ідентифікаторами MovieLens і TMDB.

Дані про користувацькі оцінки зберігаються у файлі ratings_small.csv у форматі userId, movieId, rating, timestamp. Після первинного завантаження ці файли обробляються та перетворюються у структури pandas DataFrame, а також у підготовлені таблиці processed_movies.csv і processed_ratings.csv, які надалі використовуються для навчання моделей і формування рекомендацій.

Після навчання моделей у каталозі models зберігаються основні артефакти, необхідні для подальшої роботи системи. До них належать movies.pkl, tfidf.pkl, content_similarity.pkl, user_movie_matrix.pkl, user_similarity.pkl, svd_baseline.pkl та training_report.json. Файл movies.pkl містить підготовлені дані про фільми, tfidf.pkl зберігає навчений TF-IDF-векторизатор, а content_similarity.pkl містить матрицю контентної подібності між фільмами. Файли user_movie_matrix.pkl та user_similarity.pkl використовуються для колаборативної складової, оскільки

містять матрицю «користувач — фільм» і матрицю подібності користувачів. Файл svd_baseline.pkl містить дані SVD-бейзлайну, зокрема зміщення користувачів і фільмів, латентні представлення та допоміжні параметри моделі. Файл training_report.json містить зведену інформацію про процес навчання, результати оцінювання моделей і підібрані ваги гібридного ранжування.

Модель даних для поточної взаємодії користувача зберігається не у вигляді окремого файлу користувацьких профілів, а через стан сесії Streamlit. У цьому стані фіксуються обраний демонстраційний профіль, додані або змінені оцінки, події поточної сесії та результати останньої генерації рекомендацій. Завдяки цьому система може враховувати дії користувача під час поточної роботи із застосунком і оновлювати рекомендації без повторного навчання моделей [12],[19].

3.4. Реалізація pipeline підготовки даних і навчання моделей

Pipeline підготовки даних і навчання моделей реалізовано через файл train.py, який запускає основну логіку з модуля src/train_models.py. У межах цього процесу система завантажує початкові CSV-файли з каталогу data, виконує очищення даних, узгодження ідентифікаторів фільмів, обробку жанрів, ключових слів, акторського складу, режисерів і коротких описів. Після цього створюються підготовлені таблиці processed_movies.csv та processed_ratings.csv, які використовуються для подальшого навчання моделей.

Для побудови контентної моделі використовується TfidfVectorizer з бібліотеки scikit-learn. Він перетворює об’єднані текстові описи фільмів у TF-IDF-представлення. На основі цих представлень обчислюється cosine_similarity, що дозволяє визначати змістову подібність між фільмами. Для колаборативної частини формується матриця «користувач — фільм», після чого обчислюється подібність між користувачами на основі їхніх оцінок.

SVD-бейзлайн реалізовано за допомогою TruncatedSVD з бібліотеки scikit-learn. Модель використовується для побудови латентних представлень фільмів і формування прогнозів очікуваних оцінок. Після завершення навчання основні артефакти зберігаються за допомогою joblib у каталозі models. До них належать TF-IDF-векторизатор, матриця контентної подібності, матриця «користувач — фільм», матриця подібності користувачів і SVD-бейзлайн. Додатково формується файл training_report.json, у якому зберігаються результати навчання, оцінювання моделей і підібрані ваги гібридного ранжування [10],[9].

3.5. Реалізація алгоритмів генерації рекомендацій

Функція генерації рекомендацій приймає ID користувача і повертає список топ-N фільмів із поясненнями. Спочатку вибирається алгоритм в залежності від сценарію: якщо мало даних – використовується тільки контентний підхід, інакше – гібрид. Для кожної кандидатної пари (користувач, фільм) обчислюється оцінка:

Контентна оцінка: наприклад, косинусна подібність середнього профілю користувача (який може формуватися як середнє TF-IDF оцінених фільмів) з TF-IDF фільму.

CF-оцінка: для SVD моделі використовується прогноз очікуваної оцінки фільму конкретним користувачем.

Для колаборативної складової використовується середньозважена оцінка на основі схожих профілів користувачів.

Підсумкова оцінка: лінійна комбінація зважених значень двох складових.

Рейтинги всіх можливих фільмів, що не були переглянуті користувачем, обчислюються одночасно (використовуючи бібліотеки для векторних обчислень чи оптимізований цикл). Після цього результати сортуються за спаданням підсумкового показника, і до фінального списку потрапляє задана кількість найрелевантніших фільмів.

Для кожного рекомендованого фільму одразу формується пояснення на основі факторів, які вплинули на його позицію у списку. Якщо рекомендація пов’язана з контентною складовою, пояснення формується через спільні ознаки фільмів, зокрема жанри, ключові слова, акторський склад або режисера. Якщо рекомендація пов’язана з колаборативною складовою, у поясненні враховується поведінка користувачів зі схожими вподобаннями. Для SVD-підходів додатково враховується прогнозована оцінка та зв’язок із фільмами, які користувач уже оцінив позитивно.

Важливо, що весь цей процес реалізовано з кешуванням: якщо одного користувача запускається рекомендація багато разів без змін профілю, результати швидко дістаються з кешу [1],[2],[8],[3],[18].

3.6. Реалізація механізмів пояснення результатів

Механізм пояснення побудовано на правилах, заснованих на характеристиках даних. Для кожного рекомендованого фільму розглядаються джерела його рейтингу:

Контентні ознаки: з аналізу фільму витягуються жанри і найбільш значущі ключові слова (вигляд: топ-3 за вагами TF-IDF). Якщо у профілі користувача вже містилися фільми з цим жанром, генеруємо фразу «Рекомендується, тому що обидва фільми мають жанр X» або «ці фільми містять спільні ключові слова Y, які ви високо оцінили».

Схожі користувачі: знаходимо k найближчих користувачів (за профілем оцінок, наприклад, в латентному просторі SVD) і перевіряємо, чи вони високо оцінили рекомендований фільм. Якщо так, то виписуємо: «Користувачі зі схожими оцінками до ваших також поставили високу оцінку цьому фільму».

Спільні атрибути: наприклад, якщо і рекомендований фільм, і один з улюблених користувача містять одного виконавця чи режисера, можемо згадати це як фактор «має того ж актора».

Статистика: у випадках, коли рекомендується дуже популярний фільм, пояснення може бути формально-статистичним: «Цей фільм переглянули та високо оцінили 90% користувачів з вашою демографією».

Система автоматично комбінує і ранжує ці аргументи, щоб сформувати одне сумарне пояснення. Згідно з попереднім дослідженням, навіть коротка фраза, схожа на «Бо ви дивилися X» або перелік 1–2 характеристик, суттєво підвищує задоволеність користувача. Функція explain_recommendation(user, movie) повертає рядок з поясненням.

Приклад: користувачу рекомендовано фільм «Inception»; його пояснення: «Рекомендується, оскільки цей фільм із жанрів: Sci-Fi, Action, та тут грає актор Leonardo DiCaprio, якого ви оцінили у фільмі Titanic». Таке пояснення дає зрозуміти основу рекомендації: жанр і актор пов’язують два фільми, що допомагає користувачеві прийняти чи відхилити рекомендацію [5],[6].

3.7. Реалізація веб-інтерфейсу користувача засобами Streamlit

Інтерфейс реалізовано на основі Streamlit – відкритого Python-фреймворку для швидкої побудови веб-додатків. Streamlit дозволяє описувати UI у скрипті, і він автоматично генерує веб-сторінку. У нашому проєкті головний файл app.py складається з декількох компонентів: інпут для вибору користувача (випадаюче меню з існуючими IDs), форми для додавання оцінок, кнопки «Показати рекомендації», та область виводу результатів.

При старті додатка виконується механізм кешування Streamlit завантаження всіх артефактів (моделей, словників) у пам’ять, щоб не робити це знову при кожній зміні UI. Далі користувач може увійти: якщо вибирає існуючий ID,

система завантажує збережений профіль, якщо задає новий – може надати базові рекомендації на основі заданих жанрів чи початкових оцінок (cold-start). Після натискання кнопки «Рекомендувати» викликається функція згенерації рекомендацій і пояснень, результати виводяться в тій же сторінці – у вигляді таблиці з колонками «Фільм», «Оцінка (прогноз)», «Чому?». Таблиці Streamlit автоматично рендеряться в HTML-таблиці.

Додатково, інтерфейс містить можливість фільтрування жанрів: користувач може відмітити жанри, які йому подобаються, і система підкреслює їх у поясненнях. Також реалізовано можливість оновлення профілю: введені оцінки зберігаються, і при повторній сесії з тим самим користувачем результат буде враховано нові оцінки.

Таким чином, Streamlit показав себе ефективним інструментом: не потрібно створювати шаблони HTML чи маршрути, можна просто описувати логіку і візуальні елементи через Python. Створені компоненти виглядають як звичні елементи веб-сторінки, але вони генеруються з коду на льоту [11].

3.8. Організація профілів користувачів, оцінок і стану сесії

Профіль користувача у веб-застосунку формується на основі обраного демонстраційного профілю, історичних оцінок і дій, виконаних під час поточної сесії. У межах сесії зберігаються оцінені фільми, значення оцінок, обрані параметри рекомендацій та результати останньої генерації. Такий підхід дозволяє системі враховувати нові дії користувача без повторного навчання моделей.

Щодо стану сесії: Streamlit пропонує механізм st.session_state для зберігання змінних між перезавантаженнями сторінки. Ми використовуємо його для збереження тимчасових даних (наприклад, поточного профілю чи фільтрів). Це означає, що якщо користувач оновить сторінку або перейде між вкладками додатку, його введені оцінки не зникнуть. Після завершення роботи користувача

сесія завершується, але профіль можна зберегти у файл, якщо потрібно зберегти інформацію для наступних входів.

Таким чином, система гарантує збереження стану взаємодії до моменту закриття програми, а профілі користувачів при необхідності можна експортовувати/імпортувати (наприклад, у CSV) для аналізу. Це дозволяє підтримувати послідовність рекомендацій: однаковий профіль – однаковий результат [19].

3.9. Логіка взаємодії користувача з рекомендаційною системою

У програмній реалізації важливим є не лише сам алгоритм рекомендацій, а й спосіб взаємодії користувача із системою. Користувач повинен мати можливість поступово формувати власний профіль, переглядати доступні фільми, додавати оцінки, змінювати їх і після цього отримувати оновлені рекомендації. Саме така логіка відповідає практичному сценарію роботи рекомендаційних сервісів, де результат залежить від дій користувача та накопиченої історії його вподобань.

Після входу до системи користувач працює з одним із підготовлених демо-профілів. У реалізації передбачено декілька демонстраційних користувачів, а також окремий новий профіль, який використовується для перевірки сценарію cold-start. Такий підхід дозволяє показати різницю між користувачем, який уже має історію оцінювання, і користувачем, який тільки починає взаємодію із системою. У межах активного профілю зберігаються оцінені фільми, поточний стан сесії та результати останньої генерації рекомендацій.

Якщо користувач ще не має оцінок, система працює в режимі холодного старту. У цьому випадку рекомендації формуються переважно на основі загальних характеристик фільмів, їхньої популярності, рейтингу та доступних

контентних ознак. Після того як користувач оцінює кілька фільмів, система отримує більше інформації про його смаки та може перейти до більш персоналізованого режиму. Це видно в роботі застосунку: після додавання оцінок список рекомендацій змінюється, а пояснення починають активніше посилатися на конкретні вподобані користувачем фільми.

Особливістю реалізації є те, що зміни в діях користувача одразу впливають на рекомендаційний результат. Якщо користувач додає нову оцінку, змінює попередню або видаляє її, профіль оновлюється, а наступна генерація рекомендацій уже враховує змінені дані. Такий підхід дозволяє продемонструвати динамічність системи: рекомендації не є статичним списком, а змінюються відповідно до поведінки користувача.

Інтерфейс Streamlit використовується як зручний спосіб показати роботу системи без розробки окремого складного фронтенду. Його перевага полягає в тому, що логіка Python-коду, моделі рекомендацій і елементи інтерфейсу можуть бути об’єднані в одному застосунку. Це спрощує демонстрацію результатів, оскільки користувач може безпосередньо взаємодіяти із системою: натискати кнопки, обирати фільми, змінювати оцінки та одразу бачити оновлений список рекомендацій.

3.10. Обробка поточної сесії та збереження дій користувача

Для коректної роботи рекомендаційної системи необхідно зберігати стан поточної сесії. Це потрібно для того, щоб система пам’ятала, які дії вже виконав користувач: який профіль обрано, які фільми були оцінені, які рекомендації сформовані та які параметри ранжування використовувалися. Без такого механізму інтерфейс оновлювався б після кожної дії, а користувач втрачав би результати попередніх кроків.

У реалізації стан сесії використовується для зв’язку між інтерфейсом і рекомендаційною логікою. Коли користувач додає оцінку фільму, ця дія фіксується в поточному профілі. Коли він запускає генерацію рекомендацій, система звертається до оновлених даних і формує список фільмів відповідно до актуального стану. Якщо користувач змінює оцінку, попереднє значення замінюється новим, а рекомендаційний профіль фактично перебудовується. Це дозволяє перевірити, як саме зміна вподобань впливає на кінцевий результат.

Окремо в системі фіксуються події поточної сесії. Це можуть бути дії додавання, зміни або видалення оцінки. Відображення таких подій у боковій панелі робить взаємодію прозорішою: користувач бачить, які саме кроки були виконані та як вони впливають на подальше формування рекомендацій. Для демонстраційної роботи це особливо важливо, оскільки дозволяє наочно показати, що система не просто виводить заздалегідь підготовлений список, а реагує на фактичні дії користувача.

Також стан сесії корисний для демонстрації пояснень. Пояснення до рекомендацій мають бути пов’язані з конкретними діями користувача. Наприклад, якщо система рекомендує фільм через схожість із раніше оціненими стрічками, вона повинна мати доступ до списку цих оцінених фільмів. Якщо рекомендація формується через гібридне ранжування, у поясненні можна вказати, які компоненти зробили найбільший внесок: контентна подібність, колаборативний прогноз, популярність або середній рейтинг.

У системі реалізовано порівняння трьох рекомендаційних підходів у межах одного інтерфейсу. Гібридна модель показує результат поєднання різних сигналів, Classic SVD демонструє поведінку факторизаційної моделі, а Personalized SVD показує рекомендації, посилені пошуком сусідів у латентному просторі. Така організація інтерфейсу дозволяє не лише отримати список фільмів, а й порівняти, як різні алгоритмічні підходи реагують на один і той самий профіль користувача.

Завдяки такій організації система стає не просто набором алгоритмів, а повноцінним інтерактивним застосунком. Користувач бачить причинно-наслідковий зв’язок між власними діями та результатом: спочатку він оцінює фільми, потім система аналізує ці оцінки, після чого формує персоналізовану добірку й пояснює її. Це відповідає головній меті роботи — створити систему рекомендацій фільмів не лише з персоналізацією, а й із зрозумілим поясненням отриманих результатів [19].

3.11. Автоматичний підбір ваг та формування звіту навчання

Однією з важливих особливостей програмної реалізації є автоматичний підбір ваг гібридної моделі. У системі рекомендація формується не одним показником, а поєднанням collaborative, content, popularity і rating. Якщо задати ці ваги випадково, система може надмірно спиратися на один фактор і втрачати баланс. Саме тому в навчальному pipeline передбачено перевірку кількох комбінацій ваг, після чого система обирає ту, яка показала найкращий результат за метриками оцінювання.

У коді реалізовано набір кандидатних комбінацій ваг. Для кожної комбінації виконується оцінювання за holdout-підходом. Система приховує один високо оцінений фільм для частини користувачів, формує рекомендації та перевіряє, чи потрапив прихований фільм до top-10. Після цього розраховуються HitRate, Recall, Precision, NDCG і Coverage. Найкраща комбінація визначається за NDCG, потім за HitRate, а потім за Coverage. Така послідовність вибору дозволяє надати пріоритет якості ранжування, а не лише самому факту потрапляння релевантного фільму до списку.

Після завершення навчання результати зберігаються у training_report.json. У цьому файлі міститься дата навчання, конфігурація запуску, кількість оброблених фільмів, оцінок і користувачів, опис побудованих моделей, підібрані ваги,

результати підбору ваг і порівняння гібридної моделі з SVD-бейзлайном. Наявність такого файлу є важливою для дипломного проєкту, оскільки дозволяє показати, що параметри системи були не лише задані вручну, а й перевірені на даних.

Звіт навчання також використовується веб-застосунком. Під час запуску Streamlit-інтерфейс завантажує training_report.json і використовує автоматично підібрані ваги як базові. Користувач може залишити автоматичний режим або вимкнути його й обрати один із ручних режимів фокусування рекомендацій. Наприклад, можна зробити акцент на схожих користувачах, схожому контенті, популярності або високому TMDB-рейтингу. Така реалізація робить систему гнучкою, оскільки вона підтримує як автоматичне ранжування, так і демонстраційне налаштування поведінки алгоритму.

Використання автоматичного підбору ваг є важливою перевагою розробленої системи. Воно дозволяє уникнути повністю суб’єктивного вибору параметрів і показує, що остаточна конфігурація гібридної моделі ґрунтується на результатах перевірки. У межах навчального pipeline це також робить систему більш відтворюваною: при зміні даних або параметрів навчання можна повторно запустити train.py і отримати оновлений звіт з новими вагами.

3.12. Реалізація пояснень через внесок окремих факторів

У розробленій системі механізм пояснень реалізовано не лише у вигляді загального тексту, а через показ внеску окремих факторів у підсумковий результат. Для гібридної моделі система обчислює normalized collaborative score, normalized content score, normalized popularity score і normalized rating score. Після цього кожен показник множиться на відповідну вагу, а підсумковий результат використовується для ранжування фільмів. Така структура дає змогу пояснити рекомендацію через складові, які безпосередньо вплинули на її позицію.

У веб-інтерфейсі ці складові відображаються у картці рекомендованого фільму. Користувач бачить фінальний score, значення collaborative, content, popularity і TMDB rating, а також коротке пояснення домінуючого фактора. Наприклад, якщо найбільший позитивний внесок зробила контентна складова, система показує, що рекомендація пов’язана зі схожістю змісту. Якщо більший внесок має collaborative, рекомендація пояснюється поведінкою схожих користувачів. Якщо позитивний внесок має popularity або rating, система показує, що фільм має сильний загальний сигнал.

Для формування пояснень використовується ідея порівняння значення фактора для конкретного фільму із середнім рівнем серед кандидатів. Якщо певний показник вищий за середній рівень, він вважається позитивним внеском у рекомендацію. Якщо нижчий — його внесок є слабшим або негативним. Завдяки цьому пояснення не зводиться до простого повторення оцінки, а показує, чим саме рекомендований фільм відрізняється від інших кандидатів.

Контентне пояснення доповнюється пошуком спільних TF-IDF-термінів між рекомендованим фільмом і тими фільмами, які користувач уже високо оцінив. Це дозволяє системі показати не лише числову подібність, а й конкретні слова або словосполучення, які об’єднують фільми. Наприклад, якщо рекомендований фільм і вподобаний фільм мають спільні жанрові або сюжетні терміни, система може використати це як текстовий доказ контентної близькості. Такий підхід узгоджується з призначенням TF-IDF, який використовується для перетворення текстових документів на матрицю ознак.

Колаборативне пояснення формується через аналіз схожих користувачів і session-collaborative сигналу. Якщо в історичних даних знайдено користувачів зі схожою поведінкою, система може пояснити рекомендацію через їхні позитивні оцінки. Якщо ж у поточній сесії з’являються нові оцінки, вони також можуть впливати на колаборативний сигнал. Це особливо важливо для демонстраційних профілів, оскільки дає змогу показати, як нові дії користувача змінюють не тільки список рекомендацій, а й пояснення до них.

Для Classic SVD пояснення будується навколо прогнозованої оцінки. Користувачу показується, що результат складається з базового середнього рівня, зміщення користувача, зміщення фільму та латентної відповідності. Для Personalized SVD пояснення доповнюється інформацією про найближчий вподобаний фільм у латентному просторі. Це робить SVD-модель більш зрозумілою, хоча її внутрішні фактори залишаються менш інтерпретованими, ніж жанри або ключові слова. Використання TruncatedSVD є доцільним для таких задач, оскільки цей метод працює з розрідженими матрицями та виконує зниження розмірності без необхідності щільного представлення даних.

4 МЕТОДИКА РОБОТИ КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

У четвертому розділі наведено демонстрацію роботи розробленої системи рекомендацій фільмів з поясненнями. Розглянуто процес входу до веб-застосунку, вибір демонстраційного профілю, структуру головного меню користувача та основні елементи інтерфейсу. Окремо показано сценарій формування рекомендацій для нового користувача, який не має достатньої історії оцінювання, а також сценарій поступового переходу від cold-start до персоналізованих рекомендацій після додавання користувацьких оцінок. У розділі також описано можливість додавання, зміни та видалення оцінок фільмів, порівняння Hybrid model, Classic SVD та Personalized SVD, а також перегляд пояснень до рекомендованих фільмів. Завершальна частина розділу присвячена аналізу отриманих результатів демонстрації та підтвердженню того, що реалізована система виконує поставлені функціональні вимоги.

4.1. Вхід до системи та вибір демонстраційного профілю

Перед початком роботи з рекомендаційною системою користувач потрапляє на сторінку авторизації. На цій сторінці представлено назву застосунку, коротке пояснення його призначення та форму входу через демо-профіль. Користувач може обрати один із доступних профілів, ввести пароль і натиснути кнопку «Увійти». Оскільки система має демонстраційний характер, окрема реєстрація нового облікового запису не передбачена, а авторизація виконується через заздалегідь підготовлені профілі. Такий підхід дозволяє швидко перейти до перевірки персоналізованих рекомендацій і продемонструвати роботу моделей для різних типів користувачів.

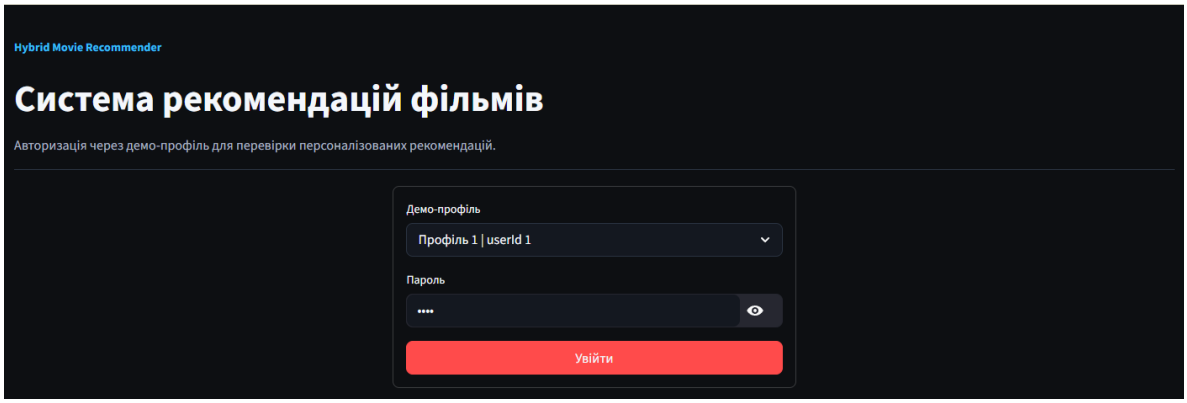

Рисунок 4.1 – Сторінка входу до системи рекомендацій фільмів

4.2. Головне меню користувача та структура інтерфейсу

Після авторизації користувач переходить до основного інтерфейсу системи рекомендацій. На цій сторінці відображається робоче середовище поточного профілю, у межах якого надалі виконуються всі основні дії: оцінювання фільмів, налаштування параметрів рекомендацій та генерація персоналізованої добірки. Для демонстрації роботи системи використовується новий демо-профіль, який не має попередньої історії оцінювання. Такий сценарій дозволяє показати, як рекомендаційна система працює з новим користувачем і як результати змінюються після поступового додавання оцінок у поточній сесії.

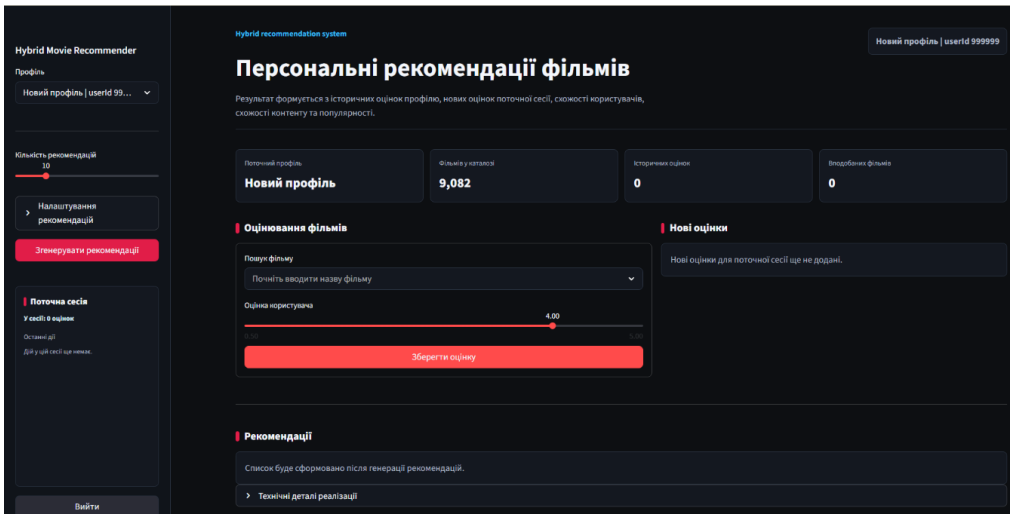

Рисунок 4.2 – Основна сторінка нового профілю користувача

4.3. Формування рекомендацій для нового користувача

Після переходу до основного інтерфейсу користувач може натиснути кнопку «Згенерувати рекомендації». У результаті система формує початковий список рекомендованих фільмів для нового профілю. На екрані результати подано у трьох окремих блоках, що відповідають різним підходам до формування рекомендацій: гібридній моделі, класичній SVD-моделі та персоналізованій SVD-моделі. Такий формат дозволяє порівняти роботу різних рекомендаційних механізмів у межах одного інтерфейсу.

Оскільки новий профіль ще не має власної історії оцінювання, система не може повною мірою використати персональні вподобання користувача. Зокрема, персоналізована SVD-модель у цьому випадку не має достатньої кількості фільмів, які користувач оцінив би високо, тому тимчасово використовує результати класичної SVD-моделі як базовий варіант. Таким чином, на початковому етапі рекомендації мають менш персоналізований характер, а подальше додавання оцінок дозволить системі точніше адаптувати результати до смаків конкретного користувача.

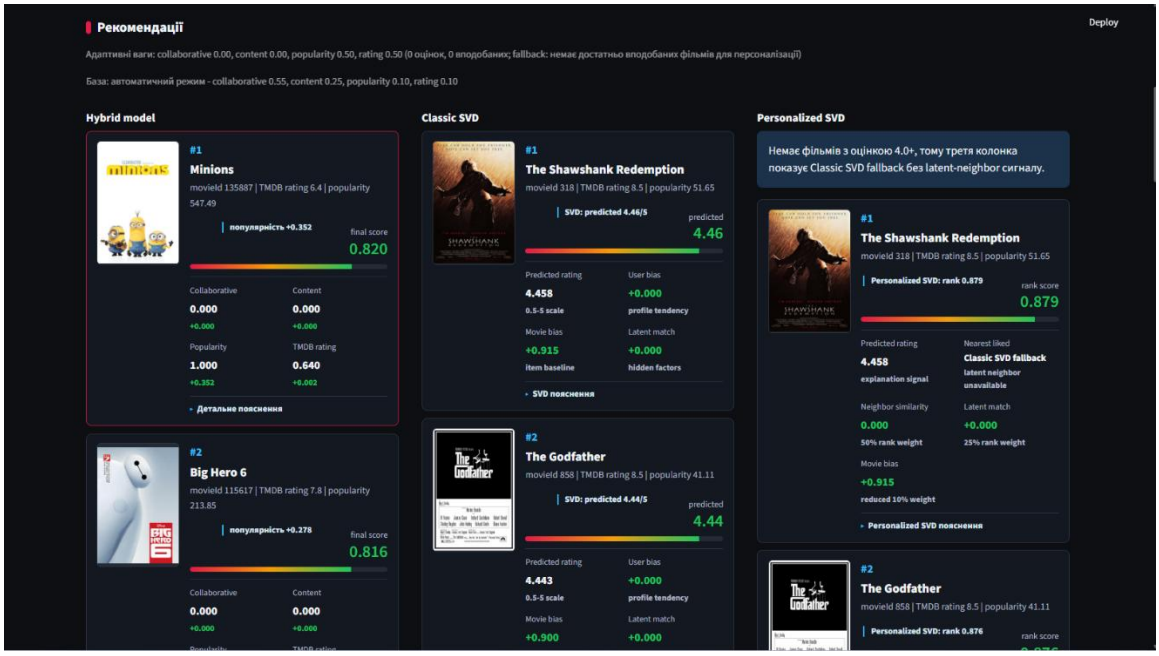

Рисунок 4.3 – Початкова генерація рекомендацій для нового профілю

4.4. Додавання, зміна та видалення оцінок фільмів

Однією з ключових функцій системи є можливість додавання користувацьких оцінок для фільмів. У відповідному блоці користувач може знайти потрібний фільм, вибрати значення оцінки за встановленою шкалою та зберегти її в межах поточної сесії. Після цього система відображає повідомлення про успішне збереження оцінки, а також додає відповідний запис до таблиці нових оцінок.

Наявність цього механізму є принципово важливою, оскільки саме введені користувачем оцінки надалі використовуються системою для формування більш персоналізованих рекомендацій.

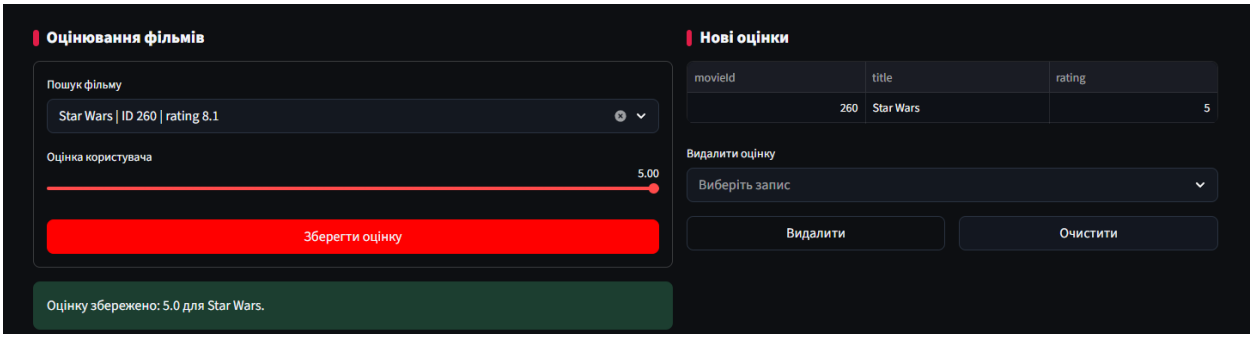

Рисунок 4.4 – Збереження користувацької оцінки для обраного фільму

Система підтримує не лише додавання нових оцінок, а й подальше редагування та видалення вже створених записів. На поданому етапі демонстрації користувач працює з раніше оціненим фільмом, змінює значення оцінки, а за потреби може повністю видалити її з поточної сесії. Після виконання відповідної дії система відображає службове повідомлення про успішне видалення запису, а блок нових оцінок оновлюється відповідно до поточного стану даних.

Окрему роль у взаємодії з інтерфейсом відіграє блок «Поточна сесія», розташований у лівій частині сторінки. У ньому фіксуються останні дії користувача, зокрема додавання оцінки, її подальша зміна та видалення. Такий журнал дій дає змогу наочно простежити послідовність взаємодії користувача із

системою та підтверджує, що внесені зміни враховуються в межах поточної сесії. Отже, інтерфейс забезпечує повний базовий цикл роботи з оцінками: створення, коригування та видалення.

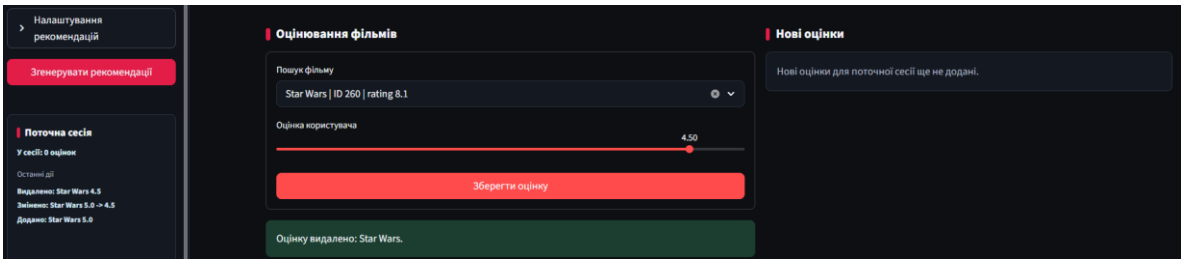

Рисунок 4.5 – Видалення оцінки та відображення історії дій у поточній сесії

4.5. Перехід від cold-start до персоналізованих рекомендацій

Після додавання кількох оцінок система повторно формує рекомендації вже з урахуванням початкових уподобань користувача. На цьому етапі профіль усе ще перебуває на ранній стадії персоналізації, тобто фактично в умовах cold start: хоча певні оцінки вже додано, їх кількість ще недостатня для повноцінного використання колаборативного сигналу. Саме тому система адаптує ваги моделі таким чином, щоб більший вплив мали контентна складова та додаткові загальні фактори, тоді як роль collaborative-компонента поки що залишається обмеженою. Такий підхід є логічним, оскільки за невеликої кількості оцінених фільмів система ще не має достатньо даних для надійного порівняння нового профілю з поведінкою інших користувачів.

На сформованих результатах можна побачити відмінності між трьома підходами. Hybrid model і Personalized SVD демонструють більш виражену персоналізацію, тому їх рекомендації виявляються ближчими до тих фільмів, які користувач уже оцінив. Натомість Classic SVD на цьому етапі поводиться більш загально: модель значною мірою спирається на загальні закономірності рейтингової матриці, базові оцінкові зсуви та загальний стиль оцінювання

користувача. Через це її результати ще не настільки чітко відображають конкретну тематичну спрямованість доданих фільмів. Для повнішого розкриття потенціалу Classic SVD, як правило, потрібна більша кількість оцінених позицій, що дає змогу моделі точніше виявити індивідуальні уподобання користувача.

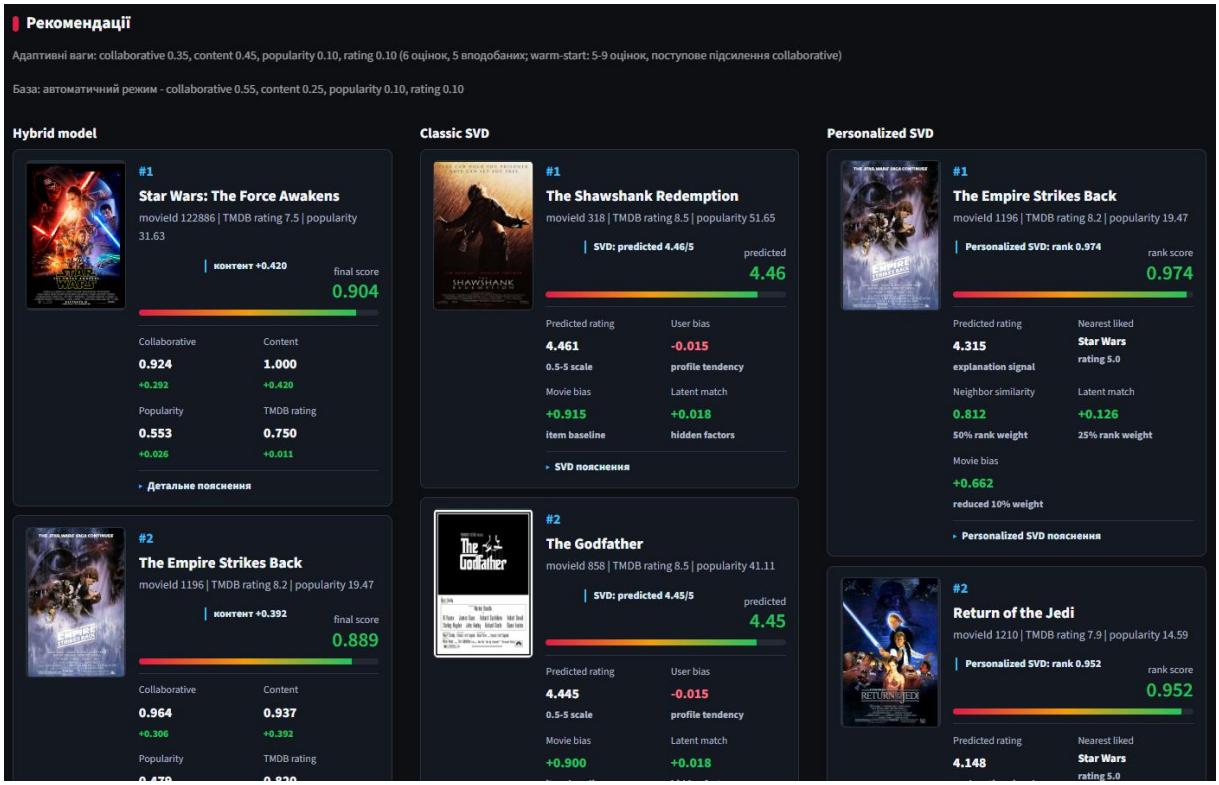

Рисунок 4.6 – Генерація рекомендацій після додавання початкових оцінок у режимі cold start

Окремо в інтерфейсі передбачено блок налаштування ваг для гібридної моделі рекомендацій. У стандартному автоматичному режимі система самостійно визначає співвідношення між основними компонентами моделі: collaborative, content, popularity та rating. У цьому випадку для гібридного ранжування використовуються ваги, за яких найбільший вплив має collaborative-компонент, а саме: collaborative — 0,70, content — 0,15, popularity — 0,10, rating — 0,05. Такий режим дозволяє системі автоматично застосовувати попередньо визначену логіку ранжування без додаткового втручання користувача.

Якщо автоматичний режим вимкнути, користувач отримує можливість самостійно обрати один із доступних пресетів фокусу рекомендацій. У системі передбачено чотири варіанти: орієнтація на схожих користувачів, схожий контент, популярні фільми або високий TMDB-рейтинг. Це дає змогу змінювати поведінку саме гібридної моделі залежно від того, який принцип добору користувач вважає більш доречним для себе. Наприклад, можна посилити вплив фільмів, схожих за змістом, або надати перевагу фільмам із вищими загальними рейтингами.

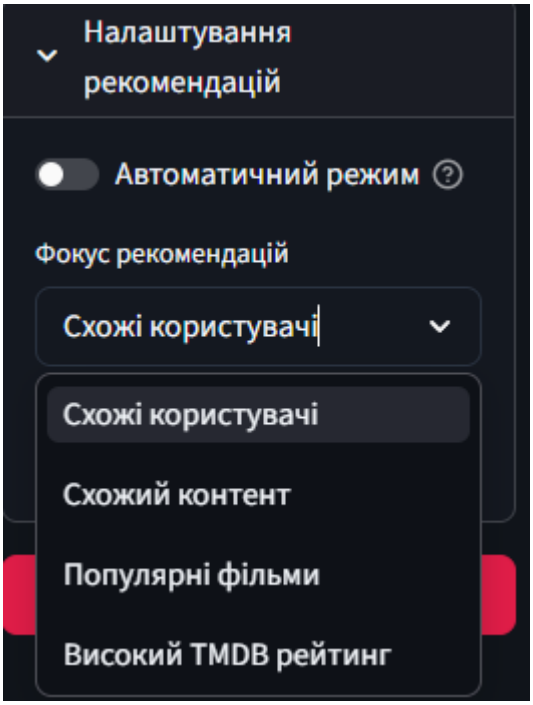

Рисунок 4.7 – Налаштування ваг гібридної моделі рекомендацій

4.6. Порівняння моделей та режимів ранжування

У картці рекомендації гібридної моделі відображається фільм, який система визначила як найбільш релевантний для поточного профілю користувача. У прикладі першу позицію займає фільм Star Wars: The Force Awakens. Для нього показано основні службові характеристики: ідентифікатор фільму, рейтинг

TMDB, показник популярності, а також підсумковий бал final score, за яким фільм ранжується в загальному списку рекомендацій.

Гібридна модель формує рекомендацію на основі поєднання кількох факторів: collaborative, content, popularity та TMDB rating. Показник collaborative відображає, наскільки фільм відповідає поведінковим закономірностям користувачів зі схожими оцінками. Показник content характеризує тематичну або змістову подібність фільму до тих фільмів, які вже були оцінені користувачем. Popularity враховує загальну популярність фільму, а TMDB rating — його середню оцінку за зовнішніми метаданими.

Зелені значення під кожним фактором показують позитивний внесок відповідної складової у підсумковий результат. У наведеному прикладі найбільший вплив має контентна складова, оскільки фільм має максимальну змістову відповідність до вже оцінених користувачем фільмів. Також суттєвий внесок дає collaborative-компонент, що свідчить про наявність поведінкової подібності. Після об’єднання цих нормалізованих сигналів система формує підсумкове значення final score, яке в цьому випадку дорівнює 0.904. Саме за цим показником гібридна модель визначає позицію фільму в рекомендаційному списку.

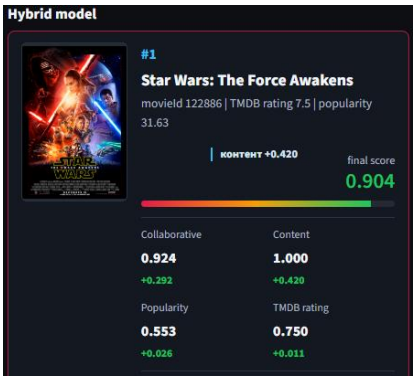

Рисунок 4.8 – Картка рекомендації гібридної моделі

У картці рекомендації Classic SVD відображається фільм, якому факторизаційна модель прогнозує найвищу очікувану оцінку з боку поточного

користувача. У наведеному прикладі першу позицію займає фільм The Shawshank Redemption. Картка містить основні довідкові відомості про фільм, зокрема його ідентифікатор, рейтинг TMDB, популярність, а також підсумковий показник predicted, який у цьому випадку дорівнює 4.46. Саме це значення використовується для ранжування рекомендацій у межах моделі Classic SVD.

На відміну від гібридної моделі, у цьому випадку оцінка формується не як зважена сума кількох явних ознак, а як прогноз рейтингу на основі латентних факторів. Показник Predicted rating відображає очікувану оцінку фільму для користувача за шкалою від 0.5 до 5.0. Компонент User bias характеризує загальну схильність користувача ставити дещо вищі або нижчі оцінки від середнього рівня, Movie bias показує базову тенденцію самого фільму отримувати вищі оцінки серед усіх користувачів, а Latent match відображає приховану відповідність між профілем користувача та властивостями фільму у просторі латентних факторів. Таким чином, підсумкова прогнозована оцінка формується як поєднання загального середнього рівня, індивідуального зсуву користувача, зсуву фільму та латентної відповідності між ними.

У наведеному прикладі найбільший внесок у високий результат дає movie bias, тобто сам фільм має сильну базову схильність отримувати високі оцінки. Водночас user bias є незначним і від’ємним, що свідчить про майже нейтральний загальний стиль оцінювання поточного профілю, а latent match має невелике додатне значення. У сукупності це дає підсумковий прогноз 4.46, завдяки чому фільм посідає верхню позицію в списку рекомендацій Classic SVD.

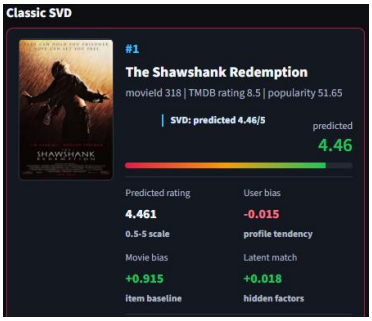

Рисунок 4.9 – Картка рекомендації моделі Classic SVD

У картці рекомендації Personalized SVD відображається фільм, який система вважає найбільш релевантним для користувача з урахуванням не лише загального факторизаційного прогнозу, а й подібності до тих фільмів, які користувач уже високо оцінив. У наведеному прикладі першу позицію займає фільм The Empire Strikes Back. У картці подано основні відомості про фільм, зокрема його ідентифікатор, рейтинг TMDB, популярність, а також підсумковий показник rank score, який у цьому випадку дорівнює 0.974. Саме цей показник використовується для ранжування рекомендацій у межах моделі Personalized SVD.

На відміну від Classic SVD, персоналізована SVD-модель не обмежується лише прогнозуванням оцінки, а додатково враховує зв’язок кандидата з уже вподобаними користувачем фільмами. Показник Predicted rating відображає очікувану оцінку фільму для користувача, тоді як Nearest liked указує на той раніше високо оцінений фільм, з яким поточна рекомендація має найближчий зв’язок. Компонент Neighbor similarity характеризує ступінь подібності між рекомендованим фільмом і найближчим уподобаним фільмом користувача та має найбільшу вагу у формуванні рангу. Показник Latent match відображає приховану відповідність у просторі латентних факторів, а Movie bias враховує загальну схильність самого фільму отримувати вищі оцінки. Отже, підсумковий rank score формується як зважене поєднання кількох складових, серед яких головну роль відіграє подібність до вже вподобаного фільму.

У наведеному прикладі система визначає, що фільм The Empire Strikes Back є близьким до фільму Star Wars, який користувач раніше оцінив на максимальний бал. Саме тому високі значення neighbor similarity і latent match забезпечують суттєвий внесок у фінальний результат. Таким чином, Personalized SVD демонструє більш виражену персоналізацію, оскільки орієнтується не лише на загальні закономірності оцінювання, а й на конкретні вподобання користувача.

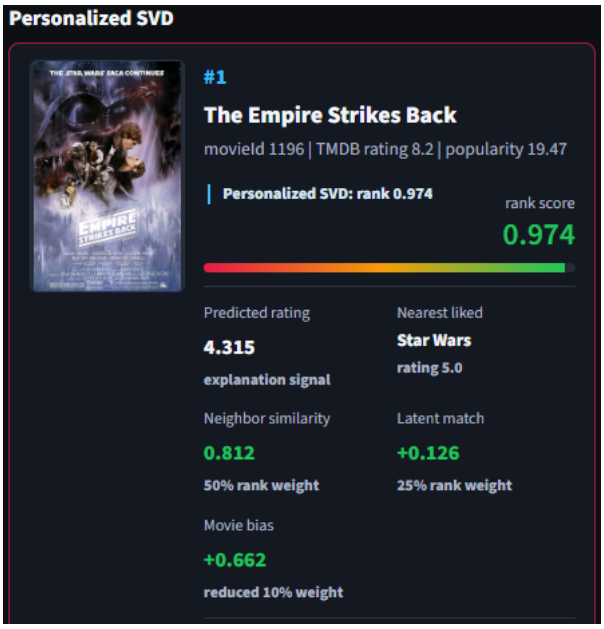

Рисунок 4.10 – Картка рекомендації моделі Personalized SVD

4.7. Перегляд пояснень до рекомендованих фільмів

У картці гібридної моделі користувач може розкрити блок «Детальні пояснення», у якому система надає більш повний розбір причин появи конкретного фільму в рекомендаціях. У цьому блоці окремо показано внесок кожної складової гібридної моделі: collaborative, content, popularity та TMDB rating. Такий формат дозволяє не лише побачити фінальний результат, а й зрозуміти, які саме фактори найбільше вплинули на позицію фільму в списку.

У наведеному прикладі система пояснює, що collaborative-компонент сформовано на основі схожості з історичними користувачами, які мали подібні оцінки. Контентна складова показує, з якими вже оціненими фільмами рекомендація має найбільшу подібність, а також які спільні TF-IDF-ознаки були враховані. Окремо подано вплив популярності фільму та його TMDB-рейтингу порівняно із середніми значеннями кандидатів. Завдяки цьому користувач отримує прозоре пояснення того, чому саме цей фільм був рекомендований гібридною моделлю.

Рисунок 4.11 – Детальне пояснення рекомендації гібридної моделі

У картці Classic SVD також передбачено блок «SVD пояснення», у якому користувач може переглянути детальнішу структуру прогнозу. На відміну від гібридної моделі, тут пояснення будується не через окремі явні ознаки на кшталт контентної схожості чи популярності, а через складові прогнозованої оцінки. Система показує, з яких компонентів формується очікуваний рейтинг фільму для поточного користувача.

У цьому блоці відображається Global mean — базова середня оцінка у навчальних даних, від якої починається прогноз. Далі враховується User bias, тобто індивідуальна схильність поточного профілю ставити оцінки вище або нижче середнього рівня. Компонент Movie bias показує, наскільки сам фільм у середньому оцінюється вище або нижче базового рівня. Окремо подано Latent match — приховану відповідність між користувачем і фільмом, обчислену у просторі латентних факторів. Саме поєднання цих компонентів дозволяє Classic SVD сформувати персональний прогноз оцінки для конкретного фільму.

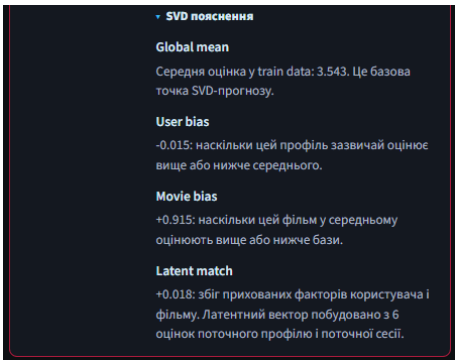

Рисунок 4.12 – Детальне пояснення рекомендації моделі Classic SVD

У картці Personalized SVD користувач може розкрити блок «Personalized SVD пояснення», у якому подається детальніший розбір причин, через які фільм потрапив до рекомендацій. Цей блок орієнтований на пояснення саме персоналізованої логіки моделі, тобто показує не лише загальний прогноз оцінки, а й зв’язок рекомендованого фільму з уже вподобаними користувачем позиціями.

У межах цього блоку відображаються такі ключові складові: Latent neighbor — фільм, який є найближчим до рекомендації у просторі латентних ознак і вже був високо оцінений користувачем; Latent match — міра прихованої відповідності між користувачем і рекомендованим фільмом; Movie bias — базова схильність фільму отримувати вищі оцінки; Predicted rating — очікувана оцінка, яку модель прогнозує для цього фільму. Таким чином, користувач отримує більш прозоре пояснення того, що рекомендація сформована не випадково, а на основі поєднання загального прогнозу моделі та подібності до вже вподобаних фільмів.

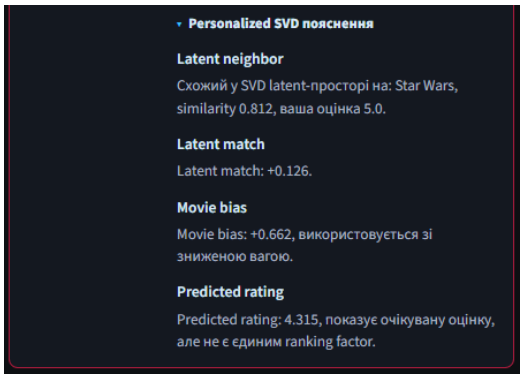

Рисунок 4.13 – Детальне пояснення рекомендації моделі Personalized SVD

4.8. Аналіз отриманих результатів демонстрації

Проведена демонстрація роботи системи показала, що розроблений веб-застосунок забезпечує повний цикл взаємодії користувача з рекомендаційною системою. Користувач може увійти до системи через демо-профіль, переглянути основний інтерфейс, додати оцінки фільмів, змінити або видалити їх, а також сформувати рекомендації за допомогою різних алгоритмічних підходів. Така послідовність дій підтверджує, що реалізований інтерфейс є не лише засобом відображення результатів, а й інструментом активного формування користувацького профілю.

На початковому етапі, коли використовується новий профіль без історії оцінювання, рекомендації мають більш загальний характер. Це є очікуваною поведінкою системи, оскільки за відсутності оцінок неможливо повністю визначити індивідуальні вподобання користувача. У такій ситуації система використовує доступні загальні сигнали, зокрема популярність фільмів, середній рейтинг і контентні характеристики. Персоналізована SVD-модель у цьому випадку не має достатньої кількості високо оцінених фільмів для пошуку латентних сусідів, тому застосовує резервну логіку на основі класичного SVD-прогнозу.

Після додавання користувацьких оцінок поведінка системи змінюється. Гібридна модель починає активніше враховувати контентну подібність і колаборативний сигнал, а Personalized SVD отримує опорні фільми для пошуку близьких кандидатів у латентному просторі. Це дозволяє системі перейти від загальних рекомендацій до більш персоналізованих. Саме цей перехід є важливим результатом демонстрації, оскільки він показує, що рекомендації залежать від дій користувача і змінюються відповідно до його вподобань.

Порівняння Hybrid model, Classic SVD і Personalized SVD показує різницю між підходами до ранжування. Гібридна модель є найбільш прозорою з погляду явних складових, оскільки для неї можна окремо показати внесок collaborative,

content, popularity і TMDB rating. Classic SVD формує прогноз на основі глобального середнього рівня, зміщення користувача, зміщення фільму та латентної відповідності, тому його результати можуть бути сильніше пов’язані з фільмами, які мають стабільно високі оцінки. Personalized SVD, у свою чергу, демонструє більш виражену персоналізацію, оскільки враховує не лише прогноз оцінки, а й зв’язок рекомендованого фільму з уже вподобаними користувачем стрічками.

Окремим результатом демонстрації є перевірка механізму пояснень. У системі пояснення подаються не як загальний текст, а як деталізація конкретних факторів, що вплинули на рекомендацію. Для гібридної моделі користувач бачить внесок окремих складових і може зрозуміти, чи фільм рекомендовано через контентну схожість, поведінку схожих користувачів, популярність або рейтинг. Для Classic SVD пояснення розкриває структуру прогнозованої оцінки. Для Personalized SVD пояснення додатково показує найближчий уже вподобаний фільм, з яким пов’язана рекомендація. Це робить систему прозорішою та зрозумілішою для користувача.

Таким чином, результати демонстрації підтверджують працездатність розробленої системи. Веб-інтерфейс забезпечує зручну взаємодію з моделями, механізм сесії дозволяє зберігати дії користувача, а рекомендаційні алгоритми реагують на зміну профілю. Система не лише формує список фільмів, а й пояснює причини їх вибору, що відповідає темі роботи та поставленій меті.

ВИСНОВКИ

На основі аналізу підходів, методів та алгоритмів розв’язання поставленої задачі було обґрунтовано використання гібридного підходу до формування рекомендацій фільмів. У роботі використано контентну фільтрацію на основі TF-IDF і косинусної подібності, колаборативну фільтрацію на основі матриці «користувач — фільм», SVD-бейзлайн з урахуванням зміщень користувачів і фільмів, а також персоналізований SVD із пошуком сусідів у латентному просторі. Поєднання цих методів дозволило врахувати як описові характеристики фільмів, так і поведінкові закономірності користувачів.

На основі аналізу програмних засобів було обґрунтовано використання мови програмування Python і бібліотек pandas, NumPy, scikit-learn, scipy, joblib та Streamlit. pandas і NumPy використано для обробки табличних даних і числових операцій. scikit-learn використано для побудови TF-IDF-моделі, обчислення косинусної подібності та реалізації TruncatedSVD. scipy застосовано для роботи з розрідженими матрицями, joblib — для збереження навчених артефактів, а Streamlit — для створення інтерактивного веб-інтерфейсу користувача.

Розроблено програмну систему «Система рекомендацій фільмів з поясненнями на основі методів штучного інтелекту». Система складається з pipeline підготовки даних, модуля навчання моделей, збережених артефактів, механізму генерації рекомендацій, механізму пояснення результатів і веб-інтерфейсу користувача. У межах підготовки даних обробляються метадані фільмів, ключові слова, акторський склад, режисери, рейтинги та зв’язки між ідентифікаторами MovieLens і TMDB. У результаті створюються оброблені таблиці фільмів і оцінок, які використовуються для навчання моделей.

У роботі реалізовано три підходи до формування рекомендацій: Hybrid model, Classic SVD і Personalized SVD. Гібридна модель поєднує collaborative, content, popularity і TMDB rating. Classic SVD використовується як факторизаційний бейзлайн для прогнозування очікуваної оцінки фільму.

Personalized SVD доповнює SVD-підхід пошуком фільмів, близьких до вже вподобаних користувачем стрічок у латентному просторі. Це дозволило показати різницю між загальним факторизаційним прогнозом і персоналізованим ранжуванням.

Реалізовано механізм адаптації системи до сценаріїв cold-start та warm-start. Якщо користувач не має достатньої кількості оцінок, система зменшує вплив колаборативної складової та більше спирається на контентні й загальні фактори. Після накопичення оцінок поступово збільшується роль колаборативних сигналів і латентних факторів. Такий підхід дозволяє системі працювати як з новим користувачем, так і з користувачем, який уже має сформовану історію вподобань.

Розроблено механізм пояснення рекомендацій. Для гібридної моделі пояснення формується через внесок collaborative, content, popularity і TMDB rating у фінальний score. Для Classic SVD пояснення показує вплив глобального середнього рівня, зміщення користувача, зміщення фільму та латентної відповідності. Для Personalized SVD додатково пояснюється зв’язок рекомендації з найближчим уже вподобаним фільмом у латентному просторі. Це дозволяє зробити результати роботи системи більш прозорими для користувача.

На основі тестування доведено коректність роботи розробленого програмного забезпечення. У процесі демонстрації перевірено вибір демо-профілю, додавання, зміну та видалення оцінок, генерацію рекомендацій, роботу трьох моделей, зміну рекомендацій після оновлення профілю та відображення пояснень. Також проведено offline-оцінювання моделей на основі holdout-підходу. За результатами оцінювання гібридна модель показала вищий NDCG@10 порівняно з окремим SVD-бейзлайном, що підтверджує доцільність використання комбінованого підходу.

Практичне значення роботи полягає в тому, що розроблена система демонструє повний цикл створення рекомендаційного застосунку: від підготовки даних і навчання моделей до інтерактивної взаємодії з користувачем. Система може бути використана як основа для подальшого розвитку рекомендаційних

сервісів, навчальних проєктів або досліджень у сфері пояснюваних рекомендаційних систем.

Перспективами подальшого розвитку є розширення набору ознак фільмів, додавання нейромережевих моделей, використання історії переглядів, інтеграція з реальним API кіносервісу, удосконалення текстових пояснень і додавання механізмів оцінювання задоволеності користувачів. Також доцільно розширити інтерфейс користувача, додати фільтрацію за жанрами, роком випуску, мовою, тривалістю та іншими параметрами, що підвищить практичну цінність системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Resnick P., Iacovou N., Suchak M., Bergstrom P., Riedl J. GroupLens: An Open Architecture for Collaborative Filtering of Netnews. Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work. New York : ACM, 1994. С. 175–186. URL: https://doi.org/10.1145/192844.192905 (дата звернення: 28.05.2026).

2. Herlocker J. L., Konstan J. A., Riedl J. Explaining Collaborative Filtering Recommendations. Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work. New York : ACM, 2000. С. 241–250. URL: https://doi.org/10.1145/358916.358995 (дата звернення: 28.05.2026).

3. Sarwar B. M., Karypis G., Konstan J. A., Riedl J. Item-Based Collaborative Filtering Recommendation Algorithms. Proceedings of the 10th International Conference on World Wide Web. New York : ACM, 2001. С. 285–295. URL: https://doi.org/10.1145/371920.372071 (дата звернення: 28.05.2026).

4. Burke R. Hybrid Recommender Systems: Survey and Experiments. User Modeling and User-Adapted Interaction. 2002. Vol. 12, No. 4. С. 331–370. URL: https://doi.org/10.1023/A:1021240730564 (дата звернення: 28.05.2026).

5. Adomavicius G., Tuzhilin A. Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. IEEE Transactions on Knowledge and Data Engineering. 2005. Vol. 17, No. 6. С. 734–749. URL: https://doi.org/10.1109/TKDE.2005.99 (дата звернення: 28.05.2026).

6. Tintarev N., Masthoff J. A Survey of Explanations in Recommender Systems. Proceedings of the 2007 IEEE International Conference on Data Mining Workshops. Omaha: IEEE, 2007. С. 801–810. URL: https://doi.org/10.1109/ICDMW.2007.4401070 (дата звернення: 28.05.2026).

7. Manning C. D., Raghavan P., Schütze H. Introduction to Information Retrieval. Cambridge : Cambridge University Press, 2008. 544 с. URL: https://nlp.stanford.edu/IR-book/ (дата звернення: 28.05.2026).

8. Koren Y., Bell R., Volinsky C. Matrix Factorization Techniques for Recommender Systems. Computer. 2009. Vol. 42, No. 8. C. 30–37. URL: https://doi.org/10.1109/MC.2009.263 (дата звернення: 28.05.2026).

9. Harper F. M., Konstan J. A. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems. 2015. Vol. 5, No. 4. Article 19. URL: https://doi.org/10.1145/2827872 (дата звернення: 28.05.2026).

10. Scikit-learn developers. sklearn.decomposition.TruncatedSVD. Scikit-learn documentation. URL: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html (дата звернення: 28.05.2026).

11. Scikit-learn developers. sklearn.metrics.pairwise.cosine_similarity. Scikit-learn documentation. URL: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.cosine_similarity.html (дата звернення: 28.05.2026).

12. Joblib developers. Persistence: joblib.dump and joblib.load. Joblib documentation. URL: https://joblib.readthedocs.io/en/latest/persistence.html (дата звернення: 28.05.2026).

13. Streamlit. Streamlit Documentation. URL: https://docs.streamlit.io/ (дата звернення: 28.05.2026).

14. Gomez-Uribe C. A., Hunt N. The Netflix Recommender System: Algorithms, Business Value, and Innovation. ACM Transactions on Management Information Systems. 2015. Vol. 6, No. 4. Article 13. URL: https://doi.org/10.1145/2843948 (дата звернення: 28.05.2026).

15. Zhang Y., Chen X. Explainable Recommendation: A Survey and New Perspectives. Foundations and Trends in Information Retrieval. 2020. Vol. 14, No. 1. C. 1–101. URL: https://doi.org/10.1561/1500000066 (дата звернення: 28.05.2026).

16. Scikit-learn developers. sklearn.feature_extraction.text.TfidfVectorizer. Scikit-learn documentation. URL: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html (дата звернення: 28.05.2026).

17. Gunawardana A., Shani G. A Survey of Accuracy Evaluation Metrics of Recommendation Tasks. Journal of Machine Learning Research. 2009. Vol. 10. C.

2935–2962. URL: https://jmlr.org/papers/v10/gunawardana09a.html (дата звернення: 28.05.2026).

18. Streamlit. st.cache_data. Streamlit documentation. URL: https://docs.streamlit.io/develop/api-reference/caching-and-state/st.cache_data (дата звернення: 28.05.2026).

19. Streamlit. Session State. Streamlit documentation. URL: https://docs.streamlit.io/develop/api-reference/caching-and-state/st.session_state (дата звернення: 28.05.2026).

ДОДАТОК А

Фрагменти програмного коду

«Система рекомендацій фільмів з поясненнями на основі методів штучного інтелекту»

УКР.НТУУ «КПІ ім. Ігоря Сікорського»_ІАТЕ_ЦТЕ_ТР_21

Мови програмування — Python

Київ — 2026

Лістинг програмного коду

Програмні засоби:

— мова програмування — Python;
— фреймворк для створення веб-інтерфейсу — Streamlit;
— бібліотека для обробки табличних даних — pandas;
— бібліотека для числових обчислень — NumPy;
— бібліотека машинного навчання — scikit-learn;
— бібліотека для роботи з розрідженими матрицями — scipy;
— бібліотека для збереження навчених артефактів — joblib;
— середовище розробки — Visual Studio Code.

Лістинг програмного коду

ІМПОРТ БІБЛІОТЕК ТА БАЗОВІ НАЛАШТУВАННЯ НАВЧАННЯ

```
from __future__ import annotations

import argparse
import ast
import json
from dataclasses import asdict, dataclass
from datetime import datetime, timezone
from pathlib import Path

import joblib
import numpy as np
import pandas as pd
from scipy.sparse import csr_matrix
from sklearn.decomposition import TruncatedSVD
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
from sklearn.metrics.pairwise import cosine_similarity


MIN_LIKED_RATING = 4.0

DEFAULT_WEIGHTS = {
    "collaborative": 0.40,
    "content": 0.40,
    "popularity": 0.10,
    "rating": 0.10,
}

WEIGHT_GRID = [
    (0.40, 0.40, 0.10, 0.10),
    (0.50, 0.30, 0.10, 0.10),
    (0.30, 0.50, 0.10, 0.10),
    (0.60, 0.20, 0.10, 0.10),
    (0.20, 0.60, 0.10, 0.10),
    (0.45, 0.45, 0.05, 0.05),
    (0.35, 0.45, 0.10, 0.10),
    (0.45, 0.35, 0.10, 0.10),
    (0.30, 0.40, 0.20, 0.10),
    (0.30, 0.40, 0.10, 0.20),
    (0.25, 0.55, 0.10, 0.10),
    (0.55, 0.25, 0.10, 0.10),
]
```

ПІДГОТОВКА ДАНИХ ПРО ФІЛЬМИ ТА ОЦІНКИ

```
def load_and_prepare_data(
    data_dir: Path,
    ratings_file: str,
    links_file: str,
) -> tuple[pd.DataFrame, pd.DataFrame]:
```

```
movies = pd.read_csv(data_dir / "movies_metadata.csv", low_memory=False)
credits = pd.read_csv(data_dir / "credits.csv")
keywords = pd.read_csv(data_dir / "keywords.csv")
ratings = pd.read_csv(data_dir / ratings_file)
links = pd.read_csv(data_dir / resolve_links_file(data_dir, ratings_file, links_file))

movies = movies[movies["id"].apply(lambda value: str(value).isdigit())].copy()
movies["id"] = movies["id"].astype(int)

links = links.dropna(subset=["movieId", "tmdbId"]).copy()
links["movieId"] = links["movieId"].astype(int)
links["tmdbId"] = links["tmdbId"].astype(int)

movies = movies.merge(links, left_on="id", right_on="tmdbId", how="inner")
movies = movies.merge(credits, on="id", how="left")
movies = movies.merge(keywords, on="id", how="left")

movies["genres"] = movies["genres"].apply(parse_names)
movies["keywords"] = movies["keywords"].apply(parse_names)
movies["cast"] = movies["cast"].apply(get_cast)
movies["director"] = movies["crew"].apply(get_director)
movies["collection"] = movies["belongs_to_collection"].apply(parse_collection)
movies["overview"] = movies["overview"].fillna("").astype(str)
movies["title"] = movies["title"].fillna("Unknown title").astype(str)

movies["vote_average"] = pd.to_numeric(
    movies["vote_average"],
    errors="coerce"
).fillna(0)

movies["popularity"] = pd.to_numeric(
    movies["popularity"],
    errors="coerce"
).fillna(0)
```

```
movies["movieId"] = pd.to_numeric(movies["movieId"], errors="coerce")
movies = movies.dropna(subset=["movieId"]).copy()
movies["movieId"] = movies["movieId"].astype(int)

movies["content"] = movies.apply(combine_features, axis=1)

processed_movies = (
    movies[["movieId", "title", "content", "vote_average", "popularity"]]
    .drop_duplicates(subset=["movieId"])
    .sort_values("movieId")
    .reset_index(drop=True)
)

ratings = ratings[["userId", "movieId", "rating"]].copy()
ratings["userId"] = pd.to_numeric(ratings["userId"], errors="coerce")
ratings["movieId"] = pd.to_numeric(ratings["movieId"], errors="coerce")
ratings["rating"] = pd.to_numeric(ratings["rating"], errors="coerce")

ratings = ratings.dropna(subset=["userId", "movieId", "rating"])
ratings["userId"] = ratings["userId"].astype(int)
ratings["movieId"] = ratings["movieId"].astype(int)
ratings["rating"] = ratings["rating"].astype(float)

ratings = ratings[ratings["movieId"].isin(processed_movies["movieId"])]
processed_ratings = ratings.drop_duplicates(
    subset=["userId", "movieId"],
    keep="last"
)

return processed_movies, processed_ratings
```

ФОРМУВАННЯ КОНТЕНТНИХ ОЗНАК ФІЛЬМІВ

```
def parse_names(value) -> list[str]:
    parsed = parse_literal(value)
    if not isinstance(parsed, list):
        return []

    names = []
    for item in parsed:
        if isinstance(item, dict) and item.get("name"):
            names.append(str(item["name"]))
    return names


def get_cast(value, limit: int = 5) -> list[str]:
    parsed = parse_literal(value)
    if not isinstance(parsed, list):
        return []

    cast = []
    for item in parsed[:limit]:
        if isinstance(item, dict) and item.get("name"):
            cast.append(str(item["name"]))
    return cast


def get_director(value) -> str:
    parsed = parse_literal(value)
    if not isinstance(parsed, list):
        return ""

    for item in parsed:
        if isinstance(item, dict) and item.get("job") == "Director" and item.get("name"):
            return str(item["name"])
```

```
    return ""


def combine_features(row: pd.Series) -> str:
    collection_terms = [row["collection"]] * 3 if row["collection"] else []
    return join_terms(
        row["genres"]
        + row["keywords"]
        + row["cast"]
        + [row["director"]]
        + collection_terms
        + [row["overview"], row["title"]]
    )
```

НАВЧАННЯ КОНТЕНТНОЇ МОДЕЛІ НА ОСНОВІ TF-IDF

```
def train_content_model(
    movies: pd.DataFrame,
    max_features: int,
) -> tuple[pd.DataFrame, TfidfVectorizer, np.ndarray]:
    model_movies = movies.copy().reset_index(drop=True)
    model_movies["content"] = model_movies["content"].fillna("").astype(str)

    tfidf = TfidfVectorizer(
        stop_words="english",
        max_features=max_features,
        ngram_range=(1, 2),
        min_df=2,
    )

    tfidf_matrix = tfidf.fit_transform(model_movies["content"])
    content_similarity = cosine_similarity(tfidf_matrix).astype(np.float32)

    return model_movies, tfidf, content_similarity
```

НАВЧАННЯ КОЛАБОРАТИВНОЇ МОДЕЛІ

```
def center_user_movie_matrix(user_movie_matrix: pd.DataFrame) -> pd.DataFrame:
    user_means = user_movie_matrix.mean(axis=1)
    return user_movie_matrix.sub(user_means, axis=0).fillna(0)



def train_collaborative_model(
    ratings: pd.DataFrame,
) -> tuple[pd.DataFrame, pd.DataFrame]:
    user_movie_matrix = ratings.pivot_table(
        index="userId",
        columns="movieId",
        values="rating",
        aggfunc="mean",
    )

    centered_matrix = center_user_movie_matrix(user_movie_matrix)
    user_similarity = cosine_similarity(centered_matrix)

    user_similarity_df = pd.DataFrame(
        user_similarity.astype(np.float32),
        index=user_movie_matrix.index.astype(int),
        columns=user_movie_matrix.index.astype(int),
    )

    return user_movie_matrix, user_similarity_df
```

РЕАЛІЗАЦІЯ SVD-БЕЙЗЛАЙНУ

```
def compute_svd_biases(
    ratings: pd.DataFrame,
    user_regularization: float = 10.0,
    movie_regularization: float = 10.0,
```

```
) -> tuple[float, pd.Series, pd.Series]:
    if ratings.empty:
        return 3.5, pd.Series(dtype=float), pd.Series(dtype=float)

    global_mean = float(ratings["rating"].mean())

    movie_residual = ratings["rating"].astype(float) - global_mean
    movie_stats = movie_residual.groupby(
        ratings["movieId"].astype(int)
    ).agg(["sum", "count"])

    movie_bias = movie_stats["sum"] / (
        movie_stats["count"] + movie_regularization
    )
    movie_bias = movie_bias.astype(float)

    movie_bias_values = ratings["movieId"].astype(int).map(movie_bias).fillna(0.0)

    user_residual = (
        ratings["rating"].astype(float)
        - global_mean
        - movie_bias_values
    )

    user_stats = user_residual.groupby(
        ratings["userId"].astype(int)
    ).agg(["sum", "count"])

    user_bias = user_stats["sum"] / (
        user_stats["count"] + user_regularization
    )
    user_bias = user_bias.astype(float)

    return global_mean, user_bias, movie_bias
```

НАВЧАННЯ ЛАТЕНТНОЇ МОДЕЛІ TRUNCATEDSVD

```
def train_svd_baseline(
    ratings: pd.DataFrame,
    movies: pd.DataFrame,
    n_components: int,
    n_iter: int,
    random_state: int,
    user_regularization: float = 10.0,
    movie_regularization: float = 10.0,
) -> dict[str, object]:
    model_ratings = ratings[ratings["movieId"].isin(movies["movieId"])].copy()
    model_ratings["userId"] = model_ratings["userId"].astype(int)
    model_ratings["movieId"] = model_ratings["movieId"].astype(int)
    model_ratings["rating"] = model_ratings["rating"].astype(float)

    user_ids = np.array(sorted(model_ratings["userId"].unique()), dtype=np.int64)
    movie_ids = movies["movieId"].astype(int).to_numpy(dtype=np.int64)

    global_mean, user_bias, movie_bias = compute_svd_biases(
        model_ratings,
        user_regularization=user_regularization,
        movie_regularization=movie_regularization,
    )

    user_id_to_position = {
        int(user_id): index for index, user_id in enumerate(user_ids)
    }

    movie_id_to_position = {
        int(movie_id): index for index, movie_id in enumerate(movie_ids)
    }

    row_indices = model_ratings["userId"].map(user_id_to_position).astype(int).to_numpy()
```

```
    col_indices = model_ratings["movieId"].map(movie_id_to_position).astype(int).to_numpy()

    user_bias_values = model_ratings["userId"].map(user_bias).fillna(0.0).to_numpy(dtype=float)
    movie_bias_values =
model_ratings["movieId"].map(movie_bias).fillna(0.0).to_numpy(dtype=float)

    residual_values = (
        model_ratings["rating"].to_numpy(dtype=float)
        - global_mean
        - user_bias_values
        - movie_bias_values
    ).astype(np.float32)

    residual_matrix = csr_matrix(
        (residual_values, (row_indices, col_indices)),
        shape=(len(user_ids), len(movie_ids)),
        dtype=np.float32,
    )

    max_components = max(1, min(residual_matrix.shape) - 1)
    safe_components = int(max(1, min(n_components, max_components)))

    svd = TruncatedSVD(
        n_components=safe_components,
        n_iter=n_iter,
        random_state=random_state,
    )

    user_factors = svd.fit_transform(residual_matrix).astype(np.float32)

    return {
        "model": svd,
        "global_mean": float(global_mean),
        "user_bias": user_bias.reindex(user_ids, fill_value=0.0).astype(np.float32),
```

```
        "movie_bias": movie_bias.reindex(movie_ids, fill_value=0.0).astype(np.float32),
        "user_factors": user_factors,
        "movie_ids": movie_ids,
        "user_ids": user_ids,
        "user_id_to_position": user_id_to_position,
        "movie_id_to_position": movie_id_to_position,
        "n_components": safe_components,
        "rating_min": 0.5,
        "rating_max": 5.0,
    }
```

ФОРМУВАННЯ ГІБРИДНОЇ РЕКОМЕНДАЦІЇ

```
def recommend_for_evaluation(
    user_id: int,
    ratings: pd.DataFrame,
    movies: pd.DataFrame,
    content_similarity: np.ndarray,
    user_movie_matrix: pd.DataFrame,
    user_similarity: pd.DataFrame,
    top_k: int,
    weights: dict[str, float],
    similar_user_count: int,
) -> pd.DataFrame:
    user_ratings = ratings[ratings["userId"] == user_id]
    if user_ratings.empty:
        return pd.DataFrame()

    watched_movie_ids = set(user_ratings["movieId"].astype(int).tolist())

    liked_movie_ids = set(
        user_ratings.loc[
            user_ratings["rating"] >= MIN_LIKED_RATING,
            "movieId"
```

```
    ].astype(int).tolist()
)

candidate_movies = movies.loc[
    ~movies["movieId"].isin(watched_movie_ids)
].copy()

if candidate_movies.empty:
    return candidate_movies

movie_id_to_index = dict(zip(movies["movieId"].astype(int), movies.index))

collaborative_scores = get_collaborative_scores(
    user_id,
    user_movie_matrix,
    user_similarity,
    similar_user_count,
)

candidate_movies["collaborative_score"] = (
    candidate_movies["movieId"]
    .map(collaborative_scores)
    .fillna(0)
    .astype(float)
)

candidate_movies["content_score"] = get_content_scores(
    candidate_movies,
    liked_movie_ids,
    content_similarity,
    movie_id_to_index,
)

candidate_movies["popularity_score"] = np.log1p(
```

```
    candidate_movies["popularity"].clip(lower=0)
)

candidate_movies["rating_score"] = candidate_movies["vote_average"].clip(lower=0)

candidate_movies["collaborative_norm"] = normalize_series(
    candidate_movies["collaborative_score"]
)
candidate_movies["content_norm"] = normalize_series(
    candidate_movies["content_score"]
)
candidate_movies["popularity_norm"] = normalize_series(
    candidate_movies["popularity_score"]
)
candidate_movies["rating_norm"] = normalize_series(
    candidate_movies["rating_score"]
)

normalized_weights = normalize_weights(weights)

candidate_movies["final_score"] = (
    normalized_weights["collaborative"] * candidate_movies["collaborative_norm"]
    + normalized_weights["content"] * candidate_movies["content_norm"]
    + normalized_weights["popularity"] * candidate_movies["popularity_norm"]
    + normalized_weights["rating"] * candidate_movies["rating_norm"]
)

return candidate_movies.sort_values(
    ["final_score", "vote_average", "popularity"],
    ascending=False,
).head(top_k)
```

ОЦІНЮВАННЯ ЯКОСТІ РЕКОМЕНДАЦІЙ

```
def make_holdout_split(
    ratings: pd.DataFrame,
    random_state: int,
    max_eval_users: int,
    min_ratings_per_user: int = 5,
) -> tuple[pd.DataFrame, pd.DataFrame]:
    rng = np.random.default_rng(random_state)
    test_rows = []

    grouped = ratings.groupby("userId")
    eligible_user_ids = []

    for user_id, group in grouped:
        liked_group = group[group["rating"] >= MIN_LIKED_RATING]
        if len(group) >= min_ratings_per_user and not liked_group.empty:
            eligible_user_ids.append(int(user_id))

    if max_eval_users > 0 and len(eligible_user_ids) > max_eval_users:
        eligible_user_ids = sorted(
            rng.choice(
                eligible_user_ids,
                size=max_eval_users,
                replace=False
            ).tolist()
        )

    test_indices = []

    for user_id in eligible_user_ids:
        user_liked = ratings[
            (ratings["userId"] == user_id)
            & (ratings["rating"] >= MIN_LIKED_RATING)
```

```
        ]

        chosen_index = rng.choice(user_liked.index.to_numpy())
        test_indices.append(int(chosen_index))
        test_rows.append(ratings.loc[chosen_index])

    test_ratings = pd.DataFrame(test_rows)
    train_ratings = ratings.drop(index=test_indices).reset_index(drop=True)
    test_ratings = test_ratings.reset_index(drop=True)

    return train_ratings, test_ratings
```

ЗАВАНТАЖЕННЯ МОДЕЛЕЙ У ВЕБ-ЗАСТОСУНКУ

```
@st.cache_resource(show_spinner=False)
def load_model_artifacts():
    movies = joblib.load(MODELS_DIR / "movies.pkl")
    tfidf = joblib.load(MODELS_DIR / "tfidf.pkl")
    content_similarity = joblib.load(MODELS_DIR / "content_similarity.pkl")
    user_movie_matrix = joblib.load(MODELS_DIR / "user_movie_matrix.pkl")
    user_similarity = joblib.load(MODELS_DIR / "user_similarity.pkl")

    svd_path = MODELS_DIR / "svd_baseline.pkl"
    svd_artifacts = joblib.load(svd_path) if svd_path.exists() else None

    movies = movies.copy().reset_index(drop=True)
    movies = movies.dropna(subset=["movieId", "title"]).copy()
    movies["movieId"] = pd.to_numeric(movies["movieId"], errors="coerce")
    movies = movies.dropna(subset=["movieId"]).copy()
    movies["movieId"] = movies["movieId"].astype(int)
    movies["title"] = movies["title"].astype(str)
    movies["content"] = movies["content"].fillna("").astype(str)
    movies["popularity"] = pd.to_numeric(
        movies["popularity"],
```

```
    errors="coerce"
).fillna(0)
movies["vote_average"] = pd.to_numeric(
    movies["vote_average"],
    errors="coerce"
).fillna(0)

user_movie_matrix = user_movie_matrix.copy()
user_movie_matrix.index = user_movie_matrix.index.astype(int)
user_movie_matrix.columns = user_movie_matrix.columns.astype(int)

user_similarity = user_similarity.copy()
user_similarity.index = user_similarity.index.astype(int)
user_similarity.columns = user_similarity.columns.astype(int)

if len(movies) != content_similarity.shape[0]:
    raise ValueError(
        "Кількість фільмів не збігається з матрицею content similarity."
    )

tfidf_matrix = tfidf.transform(movies["content"])
tfidf_feature_names = tfidf.get_feature_names_out()

return (
    movies,
    content_similarity,
    tfidf_matrix,
    tfidf_feature_names,
    user_movie_matrix,
    user_similarity,
    svd_artifacts,
)
```

ЗБЕРЕЖЕННЯ СТАНУ ПОТОЧНОЇ СЕСІЇ

```
def initialize_state():
    if "authenticated" not in st.session_state:
        st.session_state.authenticated = False

    if "active_user" not in st.session_state:
        st.session_state.active_user = None

    if "session_ratings" not in st.session_state:
        st.session_state.session_ratings = {}

    if "session_events" not in st.session_state:
        st.session_state.session_events = {}

    if "recommendations" not in st.session_state:
        st.session_state.recommendations = None

    if "recommendation_params" not in st.session_state:
        st.session_state.recommendation_params = None

    if "rating_notice" not in st.session_state:
        st.session_state.rating_notice = None
```

АДАПТАЦІЯ ВАГ ДО СЦЕНАРІЇВ COLD-START ТА WARM-START

```
def get_adaptive_weights(
    base_weights: dict[str, float],
    rating_count: int,
    liked_count: int,
) -> tuple[dict[str, float], str]:
    normalized_base = normalize_weights(base_weights)

    if rating_count <= 0 or liked_count <= 0:
```

```
        popularity_weight, rating_weight = split_fallback_weights(
            normalized_base,
            1.0,
        )

        return (
            {
                "collaborative": 0.0,
                "content": 0.0,
                "popularity": popularity_weight,
                "rating": rating_weight,
            },
            "fallback: немає достатньо вподобаних фільмів для персоналізації",
        )

    if rating_count < 3:
        popularity_weight, rating_weight = split_fallback_weights(
            normalized_base,
            0.20,
        )

        return (
            {
                "collaborative": 0.10,
                "content": 0.70,
                "popularity": popularity_weight,
                "rating": rating_weight,
            },
            "cold-start: 1-2 оцінки, пріоритет content similarity",
        )

    if rating_count < 5:
        popularity_weight, rating_weight = split_fallback_weights(
            normalized_base,
```

```
            0.20,
        )

        return (
            {
                "collaborative": 0.20,
                "content": 0.60,
                "popularity": popularity_weight,
                "rating": rating_weight,
            },
            "cold-start: 3-4 оцінки, collaborative сигнал ще обмежений",
        )

    if rating_count < 10:
        popularity_weight, rating_weight = split_fallback_weights(
            normalized_base,
            0.20,
        )

        return (
            {
                "collaborative": 0.35,
                "content": 0.45,
                "popularity": popularity_weight,
                "rating": rating_weight,
            },
            "warm-start: 5-9 оцінок, поступове підсилення collaborative",
        )

    return normalized_base, "standard: достатньо історії для базових ваг"
```

ФОРМУВАННЯ СПІЛЬНИХ TF-IDF-ТЕРМІНІВ ДЛЯ ПОЯСНЕНЬ

```
def get_common_tfidf_terms(
```

```
    movie_id: int,
    liked_movie_id: int,
    tfidf_matrix,
    tfidf_feature_names: np.ndarray,
    movie_id_to_index: dict[int, int],
    top_n: int = 5,
) -> list[str]:
    movie_index = movie_id_to_index.get(int(movie_id))
    liked_index = movie_id_to_index.get(int(liked_movie_id))

    if movie_index is None or liked_index is None:
        return []

    movie_vector = tfidf_matrix[movie_index]
    liked_vector = tfidf_matrix[liked_index]

    common_vector = movie_vector.multiply(liked_vector)

    if common_vector.nnz == 0:
        return []

    common_scores = common_vector.toarray().ravel()
    top_indices = common_scores.argsort()[::-1]

    terms = []
    for index in top_indices:
        if common_scores[index] <= 0:
            break

        term = str(tfidf_feature_names[index])
        if term and term not in terms:
            terms.append(term)

        if len(terms) >= top_n:
```

```
            break

    return terms
```

ФОРМУВАННЯ ТЕКСТОВИХ ПОЯСНЕНЬ ДО РЕКОМЕНДАЦІЙ

```
def build_reason_details_html(
    row: pd.Series,
    user_id: int,
    rating_count: int,
    liked_count: int,
    merged_ratings: pd.Series,
    liked_movie_ids: set[int],
    content_similarity: np.ndarray,
    tfidf_matrix,
    tfidf_feature_names: np.ndarray,
    movie_id_to_index: dict[int, int],
    title_by_id: dict[int, str],
    runtime_user_movie_matrix: pd.DataFrame,
    historical_similarities: pd.Series,
    all_custom_ratings: dict[int, dict[int, float]],
) -> str:
    content_text = get_content_evidence_text(
        int(row["movieId"]),
        liked_movie_ids,
        merged_ratings,
        content_similarity,
        tfidf_matrix,
        tfidf_feature_names,
        movie_id_to_index,
        title_by_id,
    )

    collaborative_text = get_collaborative_evidence_text(
```

```
    row,
    user_id,
    rating_count,
    liked_count,
    runtime_user_movie_matrix,
    historical_similarities,
    all_custom_ratings,
)

popularity_norm = float(row.get("popularity_norm", 0.0))
popularity_background = float(row.get("popularity_background", 0.0))

rating_norm = float(row.get("rating_norm", 0.0))
rating_background = float(row.get("rating_background", 0.0))

popularity_text = (
    f"Popularity {float(row['popularity']):.2f}, "
    f"normalized {popularity_norm:.3f}: "
    f"{get_relative_text(popularity_norm, popularity_background)}."
)

rating_text = (
    f"TMDB rating {float(row['vote_average']):.1f}/10, "
    f"normalized {rating_norm:.3f}: "
    f"{get_relative_text(rating_norm, rating_background)}."
)

return (
    '<div class="reason-grid">'
    f'<div class="reason-block"><b>Content</b><br>{escape(content_text)}</div>'
    f'<div class="reason-block"><b>Collaborative</b><br>{escape(collaborative_text)}</div>'
    f'<div class="reason-block"><b>Popularity</b><br>{escape(popularity_text)}</div>'
    f'<div class="reason-block"><b>TMDB rating</b><br>{escape(rating_text)}</div>'
    "</div>"
```

```
    )
```

ГЕНЕРАЦІЯ ГІБРИДНИХ РЕКОМЕНДАЦІЙ У ВЕБ-ЗАСТОСУНКУ

```
def recommend_movies(
    user_id: int,
    ratings: pd.DataFrame,
    movies: pd.DataFrame,
    content_similarity: np.ndarray,
    tfidf_matrix,
    tfidf_feature_names: np.ndarray,
    user_movie_matrix: pd.DataFrame,
    user_similarity: pd.DataFrame,
    custom_ratings: dict[int, float],
    all_custom_ratings: dict[int, dict[int, float]],
    top_n: int,
    weights: dict[str, float],
) -> pd.DataFrame:
    normalized_weights = normalize_weights(weights)
    merged_ratings = merge_user_ratings(user_id, ratings, custom_ratings)

    watched_movie_ids = set(merged_ratings.index.astype(int).tolist())

    liked_movie_ids = set(
        merged_ratings[
            merged_ratings >= MIN_LIKED_RATING
        ].index.astype(int).tolist()
    )

    candidate_movies = movies.loc[
        ~movies["movieId"].isin(watched_movie_ids)
    ].copy()

    if candidate_movies.empty:
```

```
    return candidate_movies

movie_id_to_index = dict(zip(movies["movieId"].astype(int), movies.index))
rating_count = int(len(merged_ratings))
liked_count = int(len(liked_movie_ids))

adaptive_weights, adaptive_reason = get_adaptive_weights(
    normalized_weights,
    rating_count,
    liked_count,
)

collaborative_scores = get_collaborative_scores(
    user_id=user_id,
    ratings=ratings,
    user_movie_matrix=user_movie_matrix,
    user_similarity=user_similarity,
    custom_ratings=custom_ratings,
    all_custom_ratings=all_custom_ratings,
)

candidate_movies["collaborative_score"] = (
    candidate_movies["movieId"]
    .map(collaborative_scores)
    .fillna(0)
    .astype(float)
)

candidate_movies["content_score"] = get_content_scores(
    candidate_movies,
    liked_movie_ids,
    content_similarity,
    movie_id_to_index,
)
```

```
candidate_movies["popularity_score"] = np.log1p(
    candidate_movies["popularity"].clip(lower=0)
)

candidate_movies["rating_score"] = candidate_movies["vote_average"].clip(lower=0)

candidate_movies["collaborative_norm"] = normalize_series(
    candidate_movies["collaborative_score"]
)
candidate_movies["content_norm"] = normalize_series(
    candidate_movies["content_score"]
)
candidate_movies["popularity_norm"] = normalize_series(
    candidate_movies["popularity_score"]
)
candidate_movies["rating_norm"] = normalize_series(
    candidate_movies["rating_score"]
)

candidate_movies["final_score"] = (
    adaptive_weights["collaborative"] * candidate_movies["collaborative_norm"]
    + adaptive_weights["content"] * candidate_movies["content_norm"]
    + adaptive_weights["popularity"] * candidate_movies["popularity_norm"]
    + adaptive_weights["rating"] * candidate_movies["rating_norm"]
)

candidate_movies = add_hybrid_shap_explanations(
    candidate_movies,
    adaptive_weights,
)

recommendations = candidate_movies.sort_values(
    ["final_score", "vote_average", "popularity"],
```

```
        ascending=False,
    ).head(top_n).copy()

    recommendations["adaptive_reason"] = adaptive_reason

    return recommendations
```

ГЕНЕРАЦІЯ РЕКОМЕНДАЦІЙ CLASSIC SVD

```
def recommend_movies_svd(
    user_id: int,
    ratings: pd.DataFrame,
    movies: pd.DataFrame,
    svd_artifacts: dict[str, object] | None,
    custom_ratings: dict[int, float],
    top_n: int,
) -> pd.DataFrame:
    if not isinstance(svd_artifacts, dict) or svd_artifacts.get("model") is None:
        return pd.DataFrame()

    merged_ratings = merge_user_ratings(user_id, ratings, custom_ratings)
    watched_movie_ids = set(merged_ratings.index.astype(int).tolist())

    candidate_movies = movies.loc[
        ~movies["movieId"].isin(watched_movie_ids)
    ].copy()

    if candidate_movies.empty:
        return candidate_movies

    candidate_movies, factor_source, global_mean, _, _ = score_svd_candidate_movies(
        user_id,
        merged_ratings,
        candidate_movies,
```

```
        svd_artifacts,
    )

    recommendations = (
        candidate_movies.sort_values(
            ["svd_predicted_rating", "vote_average", "popularity"],
            ascending=False,
        )
        .head(top_n)
        .copy()
    )

    recommendations["svd_reason_details_html"] = recommendations.apply(
        lambda row: build_svd_reason_details_html(
            row=row,
            known_rating_count=int(len(merged_ratings)),
            factor_source=factor_source,
            global_mean=global_mean,
        ),
        axis=1,
    )

    return recommendations
```

ГЕНЕРАЦІЯ ПЕРСОНАЛІЗОВАНИХ РЕКОМЕНДАЦІЙ SVD

```
PERSONALIZED_SVD_WEIGHTS = {
    "neighbor_similarity": 0.50,
    "latent_match": 0.25,
    "predicted_rating": 0.15,
    "movie_bias": 0.10,
}
```

```
def recommend_movies_personalized_svd(
    user_id: int,
    ratings: pd.DataFrame,
    movies: pd.DataFrame,
    svd_artifacts: dict[str, object] | None,
    custom_ratings: dict[int, float],
    top_n: int,
) -> pd.DataFrame:
    if not isinstance(svd_artifacts, dict) or svd_artifacts.get("model") is None:
        return pd.DataFrame()

    merged_ratings = merge_user_ratings(user_id, ratings, custom_ratings)
    watched_movie_ids = set(merged_ratings.index.astype(int).tolist())

    candidate_movies = movies.loc[
        ~movies["movieId"].isin(watched_movie_ids)
    ].copy()

    if candidate_movies.empty:
        return candidate_movies

    scored_candidates, _, _, rating_min, rating_max = score_svd_candidate_movies(
        user_id,
        merged_ratings,
        candidate_movies,
        svd_artifacts,
    )

    liked_ratings = merged_ratings[merged_ratings >= MIN_LIKED_RATING]

    candidate_movie_ids = set(
        scored_candidates["movieId"].astype(int).tolist()
    )
```

```
neighbor_metadata = build_svd_latent_neighbor_candidates(
    liked_ratings,
    candidate_movie_ids,
    movies,
    svd_artifacts,
)

if liked_ratings.empty:
    personalized_status = "no-liked-fallback"
    personalized_candidates = scored_candidates.copy()
    personalized_candidates["neighbor_similarity"] = 0.0
    personalized_candidates["personalized_source"] = "Classic SVD fallback"
else:
    personalized_status = "personalized"
    personalized_candidates = scored_candidates.copy()

personalized_candidates["personalized_rank_score"] = (
    PERSONALIZED_SVD_WEIGHTS["predicted_rating"]
    * normalize_series(personalized_candidates["svd_predicted_rating"])
    + PERSONALIZED_SVD_WEIGHTS["movie_bias"]
    * normalize_series(personalized_candidates["svd_movie_bias"])
).clip(lower=0, upper=1)

recommendations = personalized_candidates.sort_values(
    ["personalized_rank_score", "svd_predicted_rating", "vote_average"],
    ascending=False,
).head(top_n).copy()

recommendations["personalized_status"] = personalized_status

return recommendations
```