Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мережних технологій факультету інформатики

РОЗРОБКА АРХІТЕКТУРИ РЕКОМЕНДАЦІЙНОЇ СИСТЕМИ ІЗ ЗАСТОСУВАННЯМ ТЕХНІК МАШИННОГО НАВЧАННЯ

Текстова частина до курсової роботи
за спеціальністю «Комп'ютерні науки» 122

Керівник курсової роботи
доктор технічних наук,
доцент Глибовець А.М.

_____

(підпис)

«____» _____ 2022 р.

Виконав студент Грицюк О.О.

«____» _____ 2022 р.

Київ – 2022

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мережних технологій факультету інформатики

ЗАТВЕРДЖУЮ

Завідувач кафедри мережних технологій,

доктор ф.-м. наук

Малашонок Г. І.

_____

(підпис)

«____» _____ 2022 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студента 4 курсу факультету інформатики

Грицюка Олександра Олександровича

Тема: Розробка архітектури рекомендаційної системи із застосуванням технік машинного навчання

Зміст текстової частини до курсової роботи:

- Анотація
- Вступ
- Розділ 1. Підходи до розробки рекомендаційних систем
- Розділ 2. Створення гібридної рекомендаційної системи
- Список використаних джерел

Дата видачі «____» _____ 2022 р.

Керівник _____

(підпис)

Завдання отримав _____

(підпис)

Тема: Розробка архітектури рекомендаційної системи із застосуванням технік машинного навчання

Календарний план виконання курсової роботи:

	Назва етапу курсової роботи	Термін виконання етапу	Примітка
	Отримання завдання на курсову роботу	10.11.2021	
	Пошук джерел за темою роботи	01.02.2021	
	Ознайомлення з джерелами за темою роботи	01.03.2022	
	Написання першого розділу текстової частини	01.04.2022	
	Розробка програмної системи	01.05.2022	
	Написання другого розділу текстової частини	10.05.2022	
	Захист курсової роботи	20.05.2022	

Зміст

Анотація

Дана робота присвячена розробці архітектури рекомендаційних систем із застосуванням технік машинного навчання для розв’язання задачі рекомендацій - виділення релевантної частини каталогу контенту для користувача на основі його уподобань. Розглянуто процес розробки та фази роботи рекомендаційних систем. Описано різні критерії оцінки та способи тестування відповідних алгоритмів. Наведено алгоритми машинного навчання, які застосовуються в даній сфері, порівняно їхні недоліки та переваги. Поєднано деякі з них (декомпозиція матриці оцінок, колаборативна фільтрація, кластеризація та фільтрація на основі контенту) у гібридну рекомендаційну систему за допомогою лінійної регресії.

Вступ

Із зростанням кількості та масштабів автоматизованих інформаційних систем, які пропонують різні спектри пропозицій, послуг та товарів користувачам, останнім стає важко слідкувати за усім каталогом та швидко знаходити бажаний контент. Інтернет-магазини збільшують різноманіття категорій та товарів в них, стрімінгові платформи щоденно поповнюють каталог новими жанрами та релізами тощо. Клієнти таких систем все частіше відчувають необхідність фільтрувати усі пропозиції за релевантністю, мати швидкий доступ до такої інформації, яка потенційно має їм сподобатися, замість того, аби самотужки переглядати повний діапазон каталогу.

Рекомендаційні системи дозволяють вирішити дану проблему. На основі аналізу даних про користувачів, контент та результати їхньої взаємодії, такі системи дозволяють надати користувачу набір певних пропозицій, які з високою ймовірністю мають йому сподобатися. Серед вимог до них стоїть висока швидкість підбору рекомендацій, а також висока точність передбачення.

В межах даної роботи за мету поставлено розглянути етапи виконання рекомендаційних систем та способи оцінки їхньої точності та інших характеристик, порівняти відомі алгоритми машинного навчання для розв’язання даної задачі, визначити їхні недоліки та переваги, а також реалізувати власну гібридну рекомендаційну систему для пропозицій фільмів користувачам, поєднавши декілька описаних підходів.

Об’єктом дослідження є рекомендаційні системи для виділення релевантної частини каталогу, предметом дослідження - алгоритми машинного навчання для розробки таких систем, способи їхнього поєднання та оцінки результату.

Дослідження складається з двох розділів.

Перший з них з них містить загальні відомості про рекомендаційні системи, фази їхньої роботи, способи оцінки таких систем, а також порівняння алгоритмів для їхньої розробки.

Другий розділ розкриває процес розробки власної рекомендаційної системи, деталі реалізації та вибір моделей на основі їхньої точності. Також в ній наявна інформація про використані дані та засоби розробки.

Постановка задачі:

1. Проаналізувати етапи роботи та критерії якості рекомендаційних систем, навести метрики для оцінки їхньої точності.
2. Дослідити методи та алгоритми машинного навчання, які використовуються для розробки рекомендаційних систем, а також описати способи їхнього поєднання.
3. Розробити гібридну рекомендаційну систему, яка пропонуватиме користувачам фільми на основі їхніх інтересів.

1. Підходи до розробки рекомендаційних систем

1.1. Загальна інформація

Рекомендаційні системи – це такі системи, які з-поміж великого обсягу інформації виділяють таку її частину, яка є релевантною для користувача з точки зору його уподобань, поведінки та інтересів [1].

Вони є корисними як для користувачів, так і для компаній, які постачають відповідні товари/послуги, адже допомагають швидко знаходити схожі пропозиції, будувати якісну персоналізовану добірку та збільшувати обсяг продажів.

1.2. Фази роботи

Робота рекомендаційних систем зазвичай складається із декількох фаз, які виконуються циклічно одна за одною [1].

1.2.1. Фаза збору інформації

Успіх рекомендаційних систем значно залежить від якості інформації, яку вдалося зібрати про користувачів, а також від її актуальності. Застосунок має якомога точніше знати уподобання людей, аби пропонувати контент згідно з їхнім зацікавленням, які з найбільшою ймовірністю будуть їм до вподоби. Головною концепцією тут є зворотній зв’язок від користувача, який буває двох типів.

Явний зворотній зв’язок (*Explicit feedback*) характеризується більшою якістю та очевидністю інтерпретації, оскільки дає змогу користувачу самостійно оцінити певну послугу за заданою шкалою, на основі чого можна вже робити висновки про його інтереси (наприклад, сподобалося чи не сподобалося, 7 балів із 10 тощо). Але не завжди людина готова до цього: в неї може не вистачати часу та сил, може викликати складнощі оцінити предмет за небінарною шкалою тощо. Крім того, тут грає роль соціальний фактор, через який людина часто відповідає не правдиво, а так, як від неї вимагає оточення.

З іншого боку, неявний зворотній зв’язок (*Implicit feedback*) дозволяє простежити за поведінкою користувача, яка є природною для нього (наприклад, тривалість прослуховування пісні, історія перегляду сторінок тощо). Даний метод не вимагає жодних зайвих дій від людини, завдяки чому дозволяє зібрати більше інформації. Серед недоліків можна виділити складність інтерпретації різноманітної поведінки. Як приклад, довгий перегляд товару може свідчити як про зацікавленість, так і про те, що користувач просто відійшов від електронного пристрою по своїм справам.

Достатньо гарний результат може дати збір гібридного зворотнього зв’язку (*Hybrid feedback*), тобто такого, який поєднує переваги явного і неявного. У такому разі, можна давати користувачу змогу поставити оцінку лише тоді, коли застосунок робить висновок про його зацікавленість на основі поведінки. Або навпаки, перевіряти достовірність явного відгуку за допомогою неявних критеріїв.

1.2.2. Фаза навчання

Протягом даного періоду часу, система робить певні висновки про дані, знаходить закономірності в них за допомогою різноманітних алгоритмів машинного навчання (*Machine learning*). Докладніше про способи оцінки, типи алгоритмів та метрики, які використовуються в даному процесі, описано у наступних розділах.

1.2.3. Фаза передбачень/рекомендацій

Останній етап присвячений саме застосуванню навчених моделей в основному застосунку. Система починає пропонувати клієнтам товари та послуги, які можуть їм сподобатися, задовольняючи їхні потреби та збільшуючи продажі бізнесу. Наступні розділи містять також інформацію про те, які існують практики для впровадження нових версій систем.

1.3. Оцінка та тестування

1.3.1. Критерії оцінки

Задача рекомендацій не є тривіальною та потребує оцінки одразу з точки зору кількох аспектів. [2]

- Мінімізація помилки передбачення (*Minimizing prediction error*).

  Даний критерій дозволяє зрозуміти, наскільки точно система здатна розуміти уподобання користувача. Цей параметр можна виміряти порівнявши передбачену оцінку та фактично виставлену користувачем. Або можна використати аналіз неявного зворотного зв'язку, аби зробити висновки, чи коректно було рекомендовано послугу (наприклад, якщо користувач швидко вимкнув фільм, який запропонувала система, та не додивився його – це може свідчити про помилку передбачення).

- Різноманітність (*Diversity*).

  Важливою рисою запропонованої добірки товарів чи послуг є різноманітність, адже, користуючись такими системами, користувач хоче бачити більш унікальний перелік, аніж він сам може знайти, відкривши каталог певного жанру чи типу. Процедура вимірювання даної характеристики не є очевидною, а тим більше важко підтримувати результати передбачення різноманітними для кожного з користувачів окремо. Одним зі способів є розрахунок несхожості між кожною парою об'єктів множини рекомендованої добірки.

- Покриття (*Coverage*).

  Однією з цілей впровадження рекомендаційних систем є показати користувачам якомога ширше коло пропозицій, досягти високого покриття товарів чи послуг. Таким чином, варто уникати ситуацій, коли якась одиниця каталогу ніколи не показується користувачам, адже це негативно впливає на продажі або результативність в цілому самої компанії. З іншої сторони, можна розглядати покриття користувачів,

тобто щоб для кожного користувача система могла запропонувати певну добірку: хоч це щойно зареєстрований клієнт, хоч прихильник будь-якого жанру, інформація про якого вже міститься в системі.

- Випадковість (*Serendipity*).

  Користувачі зазвичай приємно вражені, коли уподобали якийсь новий контент, про який раніше навіть не здогадувалися. Це може бути книга чи пісня нового жанру, або куплений товар невідомої фірми в інтернет-магазині. Це дозволяє їм отримати новий унікальний досвід, а ресурси, знову ж таки, збільшують обсяг споживчого контенту. Дана риса є доволі суб'єктивною та не так легко вимірюється. Важливою порадою є якомога менше обмежувати результати рекомендацій, аби не пропустити потенційно якісний контент.

1.3.2. Тестування алгоритмів

Тестування та впровадження нових версій рекомендаційних систем можуть містити у собі декілька етапів. Кожен з них має свої переваги та недоліки, тому їх часто поєднують для більшої ефективності [2].

1.3.2.1. Оффлайн тестування

Оффлайн тестування (*Offline testing*) використовує наявні дані та вимірює, наскільки точним та ефективним є алгоритм. Для цього необхідно поділити вибірнку на дві частини: навчальну (*learning*) та тестову (*testing*). Перша використовуватиметься для тренування моделей та проведення інших розрахунків, в той час як на другій розраховуватиметься точність алгоритму. Часто вводять ще третю проміжну частину для перехресної валідації (*cross-validation*), аби обрати одну найкращу модель із декількох навчених, яку вже потім тестувати на незалежних даних.

Першим методом є вимірювання похибки, для цього застосовуються різні метрики (*Error Metrics*), наприклад: середнє абсолютне відхилення (*Mean Absolute Error*), середнє квадратичне відхилення (*Mean Squared Error*) та кореневе середнє

квадратичне відхилення (*Root Mean Squared Error* – формула 1.1). Таким чином, розраховується, наскільки запропонована рекомендаційною системою оцінка збігається з фактичною оцінкою користувача.

$$MAE = \frac{1}{|R|}\sum_{r \in R} |predicted(r) - actual(r)|$$

$$MSE = \frac{1}{|R|}\sum_{r \in R} (predicted(r) - actual(r))^2$$

$$RMSE = \sqrt{\frac{1}{|R|}\sum_{r \in R} (predicted(r) - actual(r))^2}$$

R – набір рекомендованих об′єктів

$predicted(r)$ – запропонована системою оцінка

$actual(r)$ – фактична оцінка користувача

Формула 1.1. Метрики помилок

Із недоліків даного підходу можна виділити відсутність нагальної потреби для користувача очікувати точну оцінку від системи, адже він хоче лише дізнатися про новий контент, який може йому сподобатися; також часто постає за мету показувати більш релевантні послуги першими у списку, в той час як дані метрики беруть до уваги кожен елемент однаково.

Іншим варіантом є використання метрик прийняття рішень (*Decision-support Metrics* – формула 1.2), в межах яких розрізняють чотири види відповідей системи: істинно-позитивні (*True Positive*), істинно-негативні (*True Negative*), хибно-позитивні (*False Positive*) та хибно-негативні (*False Negative*). Позитивними називаються відповіді, коли об’єкт є рекомендованим системою, а істинними - коли думка користувача збігається із рішенням системи. За допомогою них є змога порахувати точність (*Precision*) - яка частина рекомендованого набору є релевантною, та повноту (*Recall*) - яку частину релевантних товарів повернула рекомендаційна система. Для зручності порівняння різних результатів у вигляді однієї, а не двох величин, дані дві метрики поєднуються у F_1 міру (F_1 - *score*), яка дозволяє оцінити баланс між точністю та повнотою. З іншої сторони, часом важливо оцінити точність не всього результату, а перших k елементів у списку (*Precision at K* або *P@k*), адже саме вони є найважливішими.

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

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

$$F_1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

$$P@k(u) = \frac{p}{k}$$

p – кількість релевантних елементів серед перших k

u – набір рекомендацій

Формула 1.2. Метрики прийняття рішень

Повертаючись до проблеми різної важливості елементів у рекомендованому списку, слід згадати про метрики ранжування (*Ranking Metrics* – формула 1.3), які дозволяють оцінити не лише наповнення набору рекомендацій, а й порядок елементів в ньому. Завдяки цьому, система зможе показувати найбільш релевантні, на її думку, товари з самого початку, де існує найбільша ймовірність вибору користувача. Середня точність (*Average Precision*) надає більше ваги точності перших елементів, аніж останніх, а нормована середня точність (*Mean Average Precision*) дозволяє поєднати середню точність декількох наборів рекомендацій для загальної оцінки системи.

$$AP(u) = \frac{\sum_{k=1}^{|u|} P@k(u)}{|u|}$$

$$MAP = \frac{\sum_{u \in U} AP(u)}{|U|}$$

U – усі набори рекомендацій системи

Формула 1.3. Метрики ранжування

Як можна побачити, існує багато метрик для оффлайн тестування, за допомогою яких можна оцінювати та порівнювати результати алгоритмів з метою вибору кращого з них. Залишається лише питання, як правильно ділити наявні дані на навчальний, тестовий та валідаційні набори. Для вирішення даного питання можна поєднувати декілька способів:

- виділення частини даних із збереженням розподілу ключових ознак
- відкидання об’єктів, для яких наявно занадто мало даних
- розділення випадковим чином

- розділення на основі часу
- розділення взаємодій в межах користувача

Корисно також згадати про *k*-згортку (*k-fold*), основною ідеєю якої є розділення набору даних на *k* частин, та по черзі брати одну з них у якості тестової вибірки, навчаючи модель по іншим *k-1*; після чого, знайти середнє значення обраної метрики.

1.3.2.2. Онлайн тестування

Як тільки алгоритми протестовано на наявних даних, та обрано найкращий з них, можна починати впроваджувати оновлення у реальний застосунок.

Спочатку рекомендується виконати так звані контрольовані експерименти (*Controlled experiments*), суть яких полягає у тому, що вживу прослідковується реакція та дії реальних людей. Це може бути як випадкові люди, так і, наприклад, члени родини та друзі, колеги тощо, які можуть дати чесний зворотній зв’язок. Такі експерименти дозволяють оцінити, як користувачі реагують на рекомендації та систему в цілому, а також, мати змогу поставити додаткові питання. Щоправда, не завжди люди під наглядом поводять себе так, як наодинці, тому даний фактор необхідно також враховувати.

Достатньо поширеною технікою впровадження оновлень є *A/B* тестування (*A/B Testing*). В межах даного підходу невелика частина користувачів перенаправляються на нову версію певної частини додатку (в даному випадку, для них застосовується новий алгоритм рекомендацій), в той час як для інших нічого не змінюється. Головна ідея полягає в тому, що це дозволяє абстрагуватися від зовнішніх факторів чи подій, які також можуть вплинути на поведінку клієнтів, та порівняти алгоритми відносно одне одного. Навіть якщо нова версія рекомендаційної системи буде невдалою, то від цього постраждає лише невелика частина користувачів; а якщо алгоритм показує покращення, то він стає основним та застосовується до усіх користувачів до наступної ітерації тестування. Все більше популярних сервісів починають застосовувати дану техніку для будь-яких нововведень.

Логічним розширенням попереднього методу є підхід експлуатації та дослідження (*Exploit/Explore*), адже він тривалий час підтримує декілька версій алгоритму, які націлені на різні критерії оцінки, та застосовує різні з них час від часу. Це можна пояснити таким чином: переважну частину часу користувачі хочуть бачити надійні рекомендації, можливо навіть просто популярний контент без персоналізації, але іноді вони хочуть чогось нового, аби розширити кругозір. Даний метод позбавляє необхідності сумістити усі критерії та метрики в одній рекомендаційній системі, а навпаки, більш точно навчити моделі під конкретні задачі та дозволити користувачам взаємодіяти з ними всіма.

1.4. Типи алгоритмів

Існують декілька типів алгоритмів, які можна застосовувати для розробки рекомендаційних систем, кожен з яких має свої переваги та недоліки .

1.4.1. Фільтрація на основі контенту

Одним із основних методів рекомендації релевантних пропозицій є фільтрація на основі контенту (*Content-based Filtering*), суть якої полягає у пошуку схожих товарів до тих, які вже вподобав користувач. Для побудови моделі необхідно три елементи: аналізатор контенту, профіль користувача та засіб отримання нових позицій каталогу [3].

Для аналізу контенту необхідно виділити набір ознак, які будуть характеризувати об’єкт. Найпростішим прикладом є бінарні та числові ознаки, наприклад: чи знімався певний актор у фільмі, чи належить товар до певної категорії, якого року випущений автомобіль або яка тривалість вистави.

Але часто у наповненні контенту може бути вільно текстовий опис, як-от анотація до книги. У такому разі необхідно виконати деяку обробку даних. Для початку варто забрати з тексту так звані стоп-слова (*stop-words*) - це такі частини речень, які не несуть жодного смислового навантаження (сполучники тощо). Після чого слід застосувати стемінг (*stemming* - відкидання допоміжних частин слова) або лематизацію (*lemmatization* - знаходження первісної форми слова), що дасть змогу

нормалізувати слова та знаходити близькі за сенсом документи. Таким чином, наступним кроком ми зможемо представити текст у вигляді мішку слів (*bag of words*), де документ перетворюється у множину нормалізованих слів задля порівняння тематичного наповнення. Звісно, при такому підході ми можемо втратити важливі деталі, що містяться у порядку слів, проте ми значно спрощуємо алгоритм - аби нівелювати даний недолік, існують засоби обробки природної мови (*Natural Language Processing*), які дозволяють глибше аналізувати кожну частину речення.

Ключовою частиною є представлення мішка слів у зручній формі для подальшого аналізу. Найбільш популярними є два підходи: індекс *tf-idf (term frequency, inverse document frequency)* та *LDA* (*Latent Dirichlet allocation*).

Перший з них, *tf-idf* (формула 1.4), для кожного унікального слова в межах документа ставить у відповідність кількість - скільки разів воно зустрілося у документі (*tf - term frequency*). Потім дану величину треба помножити на логарифм оберненої частки документів (всього *n*) загалом, в якому воно зустрічається (*df - document frequency*). Таким чином, чим більше отримане число, тим частіше зустрічається дане слово в даному документі, і рідше - в інших, тобто є більш характерною та унікальною особливістю.

$$tf - idf = tf * log(\frac{n}{df})$$

Формула 1.4. tf-idf

Другий підхід, *LDA*, є певним розширенням попереднього завдяки тому, що оперує на трохи вищому рівню узагальнення. Завдяки йому можно представити текст у вигляді співвідношення різних тем (які завчасно формуються за допомогою окремих алгоритмів навчання), замість частоти окремих слів, що дозволяє краще обробляти саме семантику, а не синтаксис. Як результат, чим більшою є частина певної теми у розкладі, тим з більшою ймовірністю можна сказати, що документ належить до даної тематики. Суттєвою перевагою є зменшення кількості ознак, у порівнянні з *tf-idf*, адже кілька слів об’єднуються в одну тему.

На даному етапі вже можливо сформувати вектор із числовими значеннями для кожного документа. Числові ознаки переносяться у явному вигляді, бінарні заміняються на нулі та одиниці, а із тексту формуються ознаки у вигляді слів (*tf-idf*) або тем (*LDA*) із відповідними числовими величинами. Перед подальшою обробкою варто виконати також нормалізацію усіх векторів для більш точної обробки.

Після створення аналізатора контенту необхідно побудувати модель для представлення профіля користувача, за яким потім будуть шукатися схожі елементи каталогу до тих, які сподобалися клієнту. Це має бути так само числовий вектор, тієї ж розмірності та порядку ознак, як і векторі товарів, аби мати змогу порівнювати їх. Для його генерації можна застосовувати різні функції від елементів каталогу, які вже спожиті користувачем, як-от: застосування техніки *tf-idf* для бінарних ознак (де *tf* відповідає відношення кількості вподобаних товарів із даною ознакою, *df* - кількість користувачів, які позитивно оцінили хоч один товар із нею), а також знаходження середнього зваженого для числових, де у вектора товару тим більша вага, чим більша оцінка від користувача. Отриманий результат можна інтерпретувати як вподобання клієнта по конкретним ознаками.

Фінальною частиною фільтрації на основі контенту є власне генерація рекомендацій. Для цього необхідно взяти вектор профіля користувача, знайти контент із якомога більш схожими векторами ознак та представити його користувачу, у порядку спадання схожості або зростання відстані. Для цього можна використати, наприклад, косинусну міру або евклідову відстань (формула 1.5).

$$similarity_1(A,B) = \frac{A * B}{|A| * |B|} = \frac{\sum_i \quad A_i * B_i}{\sqrt{\sum_i (A_i)^2} * \sqrt{\sum_i (B_i)^2}}$$

$$similarity_2(A,B) = -\sqrt{\sum_i (A_i - B_i)^2}$$

Формула 1.5. косинусна міра та евклідова відстань

Отже, даний підхід до підбору рекомендацій дозволяє швидко додавати до алгоритму нові елементи каталогу, розраховуючи схожість з існуючими,

показувати рекомендації вже після першої оцінки користувача, а також показувати контент незалежно від поточної популярності. З іншого боку, рекомендований контент буде сильно прив’язаний до спожитого користувачем, а значить, алгоритм не може запропонувати новий жанр або категорію, яка потенційно сподобалася би клієнту.

1.4.2. Колаборативна фільтрація

Інший популярний підхід для підбору рекомендацій називається колаборативної фільтрацією (*Collaborative filtering*). Він значною мірою відрізняється від попереднього, адже жодним чином не враховує наповнення контенту. Натомість, загальна ідея полягає в тому, що якщо у користувачів співпадають оцінки по більшості товарів, то з високою ймовірністю вони будуть співпадати і по новим пропозиціям [4].

Для роботи із даним методом необхідно таке поняття як матриця оцінок (*Rating matrix*). Кожна комірка показує, яку оцінку поставив конкретний користувач певному елементу. Нехай надалі рядки позначають користувачів, а стовпці - позиції каталогу. Звичайно, значення може бути порожнім, якщо взаємодія даної пари ще не відбулася.

Загалом, колаборативна фільтрація дозволяє передбачити оцінку (порожню комірку матриці оцінок) та має два види: на основі користувачів та на основі елементів каталогу.

Колаборативна фільтрація на основі користувачів (*User-based collaborative filtering*) починається із розрахунку подібності користувачів між собою. Кожного користувача можна представити у вигляді нормалізованого вектора оцінок товарів (рядки матриці оцінок). Варто зазначити, що у зв’язку із наявністю порожніх значень, можна використати метрику кореляції Пірсона, яка є досить схожою із косинусною мірою, однак враховує лише ті елементи, які наявні в обох векторах (формула 1.6).

$$similarity_3(A,B) = \frac{\sum_{i \in P} A_i * B_i}{\sqrt{\sum_{i \in P}(A_i)^2} * \sqrt{\sum_{i \in P}(B_i)^2}}$$

P — множина індексів, відповідні елементи яких наявні в обох векторах

Формула 1.6. кореляція Пірсона

Для того аб передбачити оцінку користувача для певного об'єкта (заповнити порожню клітинку матриці), наступним кроком необхідно виділити для клієнта так зване сусідство (*neighbourhood*) - найбільш схожих користувачів, оцінки яких і будуть використовуватися для передбачення оцінки даного користувача для товару. Наостанок, залишається власне розрахувати прогнозовану оцінку, наприклад, методом середнього зваженого оцінок сусідів для даного елемента, де вагою виступає величина схожості із користувачем. Або можна порахувати, яка оцінка зустрічається найчастіше серед сусідів, що і буде результатом передбачення.

Колаборативна фільтрація на основі елементів каталогу (*Item-based collaborative filtering*) повторює усі наведені вище кроки, з однією лише відмінністю - джерелом оцінок, що використовуються для передбачення. Алгоритм потребує знайти подібність між усіма товарами (замість користувачів), де елементами векторів виступають оцінки клієнтів (стовпці матриці інтересів). Далі з метою передбачення оцінки користувача для об'єкта, слід обрати найближчих сусідів товару та на основі оцінок для них заданого користувача розрахувати передбачене значення (так само, чим більш схожий сусід на заданий елемент каталогу, тим суттєвішою є оцінка заданого користувача для нього).

Залишається вирішити, як саме варто обирати сусідство користувача/елементу каталогу. Існують два основні способи, одним з яких є взяття *N* найближчих сусідів (*Top-N*), а другим - встановлення порогу (*Threshold*), такого що всі об'єкти зі схожістю, що перевищує його, будуть вважатися сусідами. Перший варіант варто обирати, коли кількість важливіша (необхідно набрати достатньо сусідів), а другий - коли якість (необхідно враховувати лише дійсно схожі об'єкти). Можна провести навчання як по метрикам схожості, так і по кількості сусідів/величині порогу, аби знайти той підхід, який працює найкраще на обраних даних.

Як висновок, колаборативна фільтрація дозволяє знаходити рекомендації, які можуть значно відрізнятися від вже спожитого користувачем контенту, що добре відображається на різноманітності. Щоправда, із даним підходом з'являється проблема холодного старту (*Cold-start problem* - коли на початок роботи алгоритму недостатньо даних для його виконання, наприклад щойно доданий товар не буде ніким оцінений певний час, а отже не з'явиться ні в кого у рекомендаціях) та розрідженності даних (адже необхідно, аби оцінки перекривалися в межах користувача чи елементу каталогу для пошуку схожих інтересів на основі оцінок). Крім того, на великих обсягах даних пошук сусідства починає займати все більше часу.

1.4.3. Декомпозиція матриці

Доволі поширеним є метод декомпозиції матриці (*Matrix decomposition*), суть якого полягає в тому, що необхідно виконати факторизацію матриці оцінок, зменшивши розмірність даних. Даний підхід розділяє ідею колаборативної фільтрації щодо того, що оцінку взаємодії можна вивести із наявних оцінок у системі. Щоправда, має відмінність у реалізації - замість пошуку сусідів виконується побудова моделі, яку можна буде використати для передбачення оцінки [5].

Для вирішення даної зазвичай застосовують алгоритм *SVD* (*Singular value decomposition*), який приймає на вхід матрицю M оцінок та оптимізує її розклад - три матриці (формула 1.7). Принцип роботи даного алгоритму знаходиться поза межами даного дослідження.

$$M = U * \boldsymbol{\Sigma} * \boldsymbol{V^T}$$

Формула 1.7. Розклад матриці SVD

Матриці U та V^T в контексті рекомендаційних систем називають відповідно матрицею користувачів та предметів. Вони містять у собі приховані зв'язки, які притаманні обом сутностям - ті ознаки та значення, які визначають початкову таблицю. У свою чергу, матриця $\boldsymbol{\Sigma}$ називається сингулярною. Її головною властивістю є те, що вона діагональна, елементи якої спадають від лівого верхнього

до правого нижнього кутів. Саме вона дозволяє управляти розмірністю результату, її елементи показують наскільки суттєвими є вирахувані ознаки, а значить, дозволяє обрізати матриці. Таким чином, залишивши N перших рядків та стовпців в ній, а також взявши N стовпців з матриці U та N рядків з матриці V^T можна з певною похибкою апроксимувати початкову матрицю.

Дуже важливо знайти правильне співвідношення між тим, яку частину даних залишити, а від якої позбутися, адже необхідно одночасно як зберігати достатню точність, так і мати оптимальну розмірність для розрахунків та зберігання.

Оскільки матриця оцінок може містити порожні значення, то варто заповнити їх значеннями, адже на вхід алгоритму необхідно подати заповнену матрицю чисел. В такому випадку варто використати евристики, наприклад, заповнити комірки без чисел середнім значенням оцінки по користувачу/елементу каталогу. Або у випадку нормалізації даних можно заповнити такі комірки нулями, що і буде позначати усереднену величину.

Для того, аби передбачити оцінку, насправді, необхідно виконати прості кроки: зафіксувати відповідний користувачу рядок матриці U, відповідний елементу каталогу стовпець матриці V^T, та виконати скалярне множення векторів із сингулярною матрицею між ними. Звичайно, можна скоротити дані операції при роботі застосунку, попередньо обчисливши усю апроксимовану матрицю, із якої миттєво забирати значення будь-якої комірки, але тоді збільшиться обсяг пам’яті, що використовується. У будь-якому разі, не варто видаляти отримані матриці розкладу, задля застосування інкрементального *SVD* (*Incremented SVD*) при додаванні нових користувачів, елементів каталогу та оцінок, що дозволить уникнути повторного запуску факторизації на всьому масиві даних, натомість виконати лише невелику частину обчислень.

Декомпозиція матриці оцінок дозволяє попередньо навчити модель на вибірці, скоротити кількість дій при власне передбаченні, а також знайти приховані зв’язки у даних, які неможливо помітити, маючи лише початкову матрицю. Головною вимогою до застосування цього методу є висока щільність даних -

якомога менше пропусків у початковій таблиці, інакше, модель навчиться передбачувати всього-на-всього середні показники.

1.4.4. Кластеризація об’єктів

В задачах рекомендаційних систем часто застосовується кластеризація (*Clustering*), або сегментація, - розподіл екземплярів сутностей по групам, що мають схожі властивості. Це одна із задач машинного навчання без вчителя (*Unsupervised machine learning*), на вхід якої подається множина об’єктів та векторів значень їхніх ознак та число *K* - бажана кількість кластерів [6].

Найбільш вживаних є алгоритм *K* середніх (*K-means*). Спочатку випадковим чином обирається *K* векторів - початкові центри кластерів. Для кожного об’єкта обирається той з них, який знаходиться ближче (можна, наприклад, застосувати евклідову метрику). Після чого, кожен із початкових центрів рухається у нову точку - центр екземплярів його сегменту. Така ітерація повторюється певну задану кількість разів, або доки центри кластерів не стабілізуються. Для нових векторів процедура кластеризації є тривіально - треба лише знайти найближчий центр та поставити у відповідність даний кластер.

Даний підхід вдало застосовується із вищезгаданими алгоритмами для покращення ефективності обчислень. Слабким місцем колаборативної фільтрації є розрахунок подібності користувачів або елементів каталогу для знаходження сусідства. Вплив цього недоліку можна зменшити завдяки попередньому введенню кластерів: це дасть змогу проходитися не по всім об’єктам, а лише по данному кластеру, що значно пришвидшить роботу для великих обсягів даних. Іноді кластер використовують власне як сусідство, щоправда в такому випадку страждає якість вибірки - вона може бути занадто великою. Фільтрація на основі контенту напряму може використати переваги сегментації, адже кластери товарів будуються на основі векторів значень їхніх ознак. Тобто, якщо знайти, до якого кластеру належить профіль користувача, то простір перебору контенту значно звузиться до обраного сегменту.

Досить важливим є питання вибору *K*. Якщо дана величина буде занадто малою, то приріст у ефективності буде не таким значним, бо все ще велика кількість об’єктів залишиться для перебору в межах кластеру. Навпаки, у випадку завеликого числа, сегменти стануть досить специфічними, вони будуть об’єднувати лише дуже близькі об’єкти, що погано вплине на кількісну ознаку та різноманітність рекомендацій. Для оцінки правильності вибору *K* можна використати силуетний коефіцієнт (*Silhouette coefficient* – формула 1.8). Величина *a(i)* позначає середню відстань об’єкта до інших поточного кластеру - його густина, в той час як *b(i)* - мінімальну середню відстань до об’єктів одного з інших кластерів, тобто, віддаленість від інших сегментів. Фінальне порівняння у *S(i)* показує, як співвідносяться ці дві величини (область значень від -1 до 1): чим вона більша, тим краще обрано кластери в контексті даного об’єкта, тобто сегмент більш густий та знаходиться далі від інших. Для оцінки кластеризації загалом можно знайти середнє значення коефіцієнту по усім екзмеплярам.

$$S(i) = \frac{b(i) - a(i)}{max(a(i), b(i))}$$

$$a(i) = \frac{1}{|A| - 1} \sum_{j \in A, j \neq i} d(i, j)$$

$$b(i) = min_{C \neq A}(\frac{1}{|C|} \sum_{j \in C} d(i, j))$$

$$A - \text{поточний кластер}, d - \text{метрика}$$

Формула 1.8. Силуетний коефіцієнт

Отже, кластеризація є потужним інструментом для застосування із іншими підходами до розв’язання задач рекомендаційних систем завдяки покращенню ефективності виконання. Єдиною особливістю, на яку варто звернути увагу, є якість навчання самих кластерів, адже якщо вони побудовані невдачно, то це значно погіршить результати інших алгоритмів, незважаючи на їхню точність.

1.4.5. Гібридні системи

Як можна побачити, всі перераховані алгоритми мають свої переваги та недоліки. Аби використати найсильніші сторони кожного, проектуються гібридні

рекомендаційні системи (*Hybrid recommender systems*), або моделі-ансамблі. Їх існують декілька типів, деякими з яких є: каскадна, змішана, зважена та на основі перемикача [7].

Каскадна (*Cascade*) архітектура інкапсулює в собі декілька рекомендаційних систем послідовно, тобто вихід однієї надходить у вхід до іншої. Даний процес нагадує фільтрацію на кожному етапі, що дозволяє підвищити точність за допомогою оцінки різних факторів. Наприклад, фільтрація на основі контенту надасть схожі на профіль користувача об'єкти, в той час як колаборативна фільтрація виставить їм оцінки, базуючись на відповідних комірках матриці оцінок.

Наступною архітектурою є змішана (*Mixed*) рекомендаційна система. Її принцип трохи відрізняється тим, що рекомендаційні системи працюють незалежно, і фінальна запропонована добірка контенту складається із виходів кожної з них. Таким чином, можна першими показувати кілька результатів через фільтрацію на основі контенту (те що точно має сподобатися клієнту), потім трохи більше із колаборативної фільтрації (де буде більше товарів із нових категорій для розширення кругозору), а в кінці за необхідності доповнити найпопулярнішими серед усіх користувачів елементами каталогу (аби досягнути попрогу мінімальної кількості рекомендацій - це, до речі, допоможе вирішити проблему холодного старту).

Іншим типом є системи, які переключаються (*Switched*), які показують у результаті запропоновані об'єкти лише обраних рекомендаційних систем, залежно від заданих факторів. Наприклад, якщо немає оцінок користувача, то показувати популярний контент, якщо оцінок мало, то застосовувати рекомендації фільтрації на основі контенту, нарешті, якщо зворотнього зв'язку достатньо для уникнення проблеми холодного старту, то показати результати колаборативної фільтрацію.

Більш комплексними є зважені (*Weighted*) ансамблі, які на відміну від тих, що переключають, поєднують результати за допомогою ваг. Очевидним прикладом є поєднання декількох алгоритмів рекомендацій, які повертають прогнозовані оцінки для набору об'єктів, які потім поєднуються у вигляді лінійної або нелінійної комбінації. У фінальній формулі можуть брати участь не лише власне оцінки, а і

ознаки користувачів/елементів каталогу, які при поєднанні разом дозволять налаштувати роботу ансамбля під конкретні випадки (наприклад, залежно від кількості оцінок товару - надати перевагу одному із алгоритмів). Головною задачею є знаходження коефіцієнтів для оцінок кожної із компонент, адже метод підбору навряд дасть точний результат. Тому можна застосувати алгоритм лінійної регресії, аби навчити модель поєднувати результати кожної з рекомендаційних систем разом та повертати нову оцінку. Простором для експериментів залишається вибір ознак, під які будуть підлаштовуватися коефіцієнти.

Лінійна регресія дозволить оптимізувати функцію втрат J через їхні ваги θ - параметри, які і формують лінійну комбінацію результату h (формула 1.9). Лінійна регресія застосовує квадратичну метрику помилок, оскільки вона легко диференціюється, що використовується у власне оптимізації, наприклад, методом градієнтного спуску.

$$h(X,\theta) = \theta X^T$$

$$J(\theta) = \frac{1}{2n}\sum_{i=1}^{n}(h(X^{(i)},\theta) - Y^{(i)})^2$$

$$\theta_j = \theta_j - \alpha\frac{\partial J(\theta)}{\partial \theta_j} - \text{ крок градієнтного спуску}$$

$$n - \text{ розмір вибірки}$$

$$\alpha - \text{ коефіцієнт швидкості навчання}$$

$$X^{(i)}, Y^{(i)} - \text{ пара вхідного вектора та відповідного значення}$$

$$\theta - \text{ вектор параметрів}$$

$$h - \text{ функція передбачення значення для вектора за допомогою параметрів}$$

$$J - \text{ функція втрат для всього набору даних із заданими параметрами}$$

Формула 1.9. Лінійна регресія

Можна зазначити, що саме гібридні рекомендаційні системи зазвичай застосовуються у найбільш популярних ресурсах, адже вони дозволяють нівелювати недоліки кожного із методів до розробки рекомендаційних систем та поєднати найкращі риси. Звичайно, разом із цим росте і складність архітектури та час розробки/навчання.

2. Створення гібридної рекомендаційної системи

В межах практичної частини даного дослідження було виконано побудову гібридної рекомендаційної системи, що поєднує у собі декілька алгоритмів.

2.1. Опис датасету

Для виконання практичної частини було обрано датасет *MovieLens* (https://grouplens.org/datasets/movielens/100k/), що містить у собі дані про користувачів (1,000) та фільми (1,700), а також їхні оцінки (100,000). Для кожного користувача міститься як мінімум 20 оцінок.

В таблиці користувачів наявна інформація про вік, стать, професію та поштовий індекс. Фільми містить у собі дані про назву, дату виходу, посилання та жанри. Оцінками є числа за шкалою від 1 до 5 разом із часовою відміткою, коли вона була виставлена.

Автори датасету розподілили дані на навчальну та тестову вибірку за співвідношенням 80:20. Таким чином, навчання алгоритмів буде виконуватся на 80% даних, а тестування та перевірка точності - на 20%.

2.2. Опис засобів реалізації

Для розробки системи була використана інтерпретована мова *Python*. Вибір був зроблений завдяки великій кількості бібліотек для роботи із машинним навчанням та масивами даних, що значно прискорило створення програми.

Основним елементом у кожному з алгоритмів є робота із векторами та математичними функціями, тому критичною є ефективність виконання обчислень. Для цього було використано бібліотеку *NumPy*, реалізація якої написана на мові *C*, що дає значне прискорення виконання, в той час як інтерфейс доступний для *Python*.

2.3. Архітектура системи

Загалом, архітектура побудованої рекомендаційної системи (рис. 2.1) має такий вигляд. Три алгоритми тренуються незалежно на спільній навчальній вибірці: *SVD*, колаборативна фільтрація разом із кластеризацією та фільтрація на основі контенту. Краща модель для кожного з них обирається на основі метрик помилок (*RMSE, MSE, MAE*), після чого фіналізуються три рекомендаційних системи, які здатні передбачати оцінку користувача для фільму, який ще не був ним переглянутий. Далі необхідно побудувати зважений ансамбль для їх поєднання - створюється нова модель на основі лінійної регресії, яка навчається на тих самих даних, що і її дочірні. Остаточна версія оцінюється на тестовій вибірці, а протягом використання бере оцінки з кожного з трьох алгоритмів та поєднує їх за допомогою ваг.

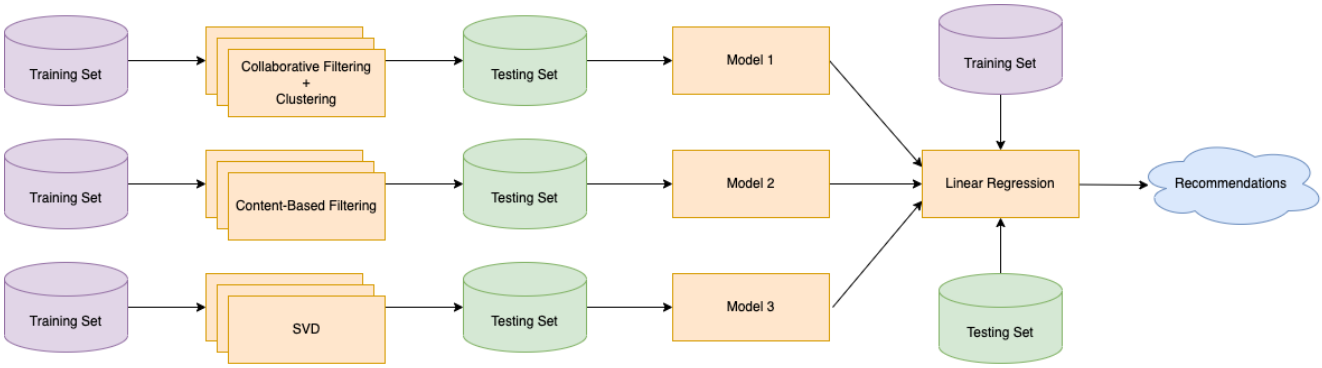

Рис. 2.1. Архітектура рекомендаційної системи

Для оцінки кожної з моделей було виконано побудову повноцінної матриці оцінок без пропусків, тобто усі передбачення були прораховані завчасно. Це зроблено через дві причини: пришвидшення тестування (залишається лише застосувати метрику помилок до очікуваного та фактичного значень) та полегшення навчання ансамблю (немає необхідності повторно застосовувати обчислення кожної з компонент, їх можна використовувати як вже готові вхідні дані). Звичайно, в такому разі витрачається більше пам’яті для зберігання, тому в реальному застосунку треба обирати між завчасними та фактичними розрахунками, беручи до уваги необхідні вимоги.

З метою порівняння алгоритмів та отриманих моделей використано оффлайн-тестування за допомогою метрик *RMSE*, *MSE*, *MAE* - акцент зроблено на мінімізацію помилки передбачень. Такі критерії як різноманіття, покриття та випадковість, а також онлайн-тестування не розглядалися протягом виконання практичної частини даної роботи.

Наступні підрозділи розглядають процес розробки кожного з кроків.

2.3.1. Компонент 1. Вибір розмірності розкладу матриці

Першою із складових системи є факторизації матриці оцінок за допомогою алгоритма *SVD*. У якості його реалізації була використана функція із згаданої бібліотеки *NumPy*. Спочатку побудувалась матриця оцінок із навчальної частини датасету, після чого було виконано нормалізацію (*Mean normalization*) для кожного рядка окремо. Це дало змогу заповнити пропуски нулями, що на нормалізованих числах позначає середню величину. Крім того, такий вибір замість глобальної нормалізації був зроблений задля того, аби для кожного користувача від'ємні та додатні числа позначали відповідну оцінку нижче та вище середнього, адже різні користувачі користуються різними діапазонами відгуків: для когось оцінка 3 це добре, для іншого - погано.

Для вибору оптимальної розмірності розкладу матриці, було проведено кілька експериментів для максимальної та зменшених розмірах, в результаті чого обрано число $N = 10$, що характеризується найменшою помилкою передбачень на тестовій вибірці на основі різних метрик (рис. 2.2).

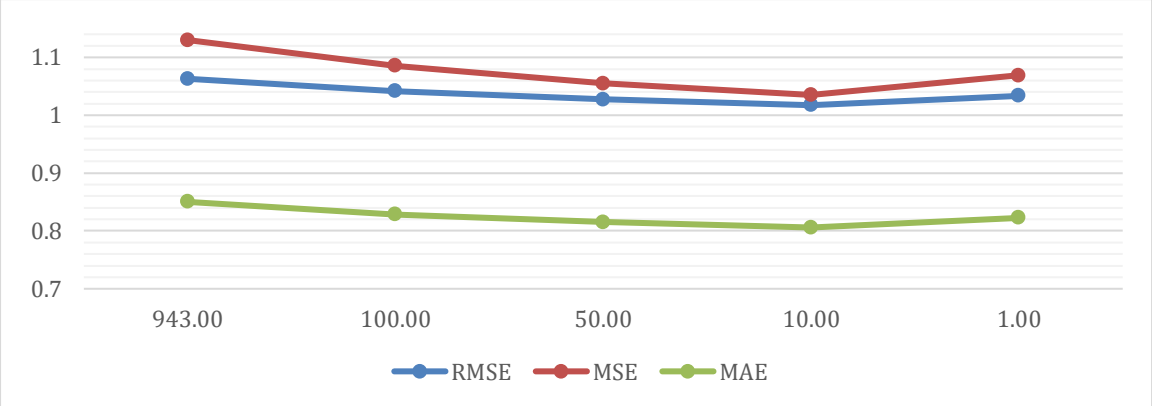

Рис. 2.2. Порівняння розмірностей розкладу матриці

2.3.2. Компонент 2. Вибір сусідства для колаборативної фільтрації

Наступною частиною системи є алгоритм колаборативної фільтрації на основі користувачів. Саме цей підтип було обрано, оскільки нормалізація вже провелася на основі користувачів, то краще порівнювати патерни поведінки саме за ними. Варто попередньо розрахувати подібність між усіма парами користувачів, адже саме цей крок займає більшу частину часу навчання (до речі, кількість користувачів менша за кількість фільмів у даній вибірці, тому це ефективніше за колаборативну фільтрацію на основі елементів каталогу).

Провівши кілька експериментів, було вирішено застосувати кластеризацію (засобами бібліотеки *SkLearn*) в парі із цим підходом, адже розрахунок схожості та генерація сусідства займала надто багато часу (із впровадженням кластерів час виконання поліпшився на 40% та 55% відповідно). Оскільки для пошуку сусідів використовується подібність виставлення оцінок, то і кластери формувалися для векторів оцінок користувачів. Для вибору кількості кластерів було обрано силуетний коефіцієнт із додатковою умовою, що в сегменті має бути не менш ніж 2 об’єкти (адже при багатьох запусках деякі групи складалися всього з одного елемента). Незважаючи на те, що найбільш оптимальним з точки зору даної метрики було кількість кластерів 2, було обрано 3, адже саме в такому випадку досягалося визначене прискорення, не надто жертвуючи точністю (рис. 2.3).

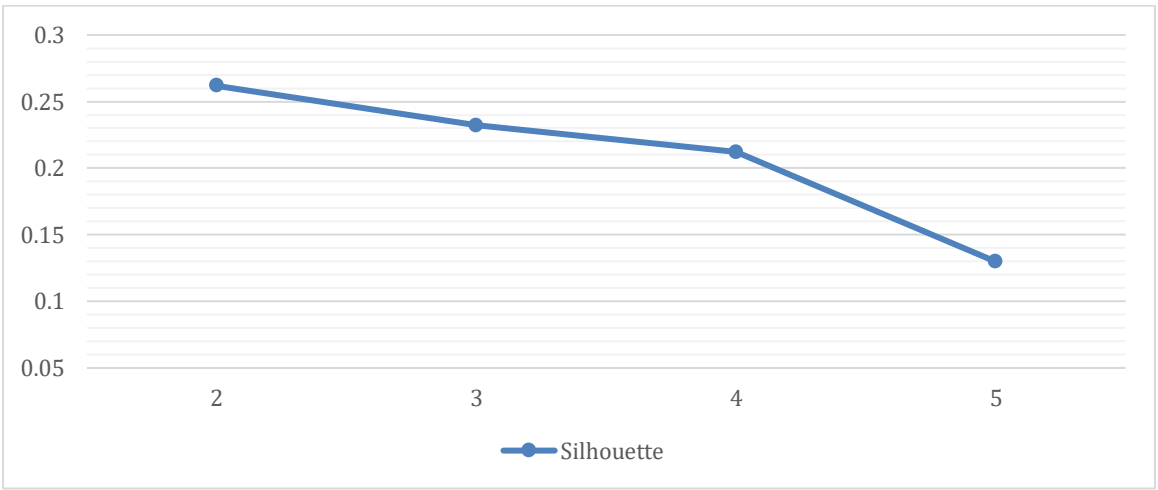

Рис. 2.3. Порівняння різної кількості кластерів

Далі ці кластери використовувалися як кандидати у сусідство. Важливо додати, що не всі інші користувачі могли претендувати на роль сусіда, а лише ті, які вже оцінили цільовий фільм, аби не використовувати середню оцінку іншого користувача, адже це лише погіршить точність - перевірено декількома запусками алгоритму.

Для вибору сусідства в межах даного підходу було обрано використання порогу, адже матриця достатньо розріджена, тому навіть при невеликих кількостях сусідів N, для деяких елементів множина наповнювалась такими об'єктами, кореляція з якими прямувала до мінус одиниці.

Тому проведено декілька експериментів із різними значеннями порогу, аби обрати найкращий поріг 0.1 (рис. 2.4).

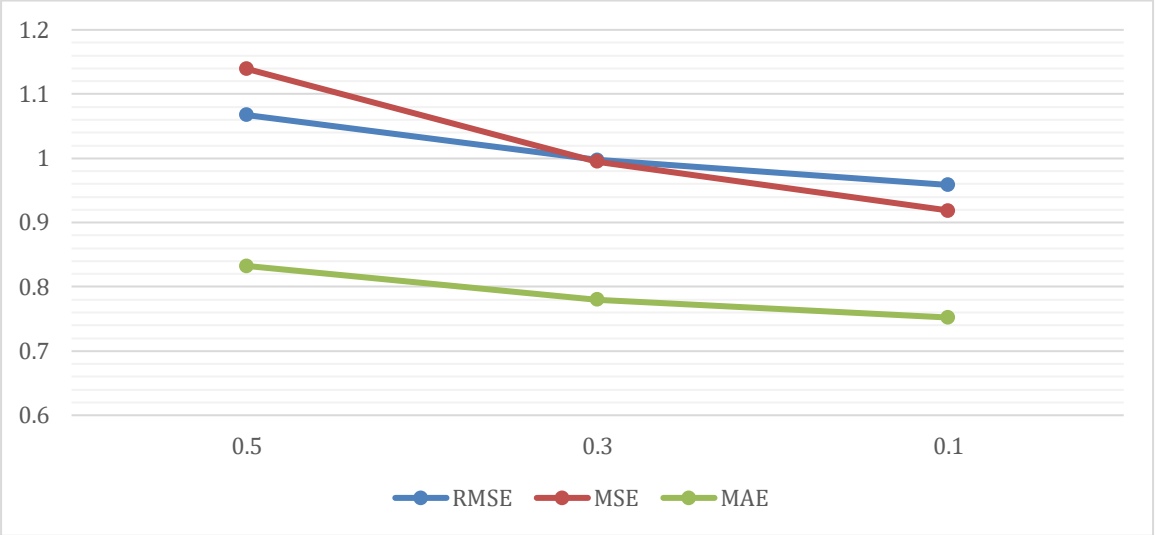

Рис. 2.4. Порівняння порогового значення для відбору сусідства

2.3.3. Компонент 3. Вибір метрики у фільтрації на основі контенту

Останнім алгоритмом було застосовано фільтрацію на основі контенту. Із даних про фільми було використано рік випуску, а також бінарні ознаки жанрів. Для кожного користувача попередньо розраховувався профіль як середнє зважене оцінених фільмів, де в якості ваг виступали оцінки.

У класичному алгоритмі необхідно для запиту на користувача віддати список найбільш схожих фільмів на його профіль, які він ще не бачив. Тобто, немає загального правила як передбачити саме числову оцінку. Очевидно, що загальний

підхід полягає в тому, що чим ближчий фільм до профіля користувача, тим більша оцінка. Тому було вирішено рівномірно поставити фільмам у відповідність оцінку, пропорційну їхній подібності із профілем користувача, за шкалою від мінімальної до максимальної оцінки, які вже наявні у користувача.

З метою експеримента було використано дві метрики: косинусна схожість та евклідова відстань. Друга з них виявилася більш точною, хоча результат все ще значно гірший за попередні алгоритми (рис. 2.5).

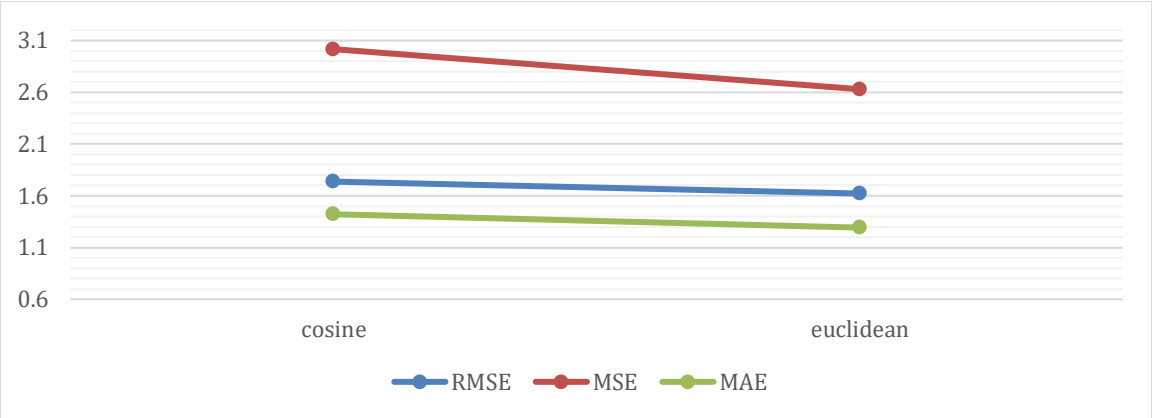

Рис. 2.5. Порівняння метрик для фільтрації на основі контенту

2.3.4. Поєднання алгоритмів

Після готовності трьох компонентів, необхідно об’єднати їх в одну модель. Для цього обрано застосування зваженого ансамблю на основі лінійної регресії. Як було сказано в описі системи, для кожного з алгоритмів будується повна матриця передбачень, а отже, її елементи можна використовувати як результат роботи кожного з алгоритмів - прогнозовану оцінку, які можна поєднати між собою. Таким чином, для регресії на вхід подається множина із елементів, що мають три характеристики (оцінки кожного із алгоритмів), яким відповідає оцінка з навчальної вибірки. Модель має апроксимувати їхні ваги, після чого використовувати їх для передбачення нових пар взаємодії.

У якості реалізації було використану бібліотеку *Sklearn*, що дозволяє швидко налаштувати навчання моделі із заданою навчальною вибіркою та параметрами. На практиці перевірено, що для даного датасету кращий результат дає створення поліноміальних комбінацій ознак степеня 2, проте додавання нових ознак, як-от

дані про користувача та фільма, майже не впливають на точність. Було проведено експерименти для визначення параметра регуляризації, аби не допустити перенавчання. Найкращий результат показала величина 0.1 (рис. 2.6).

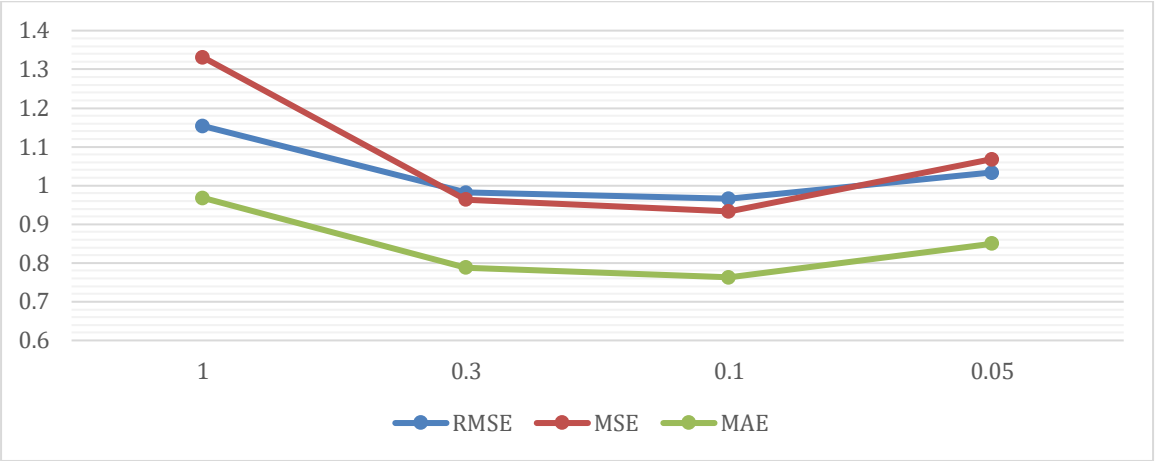

Рис. 2.6. Порівняння параметрів регуляризації для лінійної регресії

2.3.5. Порівняння результатів

Для порівняння точності отриманих алгоритмів було побудовано наївну рекомендаційну систему - таку що в кожній клітинці матриці оцінок стоїть середня оцінка конкретного користувача - для базових передбачень (*Baseline predictor*), відносно якої будуть порівнюватися отримані моделі (рис. 2.7).

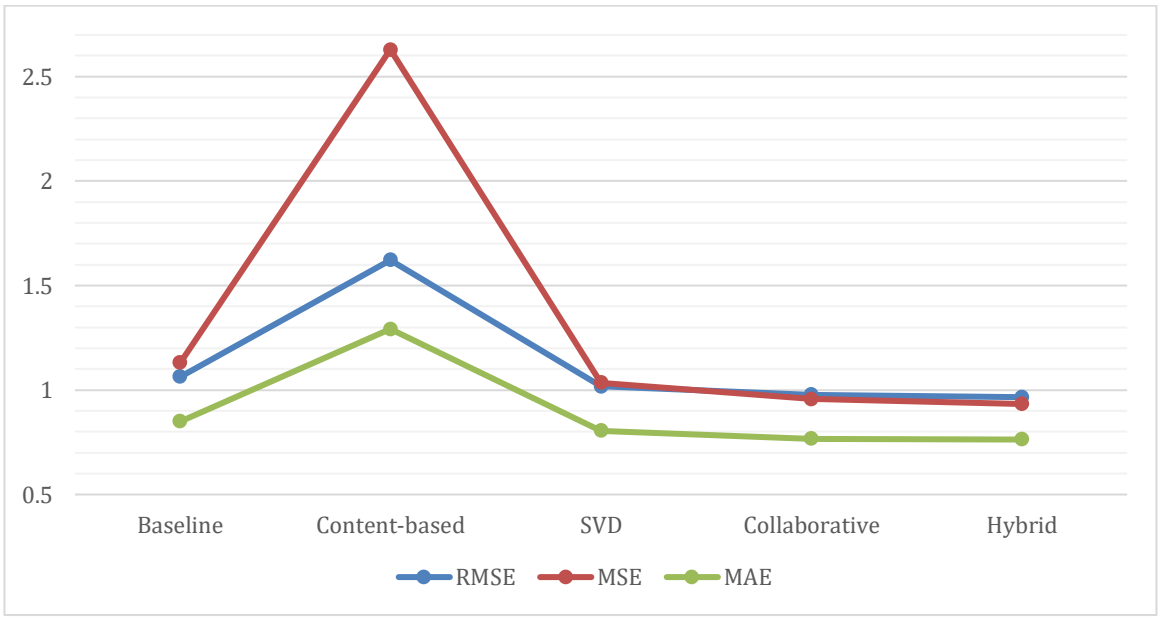

Рис. 2.7. Порівняння отриманих алгоритмів

Отже, отримані системи (крім фільтрації на основі контенту) покращують базовий варіант. Фільтрація на основі контенту показала себе невдало на обраних даних – можливо, через нестачу інформації про самі фільми. Колаборативна фільтрація зайняла більше всього часу на навчання серед усіх алгоритмів (навіть враховуючи кластеризацію), проте характеризується найкращою точністю, не враховуючи гібридну систему. Розклад матриці трохи поступається колаборативній фільтрації за метриками помилок, але займає у декілька десятків разів менше часу на навчання, що буде суттєво на даних більшого обсягу. Наостанок, гібридна система показала найточніший результат серед усіх алгоритмів (хоч і не з великою різницею), а отже, можна вважати, що за допомогою регресії було знайдено приховані залежності та поєднано недоліки та переваги кожного із алгоритмів.

Висновки

В результаті виконання роботи було досягнуто поставленої мети.

Проаналізовано фази виконання рекомендаційних систем: збір інформації, навчання та передбачення. Розглянуто критерії їх оцінки: мінімізація помилки передбачень, різноманітність, покриття та випадковість. А також перераховано способи впровадження нових версій та метрики для порівняння їхньої точності.

Порівняно різні методи машинного навчання для розробки рекомендаційних систем: фільтрація на основі контенту, колаборативна фільтрація, декомпозиція матриці оцінок, кластеризація; розглянуто їхні недоліки, переваги та вимоги до застосування. Наведено декілька способів утворення гібридних систем для поєднання алгоритмів та підвищення якості загалом.

Розроблено гібридну рекомендаційну систему - зважений ансамбль із перерахованих алгоритмів, який було утворено шляхом навчання моделі лінійної регресії. Порівняно точність кожної із компонент системи із системою загалом, використовуючи метрики *RMSE, MSE, MAE*.

Проект можна потенційно покращити шляхом:

- збільшення обсягу даних для навчання та тестування
- наближення даних до реальних умов (наприклад, із наявною проблемою холодного старту) та застосуванням додаткових підходів до створення гібридних систем
- тестування за допомогою інших метрик та критеріїв
- застосування глибокого навчання та нейронних мереж
- розгортання системи у хмарному середовищі для кожної із фаз роботи
- модифікації алгоритмів для виконання обчислень паралельно або на розподілених машинах, уникаючи побудови усієї матриці оцінок в оперативній пам’яті - що несе величезні витрати, а іноді унеможливлює роботу з даними, обсягом на порядок більшим обраного датасету

Джерела

[1] Isinkaye F.O. Recommendation systems: Principles, methods and evaluation / F.O. Isinkaye, Y.O. Folajimi, B.A. Ojokoh. - 2015.

[2] Falk K. Practical Recommender Systems / Kim Falk. - Shelter Island: Manning Publications Co., 2019. - 432c.

[3] Perez-Almaguer Y. Content-based group recommender systems: A general taxonomy and further improvements / Y. Perez-Almaguer, R. Yera, A. A. Alzahrani, L. Martinez. - 2021.

[4] Ayyaz S. Improving Collaborative Filtering by Selecting an Effective User Neighborhood for Recommender Systems / S. Ayyaz, U. Wamar. - 2017.

[5] Sarwar B. Incremental singular value decomposition algorithms for highly scalable recommender systems / B. Sarwar, G. Karypis, J. Riedl. - 2002.

[6] Widiyaningtyas T. Recommendation Algorithm Using Clustering-Based UPCSim (CB-UPCSim) / T. Widiyaningtyas, I. Hidayah, T.B. Adji. - 2021.

[7] Cano E. Hybrid Recommender Systems: A Systematic Literature Review / E. Cano. - 2017