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

Інститут прикладного системного аналізу

Кафедра математичних методів системного аналізу

До захисту допущено:

Завідувач кафедри

_____ О.Л. Тимощук

«___»_____20__ р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системний аналіз і управління

спеціальності 124 "Системний аналіз"

на тему: «Аналіз та порівняння рекомендаційних систем колаборативної фільтрації»

Виконав:
студент IV курсу, групи КА-64
Станков Іван Сергійович _____

Керівник:
к.ф-м.н.,
Статкевич Віталій Михайлович _____

Консультант з нормоконтролю:
доцент, к.т.н., доцент кафедри ММСА
Коваленко Анатолій Єпіфанович _____

Рецензент:
математик-аналітик з дослідження операцій, к.ф-м.н.
Потапенко Олексій Юрійович _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент: Станков Іван Сергійович

Київ – 2020 року

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

Рівень вищої освіти – перший (бакалаврський)
Спеціальність – 124 «Системний аналіз»
Освітньо-професійна програма «Системний аналіз і управління»

ЗАТВЕРДЖУЮ
В.о.завідувача кафедри
_____ Оксана ТИМОЩУК
«___»_____20__ р.

ЗАВДАННЯ
на дипломну роботу студенту
Станкову Івану Сергійовичу

1. Тема роботи «Аналіз та порівняння рекомендаційних систем колаборативної фільтрації», керівник роботи Статкевич Віталій Михайлович, к.ф-м.н., затверджені наказом по університету від «_25_» _травня_ 20_20_ р. №_1143-с__
2. Термін подання студентом роботи 08 травня 2020 року
3. Вихідні дані до роботи

   1. Операційна система Windows 10

   2. Частота процессора 2.3 ГГц

   3. Мова програмування Python

   4.Середовище розробки – PyCharm Community Edition 2018

   5. Бібліотеки, що використовувалися: NumPy, SciPy, Pandas, Sklearn, Surprise.

4. Зміст роботи

   1. Проаналізувати типи рекомендаційних алгоритмів.

   2. Проаналізувати основні види рекомендаційних систем колаборативної фільтрації.

   3. Розробити набір ключових рекомендаційних систем колаборативної фільтрації.

   4. Розробити пакет для тестування рекомендаційних систем.

   5. Виконати економічний аналіз програмного продукту.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо)

   1. Презентація

6. Консультанти розділів роботи[1]*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	к.е.н., доц. Шевчук О. А.	21.04.20	18.05.20

7. Дата видачі завдання

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання	13.04.20	
2	Вивчення існуючих алгоритмів	17.04.20	
3	Підбір метрик для оцінювання рекомендаційних систем.	24.04.20	
4	Розробка алгоритму введення даних, побудови зображень та виведення інформації	29.04.20	

[1]* Якщо визначені консультанти. Консультантом не може бути зазначено керівника дипломної роботи.

5	Розробка системи тестування, побудова порівнюваних алгоритмів	3.05.20	
6	Тестування додатку та отримання результатів	10.05.20	
7	Оформлення дипломної роботи	17.05.20	

Студент Іван Сергійович СТАНКОВ

Керівник Віталій Михайлович СТАТКЕВИЧ

РЕФЕРАТ

Дипломна робота: 100 с., 24 рис., 14 табл., 2 дод., 8 джерел.

РЕКОМЕНДАЦІЙНІ СИСТЕМИ, КОЛАБОРАТИВНА ФІЛЬТРАЦІЯ, ТОЧНІСТЬ, ПОВНОТА.

Предмет дослідження — пакет оцінок необхідних для вивчення такого класу алгоритмів, як рекомендаційні системи та система їх тестування.

Об’єкт дослідження — різні види рекомендаційних систем колаборативної фільтрації.

Мета роботи — побудова вичерпного набору ключових алгоритмів колаборативної фільтрації та їх оцінок, задля отримання чіткого розуміння цього класу алгоритмів.

Методи дослідження — побудова 5 різних алгоритмів колаборативної фільтрації та тестів, для їх аналізу та порівняння.

Побудовано набір унікальних тестів роботи рекомендаційних систем в залежності від кількості даних, смаків користувача та інших параметрів.

Проведено детальний огляд конкретних алгоритмів та їх результативності в залежності від різних умов.

Користуючись розробленим програмним забезпеченням, було порівняно ключові види систем колаборативної фільтрації.

Використання результатів цієї роботи дозволяє покращити роботу вже існуючих систем та полегшити створення нових.

ABSTRACT

Thesis work: 100 p., 24 fig., 14 tables, 2 app., 8 sources.

RECOMMENDATION SYSTEMS, COLLABORATIVE FILTRATION, ACCURACY, COMPLETENESS.

The subject of research is a package of evaluation metrics for recommendation systems and their testing system.

The object of research is different types of recommended systems of collaborative filtration.

The purpose of the work is to build a comprehensive set of key algorithms for collaborative filtering and their estimates, in order to obtain a clear understanding of this class of algorithms.

Research methods — construction of 5 different algorithms of collaborative filtering and tests, for their analysis and comparison.

A set of unique tests of recommendation systems depending on the amount of data, user tastes and other parameters had been built.

A detailed review of specific algorithms and their effectiveness depending on different conditions was proposed.

Using the developed software, the key types of collaborative filtering systems were compared.

Using the results of this work improved performance of existing systems and better development of new ones can be achieved.

ЗМІСТ

с.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ ... 9
ВСТУП ... 10

1 ЗАГАЛЬНІ ВІДОМОСТІ ПРО РЕКОМЕНДАЦІЙНІ СИСТЕМИ **12**
1.1 Види рекомендаційних систем ... 13
1.1.1 Фільтрація вмісту ... 13
1.1.2 Колаборативна фільтрація ... 14
1.2 Функції подібності у рекомендаційних системах ... 16
1.2.1 Кореляція Пірсона ... 16
1.2.2 Середня квадратична відстань ... 17
1.2.3 Косинус подібності ... 17
1.3 Висновки до розділу 1 ... 18

2 МЕТРИКИ ДЛЯ ОЦІНЮВАННЯ ЯКОСТІ РЕКОМЕНДАЦІЙНИХ СИСТЕМ **19**
2.1 Оцінки часу ... 19
2.2 Оцінки точності ... 21
2.2.1 Оцінка MAE ... 21
2.2.2 Оцінка RMSE ... 22
2.2.3 Оцінка RMSLE ... 23
2.3 Точність та повнота ... 25
2.3.1 Precision@k ... 27
2.3.2 Recall@k ... 27
2.3.3 F1 ... 27
2.3.4 Криві точності та повноти ... 28
2.4 Інші оцінки ... 32
2.4.1 Покриття ... 32
2.4.2 Персоналізація ... 32
2.4.3 Подібність рекомендованих об'єктів ... 33
2.5 Висновки до розділу 2 ... 33

3 ВИБІР ДАНИХ **34**
3.1 Аналіз обраного набору даних ... 34

3.2 Підходи до побудови тестувальної та тренувальної вибірок 35
3.3 Висновки до розділу 3 37

4 АНАЛІЗ ТА ПОРІВНЯННЯ АЛГОРИТМІВ 38
4.1 Огляд окремих рекомендаційних систем 38
4.1.1 Випадковий фільтр 38
4.1.2 Простий user-based фільтр 41
4.1.3 Простий item-based фільтр 44
4.1.4 SVD фільтр 46
4.1.5 Slope One алгоритм 49
4.2 Порівняння алгоритмів 50
4.2.1 Витрати часу 51
4.2.2 Оцінки точності 54
4.2.3 Точність та повнота 60
4.3 Висновки до розділу 4 62

5 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПАКЕТУ ДЛЯ АНАЛІЗУ ТА ПОРІВНЯННЯ РЕКОМЕНДАЦІЙНИХ СИСТЕМ 64
5.1 Постановка задачі 64
5.2 Обґрунтування функцій та параметрів ПП 64
5.3 Обгрунтування системи параметрів досліджень 66
5.4 Аналіз рівня якості варіантів реалізації функцій 69
5.5 Економічний аналіз варіантів розробки ПП 72
5.6 Висновки до розділу 5 75
ВИСНОВКИ 76
ПЕРЕЛІК ПОСИЛАНЬ 77
Додаток А Програмний модуль розроблених алгоритмів 78
Додаток Б Презентація 92

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

U – множина користувачів

u_i – позначення окремого користувача

I – множина об'єктів, які рекомендуються

i_i – позначення окремого об'єкта

V – матриця рейтингів розмірності $n \times m$, де $n = |U|$ – кількість користувачів та $m = |I|$ – кількість об'єктів. У деяких випадках також позначатимемо за V загальну множину рейтингів

v_{ui} – рейтинг, який користувач $u \in U$ надав об'єкту $i \in I$

p_{ui} – оцінений рейтинг, який користувач $u \in U$ надасть об'єкту $i \in I$

$V_u = \{v_{ui} \in V / i \in I_u\}$ – підмножина рейтингів які надав користувач $u \in U$, де I_u – множина пов'язаних з цим користувачем об'єктів.

$V_i = \{v_{ui} \in V / u \in U_i\}$ – підмножина рейтингів які надали об'єкту $i \in I$ користувачі $u \in U$, де U_i -множина пов'язаних з цим об'єктом користувачів.

V_{train} – навчальна підмножина рейтингів

V_{test} – тестова підмножина рейтингів

θ – порогова оцінка

MAE – середня абсолютна помилка (англ. Mean Absolute Error)

RMSE – усереднена квадратична помилка (англ. Root Mean Absolute Error)

RMSLE – усереднена квадратична логарифмована помилка (англ. Root Mean Squared Log Error)

ROC-крива – (англ. Receiver Operating Characteristic curve) аналог кривої точності/повноти, розглянутої у цій роботі.

SVD – сингулярний розклад матриці (англ. singular-value decomposition)

ПП – програмний продукт

ВСТУП

Основними явищами інформаційної епохи є постійне збільшення інформації та знань, з якими доводиться працювати людям, а також вдосконалення систем, забезпечуючих доступ та фільтрацію інформації. Розвиток інтернету та побудованих на ньому медіа-систем також привів до значних змін у повсякденному житті кожної людини. Постійне збільшення обсягів інформації, а також розвиток комерційної складової інтернету призвело до стрімкого збільшення попиту на різновидні алгоритми сортування, пошуку та фільтрації.

Розвиток соціальної складової інтернету також викликав безпосередній розвиток соціології та суспільної психології, що в свою чергу призвело до створення цілої течії прикладних теорій щодо рекомендації даних, алгоритмів пошуку та великої кількості інших інструментів, які мають безпосередній вплив на соціальне сприйняття та комерційних успіх сервісів, які їх використовують.

Одним з найважливіших класів алгоритмів, призначених для фільтрації даних, є рекомендаційні системи, основною задачею яких є побудова рекомендацій таким чином, що користувач відчуває потребу у рекомендованих продуктах чи сервісах, які їх рекомендують. Вже зазначений розвиток інтернет торгівлі привів до одночасних зміни формату торгівлі, а також до значного збільшення кількості різних товарів, які окремі онлайн-магазини здатні продавати. Таким чином, розвиток сучасних інформаційних технологій дозволяє вести роздрібний бізнес з мільйонами потенційних користувачів, при цьому пропонуючи мільйони унікальних товарів. З однієї сторони, це надало надзвичайну можливість для розвитку сегменту послуг загалом. З іншої, викликало потребу до обробки надзвичайно великих обсягів даних, з метою не тільки покращення фінансових показників, а й взагалі, з метою підтримки постійно зростаючих платформ.

Таким великим підприємствам на допомогу й прийшли рекомендаційні системи, основний розвиток теоретичного підґрунтя яких

прийшовся на середину 90-х років. Незважаючи на те, що рекомендаційні системи існували на папері кілька десятиріч, їх практична імплементація була обмежена рівнем обчислювальної техніки, тому активний розвиток рекомендаційних систем продовжується й у наші дні.

Метою цієї роботи є вивчення основних видів рекомендаційних систем, аналіз та порівняння найбільш значимих рекомендаційних систем колаборативної фільтрації та створення повноцінного пакету оцінки та тестування рекомендаційних систем. Побудова такого пакету дозволяє провести ретельне дослідження поведінки окремих рекомендаційних систем при різних параметрах запуску та інших налаштуваннях. Для створення такого пакету була вибрана мова Python, адже вона є загальноприйнятою мовою програмування серед спеціалістів, працюючих з подібними алгоритмами. Розроблений пакет реалізує випадковий фільтр, простий item-based фільтр, простий user-based фільтр, SVD фільтр та Slope One фільтр, а також надає можливість візуалізації отриманих результатів для їх подальшого порівняння.

Отримані результати аналізу рекомендаційних систем, а також розроблені інструменти на мові Python мають велику корисність як для людей, вивчаючих такі алгоритми, так і спеціалістів у цій галузі.

1 ЗАГАЛЬНІ ВІДОМОСТІ ПРО РЕКОМЕНДАЦІЙНІ СИСТЕМИ

Рекомендаційні системи становлять підкласс систем фільтрації інформації. Основним завданням таких систем є вибір та оцінка деяких даних таким чином, що результати вибору співпадають з побажаннями користувача.

Рекомендаційні системи набули величезного попиту разом з розвитком інтернету та, особливо, з розвитком онлайн торгівлі. Різновидні рекомендаційні системи використовуються такими компніям як Google, Amazon та Netflix, з метою покращення продажів послуг та товарів, які вони пропонують користувачам. Остання компанія (Netflix) навіть запровадила власний конкурс, на якому змагались розробники усього світу, з метою створити найкращу рекомендаційну систему. Сьогодні визначити конкретного винахідника, або дату винаходу — неможливо, адже рекомендаційні системи це набір різновидних алгоритмів та систем, яких споріднюють задачі оцінки та рекомендації деяких товарів.

Будь-яка рекомендаційна система використовує усі істотні відомості про користувача, що мають допомогти при підборі деяких об'єктів. Надалі об'єктом називатимемо одиницю товару, послугу чи, наприклад, фільм, або пісню — будь-що, що можливо рекомендувати. Ця назва є найбільш логічною, адже рекомендаційні системи можуть бути некомерційними, тобто пропонувати не товар. Можуть рекомендувати як матеріальні, так і не матеріальні одиниці. Взагалі, рекомендаційні системи використовуються для побудови рекомендацій для великої кількості різних благ. Тим не менш, не зважаючи на спорідненість вхідних даних та бажаних результатів, рекомендаційні системи сильно відрізняються одна від одної.

1.1 Види рекомендаційних систем

Основними стратегіями в формуванні оцінок, або рейтингів та фільтрації найкращих об'єктів є фільтрація вмісту та колаборативна фільтрація. Також існують і змішані типи, або ж зовсім езотеричні та специфічні відгалуження, втім зазвичай вони є запатентованими винаходами та/або дуже специфічними до якоїсь конкретної галузі.

У цій роботі в першу чергу розглядатимуться системи колаборативної фільтрації, адже вони, так само як і системи контентної фільтрації, становлять ключовий класс рекомендаційних систем. Детальна інформація щодо конкретних алгоритмів рекомендаційних систем, а також їх характерні ознаки наведена у роботах [1]—[4].

1.1.1 Фільтрація вмісту

Принципом функціонування рекомендаційних систем на основі фільтрації вмісту є пошук об'єків, які найбільш точно відповідають смаку користувача, спираючись на зміст об'єктів. Наприклад, рекомендація фільмів, які вийшли у тому ж десятиріччі, фільмам якого користувач віддав найбільшу перевагу. При побудові таких рекомендацій використовують так званий профіль об'єкта. Знову ж, наприклад у кіно — це деякий вектор, який містить інформацію про фільм (дата випуску, режисер, жанр тощо) та профіль користувача, який має ту ж розмірність, що і профіль об'єкта, та містить інформацію про те, об'єктам з якими характеристиками цей користувач віддає найбільшу перевагу. Завданнями, які виникають при побудові таких рекомендаційних систем, є побудова профілю користувача та подальший вибір найбільш наближених за характеристиками до цього профіля об'єктів. Більш істотно питання фільтрації вмісту розглянуто у роботі [5].

Класичним прикладом рекомендаційної системи на основі фільтрації

контенту є так званий алгоритм Роккіо:

на початку для кожного користувача будується його профіль x_u. Для цього використовують рейтинги, які користувача надав деяким об'єктам, та, власне, інформацію про них, яку надано у вигляді векторів $x_i, i \in I_u$. Профіль користувача знаходять наступним чином:

$$x_u = \sum_{i \in I_u} r_{ui} x_i, \tag{1.1}$$

де r_{ui} — рейтинг, який користувач u надав предмету i.

Потім обчислений профіль користувача порівнюється з профілем об'єктів, які користувачу треба порекомендувати. З них обираються найкращі. Втім є велика кількість більш витончених систем, які використовують методи машинного навчання, наприклад штучні нейронні мережі, або кластерний аналіз.

У цій роботі такі системи не розглядаються, втім варто зазначити, що у деяких випадках вони здатні досягти більш якісних результатів у порівнянні з системами колаборативної фільтрації, адже вони грунтуються на більш детальній характеристиці користувача. Недоліками таких систем нерідко є їх складність та те, що майже завжди модель таких фільтрів ґрунтується на деяких спрощеннях та ідеях людей, які їх створюють. Тому можуть виникати помилки та неточності, спричинені людським фактором. Іншою проблемою таких систем є те, що вони визначають інтереси на вже наявній про користувача інформації. Це унеможливлює, та й взагалі не передбачає можливості прогнозування зміни користувацьких інтересів у майбутньому. Через специфіку таких рекомендаційних систем, а також доступні дані для тестування, було обрано інший вид ркомендаційних систем, що дозволило провести більш ретельний аналіз.

1.1.2 Колаборативна фільтрація

Класична колаборативна фільтрація на сьогоднішній день вже ретельно вивчена. Вчені розумають недоліки та переваги рекомендаційних

систем, які використвоують колаборативну фільтрацію. Незважаючи на деякі труднощі при використанні таких систем, вони набули значного попиту, а разом із тим і розвитку, який привів до фрагментування класичного алгоритма у велику кількість альтернативних, які призначені для покращення якості та швидкості роботи.

У більшості випадків на вхід рекомендаційної системи колаборативної фільтрації подаються відомості про вже наявні відгуки, або іншими словами рейтинги чи оцінки користувачів. Зазвичай ці відомості подаються у вигляді так званої матриці користувача та об'єкта (user-item matrix) елементами якої є відповідні оцінки користувача. Приклад такої матриці наведено у таблиці 1.1.

Таблиця 1.1 — Матриця користувача–об'єкта

	i_1	i_2	...	i_n
u_1	v_{11}	v_{12}	...	v_{1n}
u_2	v_{21}	v_{22}	...	v_{2n}
...	...	...	...	...
u_m	v_{m1}	v_{m2}	...	v_{mn}

Саме цей тип рекомендаційних систем детально розглядатиметься у наступних розділах. Основною відмінністю від фільтрації вмісту є відсутність, або ж не використання даних про зміст об'єктів та детального опису профіля користувача. Втім, у таких системах використовують інший важливий ресурс інформації – наявні оцінки інших користувачів. Рекомендаційні системи прийнято поділяти на кілька підтипів.

Системи, засновані на пам'яті, – це підтип колаборативної фільтрації, який базується на відомих даних про користувача. Тобто пам'яті про нього. Такий підхід вважався найкращим та застосовувався у великій кількості різних комерційних систем. Він є відносно простим, однак одночасно залишається ефективним. Прикладом такої системи може виступати звичайний User-based або Item-based фільтр.

Системи, засновані на моделі, – це підтип більш тонких алгоритмів, які базуються на витончених алгоритмах машинного навчання, метою яких

є дослідження даних, закономірностей. Такі моделі, зазвичай створюються у досить специфічній області в індивідуальному порядку та їх застосування обмежено саме цією областю. У той самий час перевагою таких алгоритмів є більш якісна обробка великих скупчень неідеальних даних. Теоретично, примітивним алгоритмом колаборативної фільтрації заснованим на моделі є випадковий фільтр, який буде розглянуто далі.

1.2 Функції подібності у рекомендаційних системах

У багатьох різних алгоритмах колаборативної фільтрації, як user-based, так і item-based, виникає проблема вибору схожих користувачів (або, відповідно, об'єктів).

Як вже відомо, рекомендаційним системам колаборативної фільтрації доступні лише список користувачів, список об'єктів та рейтингів, які ці списки між собою пов'язують. Для обчислення рівня подібності між двома користувачами або об'єктами була запропонована велика кількість різних методів. Всіх їх поєднує те, що вони використовують вектори рейтингів, які потім порівнюються між собою.

Основними та найбільш популярними методами обчислення подібності є кореляція Пірсона, косинус подібності та квадратична відстань. Детально питання вибору функцій подібності освітелно роботі [6].

1.2.1 Кореляція Пірсона

Кореляція Пірсона. Цей підхід до обчислення подібності є одним з найперших. Ідея полягає у обчисленні коефіцієнта Пірсона між двома порівнюваними об'єктами або користувачами. Позначимо за a та u двох порівнюваних користувачів.

$$s(a,u) = \frac{\sum_{i \in I_a \cap I_u}(v_{ai} - \bar{v_a})(v_{ui} - \bar{v_u})}{\sqrt{\sum_{i \in I_a \cap I_u}(v_{ai} - \bar{v_a})^2 (v_{ui} - \bar{v_u})^2}}. \tag{1.2}$$

Така ідея обчислення подібності виявилась доволі популярною, однак також породила декілька альтернативних підходів як, наприклад, Обмежена кореляція Пірсона, у якій середні рейтинги користувачів замінено на центральну можливу оцінку. Центральну оцінку, позначимо, наприклад κ.

$$s(a,u) = \frac{\sum_{i \in I_a \cap I_u}(v_{ai} - \kappa)(v_{ui} - \kappa)}{\sqrt{\sum_{i \in I_a \cap I_u}(v_{ai} - \kappa)^2 (v_{ui} - \kappa)^2}}. \tag{1.3}$$

Така альтернатива була запропонована з метою більшої концентрації на відхиленнях від центральної оцінки, не спираючись на усереднені показники окремих користувачів.

1.2.2 Середня квадратична відстань

Середня квадратична відстань (англ. Mean Squared Difference) є більш звичною та більш подібною до класичного розуміння відстані:

$$\mathrm{msd}(a,u) = \frac{\sum_{i \in I_a \cap I_u}(v_{ai} - v_{ui})^2}{|I_a \cap I_u|}. \tag{1.4}$$

1.2.3 Косинус подібності

Косинус подібності (англ. Cosine Similarity) — визначається як косинус кута між двома векторами, а саме векторами, зіставленими з рейтингів

користувачів, чи об'єктів. На сьогоднішній день є одним з найпопулярніших методів підрахунку подібності.

$$s(a,u) = \sum_{j \in I} \frac{v_{aj}}{\sum_{k \in I_a} v_{ak}^2} \frac{v_{uj}}{\sum_{k \in I_a} v_{uk}^2}.$$

Кожна з цих ідей суттєво впливає на результати, які видає рекомендаційна система, втім неможливо визначити якусь конкретну, найкращу ідею визначення подібності. Порівняння результатів роботи деяких алгоритмів в залежності від функції подібності буде наведено в одному з наступних розділів.

1.3 Висновки до розділу 1

У цьому розділі було наведено ключові відомості про різні види рекомендаційних систем. З інформації, наведеної у цьому розділі випливає, що рекомендаційні системи складають великий набір різновидних алгоритмів, налаштування та вибір яких є складною, однак дуже важливою задачею. Рекомендаційні системи поділяють за їдеєю алгоритму, який вони реалізують. У цій роботі розглядатиметься окремий класс рекомендаційних систем, використовуючих колаборативну фільтрацію для побудови рекомендацій.

2 МЕТРИКИ ДЛЯ ОЦІНЮВАННЯ ЯКОСТІ РЕКОМЕНДАЦІЙНИХ СИСТЕМ

На сьогоднішній день рекомендаційні системи представлені у вигляді значної кількості різних методів та алгоритмів, втім, завдання у всіх рекомендаційних систем майже однакове — маючи деякі дані створити деякий набір рекомендацій. Такий уніфікований вигляд вихідних даних робить можливим тестування рекомендаційних систем при визначенні різних значень параметрів, а також порівняння різних рекомендаційних систем між собою. В той самий час, різноманіття різних підходів, відгалужень та варіацій рекомендаційних систем призвело до необхідності побудови доволі складного, втім дуже корисного підходу оцінювання якості роботи рекомендаційних систем.

У цій роботі будуть розглянуті не усі можливі оцінки, адже деякі з них є специфічними для окремих галузей, у яких рекомендаційні системи застосовуються, специфічних видів рекомендаційних систем або взагалі є не дуже інформативними. Не зважаючи на це, головні та найважливіші оцінки, за якими визначають якість будь-якої рекомендаційної системи, у цій роботі будуть наведені.

2.1 Оцінки часу

Напевно одна з найголовніших характеристик будь-якого алгоритму — це його часова складність. Однак рекомендаційні системи, фактично, складаються з декількох послідовних алгоритмів, а саме: алгоритм навчання рекомендаційної системи, тобто алгоритм збору, модифікації та обробки наявної інформації, та підготовка її до побудови рекомендацій. Незважаючи на те, що час на навчання зазвичай пов'язують з алгоритмами штучного інтелекту, як наприклад нейронні мережі, навіть у більш строгих

та простіших алгоритмах рекомендаційних систем процес пошуку сусідів (як подібних користувачів, так і об'єктів) можливо визначити як процес підготовки та обробки інформації у належний вигляд, тобто процес навчання. Так само, тоді, потрібно визначити й окремий алгоритм побудови рекомендацій. У більшості рекомендаційних систем він складається з вибору деякої кількості об'єктів або користувачів, та подальшого визначення рекомендованого об'єкта та оцінки, яка йому надається.

Таким чином, рекомендаційні системи пропонується оцінювати за трьома часовими характеристиками: час витрачений на підготовку системи (у подальшому train time тренування), час витрачений на побудову рекомендацій (у деякому сенсі побудова рекомендацій споріднена з прогнозуванням, адже рекомендаційна система, принаймні у цій роботі, також оцінює майбутню оцінку, тому цей часовий проміжок надалі буде також позначатись predict time). Накінець, незважаючи на корисність минулих характеристик, найважливішою часовою характеристикою, яка в першу чергу цікавить розробників програмного забезпечення (ПЗ) є загальний час, який витрачає рекомендаційна система. Фактично це сума попередніх двох часових характеристик, втім через відмінність часової складності витрат часу на підготовку та витрат часу на побудову результатів – результат, іноді, доволі складно обчислисти.

Нарешті, усі три часових проміжки пропонується оцінювати в залежності від кількості вхідних даних, які має обробити та використати рекомендаційна система.

У цій роботі загальний масив даних випадковим чином розбивається на дві множини: множину рейтингів, які рекомендаційній системі надаються як вхідні дані, та множину рейтингів, як рекомендаційна система не отримує, а для яких вона має побудувати власні оцінки. Далі вихід рекомендаційної системи порівнюватиметься з другою множиною "невідомих" системі рейтингів. Отже, час на підготовку системи залежить в першу чергу від кількості вхідних даних, коли час на побудову рекомендацій залежить від кількості рейтингів, яка рекомендаційна система має генерувати. Загальній же час залежить від загальної кількості

даних взагалі. Тому пропонується ввести параметр L, який визначатиме частку тестових даних від загальної кількості рейтингів. Тоді частка тренувальних даних від загальних рейтингів визначатиметься як

$$1 - L. \tag{2.1}$$

Детальний аналіз часових характеристик кожного алгоритму проводитиметься саме за такою процедурою.

2.2 Оцінки точності

Як вже зазначалоссь, рекомендаційні систем слід порівнювати за великою кількістю різних параметрів та показників, втім, показники які безпосередньо цікавлять замовників, розробників та користувачів рекомендаційний систем — час та точність. Безпосередньою оцінкою точності в рекомендаційних системах є відхилення прогнозованої оцінки користувача від реальної оцінки, яку користувач надає. Втім узагальнену точність, або похибку, можливо обчислювати по-різному.

У цій роботі пропонується використання як стандартних для рекомендаційний систем метрик, так і тих, які широко застосовуються в інших галузях пов'язаних з машинним навчанням.

2.2.1 Оцінка MAE

Середня абсолютна похибка (англ. Mean Absolute Error, скорочено MAE) — стандартна оцінка похибки майже будь якого алгоритму, у загальному випадку її визначають наступним чином:

$$|\bar{E}| = \frac{\sum_i^N |x_i - y_i|}{N}. \tag{2.2}$$

У випадку рекомендаційних систем, вектори x та y замінено на p_i та

v_i, також додано усереднення за всіма користувачами.

$$mae = \frac{1}{|U|} \sum_{u \in U} \frac{\sum_{i \in I_u} |p_{ui} - v_{ui}|}{|I_u|}. \tag{2.3}$$

Ця метрика є доволі простою, втім завдяки своїй простоті та деяким статистичним властивостям є доволі популярною при оцінці рекомендаційних систем.

2.2.2 Оцінка RMSE

Усереднена квадратична помилка (англ. Root mean squared error, скорочено RMSE)

$$|\bar{E}| = \sqrt{\frac{\sum_i^N (x_i - y_i)^2}{N}}.$$

У рекомендаційних системах RMSE набуває наступного вигляду:

$$rmse = \frac{1}{|U|} \sum_{u \in U} \sqrt{\frac{\sum_{i \in I_u} (p_{ui} - v_{ui})^2}{|I_u|}}. \tag{2.4}$$

Вона також є доволі популярною, однак, на відміну від попередньої оцінки усереднена квадратична помилка більш залежна від різновидних коливань та різких, великих помилок. Тобто RMSE виділяє більш значні помилки на фоні невеликих. Використання RMSE спричинене тим, що у деяких випадках необхідно оцінювати не лише середню помилку, а наявність різних відхилень. Так, наприклад, такі оцінки необхідно вживати там, де для користувача є чітка межа між прийнятим та неприйнятним об'єктом. Якщо RMSE мала, то це говорить про те, що ймовірність помістити прийнятний об'єкт у категорію неприйнятних, та навпаки, в рекомендаційної системи мала. Якщо ж RMSE велика, то це, фактично, означає, що у деяких випадках різниця між реальною оцінкою користувача

та прогнозом рекомендаційної системи є завеликою, тобто у таких випадках рекомендаційна система майже напевно робить невірну рекомендацію.

2.2.3 Оцінка RMSLE

Усереднена квадратична логарифмована помилка (англ. Root Mean Squared Log Error, скорочено RMSLE)

$$|\bar{E}| = \sqrt{\frac{\sum_i^N (log(x_i + 1) - log(y_i + 1))^2}{N}}.$$

У рекомендаційних системах RMSLE набуває наступного вигляду:

$$rmsle = \frac{1}{|U|} \sum_{u \in U} \sqrt{\frac{\sum_{i \in I_u} (log(p_{ui} + 1) - log(v_{ui} + 1))^2}{|I_u|}}. \tag{2.5}$$

Ця метрика використовується у багатьох різних задачах. Безпосередню популярність вона набула у різних проблемах аналізу часових рядів та прогнозування, де окрім загальною оцінки необхідно робити оцінки щодо відносних відхилень, без урахування постійної складової, тобто RMSLE можливо використовувати як характеристику відносної помилки. У випадку рекомендаційних систем це може бути використане як характеристика того, наскільки чутливою є система до смаків користувача. Наприклад, якщо рекомендаційна система робить доволі незначну помилку для кожної рекомендації, тобто робить якесь постійне відхилення для кожної рекомендації, втім дуже чітко реагує на оцінки користувача (дає нижчу оцінку там, да користувач дійсно дав погану оцінку, та навпаки) RMSLE буде доволі незначною.

Фактично RMSLE є альтернативою до RMSE у тих випадках, коли необхідно провести аналіз відносної похибки. Однак у тих випадках, коли також необхідно оцінити своєрідний зсув оцінок, RMSE та MAE є більш корисними.

Задля кращого розуміння суті кожної з метрик до розгляду пропонуються порівняльні графіки, які містять прототипи кожної з оцінок. Ці графіки, так само як і усі інші графіки у цій роботі, були спеціально створені у якості візуальної складової розробленого пакету. Графічне зображення помилок обчислених різними методами наведено на рисунках 2.1 та 2.2.

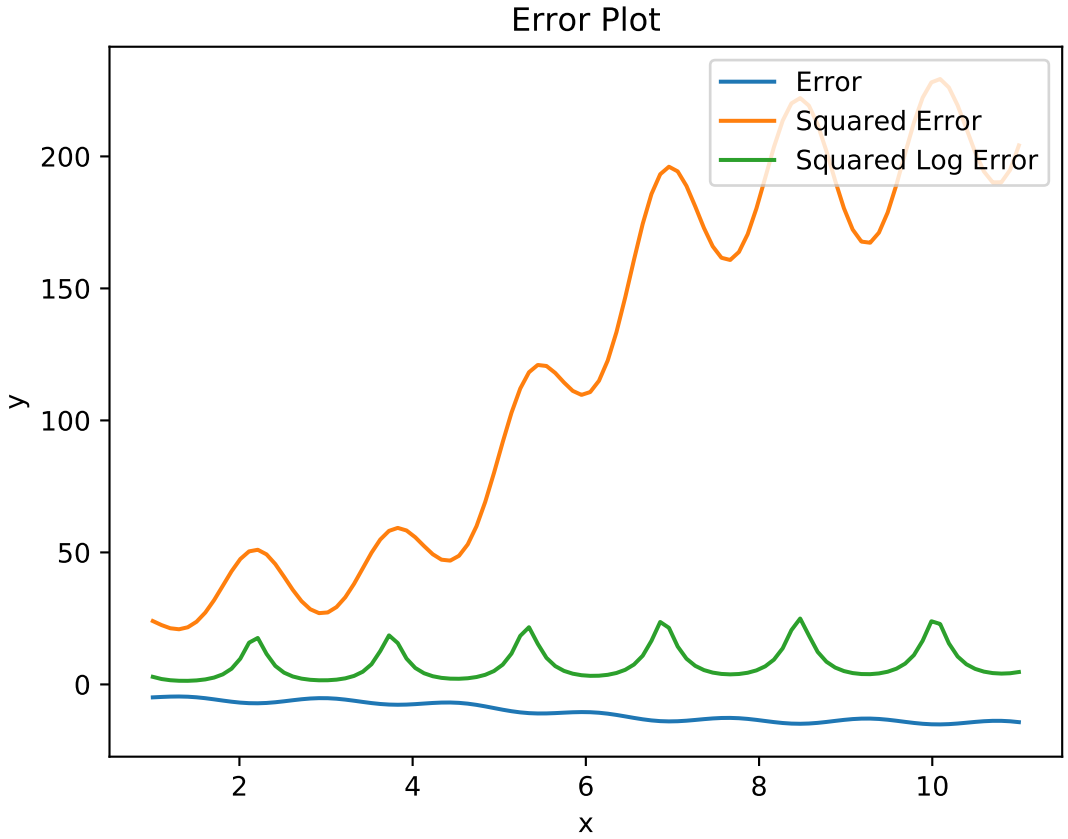

Рисунок 2.1 — Оцінки відхилень без шуму

На цьому графіку зображено звичайну помилку, квадрат помилки та помилку між логарифмами значень, які цю похибку утворюють.

Поведінку помилок у разі наявності шуму наведено на рисунку 2.2.

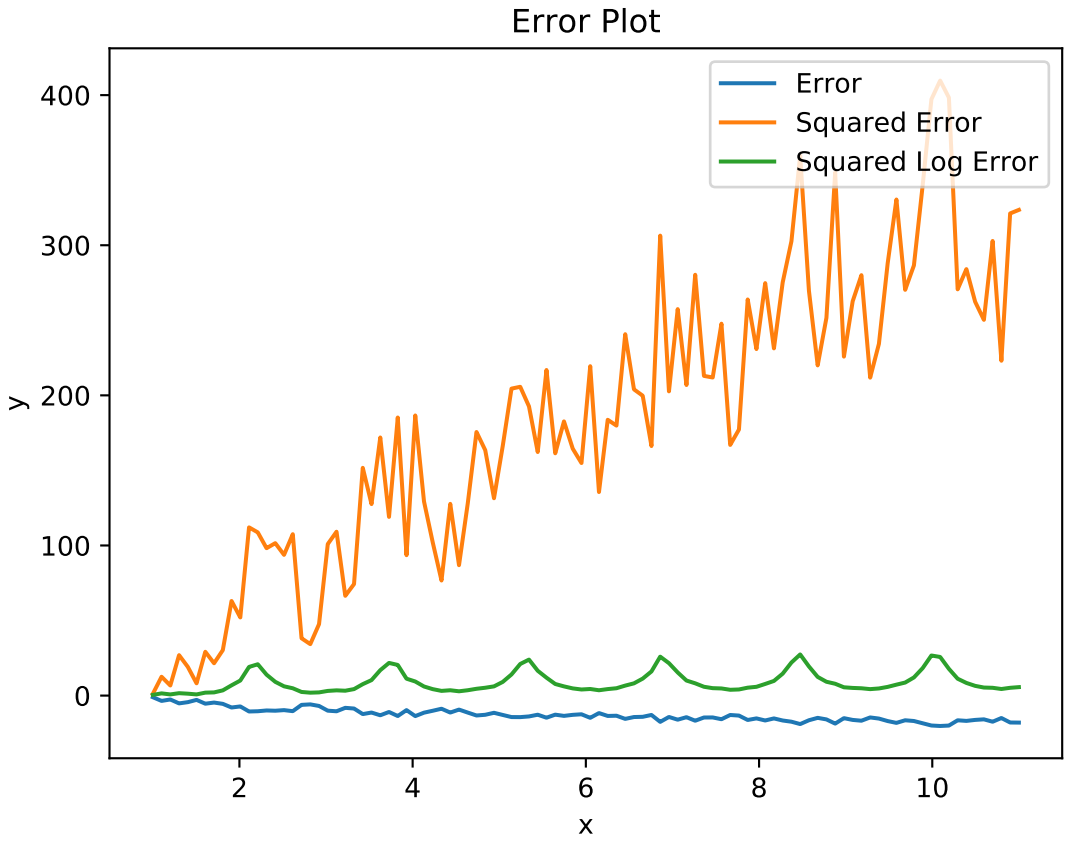

Рисунок 2.2 — Оцінки відхилень за наявності шуму

Звідси видно, що всі три метрики мають різну поведінку, тому порівняння різних видів рекомендаційних систем відбуватиметься за кожною з трьох. RMSLE реагує значно більш помірно, однак зберігає загальну поведінку, у той час коли RMSE реагує як на стрибкоподібні відхилення, так і на сталі зміни, які RMSLE практично ігнорує.

2.3 Точність та повнота

Цей набір характеристик нерідко порівнюють з оцінками бінарних класифікаторів. Дійсно, деякі з них обчислюються схожим чином, однак у випадку рекомендаційних систем деякі метрики набули доволі своєрідного вигляду.

Точність (англ. precision) – частка правильно спрогнозованих оцінок та об'єктів серед усіх спрогнозованих.

Повнота (англ Recall) – частка правильно прогнозованих результатів серед усіх релевантних об'єктів.

У випадку бінарних величин, оцінити такі характеристики доволі легко, користуючись наступними формулами:

$$precision = \frac{tp}{tp + fp}, \tag{2.6}$$

$$recall = \frac{tp}{tp + fn}, \tag{2.7}$$

де tp – число вірно визначених елементів;

fp – число невірно визначених елементів;

fn – число невірно проігнорованих елементів.

У випадку рекомендаційних систем загальна формула не змінюється, однак виникає питання того, який об'єкт вважати релевантним. Для цього вводиться поняття порогової величини (англ. threshold), яке дозволяє визначити, який об'єкт є релевантним (оцінка більше порогової), а який – ні (оцінка менше порогової).

$$precision(u) = \frac{|\{i \in I_u : v_{ui} > \theta \wedge p_{ui} > \theta\}|}{|\{i \in I_u : v_{ui} > \theta \wedge p_{ui} > \theta\}| + |\{i \in I_u : v_{ui} < \theta \wedge p_{ui} > \theta\}|}.$$

$$recall(u) = \frac{|\{i \in I_u : v_{ui} > \theta \wedge p_{ui} > \theta\}|}{|\{i \in I_u : v_{ui} > \theta \wedge p_{ui} > \theta\}| + |\{i \in I_u : v_{ui} > \theta \wedge p_{ui} < \theta\}|}.$$

Однак такі характеристики використовують інформацію лише про одного користувача. У цій роботі така оцінка проводитиметься по усій базі користувачів.

$$precision = \frac{1}{|U|} \sum_{u \in U} \frac{|\{i \in I_u : v_{ui} > \theta \wedge p_{ui} > \theta\}|}{|\{i \in I_u : v_{ui} > \theta \wedge p_{ui} > \theta\}| + |\{i \in I_u : v_{ui} < \theta \wedge p_{ui} > \theta\}|}$$

$$recall = \frac{1}{|U|} \sum_{u \in U} \frac{|\{i \in I_u : v_{ui} > \theta \wedge p_{ui} > \theta\}|}{|\{i \in I_u : v_{ui} > \theta \wedge p_{ui} > \theta\}| + |\{i \in I_u : v_{ui} > \theta \wedge p_{ui} < \theta\}|}$$

Використовуючи обидві ці характеритики рекомендаційних систем, будують одразу декілька більш складних оцінок, які будут описані далі.

2.3.1 Precision@k

Precision@k – популярне позначення повноти серед перших (найкращих) k рекомендацій. Фактично співпадає з класичним визначенням повноти, однак загальні оцінки обмежують першими k элементами, а відповідні множини мають бути відсортовані за спаданням. Формула Precision@k має наступний вигляд:

$$precision = \frac{1}{|U|} \sum_{u \in U} \frac{\sum_i^k (v_{ui} > \theta)(p_{ui} > \theta)}{\sum_i^k (v_{ui} > \theta)(p_{ui} > \theta) + \sum_i^k (v_{ui} < \theta)(p_{ui} > \theta)}. \quad (2.8)$$

2.3.2 Recall@k

Також як і Precision@k – параметр k лише обмежує вибірку, на якій обчислюють характеристику. Формально записується наступним чином:

$$precision = \frac{1}{|U|} \sum_{u \in U} \frac{\sum_i^k (v_{ui} > \theta)(p_{ui} > \theta)}{\sum_i^k (v_{ui} > \theta)(p_{ui} > \theta) + \sum_i^k (v_{ui} > \theta)(p_{ui} < \theta)}. \quad (2.9)$$

2.3.3 F1

Статистика F1 власної назви не має. Вона є комбінацією повноти та точності. Формально записується наступним чином:

$$F1 = \frac{2 \cdot precision \cdot recall}{precision + recall} \quad (2.10)$$

Однак цей вираз дещо спрощується:

$$F1 = \frac{2 \cdot \frac{tp}{tp+fp} \cdot \frac{tp}{tp+fn}}{\frac{tp}{tp+fp} + \frac{tp}{tp+fn}} = \frac{\frac{2tp^2}{(tp+fp)(tp+fn)}}{\frac{tp^2+fntp+tp^2+tpfp}{(tp+fp)(tp+fn)}} =$$
$$= \frac{2tp^2}{2tp^2 + tp(fn+fp)} = \frac{tp}{tp + \frac{fn+fp}{2}}. \tag{2.11}$$

Іноді також як повнота та точність під $F1$ розуміють $F1@k$ – $F1$ обчислену за найкращими k рекомендаціями.

2.3.4 Криві точності та повноти

Усі вище зазначені характеристики доволі інформативні самі по собі, однак, як це легко побачити з формального опису кожної із них, необхідно якимось чином вибирати параметри, такі як k та порогову оцінку θ (threshold). Замість цього вводяться графічні зображення цих характеристик, у яких відображено залежність значень оцінок рекомендаційних систем від параметрів.

Аналогом ROC – кривих бінарних класифікаторів, у рекомендаційних системах виступає крива точності/повноти. У бінарних класифікаторах ROC криві (англ. Receiver Operating Characteristic curve) будуються як графік відношення кількості вірно зроблених класифікацій, або ж рекомендацій, до кількості помилкових. У випадку рекомендаційних систем прийнято оцінювати відношення між точністю та повнотою. Формально відношення можливо дещо спростити до наступного виразу:

$$\frac{recall}{precision} = \frac{\frac{tp}{tp+fn}}{\frac{tp}{tp+fp}} = \frac{tp+fp}{tp+fn}. \tag{2.12}$$

Що у сенсі рекомендаційних систем було б коректно записати у наступному вигляді:

$$\frac{recall}{precision} = \frac{\sum_{u \in U} \sum_{i}^{k} (v_{ui} > \theta)(p_{ui} > \theta) + \sum_{i}^{k} (v_{ui} < \theta)(p_{ui} > \theta)}{\sum_{i}^{k} (v_{ui} > \theta)(p_{ui} > \theta) + \sum_{i}^{k} (v_{ui} > \theta)(p_{ui} < \theta)} \tag{2.13}$$

або

$$\frac{recall}{precision} = \\ = \frac{\sum_{u \in U} \sum_{i \in I_u} (v_{ui} > \theta)(p_{ui} > \theta) + \sum_{i \in I_u} (v_{ui} < \theta)(p_{ui} > \theta)}{\sum_{i \in I_u} (v_{ui} > \theta)(p_{ui} > \theta) + \sum_{i \in I_u} (v_{ui} > \theta)(p_{ui} < \theta)}. \tag{2.14}$$

Як це можливо побачити з останнього запису: відношення між точністю та повнотою також залежить від параметра. Звідси і випливає фактичний опис однієї з кривих точності та повноти. На графіку зображаються значення точності та повноти в залежності від різних значень граничної оцінки. Для різних рекомендаційних систем така крива прийматиме різний вигляд, та її сенс буде більш детально описаний у частині порівнянь рекомендаційних систем. На рисунку 2.3 зображається базовий вигляд такої кривої для однієї з рекомендаційних систем колаборативної фільтрації.

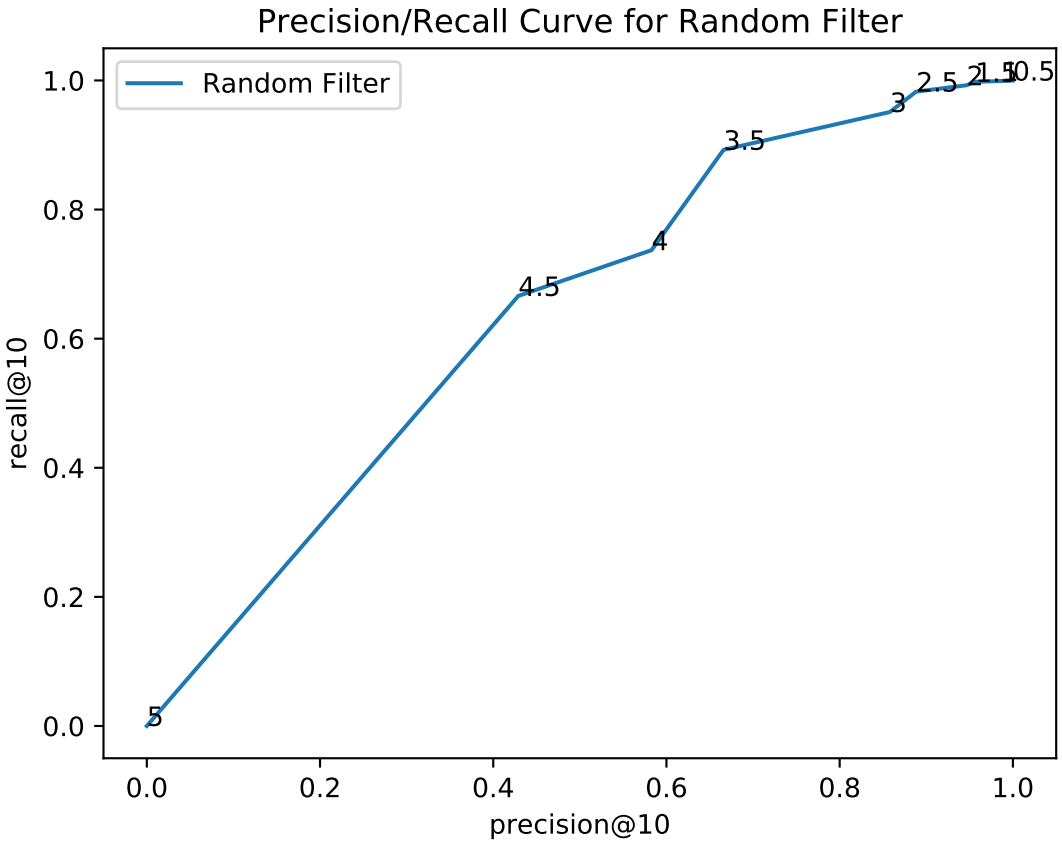

Рисунок 2.3 — Приклад кривої точності/повноти

Іншою кривою є зображення точності та повноти за фіксованого значення граничної оцінки, однак за змінного розміру вибірки, на якій точність та повнота обчислюються. Такі криві теж мають велике значення при аналізі рекомендаційних систем. У порівнянні з минулою кривою, графічне зображення такої характеристики примає дещо інакший вигляд. Приклад наведено на рисунку 2.4.

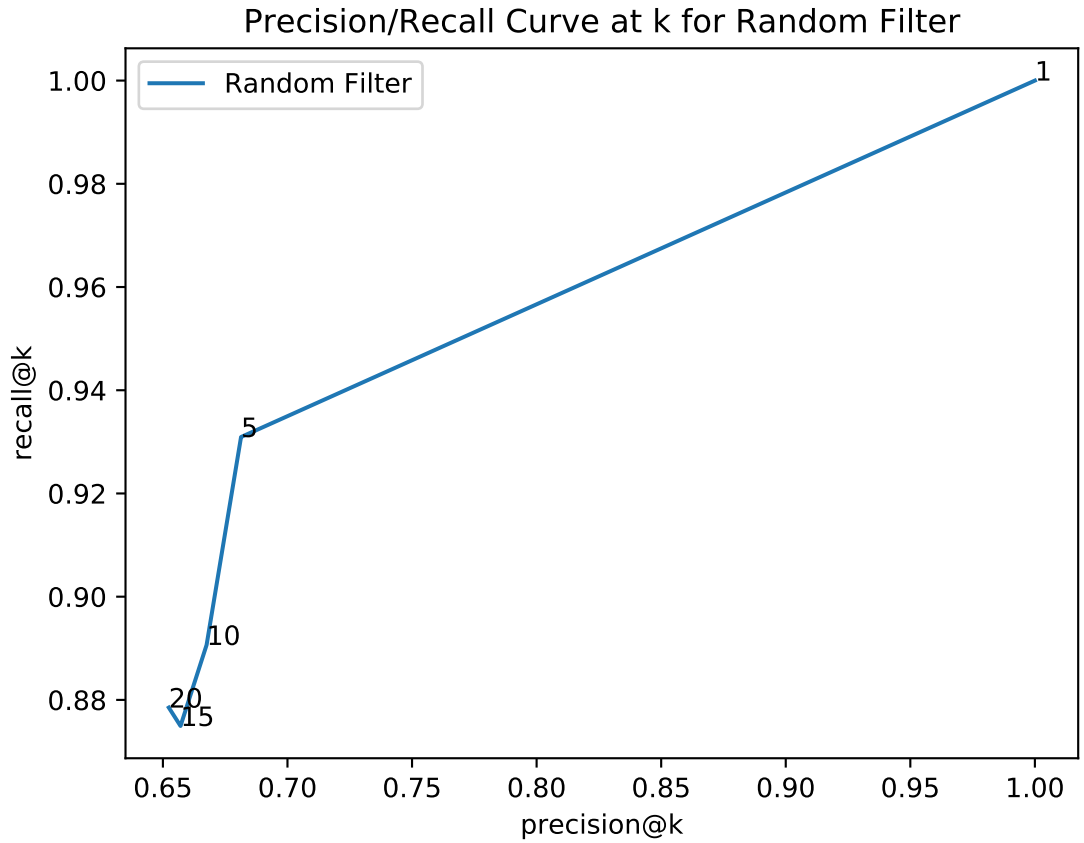

Рисунок 2.4 — Приклад кривої точності/повноти побудованої на k кращих рекомендаціях

Обидві криві мають велике значення при описі поведінки рекомендаційної системи. Перша дозволяє оцінити якість результатів в залежності від смаку користувача, у той час як друга дозволяє оцінити загальну якість системи в залежності від кількості об'єктів, які необхідно рекомендувати.

Реалізація побудови та відображення таких кривих дозволяє провести порівняння рекомендаційних систем у доволі простій та елегантній формі: крім відомостей про оцінки часу та точності необхідно побудувати лише два графічні зображення, які в іншій формі мали б бути представлені у вигляді доволі великих та незручних для вивчення таблиць.

2.4 Інші оцінки

Також існує велика кількість інших метрик, які дозволяють оцінити різноманітні специфічні показники рекомендаційної системи. Жодна з наведених нижче оцінок не використовується у подальших розділах роботи через їх специфічність, непотрібність та/або непридатність для використання у цій роботі.

2.4.1 Покриття

Під покриттям розуміють оцінку того, яку частину наявних предметів може рекомендувати система. Однією з часто трапляючихся проблем рекомендаційних систем є побудова рейтингів з найбільш популярних предметів, по яким вже існує доволі багато інформації, відгуків, тощо.

Обчислюється така характеристика як відношення кількості різних предметів, які було рекомендовано, до загальної кількості різних предметів, які можливо рекомендувати. У нашому випадку така статистика не має практичного сенсу через те, що рекомендаційну систему штучно змушено побудувати власні рейтинги або прогнози до всіх елементів тестувальної вибірки. Тобто статистика завжди дорівнювала б одиниці.

2.4.2 Персоналізація

Ще однією статистикою, яка у цій роботі не розглядається, є індивідуальність підбору рекомендацій. Проблема індивідуальності також зустрічається доволі часто і виникає, зазвичай через нестачу інформації про користувачів або неправильно обрані параметри системи. Оцінюється така метрика як порівняння векторів рекомендацій декількох користувачів між собою. Тобто, якщо ці вектори схожі між собою для деякої

кількості користувачів, то можливо зробити висновок, що система робить рекомендації, які не є індивідуальними до смаку окремих користувачів.

2.4.3 Подібність рекомендованих об'єктів

Використовується у тих випадках, коли необхідно перевірити, чи має рекомендаційна система схильність рекомендувати об'єкти з якимись характерними ознаками. Обчислюється як подібність усіх усіх рекомендованих об'єктів між собою. Таку характеристику слід використовувати при оцінці рекомендаційних систем фільтрації контенту або рекомендаційних систем, у яких можуть виникати проблеми повноти рекомендаційного набору.

2.5 Висновки до розділу 2

Рекомендаційні системи можливо оцінювати за великою кількістю показників. Деякі з них є доволі звичними та популярними, деякі є специфічними до окремих алгоритмів, а деякі, взагалі, раніше не використовувались. У цій роботі особливу увагу приділено таким метрикам рекомендаційних систем, за якими їх можливо порівняти між собою. Окремі метрики точності можливо використати при оцінці якості не тільки рекомендаційних систем колаборативної фільтрації, а й рекомендаційних систем використовуючих контентну фільтрацію, або ж, взагалі, систем, які використовують змішаний підхід.

3 ВИБІР ДАНИХ

Для отримання достовірних результатів аналізу та порівняння рекомендаційних систем, необхідно використовувати вхідні дані, які дозволяли б протестувати рекомендаційні системи як у найгірших, так і у найкращих для них умовах.

3.1 Аналіз обраного набору даних

Основними проблемами, які виникають у роботі рекомендаційних систем є відсутність необхідної для їх роботи інформації. Рекомендаційним системам доводиться обробляти розріджені матриці, якість обробки яких безпосередньо впливає на якість побудованих рекомендацій.

У цій роботі використовується доволі популярна вибірка MovieLens, яка містить оцінки фільмів реальних користувачів. Було використано версію ml-latest-small, яка містить інформацію від 610 користувачів про майже 9000 фільмів. Кількість наявних оцінок у цій вибірці складає 100000. Усі користувачі при цьому мають різні характеристики: хтось оцінив декілька сотень фільмів, а в когось наявно не більше кількох десятків оцінок. Хтось з користувачів ставив переважно «п'ятірки», а хтось навпаки. Через це середні оцінки окремих користувачів дуже сильно відрізняються одна від одної, що й дозволило провести детальний аналіз якості розглянутих рекомендаційних систем. У роботі [7] наведено детальну інформацію про інші вибріки цієї системи. Відношення наявних рейтингів, до загальної кількості можливих у обраній вибірці приблизно складає:

$$\frac{100000}{610 * 9000} = \frac{100000}{5490000} = \frac{1}{54.9} = 0.018. \tag{3.1}$$

Крім цього, необхідно розділяти всю вхідну вибірку на тестувальну та

тренувальну. На тренувальній вибірці оцінюється схожість користувачів, або фільмів, або ж робляться інші заключення необхідні для побудови оцінок. На тестувальній вибірці проводиться аналіз результатів, тобто рекомендації системи порівнюються з відповідними елементами тестувальної вибірки.

3.2 Підходи до побудови тестувальної та тренувальної вибірок

Існує декілька варіантів такого розбиття, одним з найпопулярніших варіантів є так звана перехресна перевірка (англ. cross-validation). Таке розбиття реалізовано з метою ітеративної перевірки. Тобто, усі вхідні дані розбиваються на n частин, де n – кількість разів, скільки бажано проводити перевірку. Такий алгоритм корисний тим, що дозволяє підтвердити якість оцінок декількома незалежними перевірками. Якщо результати перевірок з усіх n разів збігаються, тобто залишаються схожими між собою, то робиться висновок, що оцінки є дійсними.

Проблемою такого підходу є те, що береться не випадкова частина вибірки, а якась окрема її частина, причому це розбиття на частини є послідовним. У цій роботі таке розбиття є неприйнятним, адже використовувати для перевірки інформацію про якихось окремих користувачів, або окремі фільми – ризиковано, бо як вже зазначалось, усі користувачі різні, тому навряд оцінки отримані на якійсь одній множині користувачів збігатимуться з оцінками отриманими на повністю іншій множині. Додатково, виникає проблема технічних обмежень. Рекомендаційні системи потребують доволі значних ресурсів обчислювальної техніки. Через те, що у цій роботі розглядається не один, на три, а п'ять різних алгоритмів, одна така ітерація потребує значних витрат часу, для запуску та обробки на звичайному персональному комп'ютері. Якщо до цього додати необхідність перевірки якості роботи рекомендаційних систем в залежності від розміру тестувальної вибірки

— загальна тривалість обчислень, які необхідно провести, сягнула б декількох днів неперервної роботи, при цьому результати цих обчислень були б сумнівними.

Альтернативно було розроблено власний алгоритм розбиття вибірки. Його ідея доволі проста, хоча при цьому є найбільш вдалою для цього класу рекомендаційних систем. Отже, з загальної множини рейтингів випадковим чином обирається множина тестувальних рейтингів визначеного розміру. Для цього вводиться параметр L який визначає долю тестувальних даних у загальній вибірці:

$$L = \frac{V_{test}}{V_{all}}. \tag{3.2}$$

Таким чином, розмір тренувальної вибірки має розмір:

$$1 - L. \tag{3.3}$$

Це дозволяє провести доволі якісну оцінку роботи алгоритмів в залежності від розміру вихідних даних. Якість оцінок при цьому можливо перевірити власноруч, провівши декілька окремих тестів.

Також варто оцінювати якість та розподіл наявних рейтингів. У деяких алгоритмах робиться припущення, що рейтинги є випадковими величинами нормального розподілу з деякими параметрами. Був розроблений модуль побудови розподілу рейтингів, а також приблизної оцінки параметрів розподілу. На графіку нижче наведені зображення фактичного розподілу рейтингів, нормального розподілу з оціненими на основі датасету параметрами та ідеального теоретичного розподілу. Цей графік є дуже важливими при описі одного з алгритмів, а також дозволяє оцінити недоліки використаних даних. Графічне зображення розподілу оцінок наведене на рисунку 3.1.

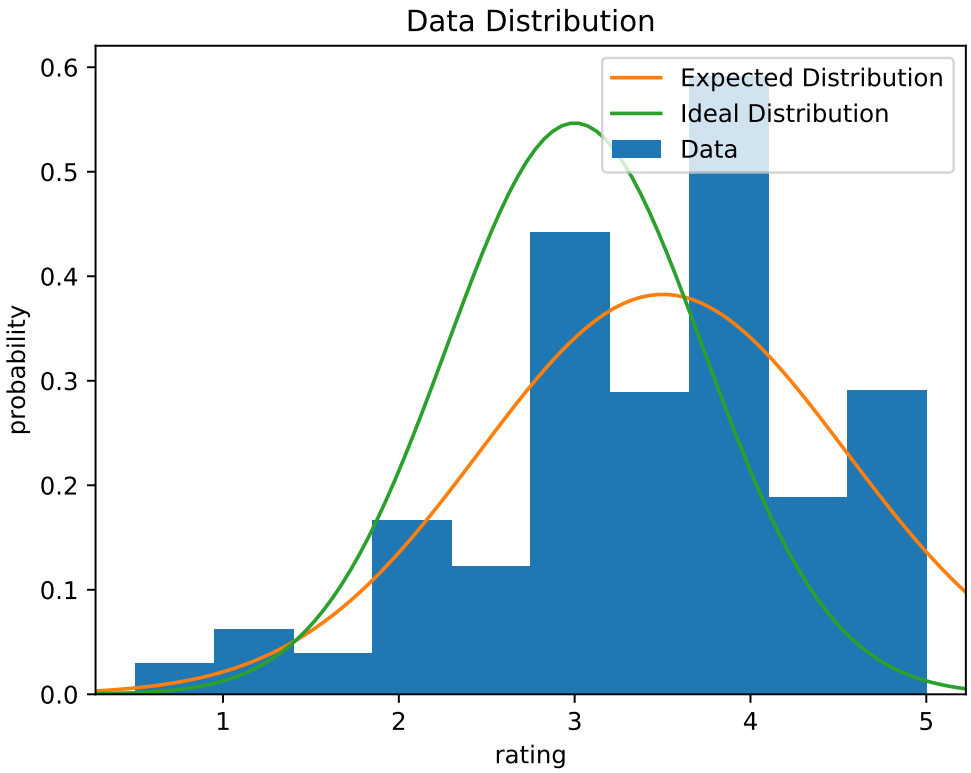

Рисунок 3.1 — Розподіл оцінок користувачів

Фактично, з наведеного графіку можливо зробити висновок, що користувачі схильні позитивно ставити оцінки фільмам, які переглянули. Адже загальна середня оцінка є більшою від середньої як такової.

3.3 Висновки до розділу 3

Датасет, на якому проводитимуться усі тести, має відповідати змісту цих тестів. Так, деякі з критеріїв оцінки рекомендаційних систем оцінють роботу рекомендаційних систем у не дуже гарних для них умовах. Задля проведеня таких тестів необхідно використовувати такий алгоритм побудови вибірок, який би дозволяв варіювати розмір, а також мав би достатньою кількість різновидних записів. Саме такий алгоритм розподілення вхідної інформації й було розроблено. З його допомогою можливо отримати велику кількість різновидних оцінок, які при тому залишатимуться достовірними.

4 АНАЛІЗ ТА ПОРІВНЯННЯ АЛГОРИТМІВ

Основну частину цієї роботи висвітлено у цьому розділі. Фактично необхідно оцінити особливості кожного з обраних алгоритмів, здійснити підбір найкращих параметрів та охарактеризувати властивості рекомендаційних систем і вже потім провести порівняння алгоритмів між собою, використовуючи систему оцінювання алгоритмів наведену в минулих розділах.

4.1 Огляд окремих рекомендаційних систем

У цьому підрозділі наведено інформацію про 5 ключових алгоритмів колаборативної фільтрації. Деякі з них виступають прототипами більш новітніх алгоритмів, деякі є варіантами базової реалізації, на якій будуються більш якісні рекомендаційні системи, а деякі використовуються як повноцінне рішення необхідних задач. Кожний з алгоритмів має свої тонкощі, переваги та недоліки. У цьому розділі основну увагу приділено особливостям алгоритмів, які впливають на їх якість.

Недоліки та переваги алгоритмів більш детально буде розглянуто при порівнянні у наступному підрозділі.

4.1.1 Випадковий фільтр

Стандартна реалізація алгоритму випадкового фільтрування передбачає оцінювання загальних характеристик датасету, вважаючи, що будь-яка оцінка є випадковою, нормально розподіленою величиною. Формально необхідні параметри визначаються наступним чином:

$$\mu = \frac{1}{|V_{train}|} \sum_{v_{ui} \in V_{train}} v_{ui},$$
$$\sigma = \sum_{v_{ui} \in V_{train}} (v_{ui} - \mu)^2,$$
$$\forall i, u \;\; p_{ui} \in N(\mu, \sigma). \tag{4.1}$$

Такий підхід має сенс, виходячи з логіки користувачів: існують чудові та зовсім жахливі фільми, а користувачі можуть ставитись як оптимістично, так і песимістично до будь-якого фільму. Навіть розуміння середньої, або граничної оцінки у всіх різне. Однак візуальна оцінка датасету довела, що до ідеального нормального розподілу датасет, на якому проводяться дослідження, доволі далекий. Тим не менш, пропонується розглянути деякі модифікації алгоритму випадкового фільтрування. Замість того, щоб оцінювати параметри розподілу на всьому датасеті, пропонується робити це на якійсь вибірці. Так випадкова величина з оціненого розподілу була б, можливо, більш близькою до конкретних користувачів або фільмів. Тому маємо два варіанти:

$$\mu_u = \frac{1}{|V_{train} \cap V_u|} \sum_{v_{ui} \in V_{train} \cap V_u} v_{ui},$$
$$\sigma_u = \sum_{v_{ui} \in V_{train} \cap V_u} (v_{ui} - \mu)^2,$$
$$\forall i, u \; p_{ui} \in N(\mu_u, \sigma_u) \tag{4.2}$$

та

$$\mu_i = \frac{1}{|V_{train} \cap V_i|} \sum_{v_{ui} \in V_{train} \cap V_i} v_{ui},$$
$$\sigma_i = \sum_{v_{ui} \in V_{train} \cap V_i} (v_{ui} - \mu)^2,$$
$$\forall i, u \; p_{ui} \in N(\mu_i, \sigma_i). \tag{4.3}$$

У першому випадку розподіл оцінюється за вибіркою фільмів, які подивився користувач. Така оцінка мала б виходити з припущення про те,

що кожен користувач оцінює усі фільми так, що його потенційна оцінка є нормальною випадковою величиною. На практиці таке припущення майже не виконується, однак підхід оцінювання параметрів для кожного користувача дозволяє зробити передбачення, яке було б не зміщене, у статистичному сенсі, відносно середньої оцінки користувача.

У другому випадку оцінки робляться відносно окремих фільмів. Прив'язки до смаків користувачів у цьому випадку вже немає. Логіка такого підходу наступна: фільми бувають різні, як якісні, так і не дуже. Якщо фільм дійсно поганий, то його середня оцінка буде нижче середньої серед усіх, а всі користувачі оцінюватимуть фільм відповідно до його якості. Тобто намагаємось знайти найближчу оцінку до параметрів кожного фільма.

Порівнюючи усі три підходи за декількома з вищезазначених характеристик, отримуємо порівняльну таблицю 4.1:

Таблиця 4.1 — Порівняння випадкових фільтрів

	MAE	RMSE	RMSLE	Precision@10	F1
Звичайний алгоритм	1.089	1.367	0.355	0.668	0.764
На оцінках користувача	0.962	1.233	0.325	0.669	0.774
На оцінках фільма	0.959	1.224	0.322	0.725	0.809

Незважаючи на більш кращі характеристики запропонованих альтернатив, далі використовуватиметься саме класичний варіант.

Час, витрачений на оцінку вхідних даних, створення рекомендацій та загальний час роботи алгоритму зазначено на графіку нижче. Тут L — частка вхідних даних, які належать до перевірочного набору. Якщо L дорівнює 0.1 то частка тренувального набору складає 90% від усіх даних, частка перевірочного — 10%. Якщо L дорівнює 0.2, — частка тренувального набору складе 80% від усіх даних, частка перевірочного 20% і т.д.. Розподіл часу випадкового фільтра зображено на рисунку 4.1.

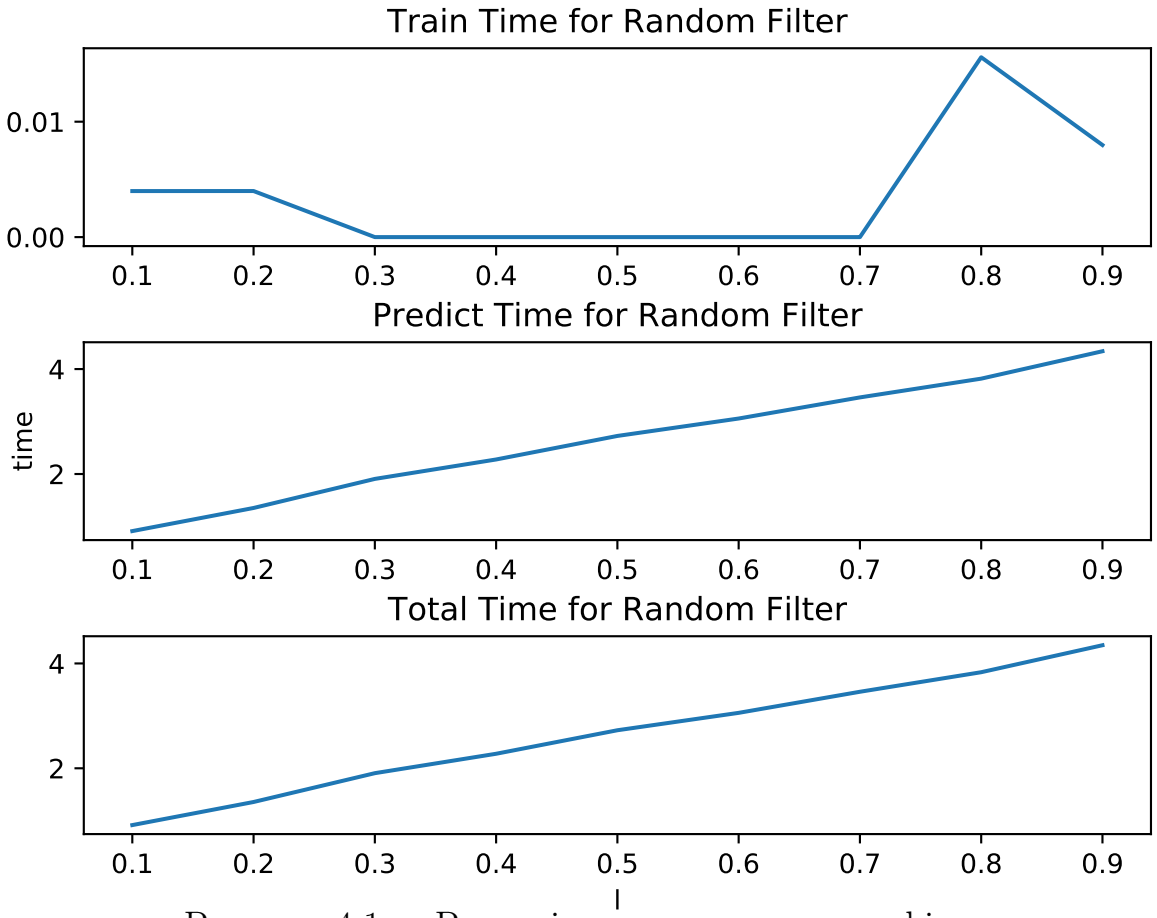

Рисунок 4.1 — Розподіл часу випадкового фільтра

4.1.2 Простий user-based фільтр

Насправді простим, у прямому сенсі цього слова, алгоритм не є. Більш справедливою була б назва прямий user-based фільтр, адже він повністю складається з кроків, описаних у теоретичній частині user-based рекомендаційних систем, без намагань спростити або прискорити обчислення. Однак, більш популярною в іноземній літературі є саме назва «простий user-based фільтр», тому її і використовуватимемо. Ані в реалізації, ані в просторовій/часовій складності цей алгоритм простим не є. Як це вже зазначалось, ідея полягає у відборі найкращих сусідів з подальшим використанням цих відомостей для побудови рейтингів. Практично цей алгоритм працює наступним чином.

На першому кроці будується матриця, приклад якої наведено у

таблиці 4.2.

Таблиця 4.2 — Матриця подібності користувачів

	u_1	u_2	...	u_n
u_1	1	$s(u_1, u_2)$	...	$s(u_1, u_n)$
u_2	$s(u_2, u_1)$	1	...	$s(u_2, u_n)$
...	...	...	...	...
u_n	$s(u_n, u_1)$	$s(u_n, u_2)$	...	1

де $s(u_i, u_j)$ – подібність, або схожість користувача u_i та u_j. Вже на цьому етапі виникають питання щодо вибору функції подібності. Насправді це є один з аспектів алгоритму, змінюючи який, можливо покращити результати.

Далі, користуючись матрицею подібності для кожного користувача, будується набір найближчих сусідів.

$$N_u = \{u_{\beta_1}, u_{\beta_2}, \ldots, u_{\beta_k}\},$$

де $u_{\beta_1}, u_{\beta_2}, \ldots, u_{\beta_k}$- – впорядковані за спаданням значення;

$s(u, u_{\beta_j}),\ j = 1, k$ — користувачі.

Тут виникає ще один параметр, який можливо обирати — кількість найближчих сусідів, які враховуються при побудові рейтинга.

Далі оцінки будуються наступним чином:

$$\forall i, u\ p_{ui} = \frac{1}{k} \sum_{u_{\beta_j} \in N_u} v_{u_{\beta_j} i}. \tag{4.4}$$

Питання, які залишаються без ґрунтовних відповідей – яким саме чином вибрати функцію подібності та параметр k. Існують роботи, у яких доведено, що саме подібність косинуса є найбільш придатною саме для цієї системи. Однак, варто перевірити це твердження ще й на практиці. Оцінки якості алгоритму при різних налаштуваннях наведено у таблиці 4.3.

Таблиця 4.3 — Якість простого user-based фільтра в залежності від функції подібності

	MAE	RMSE	RMSLE	Precision@10	F1
Cosine	0.809	1.067	0.283	0.739	0.840
Pearson	0.829	1.084	0.289	0.737	0.839
MSD	0.838	1.096	0.293	0.736	0.836

Можливо переконатись, що подібність косинуса дійсно є найкращою, хоча різниця не дуже значна. Вибір параметра k складніше оцінити. Окрім якості роботи алгоритму від нього також залежить швидкість роботи. У наступній таблиці наведено три різних значення k та характеристик систем в залежності від них. Оцінки якості алгоритму при різних налаштуваннях наведено у таблиці 4.4.

Таблиця 4.4 — Якість простого user-based фільтра в залежності від кількості «сусідів»

	MAE	RMSE	RMSLE	Precision@10	F1
5	0.861	1.114	0.296	0.721	0.834
30	0.812	1.070	0.284	0.743	0.843
100	0.759	0.996	0.266	0.782	0.862

З таблиці видно, що зі зростанням k покращується якість наданих рейтингів, однак при великих значеннях k система працює дуже довго. На рисунку 4.2 зображено час, який був витрачений рекомендаційною системою з k рівним в залежності від L.

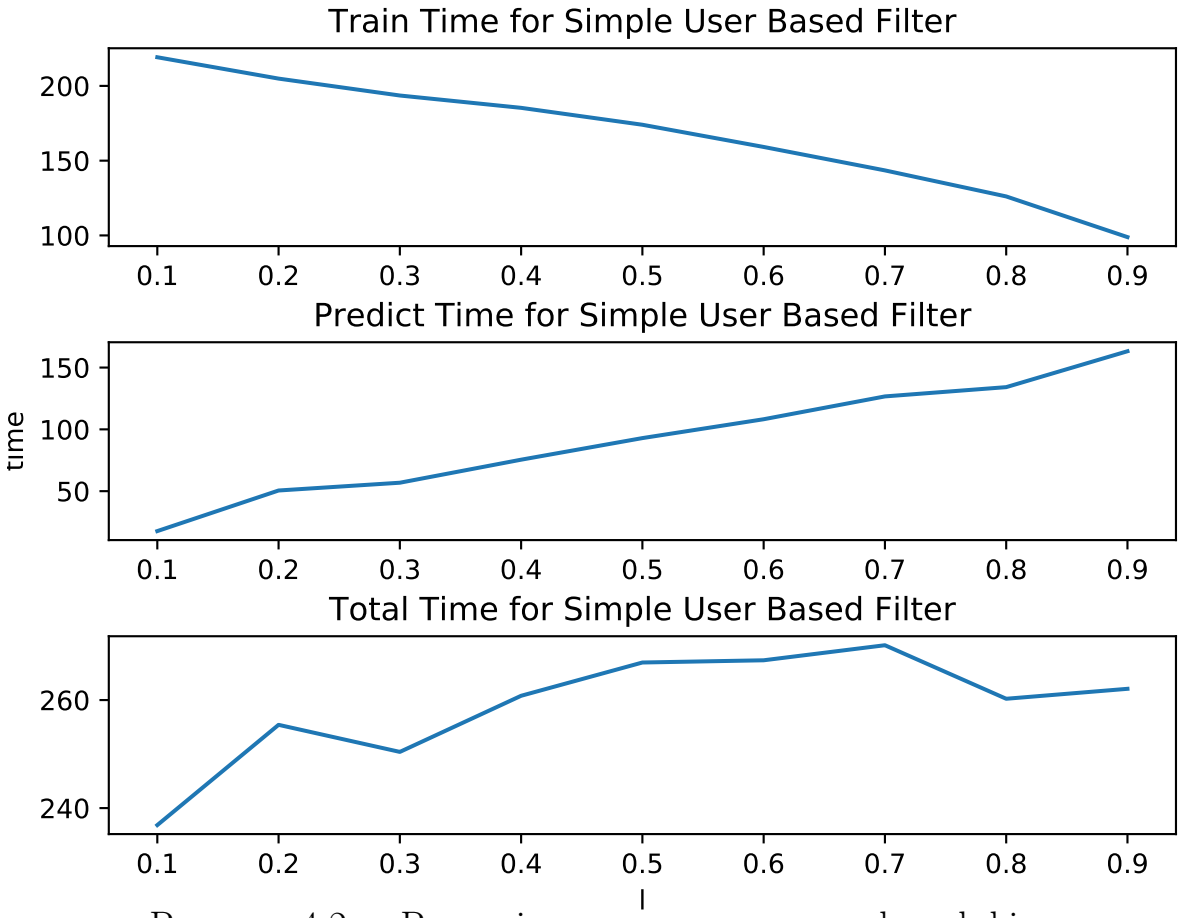

Рисунок 4.2 — Розподіл часу простого user-based фільтра

4.1.3 Простий item-based фільтр

Також як і простий user-based фільтр, «простим» він не є. Фактично такий алгоритм є транспонованим алгоритмом user-based фільтрації. На першому кроці будується матриця, приклад якої оформлено у таблиці 4.5:

Таблиця 4.5 — Матриця подібності фільмів

	i_1	i_2	...	i_m
i_1	1	$s(i_1, i_2)$	...	$s(i_1, i_m)$
i_2	$s(i_2, i_1)$	1	...	$s(i_2, i_m)$
...	...	...	...	...
i_m	$s(i_m, i_1)$	$s(i_m, i_2)$	...	1

де $s(i_i, i_j)$ – подібність, або схожість фільма i_i та i_j.

Функції подібності у цьому алгоритмі використовуються такі самі. Однак як і у минулому варіанті, тут найкращою буде саме косинус-подібність.

Далі, користуючись матрицею подібності для кожного фільму, будується набір найближчих сусідів:

$$N_i = \{i_{\beta_1}, i_{\beta_2}, \ldots, i_{\beta_k}\}. \tag{4.5}$$

де $i_{\beta_1}, i_{\beta_2}, \ldots, i_{\beta_k}$ — впорядковані за спаданням значень $s(i, i_{\beta_j})$, $j = 1, k$ фільми.

Так само як і у минулому алгоритмі, параметр k є невідомим і визначається вручну.

Далі, оцінки будуються наступним чином:

$$\forall i, u \; p_{ui} = \frac{1}{k} \sum_{i_{\beta_j} \in N_i} v_{u, i_{\beta_j}} \tag{4.6}$$

Залишається оцінити лише якість системи в залежності від k. В датасеті фільмів набагато більше, ніж користувачів. Тому й k у цьому випадку варто брати більшим. Оцінки якості алгоритму при різних налаштуваннях наведено у таблиці 4.6.

Таблиця 4.6 — Якість простого item-based фільтра в залежності від кількості «сусідів»

	MAE	RMSE	RMSLE	Precision@10	F1
100	0.821	1.068	0.281	0.672	0.798
400	0.774	1.005	0.265	0.700	0.799
800	0.754	0.975	0.259	0.714	0.801

Звідси видно, що різниця у якості між останніми двома варіаціями дуже незначна. При цьому час, який витрачається на побудову рекомендацій був у останньому випадку занадто великим.

Взагалі, ця рекомендаційна система виявилась найбільш повільною. Інформація щодо витрат часу при k рівному 400 відображена на рисунку 4.3.

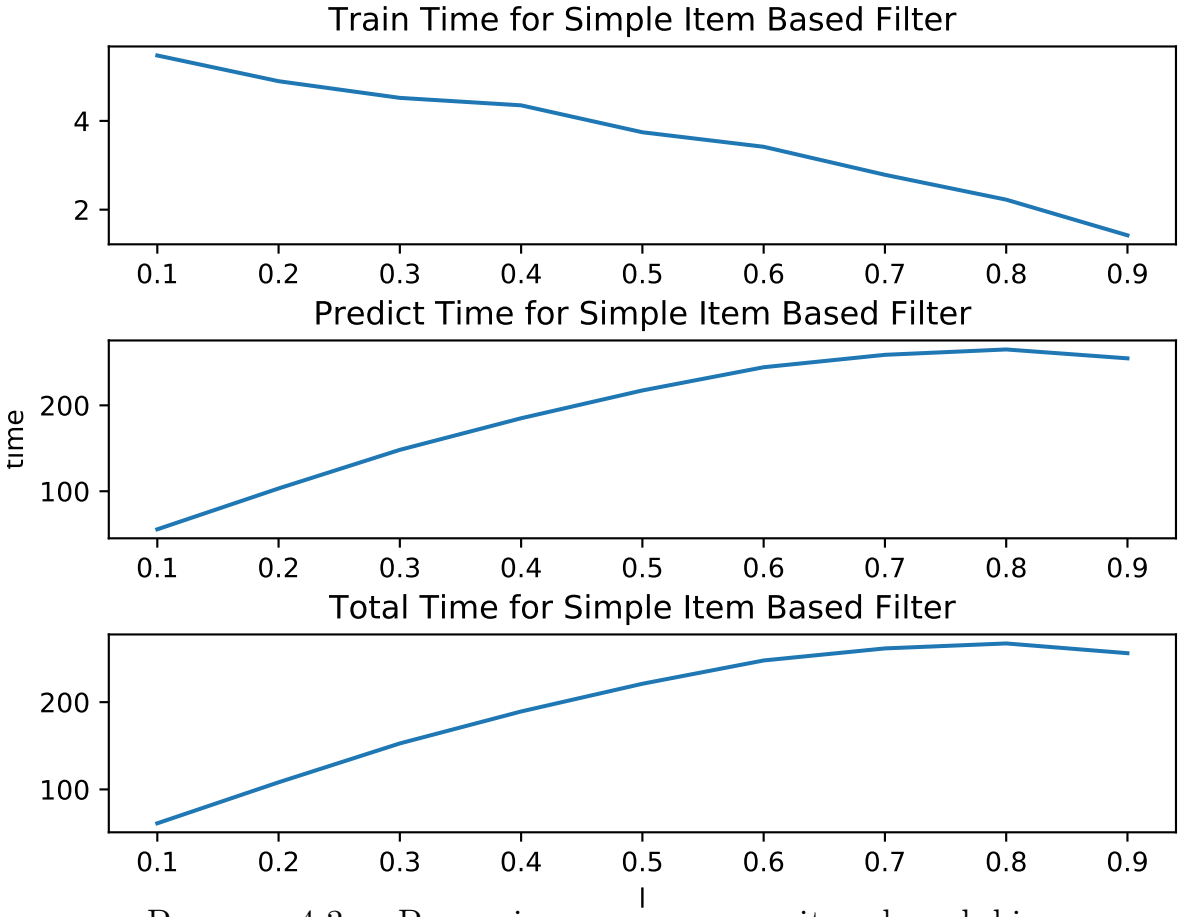

Рисунок 4.3 — Розподіл часу простого item-based фільтра

4.1.4 SVD фільтр

SVD фільтрація – один з дуже популярних прикладів рекомендаційних систем, які вживають факторизацію матриць задля покращення та прискорення обчислень. Існує сім'я подібних до SVD алгоритмів, як, наприклад, SVD++ або NMF, однак класичний SVD фільтр є найбільш відомим. Власне SVD означає singular value decomposition, що у перекладі на українську означає «сингулярний розклад матриці».

Сингулярний розклад матриці M – розмірності nxm становить собою

розклад цієї матриці на три інших, а саме: U – розмірності nxn, Σ – розмірності nxm, та V – розмірності mxm. Так

$$M = U\Sigma V \tag{4.7}$$

У рекомендаційних системах цей розклад прийнято позначати як

$$M = RV, \tag{4.8}$$

де R позначає добуток U та Σ.

При цьому, основною властивістю SVD розкладу є те, що розмірність Σ можливо зменшити до рангу матриці M, не втрачаючи інформації.

У реалізованому алгоритмі впроваджено відсікання до k елементів, тобто в результаті відсікання матриці мають наступну розмірність: U – розмірності nxk, Σ – розмірності kxk та V – розмірності kxm.

Сенс цих відсікань полягає у виборі деякої кількості найважливіших характеристик, збільшення швидкості обчислень та зменшення похибок, які виникають через неповноту даних. У нашому випадку, перед використанням SVD розкладу від усіх рейтингів віднімається їх середнє значення, а усім рейтингам, яких ще немає, присвоюється значення 0. Для побудови рекомендацій для відповідних значень з матриці, отриманої перемноженням результатів розкладу матриць, з яких відсікли непотрібні рядки та стовпці, в кінці додається усереднена оцінка рейтингів. Такі перетворення мають зменшити вплив окремих користувачів та розрізняти усіх між собою, а вплив відсутніх рейтингів повністю відсікти. Ця варіація SVD рекомендаційної системи, напевно, є не найбільш якісною, адже деякі тонкощі SVD розкладу не враховано. Втім цей алгоритм дає уяву про загальний устрій алгоритмів побудованих на факторизації матриць. Так, в цьому алгоритмі визначено параметр k, який дозволяє оцінити вплив вище зазначених відсікань. Розглянемо три різних результати, наведених у таблиці 4.7:

Таблиця 4.7 — Якість SVD фільтра в залежності від k

	MAE	RMSE	RMSLE	Precision@10	F1
10	1.015	1.285	0.336	0.680	0.753
30	1.018	1.288	0.337	0.682	0.754
100	1.020	1.293	0.338	0.675	0.750

Маємо, що загальна якість алгоритму спадає зі збільшенням k. Однак, при $k = 30$ статистика F1 та точність дещо збільшились. Алгоритм SVD фільтрації виявився дуже швидким відносно інших алгоритмів, однак його якість найкращою не була. На рисунку 4.4 зображено витрати часу алгоритма SVD фільтрації.

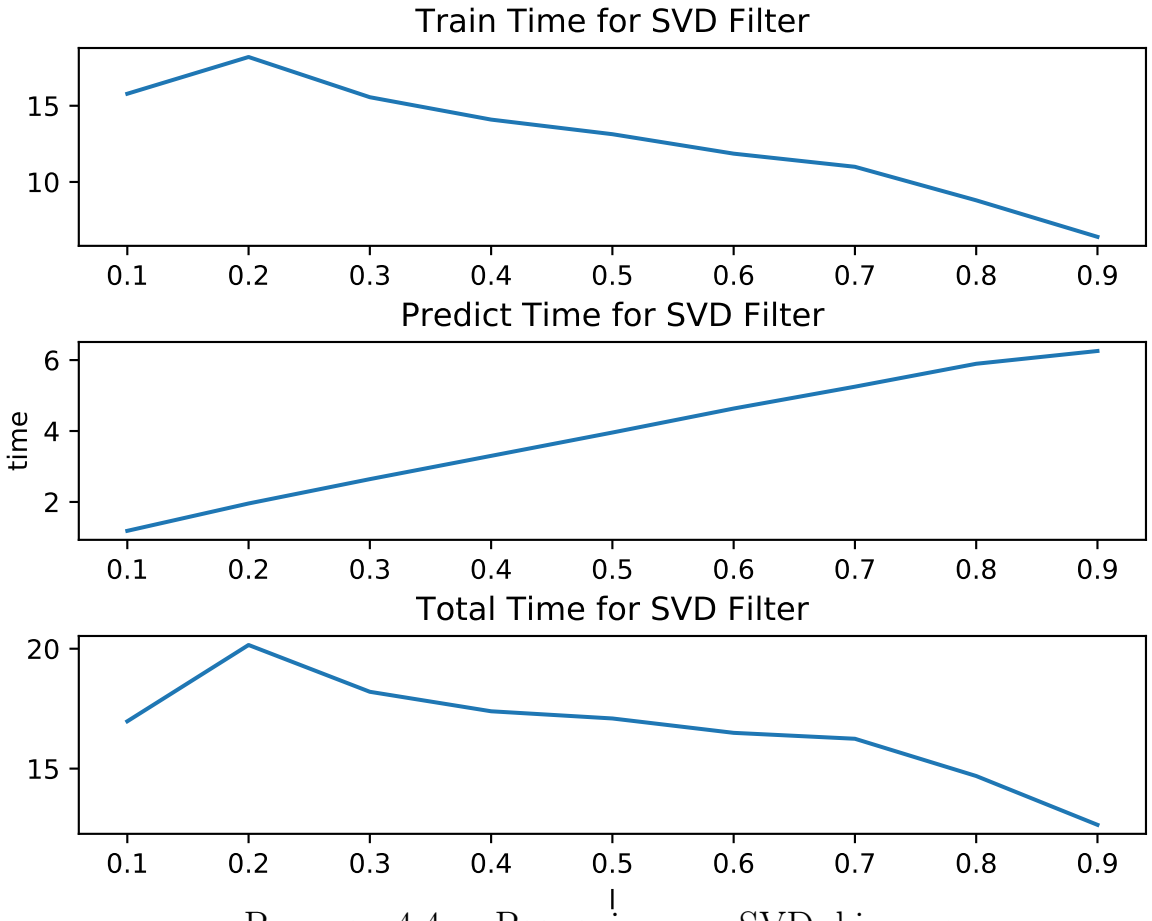

Рисунок 4.4 — Розподіл часу SVD фільтра

4.1.5 Slope One алгоритм

Задля економії часу, для реалізації алгоритму Slope One було використано бібліотеку Surprise, яка містить готові методи тренування та побудови прогнозів.

Ідейно алгоритм Slope One є хіба не найпростішим, хоча за якістю він випереджає велику кількість алгоритмів, які значно складніші від нього. Окрім цього, через виконання більшості операцій з використанням бібліотеки Surprise, яка реалізує більш тонкі та швидкі конструкції, аніж ті, які доступні у Python, Slope One виявився в додачу ще й більш швидким. Оцінка за алгоритмом Slope One рахується наступним чином:

$$p_{ui} = \mu_u + \frac{1}{|R_i(u)|} \sum_{j \in R_i(u)} dev(i,j), \tag{4.9}$$

де $R_i(u)$ – набір релевантних фільмів, таких, що рекомендовані u, та має користувачі i, який їх також оцінив;

μ_u – середня оцінка відповідного користувача.

Аналогічно подібності у інших рекомендаційних системах, Slope One розраховує свою користуючись функцією $dev(i,j)$ яка визначена, фактично, як усереднена різність між об'єктами. Тобто:

$$dev(i,j) = \frac{1}{|U_{ij}|} \sum_{u \in U_{ij} v_u i - v_u j}, \tag{4.10}$$

де U_{ij} — набір користувачів, які оцінили об'єкт i та j одночасно.

Алгоритм виявився дуже швидким та, до цього, ще й дуже точним. Інформація щодо розподілу часу, витраченого алгоритмом, знаходиться на рисунку 4.5.

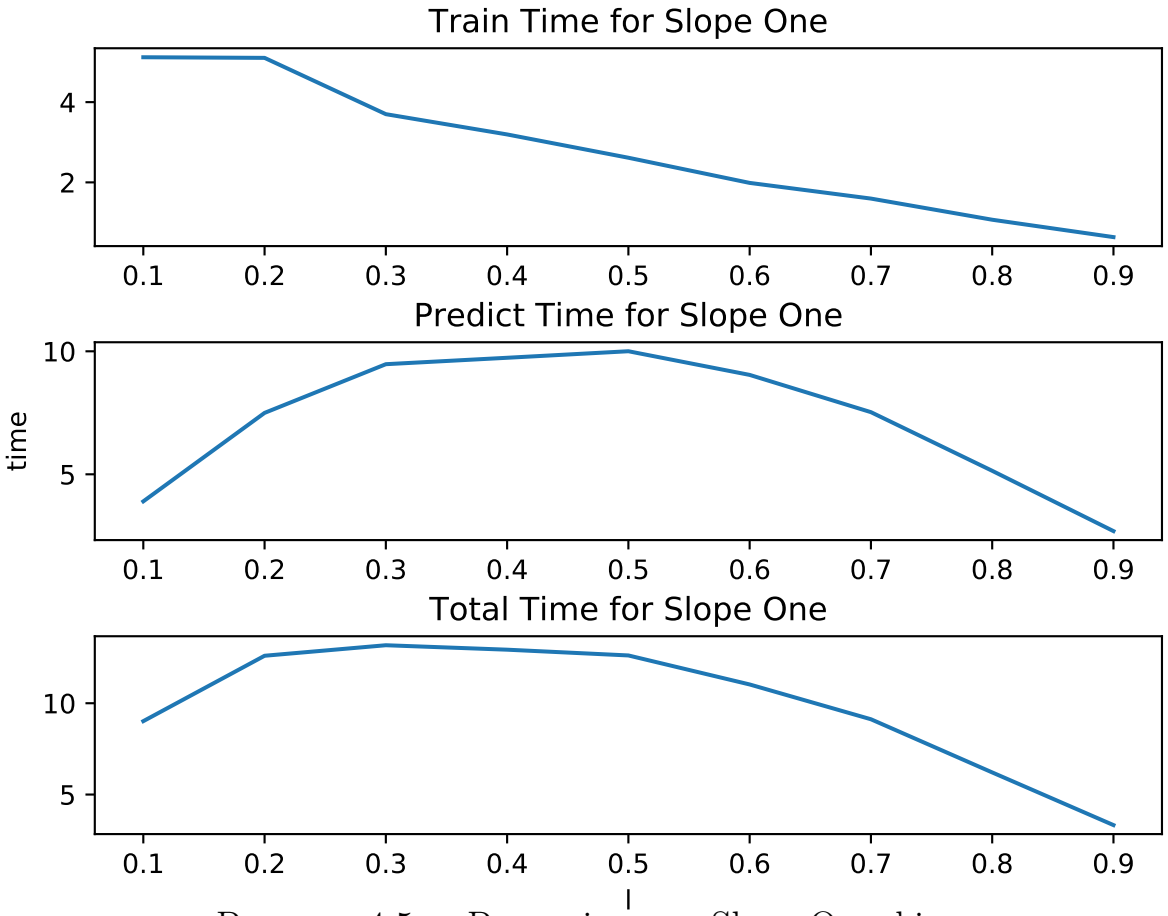

Рисунок 4.5 — Розподіл часу Slope One фільтра

4.2 Порівняння алгоритмів

Використовуючи розроблений аппарат оцінювання рекомендаційних систем, пропонується порівняти наведені вище рекомендаційні системи. Для цього було побудовано ряд тестів, які дозволяють оцінити та порівняти рекомендаційні системи між собою.

Через те, що кожен окремий тест складається з ряду запусків кожного алгоритму, та подальшої обробки результатів, побудова наведених далі результатів порівнянь потребувала значних обчислень та ресурсів, однак повторна перевірка, яка у цьому документі не наводиться, дала майже ті ж результати, що дозволяє сприймати нижче наведені результати як істинні. Порядок розгляду оцінок проведено відповідно до порядку теоретичної бази, наведеної в минулих розділах.

4.2.1 Витрати часу

Витрати часу є дуже важливою характеристикою рекомендаційних систем, адже доволі часто вони мають працювати у стаціонарному режимі, постійно обробляючи нову інформацію. Через це особливу увагу приділяють найбільш швидким алгоритмам, навіть якщо це шкодить якості. Також варто зазначити, що швидкість роботи наведених у цій роботі рекомендаційних систем зумовлена не лише алгоритмом їх функціонування, а й реалізацією. Усі 5 рекомендаційних систем у цій роботі були побудовані, використовуючи мову розробки Python та використовуючи бібліотеки обробки даних та математичні пакети, частина з яких створена на мові програмування C. Це призвело до того, що деякі алгоритми, які базувались на синтаксисі більш низького рівня та використовували окремі функції розроблені на C, працювали значно швидше. Так, наприклад, алгоритм Slope One виявився найбільш швидким через те, що був повністю реалізований, використовуючи низькорівневі обчислення. Усі інші алгоритми базувались на приблизно однаковому синтаксисі, однак вже за своєю суттю були доволі розбіжні.

Так, випадковий фільтр виявився дуже швидким через те, що майже не залежав від розміру вибірки, не потребував якихось важких обчислень для окремих користувачів, тощо.

Алгоритм SVD також виявився дуже швидким тому, що факторизація матриць виявилась більш швидкою у порівнянні з побудовою матриць подібностей. Варто також зазначити, що у випадку SVD фільтрації необхідно було побудувати одну матрицю та потім правильно обрати її елементи, у випадку «звичайних» алгоритмів та Slope One розрахунки більш складні. Для розгляду пропонуються порівняльні графіки залежностей часу роботи алгоритмів фільтрації на різних етапах, в залежності від розміру тестувальної вибірки. Час на навчання наведено на рисунку 4.6.

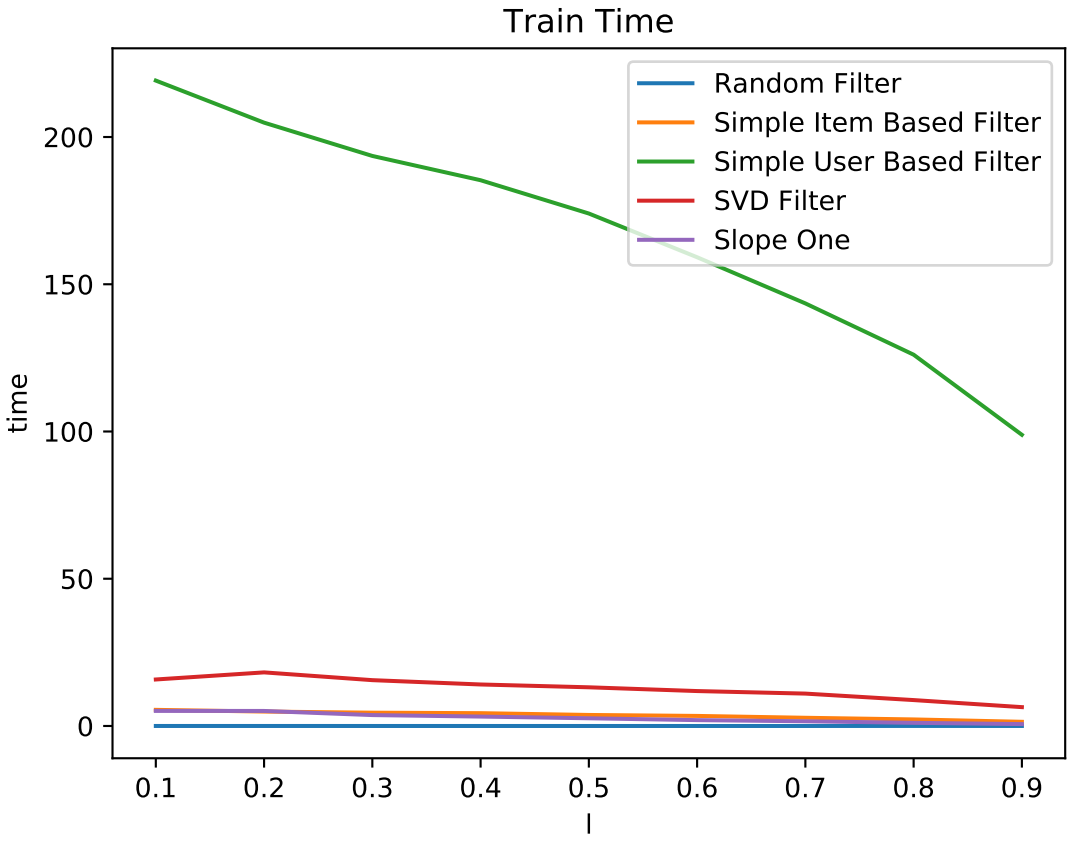

Рисунок 4.6 — Час на навчання усіх алгоритмів

Усі алгоритми, окрім випадкового фільтру, зі зменшенням розміру навчальної вибірки працювали швидше, що є закономірним.

Час на побудову рекомендацій наведено на рисунку 4.7.

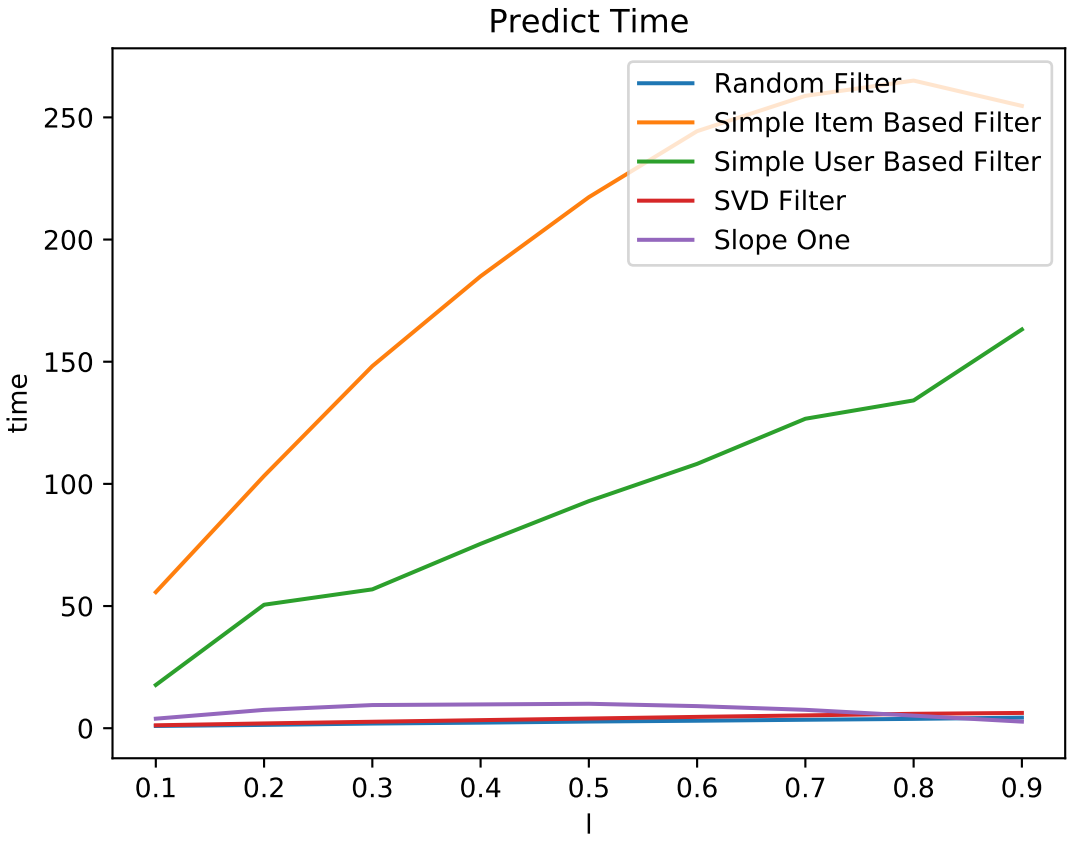

Рисунок 4.7 — Час на побудову рекомендацій усіх алгоритмів

З цього графіку випливає, що процес побудови рекомендацій за фільмами виявився значно складнішим, аніж побудова рекомендацій за користувачами. Дійсно, будувати рекомендації для 610 користувачів окремо виявляється менш складним завданням, ніж для кожного окремого з 610 користувачів робити якісь додаткові обчислення, використовуючи для цього базу фільмів, яких у цьому датасеті майже 9000. Загальні витрати часу наведено на рисунку 4.8.

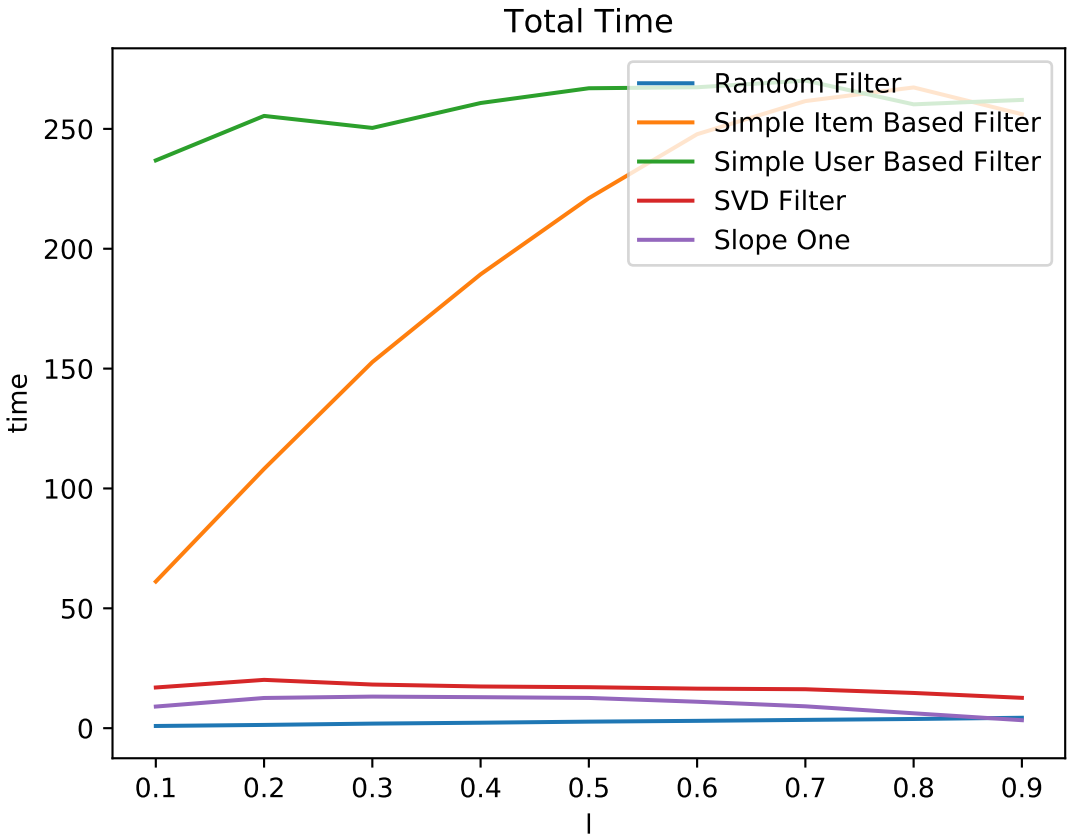

Рисунок 4.8 — Загальний час роботи усіх алгоритмів

Загальний розподіл часу. У більшості алгоритмів час необхідний для побудови рекомендацій за складністю виявлявся приблизно еквівалентним, однак для Item-based фільтра розподіл часу виявився набагато більш складним.

4.2.2 Оцінки точності

Наступні оцінки також є дуже важливими при оцінці якості рекомендаційних систем. Перші три – класичні оцінки похибки передбачень, які одна від одної відрізняються лише абсолютними значеннями, у той самий час, відношення якості алгоритмів в залежності від оцінки майже не змінюється, у чому є сенс. Усі оцінки також проведені на тренувальних вибірках різних розмірів, що також дозволяє оцінити якість алгоритму в залежності від «насиченості» інформацією. Середня

абсолютна помилка різних алгоритмів зазначена на рисунку 4.9.

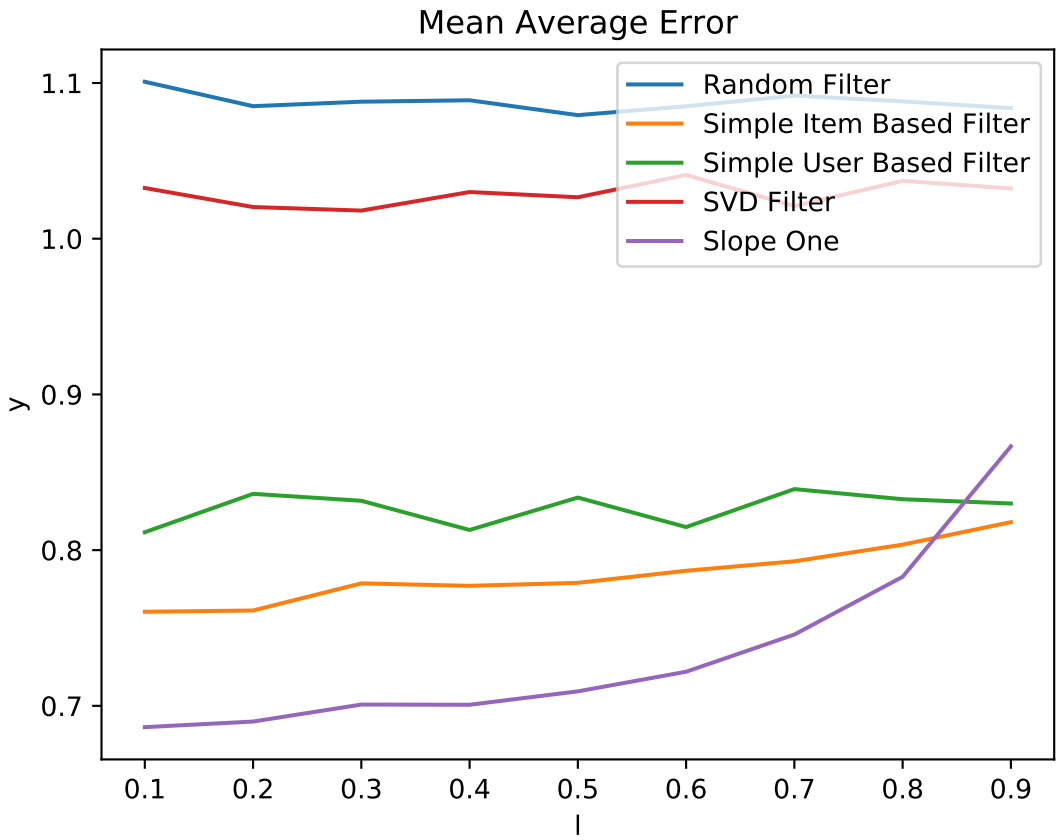

Рисунок 4.9 — Середня абсолютна похибка алгоритмів в залежності від частки тестувальних даних

Класична похибка. З цього рисунку видно, що найгіршим алгоритмом виявився випадковий фільтр. Його якість майже не залежить від змінної L (частка тестувальних даних від усього набору), SVD також виявився не дуже хорошим, його оцінка також значно не змінювалась, в залежності від кількості вхідних даних. Усі інші алгоритмі зі зменшенням інформації демонстрували гірші результати.

Усереднена квадратична помилка різних алгоритмів зазначена на рисунку 4.10.

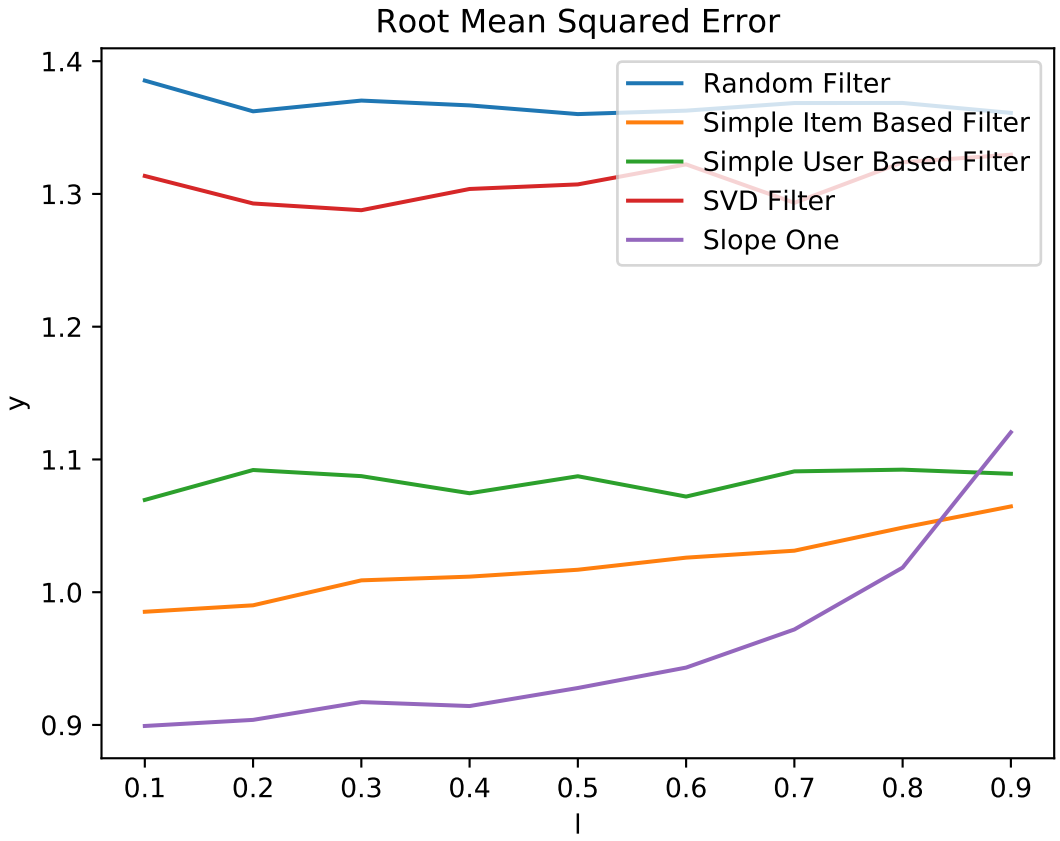

Рисунок 4.10 — Усереднена квадратична помилка алгоритмів в залежності від частки тестувальних даних

За цією оцінкою відношення алгоритмів між собою також не змінилось. User-based фільтр виявився трохи гіршим за Item-based, можливо через те, що пошук подібних об'єктів у Item-based фільтрі проводився за більшою вибіркою (фільмів більше ніж користувачів), однак, це зашкодило часу роботи алгоритму.

Усереднена квадратична логарифмована помилка різних алгоритмів зазначена на рисунку 4.11.

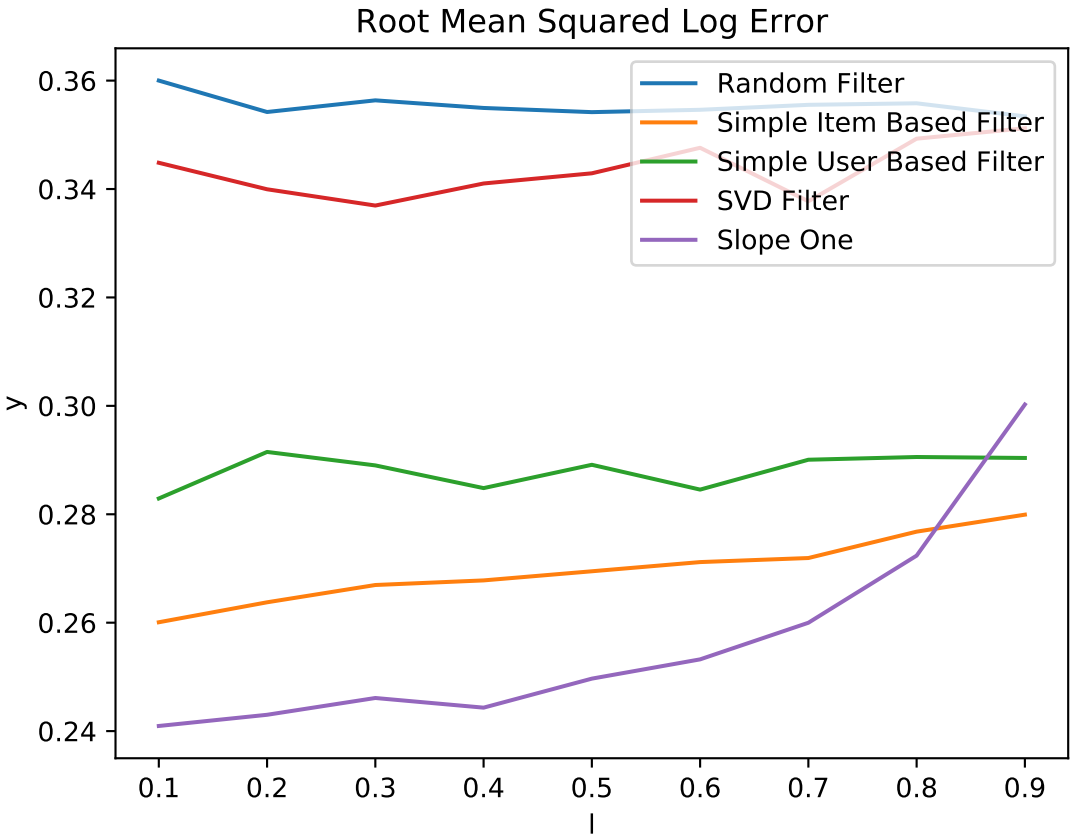

Рисунок 4.11 — Усереднена квадратична логарифмована помилка алгоритмів в залежності від частки тестувальних даних

З цієї статистики також випливає, що алгоритм Slope One виявився найкращим, однак як це можливо помітити, його якість погіршується швидше усіх, причому з цього графіку можливо зробити висновок, що йдеться не про якийсь постійний зсув, а саме про загальні помилки.

Оцінка точності різних алгоритмів зазначена на рисунку 4.12.

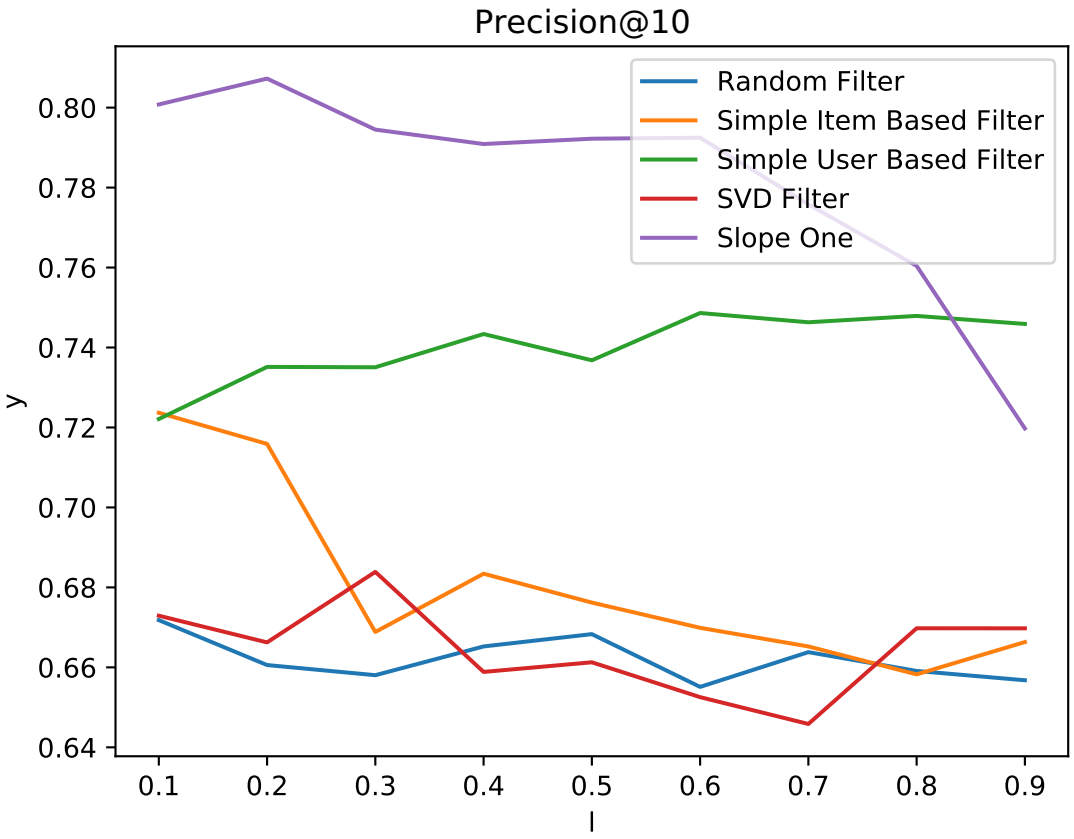

Рисунок 4.12 — Точність за десятьма найкращими рекомендаціями алгоритмів в залежності від частки тестувальних даних

Цей рисунок підтверджує висновки минулих трьох, однак з нього також слідує, що зі зменшенням інформації, яка надається на вхід рекомендаційний системі, майже всі фільтри втрачають можливість рекомендувати сталу кількість фільмів. User-based фільтр виявився найбільш стійким, та взагалі по вибірці з 10 фільмів виявився кращим за Item-based, який не тільки показав гірші результати, а ше й втрачав якість зі зменшенням тренувальної вибірки.

Оцінка F1 різних алгоритмів зображена на рисунку 4.13.

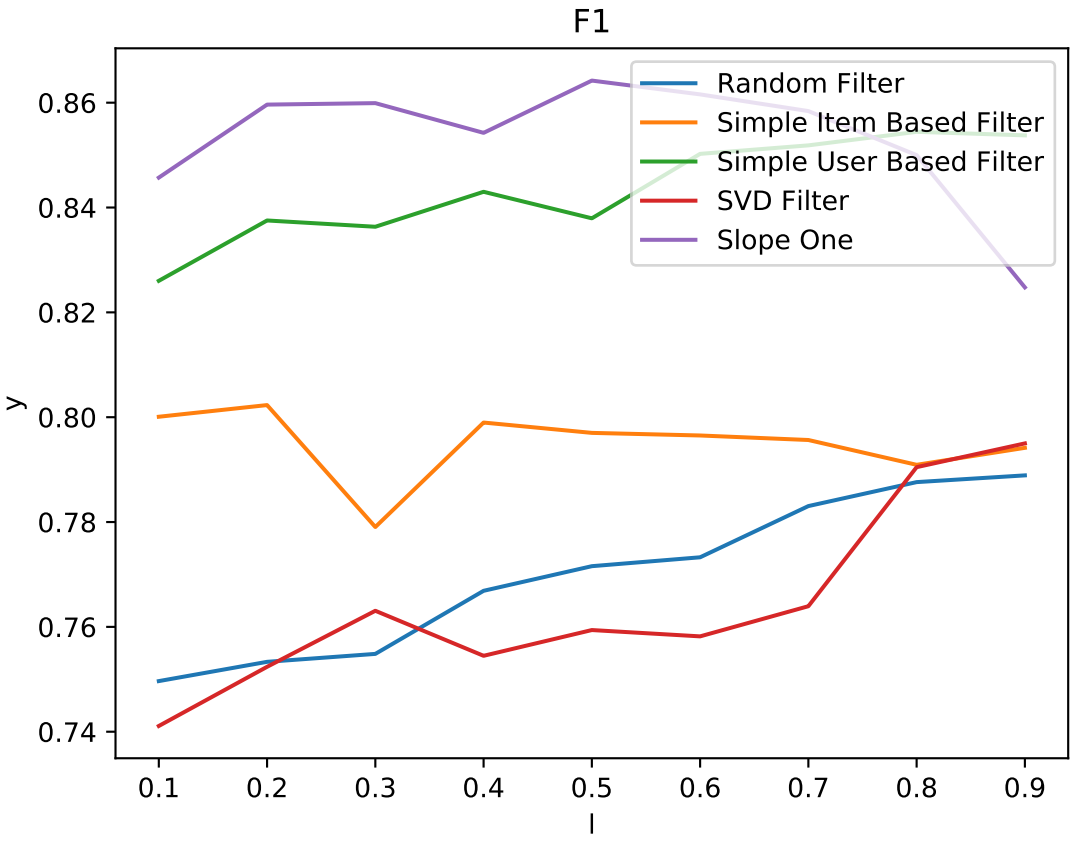

Рисунок 4.13 — Значення F1 алгоритмів в залежності від частки тестувальних даних

Статистика F1 використовується як загальна характеристика рекомендаційних систем, тому у цьому підрозділі вона є останньою, підсумовуючою. Варто зазначити, що через те, що F1 складається як з точності, так і повноти, характер поведінки F1 залежить від обох статистик. При цьому варто зазначити, що повнота, яка сама по собі не є інформативною, принаймні у цьому розділі, зростає зі зменшенням тренувальної вибірки, адже кількість фільмів які можливо рекомендувати, при цьому зростає.

За статистикою F1 найкращим виявився звичайний User-based фільтр. Алгоритм SVD подекуди виявився навіть гіршим за випадковий фільтр. Це спричинене тим, що повнота випадкового фільтра виявилась усюди більшою за повноту SVD фільтру, це обумовлене випадковою природою одного з фільтрів.

4.2.3 Точність та повнота

Графіки цього підрозділу призначені для порівняння усіх розглянутих алгоритмів, використовуючи при цьому одразу декілька показників та параметрів. Насправді такі оцінки є хіба не важливішими від ізольованих показників.

Перший графік, наведений на рисунку 4.14 – крива точності/повноти за змінним показником порогової (граничної) оцінки θ. Розмір вибірки було обрано рівним 10.

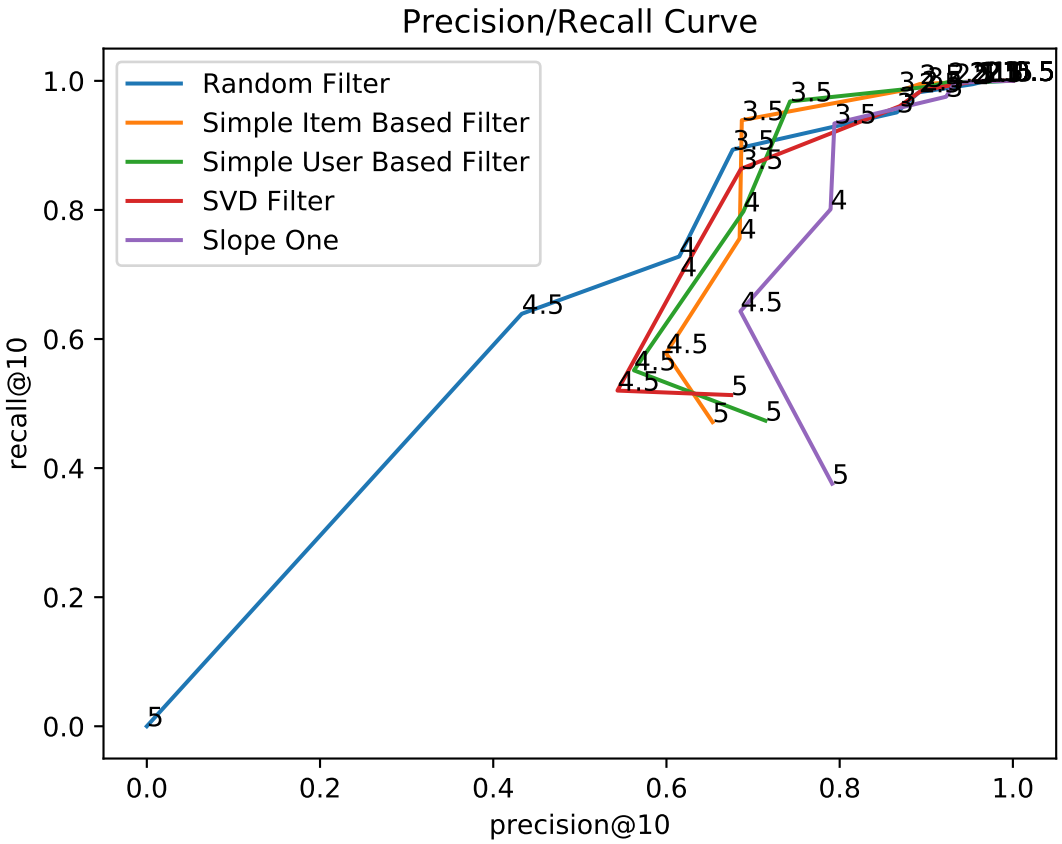

Рисунок 4.14 — Криві точності/повноти для усіх алгоритмів в залежності від граничної оцінки θ

Цей графік можливо розуміти як залежність якості системи від розуміння користувачів про «прийнятність» фільмів. Тобто, за зменшенням порогової оцінки якість систем має збільшуватись, адже при

оцінці фільмів з рейтингом більше нуля як хороших, будь-яка рекомендація виявляється вірною.

На цьому графіку найкращим положенням є правий верхній кут, система вважатиметься кращою за інші, якщо її показники знаходяться ближче до точки $(1, 1)$.

Отже, випадковий фільтр виявився найгіршим, з пороговою оцінкою рівною 5 він не зробив жодної вірної рекомендації. Зі зменшенням порогової оцінки його якість зростала, однак, в першу чергу за рахунок покриття, яке для випадкового фільтра має бути великим. Що стосується точності — для усіх значень порогової оцінки випадковий фільтр виявився найгірішим, адже усі значення знаходяться лівіше відповідних показників інших рекомендаційних систем. Серед усіх інших, напевно, найкращим виявився Slope One, хоча іноді його покриття виявилося не найкращим. User-based та Item-based фільтри виявились дуже схожими.

Наступна крива демонструє спроможність алгоритмів колаборативної фільтрації рекомендувати набір з декількох фільмів. Зрозуміло, що чим більше фільмів необхідно рекомендувати — тим гіршою буде оцінка по всій вибірці, адже рекомендувати 5 фільмів означає надати 5 найкращих, за оцінками рекомендаційної системи. Так як і на минулому графіку — найкращою точкою є позиція $(1, 1)$. Всі алгоритми здатні зробити принаймні одну успішну рекомендацію, однак на цій кривій зображено, в першу чергу, спроможність алгоритмів зберігати якість за збільшенням вибірки. Порогова оцінка була обрана рівною 3.5, та була незмінною для усіх алгоритмів.

Криві точності/повноти для усіх алгоритмів в залежності від розміру вибірки наведено на рисунку 4.15.

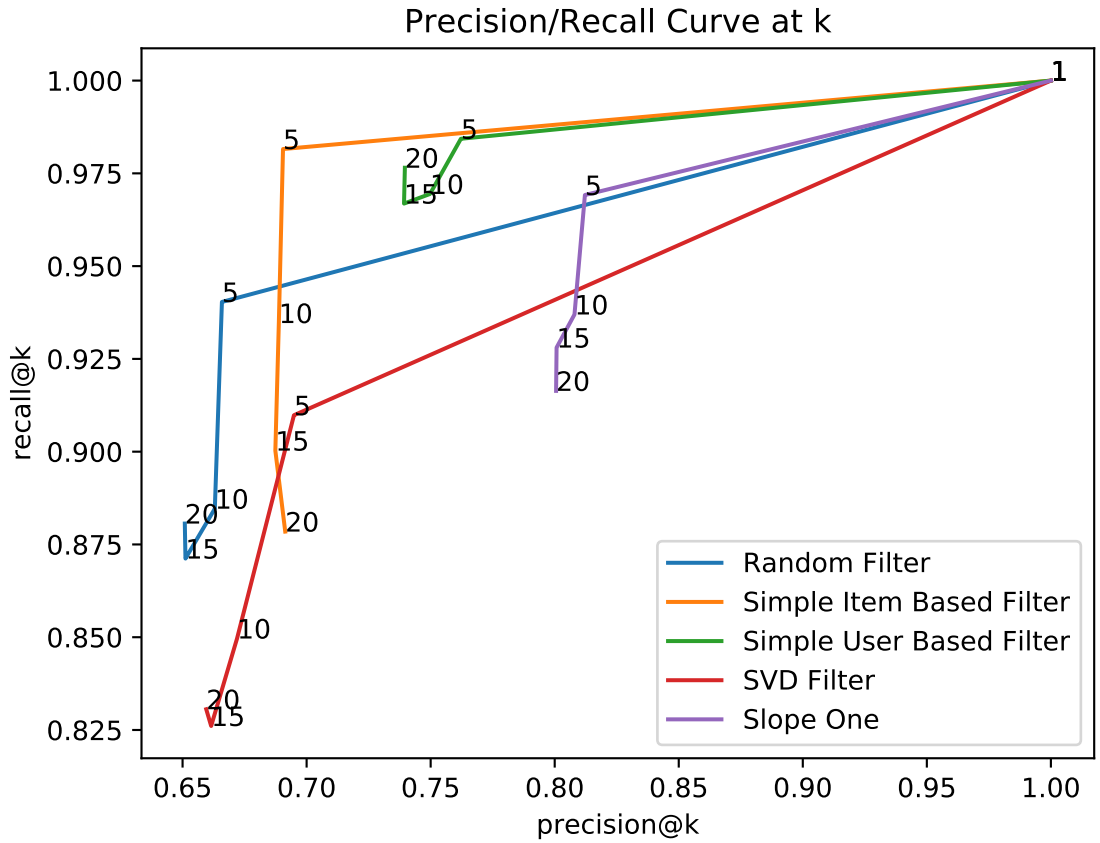

Рисунок 4.15 — Криві точності/повноти для усіх алгоритмів в залежності від розміру вибірки.

Найкращими виявились Slope One та простий User-based фільтр. Якщо перший зберігав точність, при цьому трохи втрачаючи повноту рекомендацій, то другий навпаки, трохи втрачаючи точність оцінок, зберігав майже сталою повноту рекомендацій. Усі інші виявились також доволі якісними. Наприклад, втрачаючи повноту, простий Item-based фільтр зберіг постійною точність оцінок. Найгіршим у цьому порівнянні виявився SVD фільтр.

4.3 Висновки до розділу 4

Рекомендаційні системи окремих видів дуже сильно розрізняються між собою. Окрім самого змісту алгоритму, який використовується, надважливим є також й процес його налаштування. У цьому розділі було

наведено 5 найбільш популярних, ключових алгоритмів колаборативної фільтрації, а також декілька варіацій деяких з них, які дозволяють робити більш якісні рекомендації.

Деякі з розглянутих рекомендаційних систем виявились значно кращими за інші, однак це не означає, що алгоритми, які у цій роботі поступились іншим, є застарілими або взагалі гіршими за інші. Більшість з наведених рекомендаційних систем мають покращені або доповнені версії, які можуть бути специфічними до якихось окремих галузей, або через якісь конкретні переваги є більш придатними до використання у окремих випадках.

5 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПАКЕТУ ДЛЯ АНАЛІЗУ ТА ПОРІВНЯННЯ РЕКОМЕНДАЦІЙНИХ СИСТЕМ

У даному розділі проводиться оцінка програмного продукту (ПП), призначеного для аналізу та порівняння рекомендаційних систем. Программу було створено, використовуючи мову програмування Python та такі бібліотеки як NumPy, SciPy, Pandas та Surprise.

5.1 Постановка задачі

Середовищем розробки було обрано редактор PyCharm, стандартне розширення .py дозволяє використовувати розроблений пакет на будь-якій системі, незалежно від операційної системи або бітності системи. Вищенаведені бібліотеки є відкритими для будь-якого виду використання, тому розроблений пакет не потребує додаткового ліцензування навіть при комерційному використанні.

Нижче наведено аналіз різних варіантів реалізації пакету з метою вибору оптимального, з огляду при цьому на економічні фактори та характеристики продукту, що впливають на продуктивність роботи і якість отриманого виробу. Для цього було використано апарат функціонально-вартісного аналізу.

5.2 Обґрунтування функцій та параметрів ПП

Основними функціями пакету є

а) F_1 – розробка та побудова деякого набору алгоритмів, з метою використання при тестуванні:

1) використання існуючих бібліотечних методів.
2) власна розробка алгоритмів.

б) F_2 – алгоритм підбору даних для тестування:
1) використання випадкової вибірки.
2) використання відомої частини вибірки.

в) F_3 – визначення метрик для тестування алгоритмів:
1) використання відомих, досліджених та реалізованих характеристик.
2) розробка власних або малодосліджених метрик.

У морфологічній карті наведено варіанти реалізації основних функцій. (рисунок 5.1.)

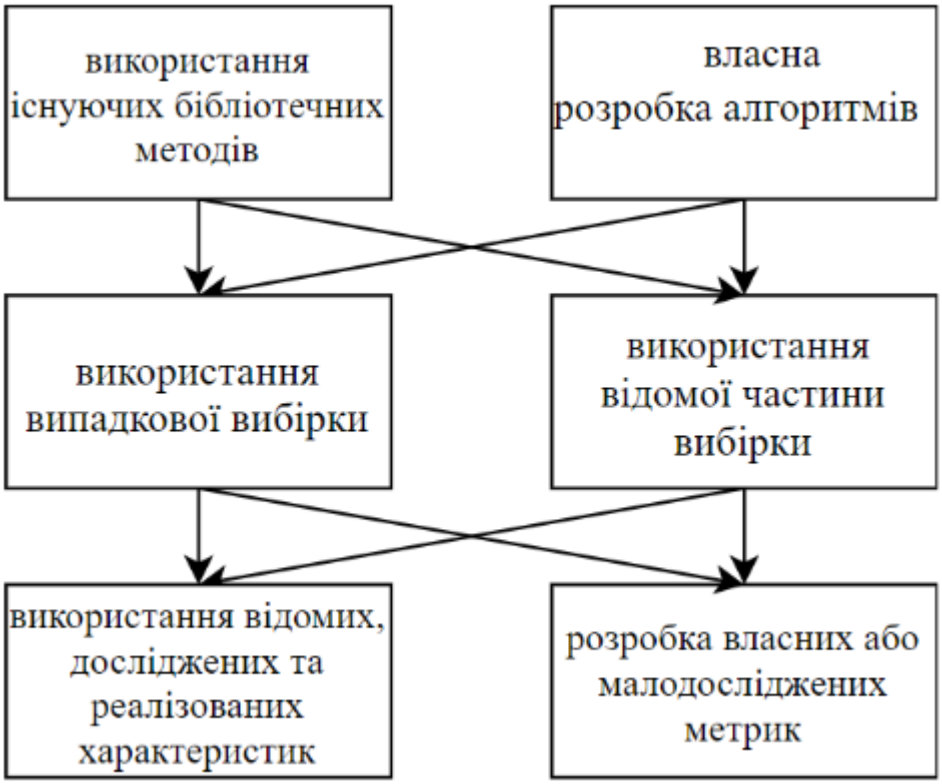

Рисунок 5.1 — Морфологічна карта

У таблиці 5.1 зображені недоліки та переваги кожного з можливих значень функцій.

Таблиця 5.1 — Переваги та недоліки варіантів реалізації

Основні функції	Варіанти реалізації	Переваги	Недоліки
F1	А	Швидкість розробки продукту	Відсутність контролю за роботою алгоритмів та можливості внесення правок
	Б	Можливість налаштовувати алгоритми під власні потреби	Час, необхідний на розробку продукту
F2	А	Більша якість тестів	Відсутність визначенності вихідних даних. Більш складний та клопітний алгоритм підбору даних
	Б	Контроль вихідних даних	Необхідність контролю вибору
F3	А	Наявність готового математичного апарату	Пристосованість конкретних алгоритмів до певних тестів
	Б	Побудова більш гнучких тестів	Час та складність побудови

За результатами аналізу позитивно-негативної матриці залишаємо 2 варіанти.

$$F_1\text{Б} - F_2\text{А} - F_3\text{А},$$

$$F_1\text{Б} - F_2\text{А} - F_3\text{Б}.$$

Для оцінювання якості розглянутих реалізацій обрані параметри, описані нижче. Основні параметри пакету, їх найкращі та найгірші оцінки наведено в таблиці.

5.3 Обгрунтування системи параметрів досліджень

У таблиці 5.2 зображені параметри можливих реалізацій.

Таблиця 5.2 — Параметри можливих реалізацій

Назва параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
час на побудову пакету	X1	год	50	25	15
час необхідний для ознайомлення з теорією	X2	год	120	80	40
кількість тестів, які можливо побудувати	X3	шт	1	5	15

За наведеними в таблиці кращими та гіршими значеннями будуються графічні характеристики параметрів (рисунок 5.2 для першого параметра, рисунок 5.3 для другого, та рисунок 5.4 для третього)

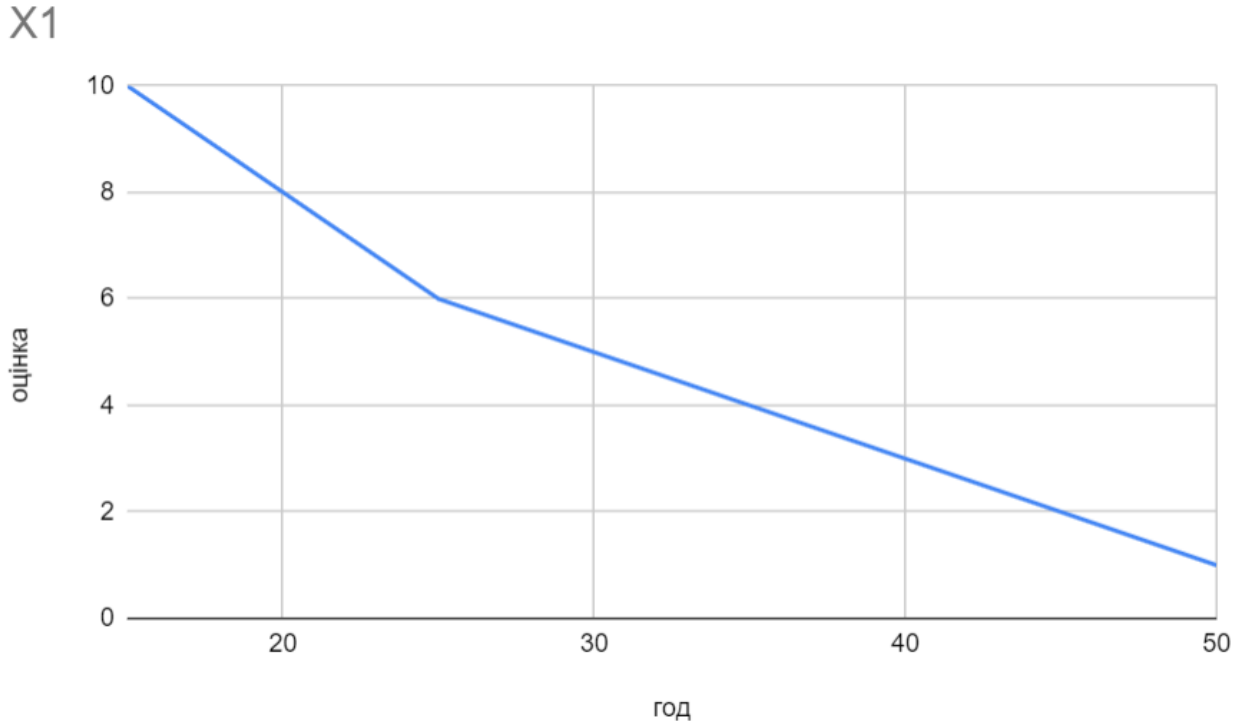

Рисунок 5.2 — Графічне зображення оцінок параметра X1

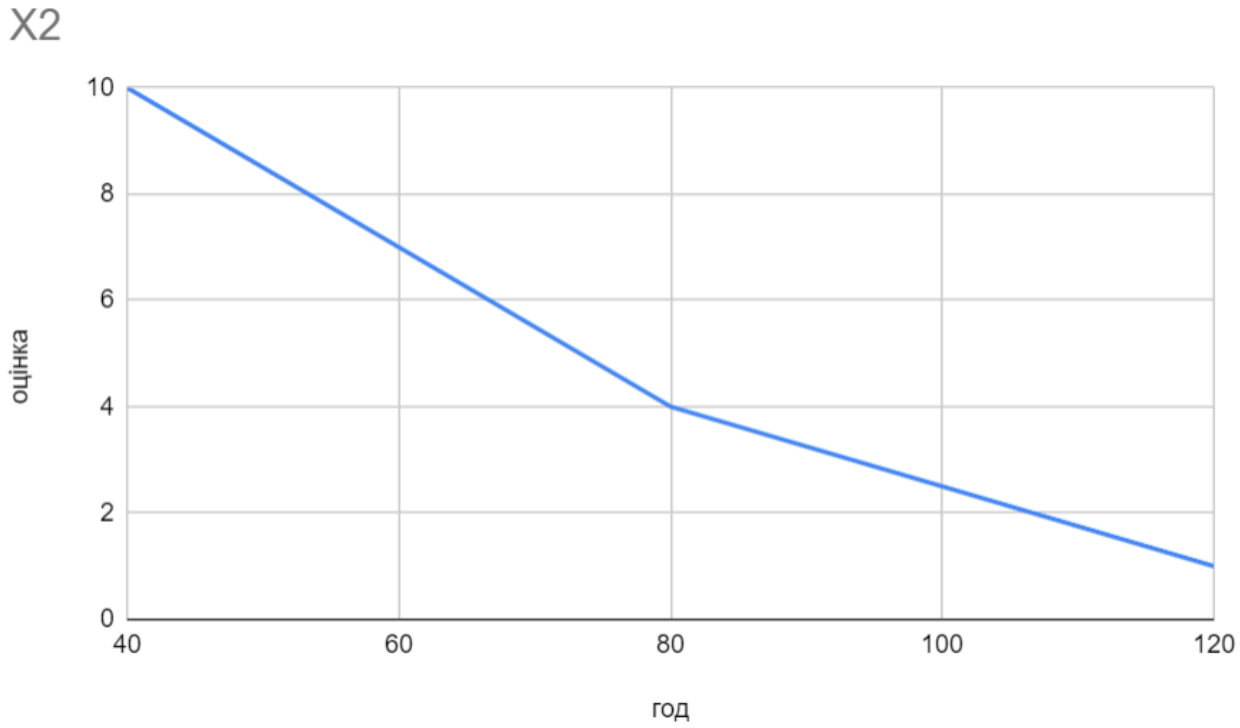

Рисунок 5.3 — Графічне зображення оцінок параметра X2

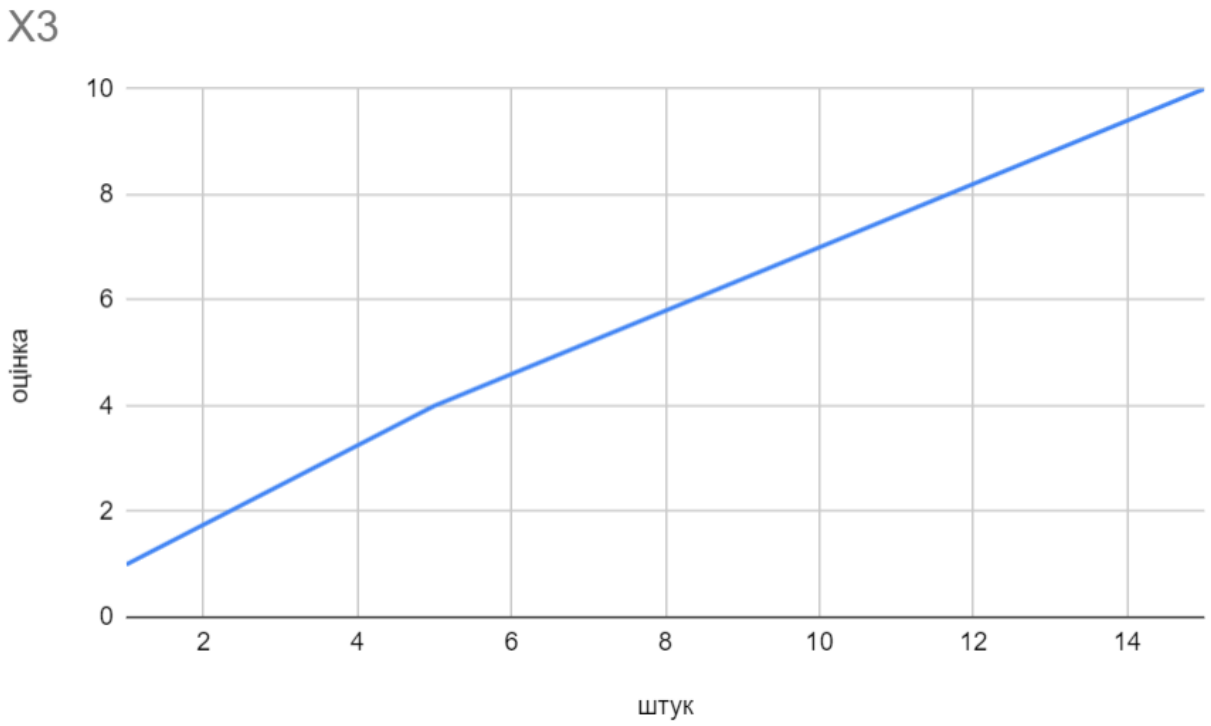

Рисунок 5.4 — Графічне зображення оцінок параметра X3

5.4 Аналіз рівня якості варіантів реалізації функцій

Результати експертного ранжування наведені у таблиці 5.3.

Таблиця 5.3 — Експертне ранжування

Параметр	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X1	час на побудову пакету	год	1	2	3	2	1	2	2	13	-1	1
X2	час для ознайомлення з теорією	год	2	1	1	1	2	1	1	9	-5	25
X3	Кількість побудованих тестів	шт.	3	3	2	3	3	3	3	20	6	36
	Разом		6	6	6	6	6	6	6	42	0	62

Найбільшим вважається ранг 3, відповідно найменшим є ранг 1. Чим більший ранг експерт присвоює параметру, тим важливішим він або вона його вважає.

Коефіцієнт узгодженості дорівнює:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 * 62}{49 * (27 - 3)} = 0.632. \quad (5.1)$$

Використовуючи отримані від кожного експерта результати ранжування параметрів, проведено попарне порівняння всіх параметрів, наведене у таблиці 5.4.

Таблиця 5.4 — Попарне порівняння параметрів

Параметри	Експерти							Підсумкова оцінка	Коефіцієнт переваги (x_{ij})
	1	2	3	4	5	6	7		
x_1 та x_2	<	>	>	>	<	>	>	>	1.5
x_2 та x_3	<	<	<	<	<	<	<	<	0.5
x_3 та x_1	>	>	<	>	>	>	>	>	1.5

Виходячи з наведеної таблиці слідує, що:

$$X3 > X1 > X2 \tag{5.2}$$

Використовуючи результати попарного порівняння обчислюється вагомість кожного з критеріїв (таблиця 5.5)

Таблиця 5.5 — Розрахунок вагомості параметрів

Параметри	Параметри			Перша ітерація		Друга ітерація		Третя ітерація	
	X1	X2	X3	b_i	K_{bi}	b_i^1	K_{bi}^1	b_i^2	K_{bi}^2
X1	1	1.5	0.5	3	0.333	8	0.32	22	0.318
X2	0.5	1	0.5	2	0.222	5.5	0.22	15.25	0.221
X3	1.5	1.5	1	4	0.444	11.5	0.46	31.75	0.460
Всього:				9	1	25	1	69	1

Розрахунок рівня якості варіантів реалізації наведено у таблиці 5.6.

Таблиця 5.6 — Розрахунок рівня якості варіантів реалізації

Основні функції	Варіант реалізації функції	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	Б	X1	35	4	0.318	1.272
F2	А	X2	60	7	0.221	1.547
F3	А	X2	60	7	0.221	1.547
		X3	4	3	0.460	1.38
	Б	X2	80	4	0.221	0.884
		X3	12	8	0.460	3.68

Функція $F3$ — визначення метрик для тестування алгоритмів, є функцією від двох параметрів – часу, який витрачається на підготовку до реалізації ПП($X2$), та бажаної кількості метрик($X3$).

Отже рівень якості для першого варіанту складе

$$K1 = 1.272 + 1.547 + 1.547 + 1.38 = 5.746,$$

а для другого

$$K2 = 1.272 + 1.547 + 0.884 + 3.68 = 7.338.$$

Звідки випливає, що другий варіант є кращим.

Для визначення вартості розробки спочатку проведемо розрахунок трудомісткості. Обидва варіанти реалізації складаються з двох окремих завдань: підготовки теоретичної бази та розробки продукту.

Складність роботи продукту від варіанту реалізації не залежить, однак об'єм теорії, яку необхідно використати при розробці, напряму залежить від варіанту.

У першому варіанті можливо реалізувати стандартний набір тестів та оцінок, що потребує менших витрат часу на вивчення теоретичного матеріалу. Другий варіант є протилежним до першого у тому сенсі, що навпаки, усі тести та оцінки будуються власні, десь нові, десь

покращені відомі тести. У цьому випадку можливо розробити значно більшу кількість інструментів для оцінки рекомендаційних систем, однак час, який необхідно витратити на освоєння необхідної теорії, є значно більшим. (У першому варіанті параметр $X2$ функції $F3$ є меншим (кращим) від відповідного значення у другому варіанті, однак параметр $X3$ функції $F3$, навпаки є більшим (кращим) у другому варіанті).

5.5 Економічний аналіз варіантів розробки ПП

Отже, для другого завдання: алгоритм групи складності 3, ступінь новизни А, вид використаної інформації — НДІ, час розробки 27 людино-днів, $К_{п} = 1.26$, $К_{ск} = 1$, $К_{стм} = 1$

Таким чином

$$Т2 = 27 * 1.26 * 1.6 = 54.432 \text{ людино-дні.}$$

Для завдання 1 (при реалізації першого варіанту): алгоритм групи складності 1, ступінь новизни Б , вид використаної інформації — ПІ, ТР = 64 людино-днів, $К_{п} = 2.01$, $К_{ск} = 1$, $К_{ст} = 0.7$, $К_{стм} = 1$.

Тому

$$Т1А = 64 * 2.01 * 0.7 = 90.048 \text{ людино-дні.}$$

Для завдання 1 (при реалізації другого варіанту): (алгоритм групи складності 1, ступінь новизни А, вид використаної інформації НДІ) ТР = 90 людино-днів, $К_{п} = 1.7$, $К_{ск} = 1$, $К_{ст} = 0.7$, $К_{стм} = 1$.

Тому

$$Т1Б = 90 * 1.7 * 0.7 = 107,1 \text{ людино-дні.}$$

Звідси виходить, що у випадку А

$$Т = 54,432 + 90,048 = 144,48$$

А у випадку Б

$$Т = 54,432 + 107.1 = 161,532$$

В розробці мають брати участь один програміст з окладом 14000 грн та один математик з окладом 10000 грн. Погодинна заробітна плата працівників складе:

$$\text{С} = \frac{14000 + 10000}{2 * 21 * 8} = 71.43 \text{ грн} \tag{5.3}$$

Поваріантно:

$$\text{С1} = 144.48 * 71.43 * 1.2 * 8 = 99073 \text{ грн} \tag{5.4}$$

та

$$\text{С2} = 161.532 * 71.43 * 1.2 * 8 = 110767 \text{ грн} \tag{5.5}$$

Відрахування ЄСВ складають 22%.

$$\text{С1'} = \text{С1} * 0.22 = 99073 * 0.22 = 21796 \text{ грн}$$

$$\text{С2'} = \text{С2} * 0.22 = 110767 * 0.22 = 24368 \text{ грн}$$

Визначимо витрати на оплату однієї машино-години. З урахуванням заробітної плати програміста в розмірі 14000 з коефіцієнтом зайнятості 0.2.

$$\text{С}_\text{г} = 12 * \text{М} * \text{К}_\text{з} = 12 * 14000 * 0.2 = 33600 \text{ грн}$$

Включаючи додаткову плату:

$$\text{С}_\text{зп} = \text{С}_\text{г} * (1 + \text{К}_\text{з}) = 33600 * 1.2 = 40320 \text{ грн} \tag{5.6}$$

Відповідно, відрахування на соціальний внесок складуть:

$$\text{С}_\text{від} = \text{С}_\text{зп} * 0.22 = 40320 * 0.22 = 8870 \text{ грн} \tag{5.7}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 7000 грн.

$$\text{С}_\text{а} = \text{К}_\text{тм} * \text{К}_\text{а} * \text{Ц}_\text{пр} = 1.15 * 0.25 * 7000 = 2012.5 \text{ грн}$$

При цьому профілактичі трати сягнуть

$$\text{С}_\text{р} = \text{К}_\text{тм} * \text{К}_\text{р} * \text{Ц}_\text{пр} = 1.15 * 0.05 * 7000 = 402.5 \text{ грн}$$

$$T_\text{ЕФ} = (\text{Д}_\text{К} - \text{Д}_\text{В} - \text{Д}_\text{С} - \text{Д}_\text{Р}) * t_\text{з} * \text{К}_\text{В} = (365 - 104 - 11 - 16) * 8 * 0,9 = 1685 \text{год}$$

Звідси виходить, що витрати на оплату електроенергії, з урахуванням ПДВ, складуть:

$$C_\text{ЕЛ} = T_\text{ЕФ} * N_\text{С} * K_\text{З} * \text{Ц}_\text{ЕН} =$$

$$= 1684.8 * 0,2 * 1,46255 * 2 * 1.75 = 1724,52 \text{ грн} \quad (5.8)$$

Накладні витрати складуть:

$$C_\text{н} = \text{Ц}_\text{пр} * 0.67 = 7000 * 0.67 = 4.690 \text{ грн} \quad (5.9)$$

Річні витрати на експлуатацію

$$C_\text{ЕКС} = C_\text{ЗП} + C_\text{ВІД} + C_\text{А} + C_\text{Р} + C_\text{ЕЛ} + C_\text{Н} =$$

$$= 40.320 + 8.870 + 2012.5 + 402.5 + 1724.52 + 4.690 = 58019,52 \text{ грн} \quad (5.10)$$

Тоді собівартість однієї машино-години складе:

$$C_\text{М-Г} = \frac{C_\text{ЕКС}}{T_\text{ЕФ}} = \frac{58019,52}{1685} = 34.4 \frac{\text{грн}}{\text{год}} \quad (5.11)$$

Тоді, в залежності від варіанту, витрати на оплату машинного часу складають:

$$C_\text{м1} = 34.4 * 144.48 = 4970.112 \text{ грн} \quad (5.12)$$

$$C_\text{м2} = 34.4 * 161,532 = 5556.7 \text{ грн} \quad (5.13)$$

Накладні витрати становитимуть 67% від заробітної плати, тобто 66378 грн для першого варіанту та 74214 для другого.

Таким чином, вартість розробки ПП та проведення дослідів для першого варіанту реалізації складає:

$$C_\text{ПП} = C_\text{ЗП} + C_\text{ВІД} + C_\text{М} + C_\text{Н} =$$

$$= 99073 + 21796 + 4970112 + 66378 = 192184 \text{ грн} \quad (5.14)$$

Та, відповідно, для другого:

$$110767 + 24368 + 55567 + 74214 = 214905 \text{ грн} \tag{5.15}$$

Коефіцієнти техніко-економічного рівня, розраховані за формулою:

$$K_{\text{ТЕРj}} = \frac{K_{\text{Kj}}}{C_{\Phi\text{j}}} \tag{5.16}$$

становлять

$$K_{\text{ТЕР1}} = \frac{5.746}{192184} = 2.99 * 10^{(-5)} \tag{5.17}$$

для першого варіанту реалізації, та, відповідно

$$K_{\text{ТЕР2}} = \frac{7.338}{214905} = 3.41 * 10^{(-5)} \tag{5.18}$$

для другого. Отже можемо зробити висновок, що другий варіант виявився більш ефективним.

5.6 Висновки до розділу 5

Завдяки функціонально-вартісному аналізу розробленого пакету для тестування рекомендаційних систем було отримано корисні висновки про найбільш перспективний та ефективний варіант реалізації ПП. Ним виявився другий варіант реалізації.

ВИСНОВКИ

Результатом проведеної роботи є як істотний аналіз популярних видів рекомендаційних систем колаборативної фільтрації, так і побудований пакет метрик, тестів та алгоритмів, які в цій роботі використовувались. Результати роботи є цінними як з точки зору теорії, тобто інформація, яка висвітлена у цій роботі, так і з прикладної.

Результати аналізу та порівняння рекомендаційних систем колаборативної фільтрації дозволили винайти слабкі та сильні місця окремих алгоритмів, оцінити їх схожість та унікальність. Детальний аналіз окремих алгоритмів дозволяє проводити більш складне та ефективне налаштування алгоритмів колаборативної фільтрації для використання у різноманітних проектах.

Побудовані тести, а також порядок їх використання та тлумачення можуть бути корисними при введенні в експлуатацію рекомендаційної системи будь-якого виду.

Рекомендаційні системи колаборативної фільтрації становлять широкий набір універсальних алгоритмів побудови рекомендацій, більшість з яких не так сильно залежить від датасету, або початкових налаштувань, як, наприклад, алгоритми контентної фільтрації. Тому розроблені метрики та тести слід вважати корисною інструкцією для покращення роботи рекомендаційних систем на реальних даних.

Оскільки як алгоритми, так і система їх аналізу представляють собою практичну цінність, дана робота може мати впровадження у реальному проекті.

Корисно було б додати ще декілька додаткових метрик, які дозволили б проводити більше ретельний аналіз на специфічних даних, або ж аналіз більш складних алгоритмів фільтрації. Реалізовані на мові Python алгоритми за високих вимог на швидкодію можна реалізовувати, користуючись більш швидкими та оптимальними мовами програмування, як, наприклад, C чи C++.

ПЕРЕЛІК ПОСИЛАНЬ

1. Dietmar J., Markus Z., Alexander F.,Gerhard F. Recommender Systems An Introduction. New York, NY: Cambridge University Press, 2011. 331 p.

2. Charu C. Aggarwal. Recommender Systems The Textbook. New York, NY: Springer International Publishing Switzerland, 2016. 498 p.

3. Ricci F., Rokach L., Shapira B. Recommender Systems Handbook. Boston, MA: Springer, 2010. 842 p.

4. Cacheda F., Carneiro V., Fernández D., Formoso V. Comparison of collaborative filtering algorithms: Limitations of current techniques and proposals for scalable, high-performance recommender systems. ACM Transactions on the Web. 2011. February 15. p. 1-33.

5. Sarwar B., Karypis G., Konstan J., and Riedl J. Item-Based Collaborative Filtering Recommendation Algorithms. Hong Kong. Hong Kong WWW10 Conference : «WWW10», 2001. January 20-25. p. 285-289.

6. Hassanieh L., Jaoude, A., Bou Abdo J., Demerjian J. Similarity measures for collaborative filtering recommender systems. Manama, Bahrain: IEEE Middle East and North Africa Communications Conference, 2018. 6 p.

7. F. Maxwell Harper, Joseph A. Konstan. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS), 2015. Vol. 5. 19 p.

8. ДСТУ 3008:2015. Документація. Звіти у сфері науки і техніки. Структура і правила оформлення. [На заміну ДСТУ 3008-95; чинний від 2015-06-22]. Вид. офіц. Київ: ДП «УкрНДНЦ», 2016. 31 с. (Інформація та документація).

Додаток А Програмний модуль розроблених алгоритмів

```
cmr
import pandas as pd
import similarities as sim
import numpy as np
import math
import scipy
from surprise import SlopeOne
from surprise import Dataset
from surprise import Reader
import time
def random_filter(test,train,all):
    start = time.time()
    mean = train["rating"].mean()
    adjusted_ratings = train["rating"]
    adjusted_ratings = (adjusted_ratings - mean)**2
    sigma = np.sqrt(adjusted_ratings.mean())
    user_list = pd.Series(test["userId"]).drop_duplicates()
    np_results = []
    fit = time.time()
    fit_time = fit=start
    for u in user_list:
        watched = list(test[test.userId == u]['movieId'])
        for m in watched:
            rating = np.random.normal(mean, sigma)
            if (rating >5) or (rating<0):
                rating = mean
            np_results.append([u,m,rating])
    res = pd.DataFrame(np_results,columns=test.columns)
```

```
    predict_time = time.time() - fit
    overall= predict_time+fit - start
    return res, [fit_time,predict_time,overall
def random_user_filter(test,train,all):
    start = time.time()
    # mean = train["rating"].mean()
    # adjusted_ratings = train["rating"]
    # adjusted_ratings = (adjusted_ratings - mean)**2
    # sigma = np.sqrt(adjusted_ratings.mean())
    user_list = pd.Series(test["userId"]).drop_duplicates()
    np_results = []
    fit = time.time()
    fit_time = fit - start
    for u in user_list:
        ur = train.loc[train.userId == u]['rating']
        um = ur.mean()
        adj = (ur - um)**2
        sgm = np.sqrt(adj.mean())
        watched = list(test[test.userId == u]['movieId'])
        for m in watched:
            rating = np.random.normal(um, sgm)
            if (rating >5) or (rating<0):
                rating = um
            np_results.append([u,m,rating])
    res = pd.DataFrame(np_results,columns=test.columns)
    predict_time = time.time() - fit
    overall= predict_time+fit - start
    return res, [fit_time,predict_time,overall]
def random_item_filter(test,train,all):
    start = time.time()
    # mean = train["rating"].mean()
    # adjusted_ratings = train["rating"]
```

```
    # adjusted_ratings = (adjusted_ratings - mean)**2
    # sigma = np.sqrt(adjusted_ratings.mean())
    user_list = pd.Series(test["userId"]).drop_duplicates()
    np_results = []
    fit = time.time()
    fit_time = fit - start
    for u in user_list:
        # ur = train.loc[all.userId == u]['rating']
        # um = ur.mean()
        # adj = (ur - um)**2
        # sgm = np.sqrt(adj.mean())
        watched = list(test[test.userId == u]['movieId'])
        for m in watched:
            ir = train.loc[train.movieId == m]['rating']
            im = ir.mean()
            adj = (ir - im) ** 2
            sgm = np.sqrt(adj.mean())
            rating = np.random.normal(im, sgm)
            if (rating >5) or (rating<0):
                rating = im
            np_results.append([u,m,rating])
    res = pd.DataFrame(np_results,columns=test.columns)
    predict_time = time.time() - fit
    overall= predict_time+fit - start
    return res, [fit_time,predict_time,overall]
def simple_user_based_filter(test,train,all):
    def rate(mid, geeks, ratings,mean):
        row = ratings[mid][ratings.index.isin(geeks)]
        ln=len(row)
        for i in row.values:
            if i==0:
                ln = ln-1
```

```
        if ln !=0:
            return row.sum()/ln
        else:
            return mean
    start = time.time()
    accelerated = train.copy()
    mean = train["rating"].mean()
    pivot = pd.pivot_table(accelerated, values='rating',
     index='userId', columns='movieId').apply(
        lambda row: row.fillna(0), axis=1)
    all_pivot = pd.pivot_table(all, values='rating',
     index='userId', columns='movieId').apply(
        lambda row: row.fillna(0),axis=1)
    similarities = sim.pearson(pivot)
    np.fill_diagonal(similarities, 0)
    fit = time.time()
    fit_time = fit - start
    user_list = pd.Series(test["userId"]).drop_duplicates()
    results =[]
    for u in user_list:
        geeks = pd.Series(similarities[u-1])
        geeks.sort_values(ascending = False,inplace= True)
        geeks = geeks.index[:30]
        watched = list(test[test.userId == u]['movieId'])
        for m in watched:
            res = rate(m,geeks,all_pivot,mean)
            results.append([u,m,res])
    out = pd.DataFrame(results, columns=test.columns, index =test.index)
    predict_time = time.time() - fit
    overall = predict_time + fit - start
    return out, [fit_time,predict_time,overall]
def simple_item_based_filter(test, train, all):
```

```
start = time.time()
accelerated = train.copy()
mean = train["rating"].mean()
pivot = pd.pivot_table(accelerated, values='rating',
 index='movieId', columns='userId').apply(
    lambda row: row.fillna(0), axis=1)
similarities = sim.cosine(pivot)
np.fill_diagonal(similarities, 0)
user_list = list(pd.Series(accelerated["userId"]).drop_duplicates())
movie_list = list(pd.Series(accelerated["movieId"]).drop_duplicates())
test_users = list(pd.Series(test["userId"]).drop_duplicates())
labeled_sim = pd.DataFrame(similarities,
index=movie_list, columns=movie_list)
out = []
fit = time.time()
fit_time = fit - start
for u in test_users:
    test_movies = list(test.loc[test.userId == u]['movieId'])
    if u in user_list:
        avg = train.loc[all.userId == u]['rating'].mean()
        for m in test_movies:
            if m in movie_list:
                similar = pd.Series(labeled_sim.loc[m])
                similar.sort_values(ascending = False,inplace= True)
                similar = similar.index[:400]
                rating = accelerated.loc[accelerated.userId == u]
                loc[accelerated.movieId.isin(similar)]['rating'].mean()
                if math.isnan(rating):
                    rating=mean
                out.append([u, m, rating])
            else:  out.append([u,m,avg])
out = pd.DataFrame(out, columns=test.columns, index=test.index)
```

```
    predict_time = time.time() - fit
    overall = predict_time + fit - start
    return out, [fit_time,predict_time,overall]
def svd_filter(test,train,all):
    feature_num = 30
    def svd(matrix, k):
        U, s, V = np.linalg.svd(matrix, full_matrices=False)
        s = np.diag(s)
        s = s[0:k, 0:k]
        U = U[:, 0:k]
        V = V[0:k, :]
        s_root = scipy.linalg.sqrtm(s)
        Usk = np.dot(U, s_root)
        skV = np.dot(s_root, V)
        UsV = np.dot(Usk, skV)
        UsV = UsV
        return UsV
    start = time.time()
    accelerated = train.copy()
    pivot = pd.pivot_table(accelerated, values='rating', i
    ndex='userId', columns='movieId')
    matrix = pivot.values
    user_num, movie_num = matrix.shape
    item_means = []
    user_means = []
    for u in range(user_num):
        user_scores = matrix[u]
        user_not_nan_scores = user_scores[~np.isnan(user_scores)]
        user_means.append(np.mean(user_not_nan_scores))
    for m in range(movie_num):
        item_scores = matrix[:, m]
        item_not_nan_scores = item_scores[~np.isnan(item_scores)]
```

```
        item_means.append(np.mean(item_not_nan_scores))
    for m in range(movie_num):
        for u in range(user_num):
            if np.isnan(matrix[u][m]):
                matrix[u][m] = item_means[m]
    x = np.tile(item_means, (matrix.shape[0], 1))
    matrix = matrix - x
    user_list = list(pd.Series(accelerated["userId"]).drop_duplicates())
    user_dict = {user_list[i]: i for i in range(user_num)}
    movie_list = list(pd.Series(accelerated["movieId"]).drop_duplicates())
    movie_dict = {movie_list[i]: i for i in range(movie_num)}
    SVD = svd(matrix, feature_num) + x
    test_users = list(pd.Series(test["userId"]).drop_duplicates())
    out =[]
    fit = time.time()
    fit_time = fit - start
    for u in test_users:
        test_movies = list(test.loc[test.userId == u]['movieId'])
        if u in user_list:
            for m in test_movies:
                if m in movie_list:
                    out.append([u,m,SVD[user_dict[u]][movie_dict[m]]])
                else:
                    out.append([u, m, user_means[user_dict[u]]])
    out = pd.DataFrame(out, columns=test.columns, index=test.index)
    predict_time = time.time() - fit
    overall = predict_time + fit - start
    return out, [fit_time,predict_time,overall]
def slope_one(test,train,all):
    start = time.time()
    reader = Reader(rating_scale=(0.5, 5))
    data = Dataset.load_from_df(train[['userId', 'movieId', 'rating']], reader)
```

```
    test_data = Dataset.load_from_df(test[['userId', 'movieId', 'rating']], reader)
    trainset = data.build_full_trainset()
    testset = test_data.build_full_trainset().build_testset()
    algo = SlopeOne()
    algo.fit(trainset)
    fit = time.time()
    fit_time = fit - start
    predictions = algo.test(testset)
    uid = []
    mid = []
    rate =[]
    for i in range(len(predictions)):
        uid.append(predictions[i].uid)
        mid.append(predictions[i].iid)
        rate.append(predictions[i].est)
    out = {'userId': uid, 'movieId': mid, 'rating':rate}
    out = pd.DataFrame.from_dict(out)
    predict_time = time.time() - fit
    overall = predict_time + fit - start
    return out, [fit_time,predict_time,overall]
def simple_ubf_var_sim(test,train,all, sim_func):
    def rate(mid, geeks, ratings,mean):
        row = ratings[mid][ratings.index.isin(geeks)]
        ln=len(row)
        for i in row.values:
            if i==0:
                ln = ln-1
        if ln !=0:
            return row.sum()/ln
        else:
            return mean
    start = time.time()
```

```
    accelerated = train.copy()
    mean = train["rating"].mean()
    pivot = pd.pivot_table(accelerated, values='rating',
    index='userId', columns='movieId').apply(
        lambda row: row.fillna(0), axis=1)
    all_pivot = pd.pivot_table(all, values='rating', inde
    x='userId', columns='movieId').apply(
        lambda row: row.fillna(0),axis=1)
    similarities = sim_func(pivot)
    print("f")
    np.fill_diagonal(similarities, 0)
    fit = time.time()
    fit_time = fit - start
    user_list = pd.Series(test["userId"]).drop_duplicates()
    results =[]
    for u in user_list:
        geeks = pd.Series(similarities[u-1])
        geeks.sort_values(ascending = False,inplace= True)
        geeks = geeks.index[:30]
        watched = list(test[test.userId == u]['movieId'])
        for m in watched:
            res = rate(m,geeks,all_pivot,mean)
            results.append([u,m,res])
    out = pd.DataFrame(results, columns=test.columns, index =test.index)
    predict_time = time.time() - fit
    overall = predict_time + fit - start
    return out, [fit_time,predict_time,overall]
def simple_ubf_var_k(test,train,all, vss):
    def rate(mid, geeks, ratings,mean):
        row = ratings[mid][ratings.index.isin(geeks)]
        ln=len(row)
        for i in row.values:
```

```
        if i==0:
            ln = ln-1
    if ln !=0:
        return row.sum()/ln
    else:
        return mean
start = time.time()
accelerated = train.copy()
mean = train["rating"].mean()
pivot = pd.pivot_table(accelerated, values='rating',
index='userId', columns='movieId').apply(
    lambda row: row.fillna(0), axis=1)
all_pivot = pd.pivot_table(all, values='rating', index=
userId', columns='movieId').apply(
    lambda row: row.fillna(0),axis=1)
similarities = sim.cosine(pivot)
print("f")
np.fill_diagonal(similarities, 0)
fit = time.time()
fit_time = fit - start
user_list = pd.Series(test["userId"]).drop_duplicates()
results =[]
for u in user_list:
    geeks = pd.Series(similarities[u-1])
    geeks.sort_values(ascending = False,inplace= True)
    geeks = geeks.index[:vss]
    watched = list(test[test.userId == u]['movieId'])
    for m in watched:
        res = rate(m,geeks,all_pivot,mean)
        results.append([u,m,res])
out = pd.DataFrame(results, columns=test.columns, index =test.index)
predict_time = time.time() - fit
```

```
    overall = predict_time + fit - start
    return out, [fit_time,predict_time,overall]
def simple_ibf_var_k(test, train, all, vss):
    start = time.time()
    accelerated = train.copy()
    mean = train["rating"].mean()
    pivot = pd.pivot_table(accelerated, values='rating'
    , index='movieId', columns='userId').apply(
        lambda row: row.fillna(0), axis=1)
    similarities = sim.cosine(pivot)
    np.fill_diagonal(similarities, 0)
    user_list = list(pd.Series(accelerated["userId"])
    .drop_duplicates())
    movie_list = list(pd.Series(accelerated["movieId"
    ]).drop_duplicates())
    test_users = list(pd.Series(test["userId"]).drop_
    duplicates())
    labeled_sim = pd.DataFrame(similarities, index=movie_list
    , columns=movie_list)
    out = []
    fit = time.time()
    fit_time = fit - start
    for u in test_users:
        test_movies = list(test.loc[test.userId == u]['movieId'])
        if u in user_list:
            avg = train.loc[all.userId == u]['rating'].mean()
            for m in test_movies:
                if m in movie_list:
                    similar = pd.Series(labeled_sim.loc[m])
                    similar.sort_values(ascending = False,inplace= True)
                    similar = similar.index[:vss]
                    rating = accelerated.loc[accelerated.userId == u].
```

```
                    loc[accelerated.movieId.isin(similar)]['rating'].mean()
                    if math.isnan(rating):
                        rating=mean
                    out.append([u, m, rating])
                else:
                    out.append([u,m,avg])
    out = pd.DataFrame(out, columns=test.columns,
    index=test.index)
    predict_time = time.time() - fit
    overall = predict_time + fit - start
    return out, [fit_time,predict_time,overall]
def svd_k(test,train,all,kdd):
    feature_num = kdd
    def svd(matrix, k):
        U, s, V = np.linalg.svd(matrix, full_matrices=False)
        s = np.diag(s)
        s = s[0:k, 0:k]
        U = U[:, 0:k]
        V = V[0:k, :]
        # s_root = scipy.linalg.sqrtm(s)
        # Usk = np.dot(U, s_root)
        # skV = np.dot(s_root, V)
        # UsV = np.dot(Usk, skV)
        # UsV = UsV
        UsV = np.dot(np.dot(U,s),V)
        return UsV
    start = time.time()
    accelerated = train.copy()
    pivot = pd.pivot_table(accelerated, values='rating',
     index='userId', columns='movieId')
    matrix = pivot.values
    user_num, movie_num = matrix.shape
```

```
item_means = []
user_means = []
for u in range(user_num):
    user_scores = matrix[u]
    user_not_nan_scores = user_scores[~np.isnan(user_scores)]
    user_means.append(np.mean(user_not_nan_scores))
for m in range(movie_num):
    item_scores = matrix[:, m]
    item_not_nan_scores = item_scores[~np.isnan(item_scores)]
    item_means.append(np.mean(item_not_nan_scores))
for m in range(movie_num):
    for u in range(user_num):
        if np.isnan(matrix[u][m]):
            matrix[u][m] = item_means[m]
x = np.tile(item_means, (matrix.shape[0], 1))
matrix = matrix - x
user_list = list(pd.Series(accelerated["userId"]).
drop_duplicates())
user_dict = {user_list[i]: i for i in range(user_num)}
movie_list = list(pd.Series(accelerated["movieId"]).d
rop_duplicates())
movie_dict = {movie_list[i]: i for i in range(movie_num)}
SVD = svd(matrix, feature_num) + x
test_users = list(pd.Series(test["userId"]).
drop_duplicates())
out =[]
fit = time.time()
fit_time = fit - start
for u in test_users:
    test_movies = list(test.loc[test.userId == u]['movieId'])
    if u in user_list:
        for m in test_movies:
```

```
            if m in movie_list:
                out.append([u,m,SVD[user_dict[u]][movie_dict[m]]])
            else:
                out.append([u, m, user_means[user_dict[u]]])
out = pd.DataFrame(out, columns=test.columns, index=test.index)
predict_time = time.time() - fit
overall = predict_time + fit - start
return out, [fit_time,predict_time,overall]
```

Додаток Б Презентація

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

Аналіз та порівняння рекомендаційних систем колаборативної фільтрації

Виконав:
студент групи КА-64
Станков Іван Сергійович
Керівник:
к.ф-м.н.,
Статкевич Віталій Михайлович

Предмет, об’єкт та мета дослідження

- Предметом дослідження є побудова пакету оцінок, необхідних для вивчення такого класу алгоритмів, як рекомендаційні системи.
- Об’єктом дослідження є різні види рекомендаційних систем колаборативної фільтрації.
- Метою роботи є вивчення алгоритмів колаборативної фільтрації, а також побудова програмного пакету для тестування та оцінки якості роботи рекомендаційних систем.

Принцип роботи рекомендаційних систем

На основі інформації щодо користувачів та об’єктів, які цікавлять користувачів та які користувачі передивились, зробити рекомендацію нових об’єктів, які їх мають зацікавити.

Рекомендаційна система будує рекомендації, які потім пропонуються користувачу.

Існують різні види рекомендаційних систем.

Види рекомендаційних систем

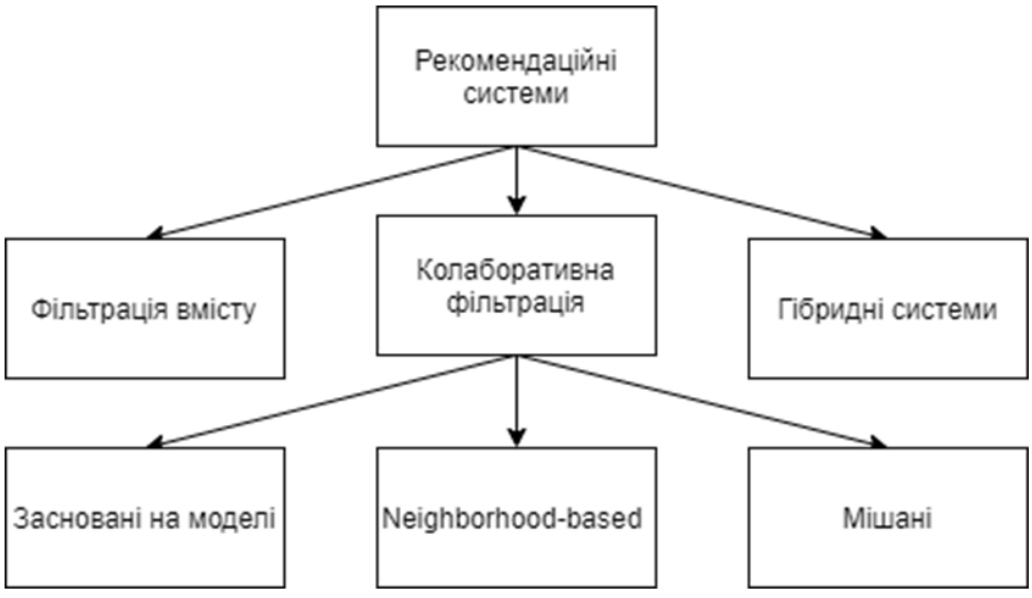

Рисунок 1 – Види рекомендаційних систем

Рекомендаційні системи колаборативної фільтрації

Не мають доступу до загальної інформації про об'єкти, тобто в таких системах не використовується інформація про рік створення, бренд, колір і т.д.

Замість цього використовуються оцінки користувачів, за допомогою яких визначається, які об'єкти є схожими, а які ні. Їх, зазвичай, зберігають у вигляді матриці наступного вигляду:

Таблиця 1 – матриця рейтингів

	i_1	i_2	...	i_n
u_1	v_{11}	v_{12}	...	v_{1n}
u_2	v_{21}	v_{22}	...	v_{2n}
...	...	...	...	...
u_m	v_{m1}	v_{m2}	...	v_{mn}

5

Реалізовані алгоритми

- **Випадковий фільтр** випадковим чином будує рекомендації.
- **Прості item- та user-based** алгоритми відшукують схожі об'єкти (відповідно користувачів), користуючись матрицею відомих оцінок. Використовуючи множину подібних об'єктів (відповідно користувачів), будуються рекомендації.
- **Slope One**, фактично, працює тим самим чином, однак відрізняється за алгоритмом пошуку схожих об'єктів (відповідно користувачів).
- **SVD фільтр** замість обробки матриці оцінок, використовує Singular Value Decomposition, що значно прискорює роботу.

6

Методика оцінювання якості рекомендаційних систем

- За витратами часу
- За похибкою передбачень (відношення між оцінкою, яку надав користувач, та оцінкою, яку прогнозувала система)
- Точність/повнота – показники подібні до однойменних оцінок бінарних класифікаторів, в яких оцінка будується в залежності від кількості вірно рекомендованих об’єктів, невірно рекомендованих та невірно не рекомендованих об’єктів.

Витрати часу

- Час на тренування, тобто час витрачений на обробку вхідної інформації.
- Час на побудову рекомендацій.
- Загальний час, який витрачає рекомендаційна система.

Оцінки точності (оцінки похибки рекомендацій)

- Оцінка MAE – середня абсолютна похибка
- Оцінка RMSE – усереднена квадратична похибка
- Оцінка RMSLE – усереднена квадратична логарифмована похибка

$$mae = \frac{1}{|U|}\sum_{u\in U}\frac{\sum_{i\in I_u}|p_{ui}-v_{ui}|}{|I_u|}, \qquad rmse = \frac{1}{|U|}\sum_{u\in U}\sqrt{\frac{\sum_{i\in I_u}(p_{ui}-v_{ui})^2}{|I_u|}},$$

$$rmsle = \frac{1}{|U|}\sum_{u\in U}\sqrt{\frac{\sum_{i\in I_u}(log(p_{ui}+1)-log(v_{ui}+1))^2}{|I_u|}}.$$

Відмінності оцінок

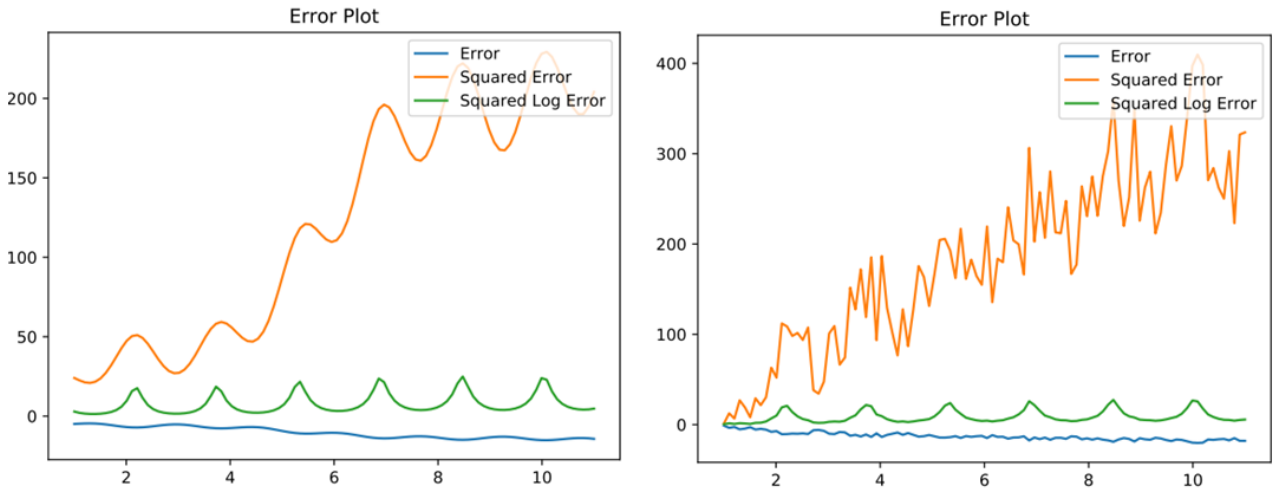

Рисунок 2 – Поведінка різних оцінок похибки А) без шуму Б) за наявності шуму

Точність/повнота

- Тут точність (precision) – у сенсі бінарних класифікаторів, визначається за наведеними нижче формулами.
- Точність та повнота за вибіркою з n кращих об'єктів.
- Точність та повнота за змінного значення граничної оцінки, яка розділяє вірні та невірні рекомендації.
- Криві точності-повноти.

$$precision = \frac{tp}{tp + fp},$$

$$recall = \frac{tp}{tp + fn},$$

де tp – число вірно визначених елементів;
fp – число невірно визначених елементів;
fn – число невірно проігнорованих елементів.

Порівняння алгоритмів

Усі оцінки будуються в залежності від параметра L, який визначає частку тестувальних даних від загальної множини вхідних даних. Більшість тестів складалась з послідовності запусків кожного алгоритму при різних початкових умовах. Окремі алгоритми оцінювались за цим же принципом, однак у їх випадку початкові умови задавались більш тонкі, направлені на більш ретельне вивчення особливостей поведінки.

Витрати часу

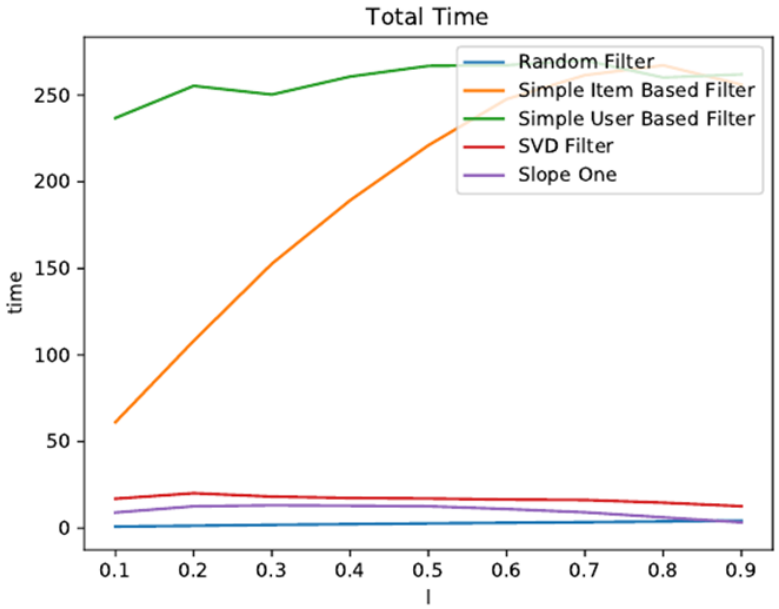

Рисунок 3 – Загальні витрати часу розроблених алгоритмів

13

Похибки моделей

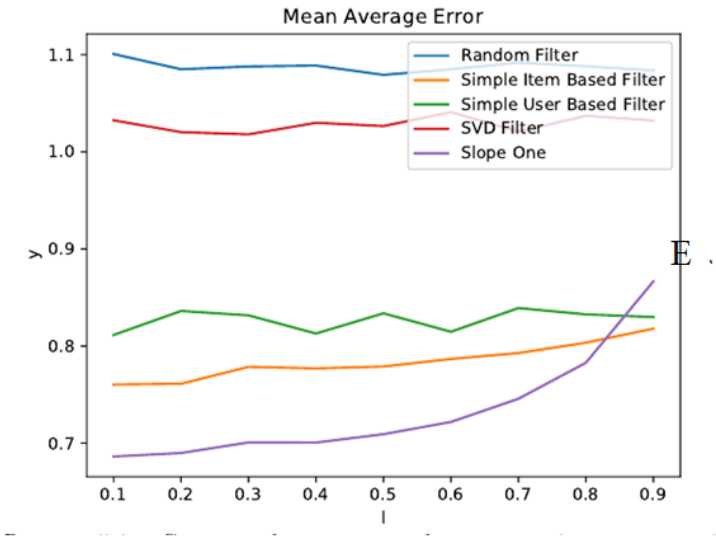

Рисунок 4 – Оцінка MAE

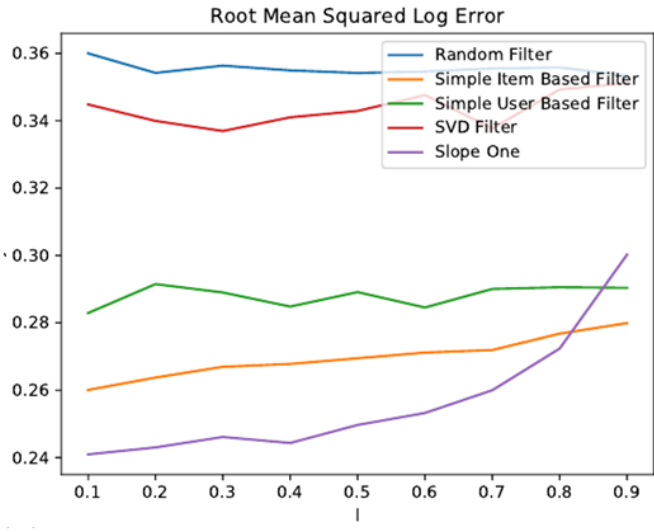

Рисунок 5 – Оцінка RMSLE

Криві точності та повноти

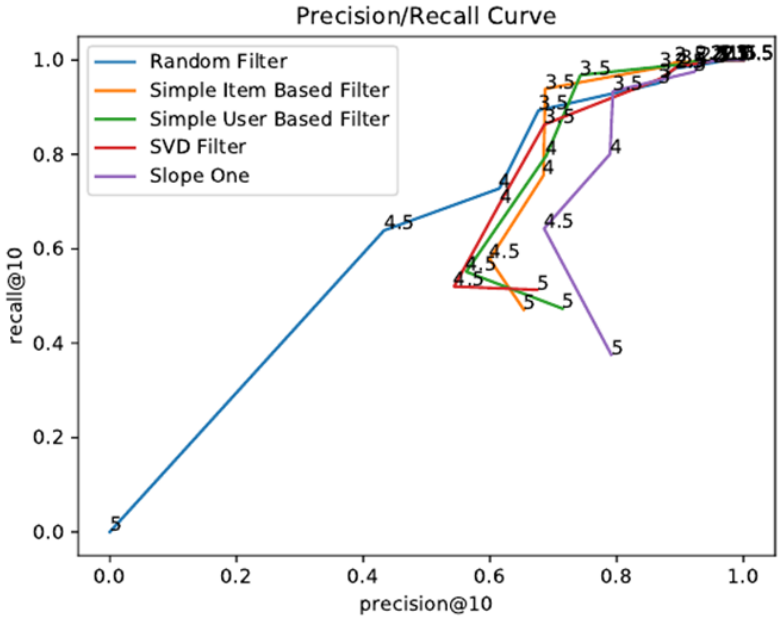

Рисунок 6 – Крива точності/повноти за змінної граничної оцінки

15

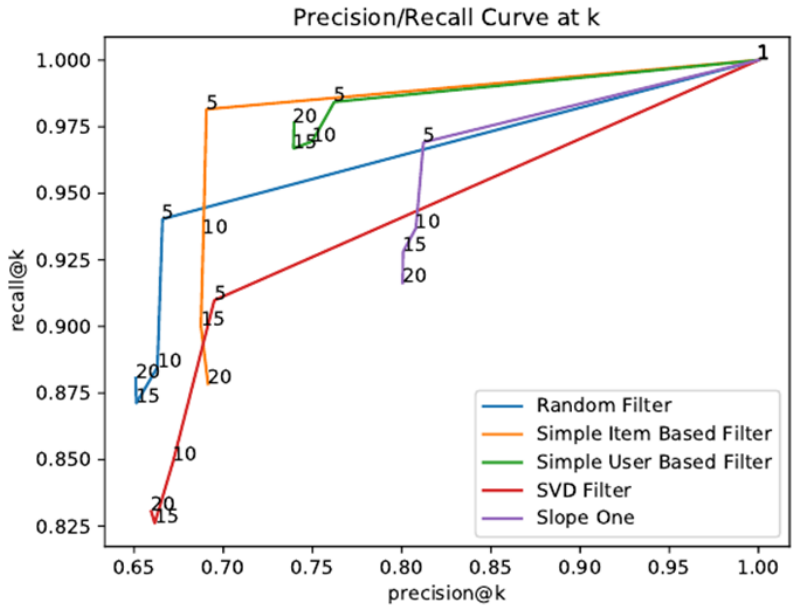

Рисунок 7 – Крива точності/повноти за змінного розміру вибірки

16

Висновки

- Проаналізовано та порівняно різні види рекомендаційних систем колаборативної фільтрації.
- Побудовано пакет метрик, тестів та алгоритмів (Python): item-based фільтр, used-based фільтр, Slope One алгоритм, SVD фільтр та випадковий фільтр.
- Розроблено систему оцінювання рекомендаційних систем, яка є корисною, зокрема, для виконання цієї роботи, а також в інших прикладних задачах.
- Вказані характерні слабкі та сильні місця окремих алгоритмів, оцінено їх схожість та унікальність.
- Практична цінність роботи: Amazon, Netflix тощо.

Дякую за увагу!