Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

Розробка двосторонньої рекомендаційної системи

Текстова частина до магістерської роботи

за спеціальністю «Інженерія програмного забеспечення»- 12**1**

Керівник курсової роботи

ст. викладач, к.н.,

Черкасов Д.І.

_____

(підпис)

“____” _____ 2021 р.

Виконав студент ІПЗ-МП2:

Баранов Д.О.

“____” _____ 2021 р.

Київ 2022

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

ЗАТВЕРДЖУЮ

доц., канд. фіз.-мат. наук

_____С. С. Гороховський

(підпис)

8 листопада 2021 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на магістерську роботу

студенту Баранову Д.О. факультету інформатики 2 курсу МП

ТЕМА Розробка двосторонньої рекомендаційної системи

Зміст ТЧ до курсової роботи:

Індивідуальне завдання

Вступ

1. Огляд рекомендаційних систем

2. Розробка системи

Висновок

Список використаних джерел

Дата видачі 8 листопада 2021 р.

Керівник _____

(підпис)

Завдання отримав _____

(підпис)

№	Назва етапу	**Термін виконання**	Примітка
1.	Отримання теми курсової	**08.11.2021**	
2.	Пошук тематичної наукової літератури	**20.10.2021**	
3.	Ознайомлення з науковою літературою	**11.12.2021**	
7.	Визначення структури програми	**05.02.2021**	
8.	Написання першої частини магістерської роботи	**23.02.2022**	
9.	Написання другої частини магістерської роботи	**27.06.2022**	
10.	Написання висновків магістерської роботи	**29.06.2019**	
11.	Правки	**03.07.2019**	
17.	Створення презентації	**05.07.2019**	
18.	Захист магістерської роботи	**7.07.2019**	

Зміст

Вступ 5
Основи рекомендаційних систем 7
Що таке рекомендаційна система та де використовується?.....7
Контент-орієнтовані рекомендаційні системи.....8
Колабораційні рекомендаційні системи 10
Що таке функція подібності?.....11
Рекомендаційна система основана на популярності 13
Гібридні рекомендаційні системи 14
Реалізація рекомендаційної системи.....15
Предметна область.....15
Аналіз проблеми.....16
Вибір алгоритму для реалізації двосторонньої системи.....17
Використані технології.....19
Релізація 21
Висновок 24
Тестування: 24
Висновки 25
Додадки 26
Список літератури:.....31

Вступ

В наш час людей охоплює величезна кількість інформації. Для того щоб не витрачати час на ручне опрацювання всієї існуючої інформації були придумані різні способи її обробки та видачі в готовому вигляді. Одним з таких засобів є рекомендаційні системи.

Для їх побудови придумано неймовірну кількість засобів для збору інформації. Кожного разу коли ви переходите з сайту на сайт зберігається історія ваших перегляді. На основі якої вже можна порекомендувати товар який вас цікавить зараз.

Взагалі практика показує що люди не завжди розуміють чого вони насправді хочуть. І іноді рекомендації допомагають людини потратити «лишні» гроші. Завдяки розвинутим алгоритмам для рекомендацій великі компанії заробляють дуже великі гроші на цьому

Знову ж таки збір даних про кожного користувача буває прихований або відкритий(заповнення анкети наприклад). Людям все більш некомфортніше відчувати що за ними завжди хтось спостерігає. На мою думку все-таки збір величезного числа інформації про кожну людину більш необхідний і зрештою завдяки хорошим рекомендаціям які зекономлять якийсь час – принесуть більше користі ніж недостатків.

Яку книгу прочитати? Який фільм подивитись? З якою людиною спілкуватися? На всі ці питання допоможе дати відповідь рекомендаційна система. Завдяки чому значною буде економія часу.

Проте потрібно мінімізувати кількість потенційно втрачених цікавих речей для конкретної людини. Для цього необхідно враховувати всі вимоги людини і підбирати відповідні результати.

В цій роботі буде описано спосіб побудови системи пошуку людей «за спільними інтересами» за допомогою двох сторонньої рекомендаційної системи.

Основи рекомендаційних систем

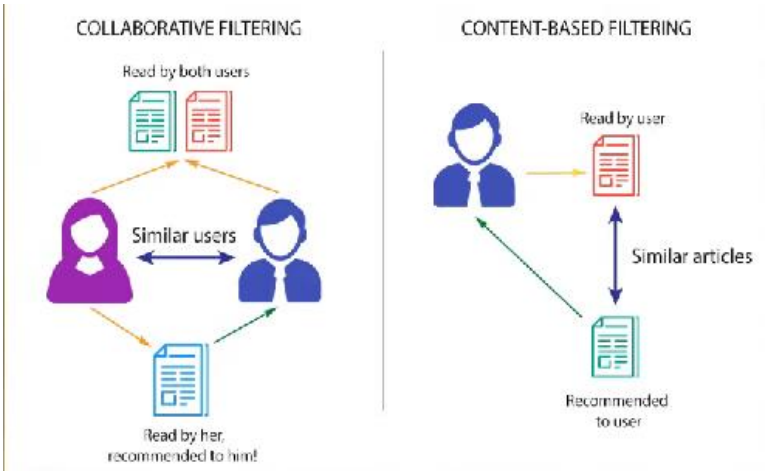
Що таке рекомендаційна система та де використовується?

З ростом різноманітної інформації та можливостей появилось бажання відфільтрувати її для кожної конкретної людини окремо. Комусь подобається Queen, комусь Metallica. Сучасні сервіси для прослуховування музики, такі як Spotify та YouTube Music наприклад, використовуються свої алгоритми які базуються на основі вже прослуханої музики та рекомендують, на їх думку, найбільш схожі на те що ви вже послухали і те що вам (приклад рекомендаційної систему заснованої на контенті про яку я розповім далі), скоріше за все сподобається. Завдяки цьому вони збільшують свої продажі, продажі музикантів та покращуються user-experience своїх користувачів.

Але рекомендаційні системи використовуються не лише при підборі музики. В наш час вони проникли майже у всі сфери життя. Навіть при пошуку нових знайомств, покупці товарів, способу знайти заняття на вихідні.

Тож, якщо узагальнювати, рекомендаційна система – це алгоритми, які допомагають значно відфільтрувати контент за якимось принципом.

Основними зараз розглядають рекомендаційні системи основані на контенті, методи колаборативної фільтрації та гібридні моделі.



Контент-орієнтовані рекомендаційні системи

У системах орієнтованих на вмісті головна ідея полягає в тому щоб подати результат який до вподоби користувачу на основі його раніше зроблених виборів. Тобто нам необхідно знати якусь інформацію про користувача. Цю інформацію можна дізнатись шляхом добровільного заповненням користувачем анкети. Дані про нього включають такі речі як: цілі, ідеї, місце проживання та інші.

Часто в таких системах використовують спосіб фільтрації вмісту для знаходження важливих слів за допомогою алгоритму TF-IDF.

Складність полягає в тому що потрібно збирати дані про дані – тобто метадані. Метадані про фільм — це все, що можна знайти на сторінці IMDb, наприклад. Розбити метадані можна умовно на дві категорії: Факти та Теги.

Для простоти можна вважати що факти – це незмінні сталі речі, як наприклад головний актор якогось фільму, рік виробництва.

Теги з іншого боку – це речі до яких різні люди ставляться по різному.

Одні з найбільших перешкод для розробки таких систем – це те що необхідних даних може і не бути. І як бути в такій ситуації? Як будувати рекомендації без уявлення про предмет.

TF-IDF – пошук важливих слів. Опис фільму або книгу можна робити на багато слів, але як зрозуміти які з них є важливими? Тут допоможе TF тобто частота з якою зустрічається слово у тексті. Далі IDF є кількістю всіх ваших документів, поділених на кількість документів, які містять слово

$$idf(term) = \log \frac{Total\ number\ of\ docs}{Number\ of\ docs\ containing\ term}$$

Переваги:

- Нові елементи легко додавати.
- Робо без «розігріву» вже з першими предметами

Мінуси:

- Поєднує вподобання з важливістю. Іноді схожі за вмістом фільми(наприклад з одинкавими акторами) будуть відрізнятися кардинально.
- Важко включити всі характеристики вмісту для того щоб зрозуміти чого саме хоче користувач

Колабораційні рекомендаційні системи

Колабораційні системи вміщують в собі оптимальні алгоритми рекомендацій які базуються на дуже простому принципі – рекомендуємо те що подобається іншим людям. Звісно які так чи інакше схожі на поточного користувача. Їх ще називаюдь сусідами.

Такі системи орієнтуються на оцінки від інших користувачів: 1-5, бінарні або інші.

Для представлення всіх речей які подобаються якомусь користувачу використовують матриці як числове представлення всього що до вподоби поточному користувачу.

Для вирішення таких задач існують різні алгоритми, їх можна віднести як користувач до користувача або ж предмет до предмету.

При підході користувач до користувача потрібно визначити схожість користувачів. Класичний спосіб оцінки. В цій роботі було розглянуто два таких підходу: відстань Евкліда та кореляція Пірсона.

При предметному підході, розглядаються предмети на основі векторів оцінок отриманих користувачами. Потім потрібно взяти список N схожих елементів які отримали таку ж оцінку. Такий підхід використовується коли користувачів значно більше ніж предметів. Наприклад Amazon.

Що таке функція подібності?

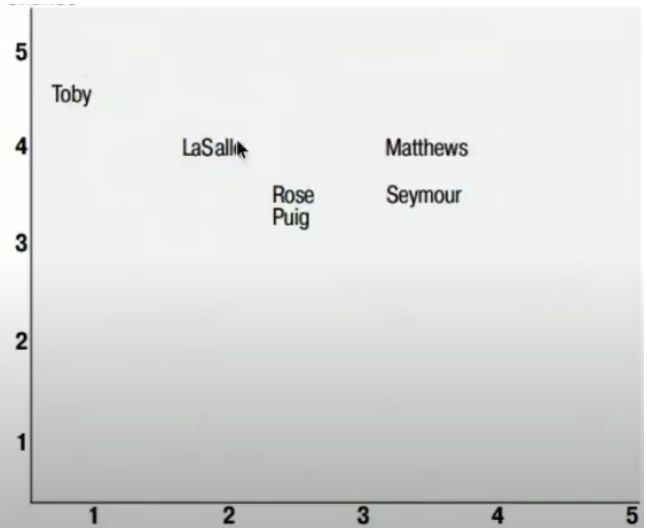
Обчислити подібність можна різними способами. Але в основному будуть два елементи та необхідно визначити функцію sim(i1, i2). Значення цієї функції зростає прямо пропорційна схожості елементів

Загалом можна сказати, що зв’язок між подібністю та віддаленістю такий:[5]

- Коли відстань збільшується, подібність наближається до нуля.
- Коли відстань наближається до нуля, подібність наближається до одиниці

Відстань Евкліда визначається як квадратна різниця сум оцінок під квадратним коренем. Якщо на графік нанести такі елементи то найбільш схожі будуть знаходитись поряд. [3]

$$d_{ij} = \sqrt{\sum_{k=1}^{m} \left(x_{ik} - x_{jk}\right)^2}$$



Кореляція Пірсона знаходиться в межах [-1,+1]. В цьому методі можна провести пряму на графіку. Якщо елементи будуть найбільш близкі до цієї лінії то два об’єкта можна вважати схожими

$$r = \frac{S_{xy}}{S_x S_y} = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \overline{y})^2}}$$

[3]

Недоліки:

- Необхідна достатня кількість користувачів, щоб знайти пару. Для подолання таких проблем із холодним пуском часто використовуються гібридні підходи між CF і відповідністю на основі вмісту.

- Навіть якщо є багато користувачів і багато елементів, які часто рекомендуються, можуть виникнути проблеми, пов’язані з розрідженістю матриці користувачів і оцінок, і буде складно дізнатися про користувачів, які оцінили той самий елемент.

Рекомендаційна система основана на популярності

Досить проста модель. Це тип системи рекомендацій, яка працює за принципом популярності та/або всього, що є в тренді. Ці системи перевіряють продукт або фільм, які є в тренді або найбільш популярні серед користувачів, і безпосередньо рекомендують їх

Наприклад, якщо продукт часто купує більшість людей, система дізнається, що цей продукт найпопулярніший, тому для кожного нового користувача , система рекомендуватиме цей продукт

Переваги системи рекомендацій на основі популярності: швидкий запуск, гарні рекомендації вже з першого дня. Немає необхідності в даних користувача

Недоліки системи рекомендацій на основі популярності: немає персоналізації. Тобто всім користувачам рекомендують одинакові речі. Наприклад «Тренди Ютуба»

Гібридні рекомендаційні системи

Гібридні рекомендаційні системи використовуються різні алгоритми рекомендацій для того щоб позбавитись слабкостей різних видів рекомендацій. Ви об’єднуєте алгоритми рекомендацій, щоб отримати більш потужний інструмент. Це не лише покращить результат і прибире кутові явища.

Розрізняють в літературі основні такі види[2]:

1. Зважені: бали, розраховані за різними компонентами рекомендацій об’єднуються в одне значення
2. Перемикання: система може вибирати між різними рекомендаціями компоненти, які потрібно застосовувати час від часу
3. Змішані: Рекомендації, створені різними компонентами рекомендацій, змішуються та подаються разом

Прикладом многолітньої систему можна вважати гібридну систему в якої може бути ще один додатковий етап попередньої обробки.

Змішаний гібрид повертає об’єднані результати двох алгоритмів. Наприклад створити першу рекомендацію максимально персоналізовану і інші на основі популярності. Часто персоналізовані рекомендації повертають кілька рекомендацій лише і таким чином можна підтримувати достатню кількість рекомендацій але трішки гіршої якості.

Реалізація рекомендаційної системи

Предметна область

Рекомендаційна система буде створюватись на основі датасету з сайту пошуку людей «за спільними інтересами».

В цьому датасеті я виділив особливу увагу таким таблицям як :

user_profile	
user_id	*unique user identificator*
height	*user`s height, cm,* ***numerical***
weight	*user`s weight, kg,* ***numerical***
hairColor	*(brown, blonde, etc),* ***categorical***
eyesColor	*(grey, blue, etc),* ***categorical***
bodyType	*(skiny, muscle, etc),* ***categorical***
appearance	*(asian, caucasian, etc),* ***categorical***
goalMoney	*life goal is money,* ***categorical***
goalRelation	*life goal is relationship,* ***categorical***
goalEvening	*life goal is entertainment,* ***categorical***
goalTravel	*life goal is travels,* ***categorical***
relationship	*(single, divorced, etc),* ***categorical***
children	*does user have children,* ***categorical***
alcohol	*attitude to alcohol,* ***categorical***
smoking	*attitude to smoking,* ***categorical***
job	*(finance, medicine, etc),* ***categorical***
ocupation	*ignore this feature*
interests	*user`s interests (tv, music, IT, etc),* ***categorical***

user_likes	*each pair of user_id - partner_id = unique like*
user_id	*unique user identificator, user that liked other user`s profile*
partner_id	*unique user identificator, owner of liked profile*

Перша таблиця необхідна для фільтрації на основі вмісту завдяки різним характеристикам користувача.

Завдяки другій таблиці я зможу застосувати колаборативну фільтрація та зрозуміти хто подобається іншим користувачам схожим на поточного.

Аналіз проблеми

Для того щоб побудувати якісну рекомендаційну систему потрібно використати гібридний підхід(в основному колабораційні алгоритми). Оскільки різних характеристик дуже багато і як сильно вони впливають на оцінку тої чи іншої людини в даний момент оцінити складно. Зате є вже стовідсоткові дані про те які користувачі подобаються іншим. Тож необхідно знайти схожих користувачів і далі вже застосувати двосторонню фільтрацію, для якої існують різні алгоритми, в цій роботі було використано комбінацію спосібів «Recommendations to groups»[4] та «Social recommendations» або «мудрість натовпу» [5]. Коротко про суть даного підходу: необхідно робити не індивідуальну рекомендацію (оскільки похибка може бути дуже високою) а створити невелику групу, наприклад схожих за інтересами людей, і побудувати рекомендації для кожного з них, і з тією вибірки за допомогою функції агрегації обрати найкращий варіант.

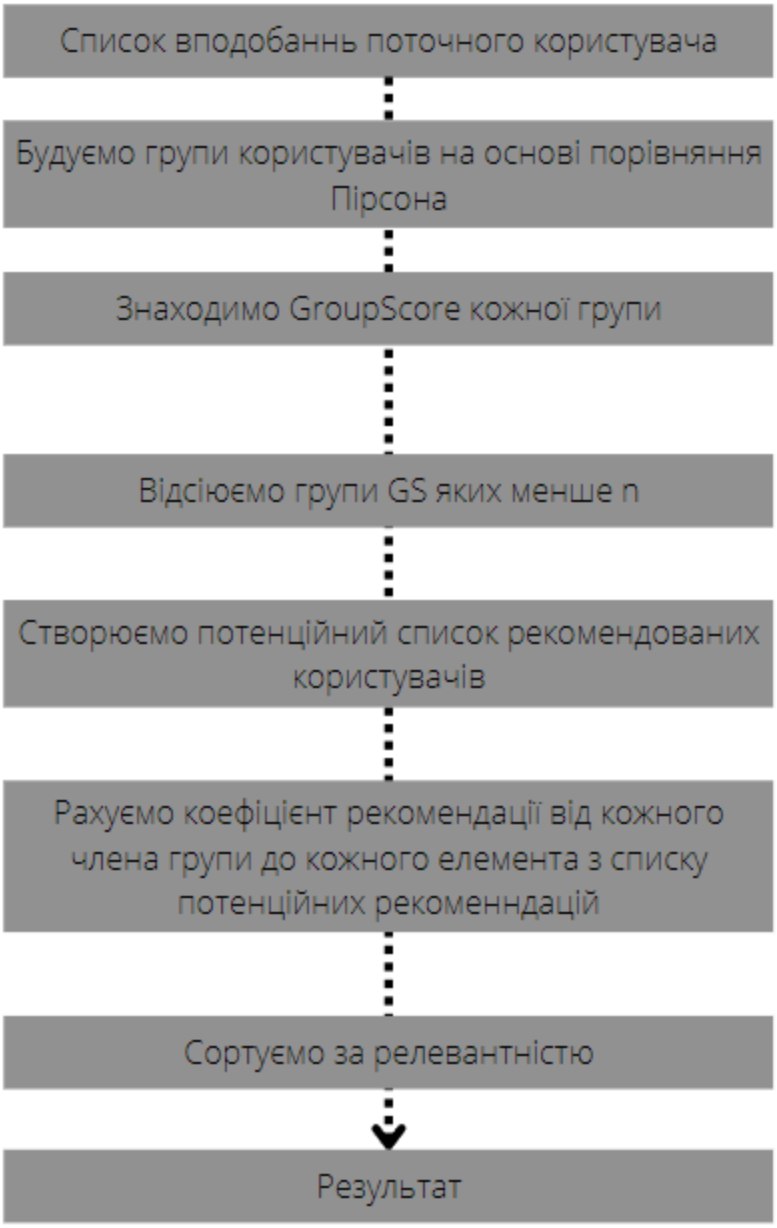
Тож двох-сторонньою рекомендаційну систему можна вважати таку, для якої рекомендація якогось предмета і співпадає з рекомендацією такого предмета в групі G.

Вибір алгоритму для реалізації двосторонньої системи.

Розглянемо детальніше алгоритми для побудови двох-сторонньої рекомендаційної системи.

Social Comparison-based method – цей метод заснований на соціальному порівнянні. Коли користувачі конфронтують один з одним та відсутня конкретна інформація про них. Для заданої предметної області алгоритм не підходить.

Group-priority recommendation – спочатку потрібно створити декілька груп користувачі, ідентифікувати найбільш оптимальну групу користувачів з цих. Далі застосовуємо функцію агрегації та шукаємо найбільше оптимальні варіанти.


Список вподобаннь поточного користувача
Будуємо групи користувачів на основі порівняння Пірсона
Знаходимо GroupScore кожної групи
Відсіюємо групи GS яких менше n
Створюємо потенційний список рекомендованих користувачів
Рахуємо коефіцієнт рекомендації від кожного члена групи до кожного елемента з списку потенційних рекоменндацій
Сортуємо за релевантністю
Результат

Використані технології

Структура роботи додатку: клієгь-серверне застосування.

Для виконання основної задачі було вирішено обрати Pyhon та бібліотеки pandas та Numphy. Ці бібліотеки дают змогу вільно користовутися великими даними а також мають деякі налаштовані функції кореляції.

Фронтенд частина – для введення необхідних параметрів написана на Angular. Виконистовував Antd design.

Наступні дані використовуються для аналізу

api/getRec/@userid number - Повертається список рекомендованих користувачів

height	*number*
weight	*number*
hairColor	*number*
eyesColor	*number*
bodyType	*number*
appearance	*number*
goalMoney	*number*
goalRelation	*number*
goalEvening	*number*
goalTravel	*number*
relationship	*number*
children	*number*
alcohol	*number*
smoking	*number*

job	*number*
ocupation	*number*
interests	*number[]*

Релізація

Крок 1. Підготовка даних.

Для початку створимо загальну таблицю. Було вирішено об’єднати параметри в 3 групи:

['height','weight','hairColor','eyesColor','bodyType','appearance'] – AppearanceGroup

['goalMoney','goalRelation','goalEvening','goalTravel','relationship','children','alcohol','smoking','job','occupation'] – LifeGroup

[‘interests'] – interestsGroup

Далі для кожного користувача будуємо масив користувачів які йому сподобалися. Тобто об’єднуємо дві таблиці

Потрібно також всім nan присвоїти 0 для того щоб далі простіше було працювати з системою

Крок 2. Реалізуємо алгоритм Пірсона. В предметній області існує таке поле я interests. В якому преліковані різні інтереси. Я вирішив їх переотворити зі звичайного переліку 1-15 в масив 0 якщо інтересу немає та 1 якщо є. Завдяки цьому алгоритм Пірсона став точніше відоражати схожих користувачів

def sim_pearson(user1_int, user2_int) - повертає число від -1 до 1

Крок 3. Реалізовуємо метод взаття n схожих елементів :

def top_similarBy(arr, user,sim_field, n=5, sim=sim_pearson) – повертає n користувачів з вагами, відсортованих

arr – композитний/основний масив

user – поточний користувач

sim_field – поточна група парметрів яка перевіряється

n – число схожих користувачів яких ми шукаємо.

sim – метод визначення схожості.

Крок 3. Описуємо функцію для створення різних груп користувачів. Як я вже вказав вище я вирішив брати користувачів в різні групи за різними показниками в профілі.

def getReccUserByField(person) – повертає масив рекомендацій для користувача

person – поточний користувач

Крок 4. Будуємо матрицю для всіх користувачів. Для цього виконуємо getReccUserByField для всіх користувачів.

Крок 5. Я вирішив об’єднати результати зі всих груп та вибрати ті які найчастіше попадаються в кожній групі – остаточний список рекомендацій.

Крок 6. Записуємо результат в словник.

```
56268  [43641276, 63065, 42800384, 42862073, 42852963...    56268
56672  [52862192, 32514224, 44687340, 53627686, 53628...    56672
59219  [54202454, 47144243, 44273260, 31027395, 31001...    59219
59504  [43016307, 31049276, 44687340, 47520729, 47404...    59504
60019  [63065, 42862073, 42845494, 42840953, 42840876...    60019
61344  [32514224, 46402729, 46461424, 46460771, 46455...    61344
61865  [32514224, 221449, 48206460, 48474688, 4846236...    61865
62825  [18544326, 32514224, 53889020, 48635811, 49657...    62825
63065  [275328, 43665417, 2765683, 65222, 47257491, 4...    63065
64892  [39502486, 43016307, 47150076, 47730710, 36521...    64892
65222  [37005593, 5906307, 43016307, 65222, 45961312,...    65222
66316  [44687340, 43665417, 48116572, 9089625, 227846...    66316
66822  [15714969, 65222, 48433987, 48653888, 48614342...    66822
67670  [18544326, 43665417, 43016307, 31049276, 43234...    67670
68054  [27387958, 32514224, 9089625, 33713680, 263507...    68054
69235  [22784633, 32514224, 37096835, 35557696, 47324...    69235
72825  [53701820, 51032220, 32514224, 34470969, 22784...    72825
74659  [32514224, 18544326, 2960844, 43051216, 485824...    74659
75497  [22784633, 39502486, 32514224, 47012313, 43665...    75497
75747  [22784633, 31049276, 43016307, 44687340, 43665...    75747
77612  [32514224, 15714969, 42297408, 131158, 4865719...    77612
81657  [5906307, 33713680, 43665417, 52862192, 395024...    81657
82099  [44687340, 48689280, 48679974, 48677030, 48665...    82099
82272  [39502486, 48614342, 48116572, 48991635, 32514...    82272
84560  [47012313, 47194688, 44687340, 47730710, 26219...    84560
86675  [32514224, 2960844, 48582473, 43051216, 63065,...    86675
87212  [53830598, 30207232, 52240182, 32514224, 52862...    87212
87417  [32514224, 65222, 53130374, 53260073, 53259148...    87417
88224  [63065, 42146217, 42106563, 42093891, 42088038...    88224
88419  [32514224, 18544326, 51521715, 38066764, 53886...    88419
91606  [1391046, 49101328, 42536659, 44164201, 495603...    91606
92102  [32514224, 18544326, 53227497, 1615600, 491013...    92102
93500  [22784633, 47012313, 38032816, 42704169, 44286...    93500
95491  [31049276, 22784633, 43665417, 18544326, 81004...    95491
95526  [32514224, 39502486, 40721148, 9150009, 538292...    95526
```

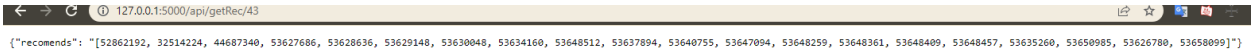
Далі за айді користувача можна дізнатись його список рекомендацій.

Список зберігається в файл, тож запити до сервера будут моментальні.

Rest Api

Зробив за допомогою фраемворку Flask – швидкий в налаштуванні та нормальний в роботі

/api/recSys/ 56672 Результат запиту:



{"recomends": "[52862192, 32514224, 44687340, 53627686, 53628636, 53629148, 53630048, 53634160, 53648512, 53637894, 53640755, 53647094, 53648259, 53648361, 53648409, 53648457, 53635260, 53650985, 53626780, 53658099]"}

Висновок

Тестування:

Перевіряв тестову вибірку на предмет співпадіння хоча б одного перегляду профіля в поточного користувача та його рекомендованих користувачів. Якщо такі співпадіння були то система працює та може використовуватися в цілях бізнесу.

Результат на 50 користувачах - 0.002331620206737163

Результат на 100 користувачах - 0.004331620206737163

Результат на 200 користувачах - 0.005287241267585081

Результат на 500 користувачах - 0.012349284205820175

Коефіцієнт розраховувався як (рекомендовані користувачі U всі корстувачі яких переглядала ця людина) довжина / всі користувачі яких переглядала ця людина

Тому коефіцієнт такі маленькі. Проте на мою думку це непоганий результат.

Висновки

Зі збільшенням інформації рекомендаційні системи все більше і більше потрібні. В наш час майже жодна покупка не проходить без рекомендацій.

В цій роботі я виділив основні проблеми різних підходів для створення рекомендаційних систем. Також описав які є підходи для вирішення тих чи інших проблем. Одне можна сказати точно, що ідеальної системи не існує і в наш час і в цей напрямок бізнесмени інвестують чималі гроші для того щоб покращити якість рекомендацій хоча б на декілька відсотків.

Також я реалізував тестову систему засновану на двосторонньому аналізі користувачів або ж груповому аналізі. Для калібрування системи необхідно протестувати які параметри сильно змінюють результати системи та змінити відповідні ваги в системі.

Реалізував restfull додаток для отримання рекомендацій по id користувача.

З мінусів моєї системи виділю, що немає миттєвої підтримки нових користувачів та доволі громіздка реалізація. Також необхідно підкорегувати ваги деяких параметрів.

З плюсів виділю швидкодію та загальну роботу системи.

Додадки

```
import numpy as np
import statistics
import scipy
import math
import random
import requests
from flask import Flask
from flask_restful import Resource, Api

app = Flask(__name__)
api = Api(app)
res = [];


def sim_pearson(user1_int, user2_int):
    equal1 = []
    equal2 = []
    for i in range(20):
        equal1.append( 1 if (i in user1_int) else 0)
        equal2.append( 1 if (i in user2_int) else 0)
    v1 = pd.Series(equal1)
    v2 = pd.Series(equal2)
    res = v1.corr(v2)
    return 0 if (res == np.nan) else res;

def top_similarBy(arr, user,sim_field, n=5, sim=sim_pearson):
    scores = []
    scores = pd.DataFrame(columns=['coef','myUserInterest', 'interests',
'user_id', 'like_list'])
    for index, row in arr.iterrows():
        if row[sim_field] != user:
            scores.loc[index] = (sim(user,
row[sim_field]),user,row[sim_field],row['user_id'], row['like_list'])
    scores = scores.sort_values(by=['coef'], ascending=False)
```

```
    return scores.head(n)

def getReccUserByField(person):
    groups = []

    res = []
    for i in ['interests', 'appearanceGroup','lifeGroup']:
        groups.append( top_similarBy(all_table,person[i],i))
    for g in groups:
        m = statistics.mean(g['coef'].values.tolist())
        if m > 0.5:
            res = res + g['like_list'].values.tolist()

    res = [item for sublist in res for item in sublist]
    resDF = pd.DataFrame(columns=['user_id']);
    resDF['user_id'] = res;
    resDF =
resDF.groupby(['user_id'])['user_id'].count().reset_index(name="count").sort_
values(by=['count'], ascending=False)
    res = (resDF.head(20)['user_id'].values.tolist())
#     print(res)
    return res

def main(arr):
    my = []
    df = pd.DataFrame(data=[],index=arr['user_id'].values.tolist(),
columns=['user_rec'])

    for index, row in arr.iterrows():
        r = getReccUserByField(row)
        df.loc[row['user_id']] = [r]

    df['user_id'] = arr['user_id'].values.tolist()

    return df
```

```
# Prepare date
df1 = pd.read_csv('data/user_profile.csv')[['user_id', 'interests',
'height','weight','hairColor','eyesColor','bodyType','appearance','goalMoney'
,'goalRelation','goalEvening','goalTravel','relationship','children','alcohol
','smoking','job','occupation']];
df1 = df1.fillna(0)
print(len(['user_id']))
df1['appearanceGroup'] = df1.loc[:,
['height','weight','hairColor','eyesColor','bodyType','appearance']].values.t
olist()
df1['lifeGroup'] = df1.loc[:,
['goalMoney','goalRelation','goalEvening','goalTravel','relationship','childr
en','alcohol','smoking','job','occupation']].values.tolist()
df1 = df1.loc[:, ['user_id', 'interests','appearanceGroup','lifeGroup',]]
df1['interests'] = df1['interests'].apply(lambda x1:[]  if (x1 == 0)  else
[int(x) for x in x1.split(',')])

df2 = pd.read_csv('data/user_likes.csv');
dft =
df2.groupby('user_id')['partner_id'].apply(list).reset_index(name='like_list'
)

all_table = df1.merge(dft)
all_table = all_table.reset_index()  # make sure indexes pair with number of
rows

# res = (main(all_table))
# print(res)
# dfCheck = pd.read_csv('data/user_profile_view.csv');
# dfCheck =
dfCheck.groupby('user_id')['partner_id'].apply(list).reset_index(name='view_l
ist')
# res = res.merge(dfCheck)
# r = []
# for index, row in res.iterrows():
#     ll = 0
#     j = 0
```

```
#     for w in row['user_rec']:
#         ll += row['view_list'].count(w)
#     r.append(ll/len(row['view_list']))

# print(statistics.mean(r))

if __name__ == '__main__':
    res = (main(all_table))
    print(res)
    dfCheck = pd.read_csv('data/user_profile_view.csv');
    dfCheck =
dfCheck.groupby('user_id')['partner_id'].apply(list).reset_index(name='view_l
ist')
    res = res.merge(dfCheck)
    r = []
    for index, row in res.iterrows():
        ll = 0
        j = 0
        for w in row['user_rec']:
            ll += row['view_list'].count(w)
        r.append(ll/len(row['view_list']))

    print(statistics.mean(r))
    app.run(debug=False)


class HelloWorld(Resource):
    def get(self):
        return {'hello': 'world'}
class GetRec(Resource):
    def get(self, user_id):
        print(res)
        return {"recomends": res.loc[user_id].values.tolist()}
api.add_resource(HelloWorld, '/')
api.add_resource(GetRec, '/api/getRec/<int:user_id>')
```

Список літератури:

1. Double-Sided Recommendations: a novel framework for exploiting social networking in recommender systems(23p.) Fabiana Vernero1. Double-Sided

2. Recommendations: a novel framework for exploiting social networking in recommender systems(31p.) Fabiana Vernero

3. Practical Recommender Systems KIM FALK (155p)

4. Recommendations: a novel framework for exploiting social networking in recommender systems(31p.) Fabiana Vernero

5. Recommendations: a novel framework for exploiting social networking in recommender systems(47p.) Fabiana Vernero

6. Practical Recommender Systems KIM FALK (178p)