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

До захисту допущено
Завідувач кафедри
_____ Оксана ТИМОЩУК
«___» _____2023 р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Системний аналіз і управління»
спеціальності 124 «Системний аналіз»
на тему «Система надання рекомендацій методами глибоких нейронних мереж»

Виконав:
Студент IV курсу, групи КА-91
Панаско Віталій Євгенович _____

Керівник:
професор, докт. техн. наук Недашківська Н.І.

Консультант з економічного розділу:
доц., к.е.н. Рощина Н. В. _____

Консультант з нормоконтролю:
доц., к.ф.-м.н. Статкевич В. М. _____

Рецензент:
професор, докт. техн. наук Данилов В.Я. _____

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

Київ 2023

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
НН 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
Завідувач кафедри
_____ Оксана ТИМОЩУК
«___» _____2023 р.

ЗАВДАННЯ
на дипломну роботу студента
Панаска Віталія Євгеновича

1. Тема роботи «Система надання рекомендацій методами глибоких нейронних мереж», керівник роботи професор, докт. техн. наук Недашківська Н.І., затверджені наказом по університету від «30» травня 2023 р. №2065-с

2. Термін подання студентом роботи – 12.06.2023 р.

3. Вихідні дані до роботи: оцінки взаємодій, таблиці характеристик

4. Зміст роботи: дослідження та порівняння наявних та імплементованих методів надання рекомендацій за допомогою метрик якості

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

6. Консультанти розділів роботи:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Рощина Н.В., доцент		

7. Дата видачі завдання 17.04.2023 _____

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Затвердження теми ДР	17.04.2023-21.04.2023	виконано
2	Ознайомлення зі структурою БДР згідно з Положенням про державну атестацію студентів НТУУ «КПІ ім. І. Сікорського»	17.04.2023-21.04.2023	виконано
3	Ознайомлення з ДСТУ 3008–95 та стандарти ЄСПД	21.04.2023-28.04.2023	виконано
4	Проведення дослідження за темою БДР під керівництвом керівника	28.04.2023-05.05.2023	виконано
5	Завершення роботи над першим варіантом частини БДР	05.05.2023-12.05.2023	виконано
6	Проведення роботи над експериментальною частиною БДР	05.05.2023-12.05.2023	виконано
7	Проведення роботи над програмним продуктом	12.05.2023-19.05.2023	виконано
8	Оформлення БДР та аналіз отриманих результатів	12.05.2023-21.05.2023	виконано

Студент Віталій ПАНАСКО

Керівник Надія НЕДАШКІВСЬКА

РЕФЕРАТ

Дипломна робота містить 149 с., 45 рис., 33 табл., 2 дод., 40 посилань.

Ключові слова: РЕКОМЕНДАЦІЙНА СИСТЕМА, ГЛИБИННЕ НАВЧАННЯ, МАТРИЧНА ФАКТОРИЗАЦІЯ, КОЛАБОРАТИВНА ФІЛЬТРАЦІЯ, ГІБРИДНА РЕКОМЕНДАЦІЙНА МОДЕЛЬ, ЗАДАЧА РАНЖУВАННЯ, ЛАТЕНТНЕ ПРЕДСТАВЛЕННЯ, МЕТОД DSSM, ПОСЛІДОВНІ НЕЙРОННІ МЕРЕЖІ

Об’єкт дослідження: рекомендаційні системи, реалізовані методами глибинного навчання.

Мета дослідження: порівняння ефективності наявних методів рекомендацій з новими методами, що використовують для надання рекомендацій нейронні мережі.

Використані моделі: модель матричної факторизації методом SVD, модель матричної факторизації методом ALS, модель матричної факторизації методом LightFM, модель DSSM, модель GRU, модель GRU4Rec, модель Caser.

Отримані результати: програмно реалізовано систему надання рекомендацій методами DSSM та GRU за допомогою Python з можливістю включати в моделі різні архітектури, оцінено точність моделей за множиною метрик якості.

В рамках подальшого дослідження пропонується розглянути роботу реалізованих методів на інших вибірках, що представляють більш широкий спектр представлень даних, а також експерименти з вбудованими в моделі архітектурами.

ABSTRACT

The diploma thesis contains 149 pages, 45 figures, 33 tables, 2 appendices, 40 references.

Keywords: RECOMMENDATION SYSTEM, DEEP LEARNING, MATRIX FACTORIZATION, COLLABORATIVE FILTERING, HYBRID RECOMMENDATION MODEL, LEARNING TO RANK, EMBEDDINGS, DSSM METHOD, RECURENT NEURAL NETWORKS

Object of research: recommendation systems implemented by deep learning methods.

Purpose: comparison of the efficiency of methods which provide recommendations using neural networks with existing recommendation methods.

Used models: SVD matrix factorization model, ALS matrix factorization model, LightFM matrix factorization model, DSSM model, GRU model, GRU4Rec model, Caser model.

Results: a system for providing recommendations using DSSM and GRU methods using Python with ability to include different architectures in the models, the accuracy of the models was evaluated according to a number of quality metrics.

As part of further research, it is proposed to consider the performance of the implemented methods on other samples representing a wider range of data representations, as well as experiments with architectures built into the models.

ЗМІСТ

ВСТУП

Тема штучного інтелекту на сьогоднішній день є актуальною та цікавою. З розвитком технологій та збільшенням використання методів машинного навчання, люди змогли прискорити рутинні процеси, а деякі навіть замінити. Постає питання можливості повної автоматизації діяльності людини, яке можна сформувати так: «чи може штучний інтелект повністю замінити роботу, яку ми звикли виконувати кожен день, і створити нам ідеальні комфортні умови, коли нам взагалі не доведеться витрачати свої ресурси: часові, фізичні та навіть розумові?».

Відповідь на це питання: «ні, не може; принаймні найближчим часом». Кожна людина є особливою, наш спосіб мислення є фактором, що з одного боку допомагає нам аналізувати інформацію, з іншого – є нашою унікальною особливістю, яка вирізнятиме нас від інших. Спосіб мислення буде задавати наш світогляд, наші інтереси, наші вподобання. Штучний інтелект, безумовно, може бути корисним інструментом для людини, але як додатковий фактор в її діяльності: так чи інакше ми маємо задавати спосіб роботи для алгоритму, де це можливо, або правильно інтерпретувати результат, який отримали.

У випадку створення рекомендацій, алгоритм не може працювати без взаємодії з людиною. В роботі ми розглянемо складні архітектури, що передбачають кодування будь-якої наявної інформації, але самі по собі ці моделі не будуть корисними – спочатку користувачі мають надати інформацію про взаємодію. Надалі алгоритми будуть працювати лише з цією інформацією, робити по ній висновки та створювати рекомендації. Те, що буде рекомендувати нам система – буде залежати лише від нас, від нашого способу мислення.

Залишилось лише зрозуміти, як представити себе системі: зібрана інформація може мати різну природу, а тому архітектура системи має бути

гнучкою та ефективною. Нейронні мережі задовольняють вказані вимоги, вони широко використовуються в сучасних практичних задачах, адже вважаються певним довершенням тих алгоритмів, які існували до. В роботі перевіримо наскільки якісним є новий підхід в сфері рекомендацій та як він співвідноситься з іншими, вже відомими методами.

1 ОСНОВНІ ТЕОРЕТИЧНІ ПОЛОЖЕННЯ

1.1 Вимоги до сучасних рекомендаційних систем

Сучасний світ важко уявити без рекомендацій. Інтернет-магазини, онлайн-застосунки, соціальні мережі – все побудовано на певних системах, що дають рекомендації – певну допомогу у виборі. Генерувати хороші рекомендації – критично для сучасного продукту, який хоче буде широко використованим. Хороша рекомендаційна система стає важливою особливістю продукту, яка буде вигідно виділяти його на фоні схожих по функціоналу, але гірше продуманих застосунків.

Чому рекомендації є актуальними? Вони є корисними для людини та можуть задовольнити певні її потреби. Враховуємо сучасне різноманіття вибору послуг, а також певні поведінкові особливості [1], або патерни, такі як інтереси певної соціальної групи. Рекомендації, та особливо хороші рекомендації, стали важливим запитом у світі, оскільки людина:

- не може тримати в голові всі можливі та схожі варіанти;
- інколи сама не знає, що вона хоче обрати;
- як правило має схожі інтереси із соціальною групою, до якої вона належить;
- а отже схильна обирати те, що обирає група;

Це аксіоми будь-якої рекомендаційної системи. На них будуються як прості, так і складні моделі. Сама складність моделі у деякому сенсі визначається тим наскільки добре вона слідує цим визначеним поняттям. Проте це далеко не повний перелік, що необхідно врахувати.

По-перше, не можемо не враховувати побажання користувача, те з цим він вже «провзаємодіяв» в минулому. Робиться фундаментальне припущення, що ці взаємодії є інформативними, тобто людина дійсно цікавилася цим фільмом/товаром/будь-яким об’єктом, який її зацікавив. А значить щось схоже їй також буде подобатися. Тут варто пояснити деякі моменти.

Користувач міг вибрати об’єкти випадково і насправді ними не цікавився. Ми не можемо врахувати все, з одного боку технологічно, а з іншого – в питанні обчислень: зробити це буде дуже важко, У випадку «випадкових» взаємодій система буде рекомендувати навіть « нерелевантні для користувача об’єкти. Ключовий момент: щоб зменшити кількість потенційних обчислень будемо орієнтуватися тільки на наявні взаємодії та припускатимемо, що вони є невипадковими та дійсно задають інтереси користувача. В будь-якому випадку, покращення якості представлення взаємодій – задача відмінна від тої, що розглядається в роботі.

Крім цього робиться логічне припущення, що користувач не може мати лише однакові інтереси: одна категорія, один вид послуг, або взагалі однаковий товар. Люди хочуть, як правило, певної різноманітності у виборі. Якщо в рамках одної категорії користувач вже вибрав свій певний «ідеальний» об’єкт і не хоче його змінювати, то для інших категорій варіанти рекомендацій можуть його зацікавити.

Також важливою особливою особливістю є потреба рекомендувати об’єкт, з яким користувач вже про взаємодіяв: запит на таку рекомендацію варіюється залежно від області, яку розглядаємо. Пам’ятаємо, що основна задача загальної рекомендаційної системи – згенерувати нові рекомендації по наявній історії взаємодії. У випадку, коли користувач знає, що хоче та наприклад може купляти декілька разів один і той самий товар ми можемо надати йому «вже відому» рекомендацію. Проте це може бути і «динамічна» рекомендація (наприклад рекомендація фільмів, де користувачі хочуть

взаємодіяти з новими об’єктами-фільмами); тут використовується принципово інший підхід, ми нехтуємо з минулих взаємодій користувача. Рекомендаційну систему у випадку усталених та стабільної сфер, де люди мають сформовані та чіткі інтереси, також називають експертною системою, вона є простим підвидом загальних рекомендаційних систем.

По-друге, серед всіх наявних взаємодій варто вибрати найбільш та найменш релевантні. Ця ідея є загальною для всіх областей, навіть вже усталених. Тут робиться акцент не на різноманітності, а на тому, що з плином часу:

- можуть з’явитися об’єкти, які є логічним довершенням об’єктів, з якими ми провзаємодіяли;
- змінилися наші інтереси;
- змінилася категорія, яка нас цікавить;

По суті ми додаємо ще один вимір – час. Усі перераховані фактори органічно входять в наше життя. Великі компанії, що перш за все розвиваються за рахунок нових технологій, будуть намагатися створити все кращий продукт, розуміючи роль конкуренції на ринку. Навіть кінокомпанії чи режисери, так чи інакше, прагнуть отримати кращі касові збори чи вищу оцінку. Світ не стоїть на місці, а постійно розвивається. Ростуть і вимоги до продукту, а значить в полі зору людини будуть потрапляти більш «довершені» об’єкти. Так вирішила людина і ринок пристосовується до її запитів.

Також варто сказати про динамічні інтереси людини: з плином часу розширюється її спектр інтересів, тому людина може змінити свої погляди/судження/вимоги до об’єкту. Можливо вона спробувала щось нове і їй сподобалось і тепер просто немає сенсу повертатися до об’єктів із старих взаємодій.

На процес докорінної зміни категорії можуть впливати творці продукту. Інколи вони прагнуть певним чином «шокувати» наявні ринки та змінити

категорію, додавши нові особливості. Інколи це призводить до створення нової категорії товарів, яка починає конкурувати з наявною. Логічним наслідком цього фактору є нові вимоги до продукту від користувача, але на відміну від варіанту з «довершенням», тобто вертикальним ростом, тут спостерігаємо горизонтальний ріст, тобто принципово нову категорію.

Крім цього додам, що сама взаємодія, навіть без врахування часу може нести певну інформацію про релевантність. Самий простий приклад – оцінки, по суті ми переходимо від простого індикатора (взаємодія чи не взаємодія) до метрики якості взаємодії. Тут можна експериментувати і комбінувати (розділ 1.3) наявні дані.

Також важливими гіпотезами є етичність системи та релевантність або актуальність для різних соціальних груп.

З одного боку ми хочемо, щоб система не давала протизаконних рекомендацій, тобто слідувала нормам законодавства [2]. З іншого, навіть там де законодавство не діє, не порушувала соціальний порядок. У випадку порушення прав певної соціальної групи з боку рекомендаційної системи ситуація отримає резонанс і буде загроза серйозної шкода репутації самого продукту та команді, яка його створила [3, 4]. Тому треба бути досить уважним та перед глобальним запуском протестувати систему на предмет етичних непорозумінь.

Окрім цього рекомендаційна система має бути релевантною для різних соціальних груп. Цей фактор є певним ідеалом для команди-розробника: система створює рекомендації однаково добре для усіх. Проте на практиці це далеко не так: на перший план виходять розмір вибірок, наскільки динамічні їх інтереси, інший контекст. Завжди буде група, з якою буде проблем найбільше, це природній фактор. В такому випадку команда розробки буде прагнути підвищити точність до певного середнього рівня і це буде вважатися хорошим результатом.

Також додам, що хороший продукт, що використовує рекомендаційну систему, має мати чітку мету і не відхилятися від неї. Дуже часто недоліки продукту, такі як реклама, поганий дизайн чи інтерфейс стають критичними та применшують позитивний вплив рекомендаційної системи, над якою довго працювали справжні професіонали-розробники. Тому не варто з головою «занурюватися» в розробку, нехтуючи іншими важливими факторами, а чітко перерозподілити ролі. Запорука успішності хорошого продукту, що використовує рекомендаційну систему – в його органічності та розумінні свого користувача.

1.2 Актуальність проблеми

Немає ніякого сумніву, що рекомендаційні системи є актуальними для користувачів. Можна вибрати хоча б одну з наступних причин:

- не потрібно пам’ятати про всі можливі варіанти;
- економія часу на пошук потенційних варіантів;
- персоналізовані рекомендації – неочевидна причина, коли користувач починає відчувати свою «особливість» від інших, а отже отримує задоволення від користування продуктом.

Остання причина цікава, адже головну роль грає вже не функціонал системи, а її позиціонування в очах користувача. По суті, рекомендаційна система стає окремим брендом, який повністю визначає продукт. Це добре працює у зв’язці з користувачами: вони знають, що система робить хороші рекомендації саме під них, а тому готові запросити в застосунок своїх друзів (нові користувачі) або просто провести більше часу в продукті, що також є хорошим сигналом. Фактично, ми розглянули вплив рекомендацій з іншої

сторони: важливість для бізнесу. Крім того, що рекомендаційна система, дійсно, може збільшувати обсяги продаж, допомагаючи користувачам витрачати більше, коли вона виконує свою безпосередню функцію – генерувати рекомендації, вона може бути інструментом для аналізу певних соціальних груп, їх вподобань та інтересів. Така інформація може бути корисною для оптимізації бізнес-процесів, переорієнтування на певну категорію. Тому вважаю, що актуальність для бізнесу точно є.

Сам ринок продуктів-рекомендаційних систем стрімко розвивається. Можна оцінити поточну капіталізацію ринку та оцінку її динаміки до 2028 року (рис. 1.1):

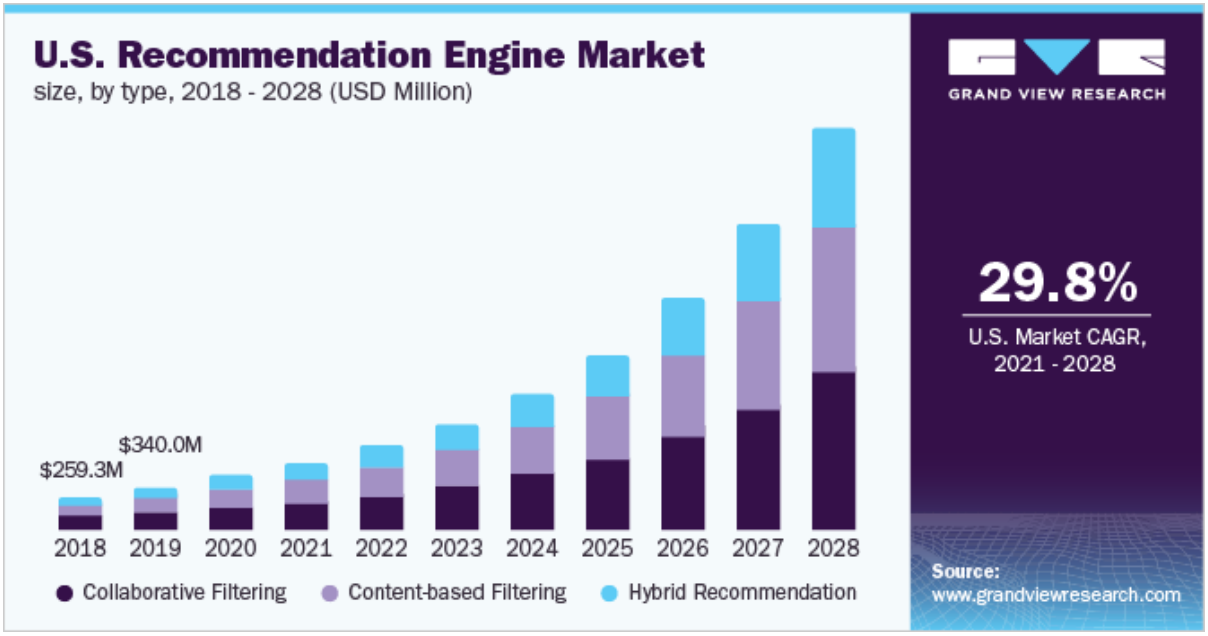


Рис.1.1 Динаміка росту загальної капіталізації ринку США для продуктів, що використовують рекомендаційні системи різних типів; прогноз до 2028 року [5].

Ріст – експоненційний, а показник росту $CAGR$ [5] – дуже високий. Основним драйвером для росту попиту стала пандемія COVID-19, коли люди не маючи змоги купувати товари в магазинах, звернули увагу на електронну

комерцію (рис. 1.2). Таким чином це дало змогу зібрати певну інформацію про історію їх взаємодій та побудувати на їх базі моделі з рекомендаціями.

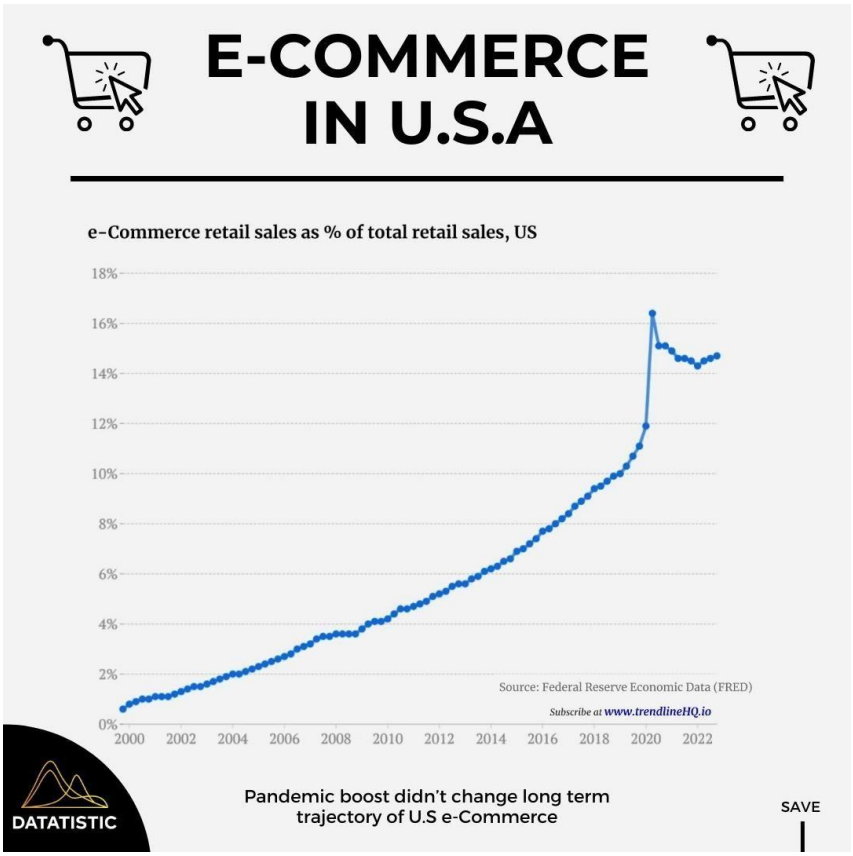


Рис.1.2 Динаміка частки роздрібних продаж через електронну комерцію порівняно з обсягом усіх розрібних продаж для ринку США за 20 років

Тому, дійсно, можна говорити про те, що запуск застосунку на базі добре продуманої та розробленої рекомендаційної системи виглядає привабливою ідеєю на перспективу. Така система має бути реалізована згідно всіх аксіом, сформованих в попередньому пункті, а також враховувати певні технічні моменти: значні обсяги пам’яті та потужне програмне забезпечення.

1.3 Постановка задачі

Розглянемо основні сутності:

U – множина користувачів, або інших суб'єктів рекомендації

I – множина об'єктів, які будемо рекомендувати

Y – простір опису взаємодій суб'єктів та об'єктів рекомендації

$D = (u_t, i_t, y_t)_{t=1}^{m} \in U \times I \times Y$ – дані для всіх взаємодій

$R = \|r_{ui}\|$ – матриця взаємодії розміру $|U| \times |I|$, *де* $r_{ui} = aggr\{(u_t, i_t, y_t) \in D \mid u_t = u, i_t = i\}$, а $aggr$ – спосіб агрегації.

Будемо вирішувати наступні задачі:

- прогнозування незаповнених клітинок r_{ui} в матриці взаємодії;
- формування списку рекомендацій для u та для i, використовуючи міру близькості p;

Без втрати загальності надалі суб'єктів рекомендацій, тобто такі сутності для яких будемо створювати рекомендації, окрім випадку item2item рекомендації [20], будемо називати *користувачами*, а об'єкти рекомендації, які будемо рекомендувати – *об'єктами*.

Висновки до розділу 1

Рекомендації важливі, як для користувача, так і для бізнесу. Проте варто враховувати особливості, певні виклики як в етапі розробки, так і в процесі їх створення. Не можна не враховувати певні поведінкові особливості людей, які задають, те як людина мислить, що вона хотіла б бачити чи не бачити у своїх

стрічці. Також існують основні правила, вимоги, яким має слідувати будь-яка рекомендаційна система – це сформовані (розділ 1.1) аксіоми. З точки зору бізнесу логіка проста: якщо система зацікавить увагу людини, вона буде користуватися нею не один раз; це впливає на продукт, оскільки ми отримаємо лояльного користувача. Крім цього ринок продуктів-рекомендаційних систем швидко росте, а тому це хороший варіант для створення компанії, що спеціалізується на створенні саме рекомендацій.

2 ОГЛЯД МАТЕМАТИЧНИХ МЕТОДІВ ВИКОРИСТАНИХ В РОБОТІ

2.1 Content-based filtering

Це підвид фільтрації, що ґрунтується рекомендації об’єктів, схожих до тих, який користувач вже вибрав. При цьому зв’язки з іншими користувачами не враховуються, будується лише певне середнє представлення про вподобання одної людини з припущення, що вона буде знати, що їй подобається. Пара (користувач, об’єкт) представляється або індикатором взаємодії (1 або 0), або певною оцінкою – це можуть бути дискретні або неперервні величини, такі як оцінка користувача для об’єкта, або наприклад кількість часу, яку користувач провів у взаємодії з об’єктом. Задача: для заданого користувача спрогнозувати найвищу оцінку серед об’єктів, з якими він не взаємодіяв лише по історії його взаємодій та дати рекомендацію того об’єкта, що матиме найвищу оцінку.

Опишемо цей алгоритм математично: зафіксуємо лише одного користувача, припустимо він має певні оцінки. Для того, щоб точно підбирати об’єкт нам необхідно буде розглянути його характеристики – це можуть бути як реальні його характеристики, так і його певне латентне семантичне представлення (розділ 2.3.4). Для прикладу наведемо наступну матрицю характеристик фільмів (табл. 2.1) – це будуть наші об’єкти та оцінки фільмів від користувача (табл. 2.2):

Таблиця 2.1 Приклад представлення матриці-характеристик фільмів

	Комедія	Пригоди	Супер-герої	Науковий
Фільм 1	0	1	1	1
Фільм 2	1	1	1	0
Фільм 3	1	0	1	0

Таблиця 2.2 Приклад представлення оцінок фільмів певним користувачем

Фільм 1	2
Фільм 2	5
Фільм 3	4

Далі порахуємо зважене значення для користувача, поєднавши його оцінки та характеристику кожного фільму перемноживши вектор-характеристику фільму на оцінку фільму.

$$M = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix}, u = \begin{pmatrix} 2 \\ 5 \\ 4 \end{pmatrix}, M \odot u = \begin{bmatrix} 0 & 2 & 2 & 2 \\ 5 & 5 & 5 & 0 \\ 4 & 0 & 4 & 0 \end{bmatrix} \xrightarrow[\text{і нормуємо}]{\text{додаємо рядки}}$$

$$[0.31 \quad 0.24 \quad 0.38 \quad 0.07]$$

Після агрегації векторів ми отримуємо середнє значення «вподобання» для конкретного користувача – це буде його представлення. Далі виберемо фільм який найбільш «схожий» за своїм представленням з представленням користувача. Алгоритм є дуже простим, але певні його модифікації можуть мати місце в сучасних рекомендаційних системах.

2.1.1 Недоліки алгоритму контентної фільтрації

Основною проблемою є «холодний старт» – відсутність достатньої кількості інформації на початку для створення якісних рекомендацій. Уявимо ситуацію, що ми додаємо в модель нового користувача, про якого є мало що відомо: не заповнені метадані, тобто характеристики, історія взаємодій є малою та недостатньою для чіткого формування списку інтересів. Та сама логіка працює може бути зворотною: для об’єкта можуть додаватися нові характеристики, яких просто не було в моделі – це «холодні» ознаки, чи з об’єктом ще просто не встигли провзаємодіяти. Для даного алгоритму у випадку «холодних» ми просто не зможемо додати їх модель без перерахування та уточнення матриці об’єкт-характеристика (табл. 2.1). А для нових користувачів просто не зрозуміло, що робити, адже оцінок немає. Придумати рішення можна, з цим справляються більш складні алгоритми, такі як наприклад LightFM [10] (розділ 2.2.3).

Нехтування взаємодією інших користувачів. Логіка алгоритму припускає той факт, що користувач знає, що він шукає і що хоче. Важливе уточнення: все відбувається лише в рамках тих категорій, де була взаємодія. Проте, знову ж таки, повертаючись до теми поведінкових особливостей людини (розділ 1.1), ми можемо зробити висновок, що людям притаманно вибирати те, що вибирають члени певної соціальної групи, до якої вони належать. Слідування підходу «експертної» системи – коли ми заповнюємо свої вподобання-фільтри і система однозначно підбирає нам об’єкт по введеним вимогам призводить до так званої проблеми «нудних» рекомендацій - модель пам’ятає лише те, що ми вибрали, повністю відкидаючи ідею, що нам може сподобатися, щось принципово інше, те, з чим вже взаємодіяли люди. Це суперечить наведеному запиту людини про

«різноманітність» у виборі, але, як ми вже з’ясували, таке рішення може варіюватися залежно від області.

Крім цього не враховується також і часова залежність. Це аксіома про релевантність, коли більш пізна оцінка має бути більш релевантною ніж більш рання.

Додам, що вказані проблеми не є особливістю лише вказаного типу рекомендаційних систем та можуть зустрічатися в будь-якій реалізації систем. Одні архітектури будуть справлятися з вказаними недоліками краще, інші – гірше.

2.2 Колаборативна фільтрація

Колаборативна фільтрація (CF) – більш загальний та принципово інший підхід, тут ми враховуємо зв’язки між користувачами. На відміну від попередньо згаданих експертних систем ми слідуємо аксіомі про те, що об’єкти, які подобаються певній групі людей, будуть подобатися кожному її члену та новим користувачам, які схожі з цією групою по своїй взаємодії. Важливе уточнення, що інтереси конкретного користувача система не «запам’ятовує», хоча і враховує, адже вважає, що ця інформація певним чином записана в поведінці інших користувачів. З одного боку така логіка має місце бути, проте ми лишаємося можливості давати експертні «персоніфіковані» рекомендації, що було б корисним при додаванні нового користувача: для нього немає історії взаємодії та оцінити його «схожість» на інші групи важко. Класичний підхід колаборативної фільтрації використовується на практиці, вважаючись моделлю, з якою варто починати створювати рекомендації. Описану проблему «холодних» користувачів можна вирішити гібридними методами рекомендацій:

поєднанням методу колаборативної фільтрації з контекстною фільтрацією (розділ 2.1) чи іншими видами, що і використовується на практиці.

Загальна модель алгоритму схожа з контекстним або експертним методом, нас цікавить взаємодія у вигляді користувач-об'єкт. Проте тут ми розглядаємо не просто вектор з оцінками конкретного користувача, а оцінки всіх користувачів. Це зручно зображувати у вигляді матриці або таблиці (табл. 2.3):

Таблиця 2.3 Приклад матриці взаємодії в колаборативній фільтрації

	Фільм 1	Фільм 2	Фільм 3	Фільм 4	Фільм 5	Фільм 6
Користувач 1	4	?	?	?	5	?
Користувач 2	4	5	?	5	?	?
Користувач 3	?	5	?	4	?	3
Користувач 4	5	4	?	?	?	?

Наша задача: спрогнозувати порожні значення в матриці (розділ 1.3), в даному випадку оцінки фільмів, які користувач ще не подивився.

2.2.1 Різні підходи в колаборативній фільтрації

Memory-based підходи: user-user та item-item – шукаємо схожих користувачів та схожі об'єкти; задача в чомусь схожа на контенту фільтрацію. Для першого варіанту треба дати рекомендацію по середній оцінці користувачів, яку вони виставили. В item-item ми навпаки підбираємо схожі об'єкти і по ним формуємо оцінку. Міра близькості може бути будь-якою:

косинусна відстань, коефіцієнт Пірсона чи наприклад жакардова близькість (добре працює коли в нас закодовані категоріальні ознаки):

$$S(u,v) = \frac{|I(u) \cap I(v)|}{|I(u) \cup I(v)|} \quad (2.1)$$

Перевага в швидкодії роботи, без прогнозування всіх порожніх значень матриці. Але в деякому сенсі він нехтує «загальною» користувацькою поведінкою і може видавати дивні і вузькоспеціалізовані результати. Проте це просте узагальнення й досі використоване у великих e-commerce продуктах, таких як Amazon, оскільки обчислювальна складність невелика, а результати можуть бути релевантними.

ML-based підхід: тут прогнозуються всі можливі значення матриці взаємодій. Має певну схожість з контентною фільтрацією, саме в представленні об’єктів по характеристикам: ми виділяємо окрему матрицю і об’єкти можна якісно розрізняти. Той самий підхід використовується і для користувачів: вони матимуть свої характеристики і матрицю та розрізняти їх можна буде не лише по взаємодії з об’єктами. Суть підходу наступна: матрицю взаємодії користувач-об’єкт можна представити у вигляді добутку двох матриць: представлення користувачів на представлення об’єктів. Задані матриці є повністю визначеними, а отже повністю будуть задавати матрицю взаємодій, тобто порожніх клітинок не буде. А самі векторні представлення як користувачів, так і об’єктів можна отримати, внаслідок тренування моделі і надалі використовувати як певне латентне представлення (рис. 2.1):

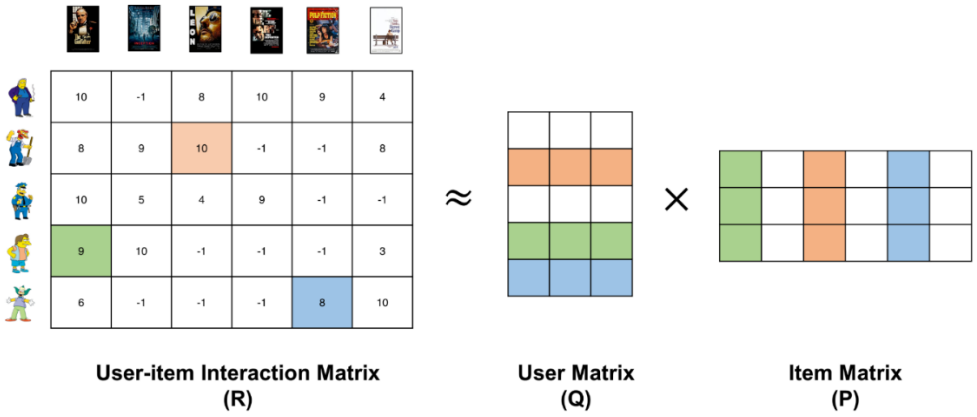


Рис. 2.1 Приклад стандартної матричної факторизації матриці з розкладом її на добуток двох матриць

Заданий метод називається матричною факторизацією – розклад матриці на декілька, у нашому випадку на дві. В цьому є користь при проектуванні системи: дві невеликі матриці зберігати набагато легше, ніж одну велику, обновляти матрицю взаємодії можна достатньо просто за рахунок перемноження. Трансформації набагато легше робити на менших матрицях, локально. В ході роботи алгоритму матриця апроксимується доволі точно. На практиці звичайна матрична факторизація досі показує себе добре, свою актуальність вона знову підтвердила внаслідок Netflix recommendation competition (2006-2009) [35]. Тоді задача зводилася до регресії оцінки:

$$RMSE^2 = \frac{1}{|obs|}\sum_{(u,i)\in obs}(r_{ui} - \hat{r}_{ui})^2 \qquad (2.2)$$

Розглянемо основні підходи до матричної факторизації, які використовуються при побудові рекомендаційних систем.

2.2.2 SVD

Даний вид матричної факторизації передбачає представлення матриці у вигляді добутку:

$$A = U\Sigma V^T \tag{2.3}$$

де матриці U та V будуть унітарними; розміри матриць наступні:

$$\boldsymbol{A}: \boldsymbol{n} \times \boldsymbol{m};\ \boldsymbol{U}: \boldsymbol{n} \times \boldsymbol{n};\ \boldsymbol{\Sigma}: \boldsymbol{n} \times \boldsymbol{m};\ \boldsymbol{V}: \boldsymbol{m} \times \boldsymbol{m}$$

При цьому на діагоналі матриці Σ будуть знаходитися сингулярні числа [6], розташовані в порядку зростання:

$$\boldsymbol{\Sigma} = \boldsymbol{diag}(\boldsymbol{\lambda_1}, \boldsymbol{\lambda_2}, \dots, \boldsymbol{\lambda_{min(m,n)}})$$
$$\boldsymbol{\lambda_1} \geq \boldsymbol{\lambda_2} \geq \cdots \boldsymbol{\lambda_{\mathbf{min}(m,n)}} \geq \mathbf{0}$$

Якщо розглянути тільки частину Σ'матриці Σ, взявши лише d найбільших сингулярних чисел, а також частини U', V' матриць U та V, що відповідають цим d сингулярним числам можемо отримати наступне представлення матриці A':

$$\boldsymbol{A} \approx \boldsymbol{A'} = \boldsymbol{U'\Sigma' V'^T}$$
$$\boldsymbol{A'}: \boldsymbol{n} \times \boldsymbol{m};\boldsymbol{U'}: \boldsymbol{n} \times \boldsymbol{d};\ \boldsymbol{\Sigma'}: \boldsymbol{d} \times \boldsymbol{d};\ \boldsymbol{V'}: \boldsymbol{d} \times \boldsymbol{m}$$

2.2.3 ALS та WALS

Стандартним підходом до побудови алгоритмів мінімізації в рекомендаційних системах вважається ALS (Alternative Least Squares) [7]. Це модифікований метод градієнтного спуску, ідея якого полягає в тому, щоб почергово робити ітерацію для матриці користувачів, вважаючи всі значення

для об’єктів константними, а потім навпаки. Припустимо, що x_u, y_i – деякі представлення користувача u та i, розміром d при запропонованій мірі близькості

$$dot\ product(a,b) = cos\theta = \frac{(a,b)}{|a||b|} \quad (2.4)$$

Тоді оцінка взаємодії користувача u та i, враховуючи, що дані вектори вже було нормовано, буде розраховуватися наступним чином:

$$\hat{r}_{ui} = x_u {y_i}^T = \sum_d x_{ud} y_{di}$$

Функція похибок – середньоквадратична функція відхилення враховуючи L_2 регуляризаційні параметри; S – множина всіх взаємодій:

$$L = \sum_{u,i\epsilon S}(r_{ui} - x_u {y_i}^T)^2 + \lambda(\sum_u \|x_u\|^2 + \sum_i \|y_i\|^2) \quad (2.5)$$

Беремо похідну по x_u, фіксуючи всі дані для y:

$$\frac{\partial L}{\partial x_u} = 0 \Rightarrow -2\sum_i (r_{ui} - x_u {y_i}^T)\, y_i + 2\boldsymbol{\lambda} x_u = 0$$

$$= -(r_u - x_u Y^T)Y + \boldsymbol{\lambda} x_u = 0$$

$$= x_u(Y^T Y + \boldsymbol{\lambda} I) = r_u Y$$

$$= x_u = r_u Y(Y^T Y + \boldsymbol{\lambda} I)^{-1} \quad (2.6)$$

Далі навпаки: робимо крок для y_i, фіксуючи всі дані для x:

$$\frac{\partial L}{\partial y_i} = 0 \Rightarrow y_i = r_i X(X^T X + \boldsymbol{\lambda} I)^{-1} \quad (2.7)$$

Існує безліч варіацій даного алгоритму, наприклад WALS (Weighted Alternative Least Squares) [9], коли ми задаємо певні ваги w_0 для непорожніх клітинок, а порожні не враховуємо.

$$\sum_{u,i\epsilon S}(r_{ui} - x_u {y_i}^T)^2 + w_0 * \sum_{u,i\notin S}(0 - x_u {y_i}^T)^2 \quad (2.8)$$

2.2.3 LightFM

LightFM [10] є нетиповою реалізацією стандартної матричної факторизації. Основна ідея: вирішення проблеми «холодного» старту, як для нових користувачів, так і для нових об’єктів. Суть методу полягає в тому, що ми представляємо користувацькі та об’єктні векторні представлення як лінійну комбінацію всіх їх характеристик. По суті крім матриці взаємодії ми вводимо ще дві матриці: характеристики користувача та характеристики об’єкта і факторизуємо їх. Потім оновлюємо вектори користувачів та об’єктів, додаючи їх по характеристикам та передаємо їх в основну матрицю взаємодії (рис. 2.2):

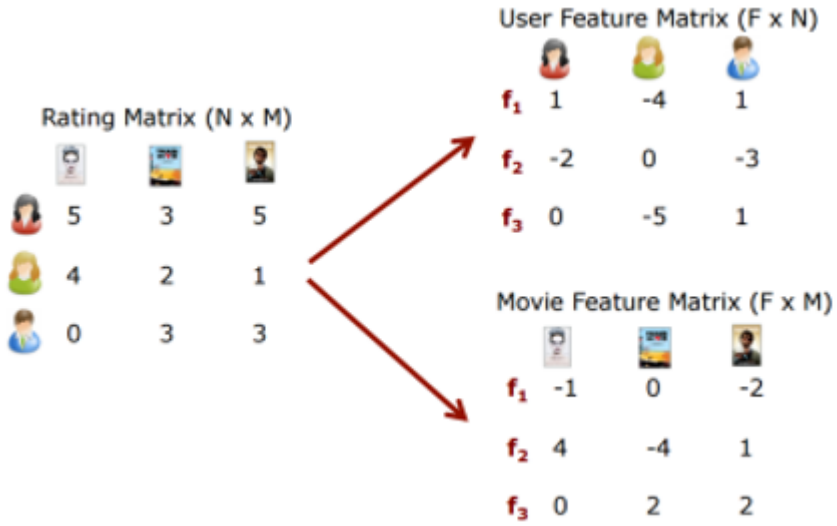


Рис. 2.2 Спосіб представлення матриці взаємодії, матриці характеристик користувача та об’єкта для матричної факторизації методом LightFM [11]

Чому це певне вдосконалення алгоритму? «Невідомого» користувача можна відразу представити явним чином представити через власні характеристики, або ввести для нього середнє значення по всім користувачам на певний момент часу – в такому в такому випадку «персоналізація» користувача буде мінімальною, але це варіант вирішення проблеми. Та сама логіка працює і

для нового об’єкту – відпадає необхідність робити декілька ітерацій матричної факторизації, щоб «актуалізувати» вектори.

2.2.4 Використання ранжування як основного підходу

Зазначу, що даний метод, хоч і є розширенням стандартної матричної факторизації, проте вводить принципово інший підхід: від стандартної класифікації/регресії ми переходимо до задачі ранжування (learn2rank) [12]. Ця ідея неможлива без існування позитивних та негативних об’єктів: наскільки якісно користувач провзаємодіяв з певним об’єктом, наскільки він був зацікавлений тощо. Як правило, позитивні об’єкти знайти просто – вони існують в явному вигляді в наших даних. Наприклад оцінки (навіть негативного характеру), індикатор взаємодії чи наявність кліку на об’єкт. Негативний об’єкт – в більшості випадків означає відсутність взаємодії, з цього робиться припущення, що користувачу він нецікавий. Ми можемо враховувати більш складні залежності, пов’язані з онлайн-метриками, такими як наприклад послідовні кліки, натискання клавіші «Подивитись контакт» для об’єктів-оголошень. Це неявна оцінка об’єкта [9]. Якщо позитивні об’єкти визначити досить просто, то негативні часто задають наскільки якісним буде фінальна рекомендація. Способи генерації бувають різними, найпоширенішим, є випадковий підбір об’єкта з вибірки: вважається, що усвідомлений вибір користувача є апріорі кращим за випадковий.

Деталізуємо процес ранжування; припустимо для одного користувача ми маємо один позитивний та один негативний об’єкт (без втрати загальності їх може бути декілька). Наша задача зводиться для того, щоб для конкретного користувача (на картинці *anchor*) привести векторний простір представлень

наших об’єктів в такий вигляд, щоб негативні представлення були якомога далі від користувача, а позитивні – якомога ближче (рис. 2.3).

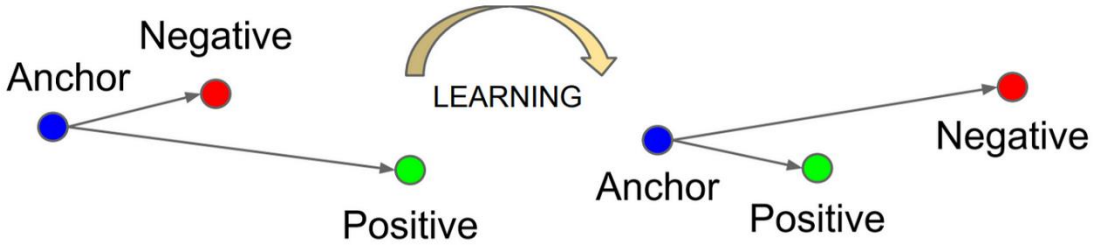


Рис. 2.3 Візуальна інтуїція ранжувальної функції похибок triplet loss [13]

Мінімізувати ми будемо наступний функціонал:

$$Loss = \sum_{i=1}^{N} \left[\left\| f_i^a - f_i^p \right\|^2 - \| f_i^a - f_i^n \|^2 + \varepsilon \right]_+ \tag{2.9}$$

$$[\cdot]_+ - Hinge\ Loss = Max(0, x)$$

ε – деякий коефіцієнт, необхідний для регуляризації.

На практиці найкраще LightFM працюватиме внаслідок мінімізації WARP Loss [14] – модифікації описаного вище підходу: він також працює з трійками (анхор-позитивний-негативний). Відмінність у тому як ми підбираємо негативний об’єкт [15]. Наведемо математичне описання:

- вибираємо користувача (u);
- вибираємо об’єкт, з яким користувач взаємодіяв (i);
- вибираємо випадковий об’єкт, з яким не було взаємодії (j);
- обраховуємо скалярні добутки $p = (u, i)$ та $n = (u, j)$;
- якщо $p > n$, повертаємося на крок 2;
- якщо $p \leq n$, обраховуємо похибку, оновлюємо ваги та повертаємося на крок 3;

Наша мета – знайти такий негативний об’єкт, з яким значення взаємодії буде вищим за позитивний. Якщо ми знайшли його швидко, робимо великий градієнтний крок, якщо довго – модель вже здатна досить добре узагальнювати, а отже сильно змінювати ваги не варто. Функція похибок розписується наступним чином:

$$Loss = \frac{\sum_{u \in users} L_u}{|users|}$$

$$L_u = \sum_{i \in Pos} L\big(rank(u,i)\big)$$

$$rank(u,i) = \sum_{j \notin Pos} I(f(u,i) \geq f(u,j))$$

b - відхилення: глобальне, по користувачу, по об’єкту

$$f(u,i) = b_g + b_u + b_i + \langle p_u, q_j \rangle,$$

В повному вигляді функції похибок для користувача виглядає так:

$$L\big(rank(u,i)\big) = \sum_{j \notin Pos} \log\left(\frac{|Neg|}{Sampled}\right) \left|1 - \langle p_u, q_i \rangle - \langle p_u, q_j \rangle\right|_+ \quad (2.10)$$

Зазначу, що вказаний метод можна вважати гібридним: він поєднує в собі можливості враховувати метадані вибраного користувача чи об’єкта, працюючи при цьому з матрицею взаємодій. Проте він легко зводиться по своїй суті до стандартної колаборативної фільтрації, якщо не передавати метадані. Користувач або об’єкт при такому варіанті представляються лише індикаторними особливостями: одиничною матрицею розміром N = загальна кількість користувачів або об’єктів.

Таке представлення кодує лише ідентифікатор і повідомляє алгоритму, що ми вибрали пару $\{n, m\}$ та відповідні їм представлення $\{u_n, i_m\}$.

2.2.5 Недоліки алгоритму колаборативної фільтрації

І з проблем алгоритму можна виділити як вже згаданий попередньо «холодний» старт для користувачів (на відміну від холодного старту в об’єктах) – неякісні рекомендації на початку, неврахування часового фактору та в загальному випадку необхідність великих обчислень (розділи 3.3.1-3.3.2).

2.3 Гібридні методи рекомендації

Задача гібридних методів – побудувати систему, що використовує переваги методів контентної та колаборативної фільтрації, мінімізуючи вплив їх слабких сторін. На допомогу приходять гібридні системи – за допомогою означеного правила ми зможемо поєднати рекомендації двох моделей в одну сильну, або означити архітектуру моделі так, щоб вона враховувала всі поставлені вимоги: давала персоніфіковані рекомендації та аналізувала поведінку інших користувачів.

2.3.1 Метод нейронних мереж для представлення міри близькості

Використання нейронних мереж дозволяють узагальнити алгоритм матричної факторизації. Вектор-представлення (ембедінг, розділ 2.3.4) користувача та об’єкта мають бути одної розмірності, а міра близькості (наприклад косинусна близькість) видавати певну оцінку взаємодії (рис. 2.4):

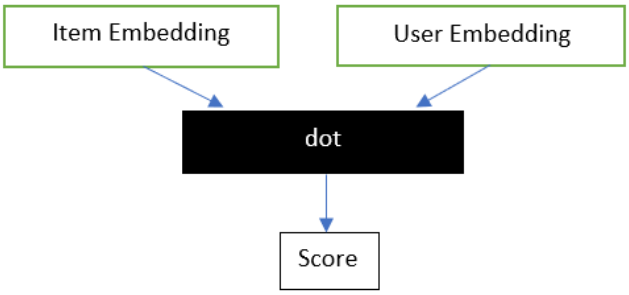


Рис. 2.4 Спосіб представлення матричної факторизації для векторів-представлень користувача та об’єкта з косинусною близькістю

Ми можемо узагальнити та дати можливість нейронній мережі визначити міру близькості вектора-представлення користувача і об’єкта. Для такої задачі, як правило, використовують MLP [33] – як приклад архітектура NeuCF [17] (рис. 2.5):

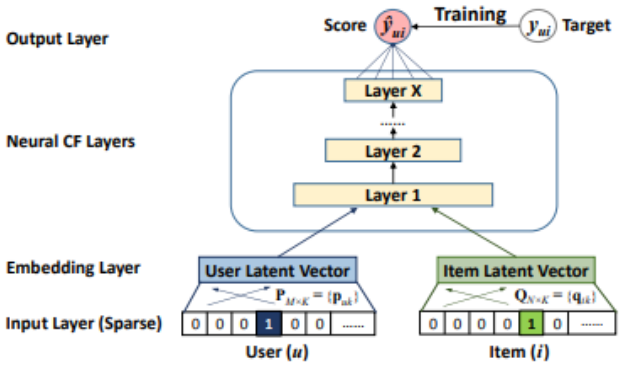


Figure 2: Neural collaborative filtering framework

Рис. 2.5 Приклад архітектури NeuCF, в ролі міри близькості якої використовується послідовний перцептрон

Зазначу, що такий підхід на практиці не використовується, оскільки потребує значної кількості обчислень для кожної пари користувач-об’єкт: таких пар буде значна кількість і для кожної з них треба бути зробити прямий та зворотній крок обчислень для тренування мережі. Відсутність складності в обчисленнях – як правило основна причина чому на практиці використовують

просту метрику близькості наприклад косинусну близькість. Інший підхід – знизити розмірність не об’єднаного вектора користувача та об’єкта (рис. 2.5), а кожного вектора представлення; цей підхід належить до сімейства DSSM [19] підходів (розділ 2.3.3) та широко використовується на практиці.

2.3.2 Двохрівневі гібридні моделі

Деталізуємо алгоритм LightFM: він вважається простим (як і будь-який підвид стандартної матричної факторизації), адже буде не таким точним, як результат отриманий за допомогою більш складної архітектури, але він дозволить швидко видати результат, з яким можна працювати надалі. Мова про двохрівневі моделі: спочатку ми «відсіюємо» кандидатів за допомогою простих алгоритмів, а потім результати, отримані для кожного користувача (певний фіксований по довжині список рекомендацій) передаємо на основне ранжування в складний алгоритм наприклад градієнтний бустинг [31]. Чому ми не використовуємо бустинг із самого початку? Найголовніша причина – обчислення, як правило потужностей для ранжування просто не вистачає. При порівнянні результатів першого етапу та другого, більш складний алгоритм у зв’язці з простим алгоритмом завжди перевершує простий [16]. При такому підході ми можемо використовувати нейронні мережі, але на другому етапі, враховуючи значний час обчислень [18].

2.3.3 Content2vec або DSSM моделі

Інший приклад складної гібридної архітектури – DSSM моделі [19, 20]. По-іншому їх називають content2vec моделі, оскільки вони дозволяють представити будь-який за усіма типами його даних у векторному просторі, де близькі по своєму змісту об’єкти матимуть високу міру близькості, а далекі – навпаки, будуть мати низьку міру. Такий підхід дуже схожий на поняття латентного представлення або ембедінгу [22]. (розділ 2.3.4) Назва походить із статті Google «Deep Semantic Structured Models» [19], де автори запропонували новий підхід для аналізу семантичної близькості текстових документів. Припустимо, що ми маємо пари документ-запит і наша задача – підібрати найрелевантніший документ по запиту. Це схоже на взаємодію користувач-об’єкт в рекомендаційних системах, де характеристики документів будуть текстовими. Ми розбиваємо кожне слово документу на триграми, наприклад слово «слон» буде кодуватися як «#слон#» та розбиватися на наступні триграми: «#сл», «сло», «лон», «он#». Чому саме триграми? Вважається, що це певне оптимальне значення, на яке ми зможемо ділити наш текст, при цьому добре «зберігаючи» його семантичне забарвлення. Крім того, якщо говорити про англійський алфавіт, кількість триграм не така велика, як кількість слів (500 тисяч проти 30 тисяч). Таким чином вважається, що ми зможемо вирішувати проблему «нових» слів в загальному словнику, адже всі можливі триграми вже були обраховані. Ідея DSSM полягає в наступному: на вхід ми подаємо два вектори, для яких хочемо порахувати певне значення близькості. Проблемою є розмірність цих векторів, адже вона занадто висока, навіть для обрахування косинусної близькості. Ідея в тому, щоб за допомогою нейронних мереж значно зменшити розмірність кожного вектора, при цьому зберігши його певне семантичне значення, і в кінці швидко порахувати їх близькість (рис. 2.6). Ідея з

триграмами – лише спосіб реалізації однієї з можливих архітектур для трансформації представлення об’єкта; перевагою DSSM методу є можливість комбінувати різні архітектури, всередині основної моделі [20] (розділ 3.2.1-3.2.2).

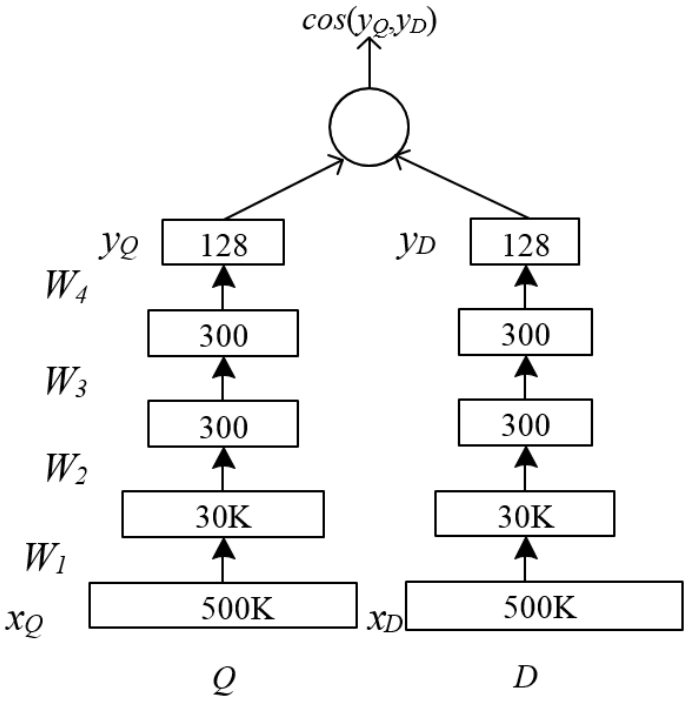


Рис. 2.6 Архітектура DSSM, запропонована Google для проблеми вирішення рекомендацій релевантних документів по введеному запиту; основною ідеєю є розбиття вхідних слів на триграми

Ми можемо проводити будь-які експерименти: включати додавати в модель рекуретні нейронні мережі [32], використовувати згорткові архітектури для тексту, об’єднувати трансформовані представлення, з трансформованими представленням даних іншої природи (відео, музика, геолокація) і проводити подальші трансформації на власний розсуд (розділ 3.2.1). Нагадаю, що основна ідея – суттєво зменшити об’єм вхідного представлення для як для користувача (функція-трансформації user2vec), так для і об’єкта (функція-трансформації item2vec), а потім на векторах меншої розмірності порахувати міру близькості.

Ми зможемо розширити задачу до ранжування, застосувавши підхід сіамських мереж [16] (рис. 2.7):

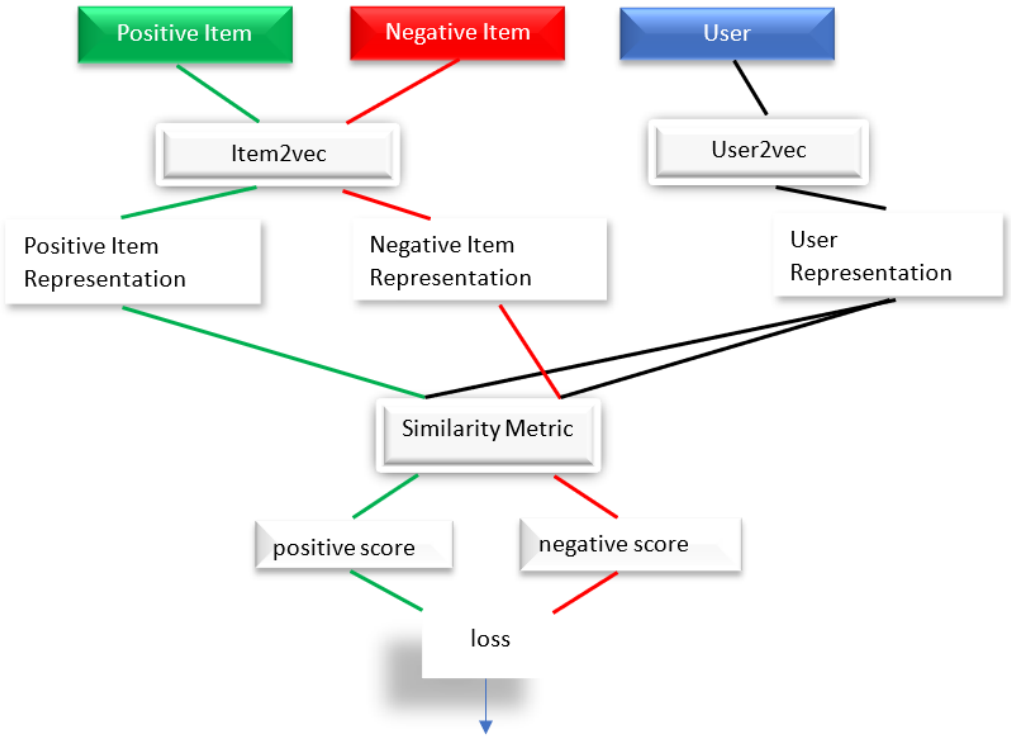


Рис. 2.7 Приклад архітектури DSSM моделі з використанням сіамських мереж для вирішенні задачі ранжування

Основна ідея сіамських мереж – одна модель item2vec одночасно для позитивних та негативних об’єктів (рис. 2.7). Наша основна архітектура, що буде містити item2vec, зможе натренувати її так, що модель бути здатна правильно «класифікувати» об’єкти – потенційно позитивний чи негативний об’єкт відносно до користувача. В основній моделі мінімізується ранжувальна метрика – як правило triplet loss (розділ 2.2.4)

2.3.4 Sequential моделі та латентні представлення (embeddings)

Приведемо пояснення як саме представляється текстова або будь-яка послідовна інформація в комп'ютері. На допомогу приходять техніки Natural Language Processing (або NLP) наприклад bag-of-words, TF-IDF, або латентне семантичне представлення (*embeddings* – ембедінги) [22, 23].

Методи NLP, створені для роботи з текстовою інформацією. Розглянемо попередній приклад з фільмами, де в нас були індикаторні характеристики. Ідея проста: розбити текст на слова і кожне слово представити як індикаторну характеристику. Наприклад текст: «*Emma is writing a letter*» представляється як (табл. 2.4):

Таблиця 2.4. Приклад індикаторного представлення речення, попередньо розбитого на одиниці – слова

emma	is	writing	a	letter
1	1	1	1	1

В реальності ми працюємо відразу з кількома текстами, тому в рядку, що відповідає заданому реченню будуть і нулі – слова яких немає в реченні (табл. 2.5):

Таблиця 2.5. Приклад індикаторного представлення декількох речень, попередньо розбитих на одиниці – слова; підхід bag-of-words [22]

emma	is	had	writing	driven	a	car	letter
1	0	1	0	1	1	1	0
1	1	0	1	0	1	0	1

Такий підхід називається bag-of-words [22] і має ряд суттєвих недоліків. По-перше, проблема «холодних» слів (розділ 2.1.1 та 2.3.2). Нам доведеться перераховувати матричне представлення (табл. 2.5) кожен раз, коли додаємо нове слово. Друга проблема – ми не можемо добре оцінити зв'язок між текстами, маючи лише індикаторне представлення слів. Логічно припустити, що в реченні будуть більш значущі слова, такі що повністю задаватимуть його контекст. Представлений підхід оцінює лише кількість входжень слів одного тексту в інший, що не є семантичною близькістю в загальному випадку.

Як рішення був придуманий TF-IDF [22, 23]: підхід коли ми модифікуємо значення слова в тексті, аналізуючи його входження в інші тексти. Якщо слово x входило лише в один текст $text$ і ні в який інший, воно свого роду є визначальним для $text$, а отже повинно мати вищу значимість ніж просто індикатор (рис. 2.11):

$$TF - IDF_{x,text} = \\ = frequency_{x,text} \times \log\left(1 + \frac{average\ number\ of\ words\ per\ text}{frequency_{x,all\ texts}}\right) \quad (2.11)$$

Бачимо, що до звичайної частоти (компонента TF, аналог індикаторного входження), додається компонента IDF, зі зворотньою логарифмічною залежністю, яка враховує входження слова не тільки по одному тексту, а по всім. Це може бути певним вирішенням проблеми, але:

- деякі слова, що повторюються в різних текстах можуть мати грати важливу роль у одному тексті, тоді як TF-IDF помічає лише «унікальність» слова;
- зв'язок між словами в реченні ніяк не встановлено, слова помічаються або як індикатор, або по кількості входжень;

В 2013 був придуманий підхід, який дозволяє подивитися на семантичне забарвлення текстів через представлення його слів. Це підхід називається Word2vec [22], коли кожне слово представляється через вектори певної заданої довжини (рис.2.8):

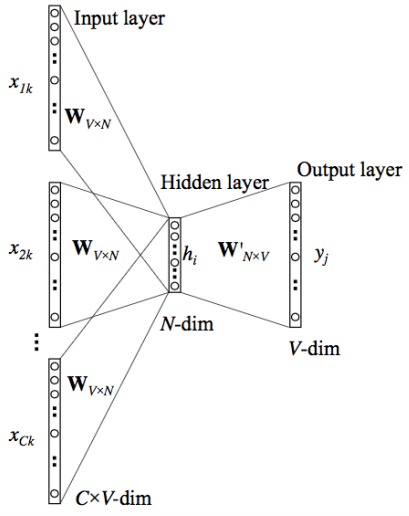


Рис. 2.8 Оригінальна архітектура Continious bag-of-words (CBOW) для отримання латентного представлення слова у тексті, реалізуючи підхід Word2vec

Отримати представлення можна одним з двох найбільш відомих методів: Continious bag-of-words (CBOW) [22] або за допомогою Skip-Gram [23]. У випадку з CBOW ми проходимо по тексту та намагаємося прогнозувати слово по словах, що знаходяться від нього зліва та справа. Skip-Gram –навпаки: по центральному слову ми робимо передбачення його контексту. Виявилось, що якщо взяти матрицю коефіцієнтів у першому шарі такої мережі, отримаємо хороше векторне представлення слів з текстового контексту, який використовується в тренуванні. Для тренування нам необхідна текстова інформація та розмічені дані: алгоритм намагатиметься закодувати контекст слова відповідно до того, як саме ми означили нашу залежну змінну y. Основна особливість представлень (табл. 2.6) в тому, що в них робиться акцент не на

«інформативності» вектору, а на те, наскільки він близький до вектору, з яким схожий семантично насправді.

Таблиця.2.6 Приклад представлення деякого слова, використовуючи його латентне представлення, отримане за допомогою методу Word2vec

0.88	-0.01	0.3	-0.1	…	0.02	0.02	-0.1	-0.33

Тепер розглянемо використання послідовних моделей для створення рекомендацій: рекурентні нейронні мережі [32] можна використовувати для цих задач. Ми можемо кодувати послідовну історію взаємодій користувача, це допоможе врахувати аксіому релевантності (розділ 1.1) та надати приорітет більш пізнім взаємодіям. Такий підхід можна використати, щоб побувати рекомендаційну систему без врахування особливостей користувача чи об’єкта (розділ 3.2.2). Ще один варіант застосування – комбінації з DSSM моделями (розділ 2.3.3), наприклад всередині блоку item2vec, передаючи як представлення об’єкта послідовну інформацію наприклад текст. Можна ускладнити алгоритм, додавши attention layers [24] – це метод запозичений з архітектури трансформера. На прикладі текстових даних ми кодуємо послідовність слів, а потім даємо змогу алгоритму виділити найбільш важливі слова в контексті тексту (рис. 2.9):

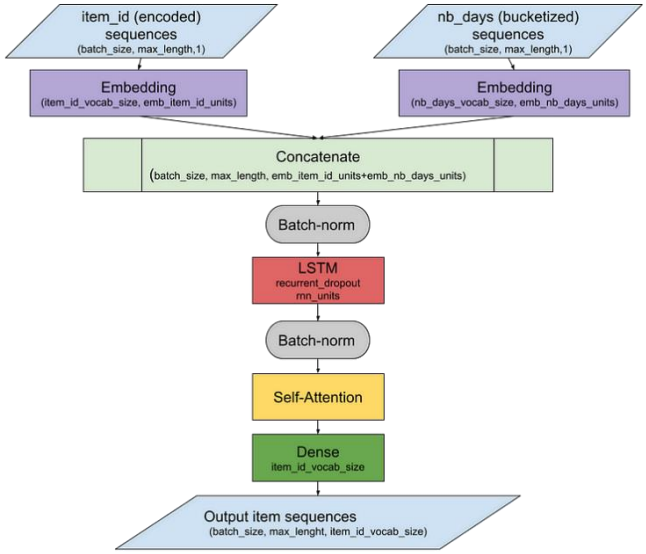


Рис. 2.9 Спосіб кодування одночасно двох послідовностей в одну з використанням послідовної архітектури LSTM та механізму Self-Attention

Загальновідомо, що моделі типу GRU [25] або LSTM [26] мають «коротку пам’ять», тобто не здатні запам’ятати інформацію надовго , особливо, якщо використовувати їх для послідовностей значної довжини. Такі архітектури можна використовувати в рамках Session-based моделей [27], коли модель тренується по активній сесії для одного чи декількох користувачів. GRU4Rec [17] являє собою спеціальну реалізацію попередньо реалізованого методу GRU, проте саме для session-based моделей, коли, як правило, користувачі не є авторизованими (недоступне поле user_id) та середня кількість взаємодій на користувача є дуже малою. На практиці це відбувається дуже часто і відомі застосунки вимушені адаптувати свої рекомендаційні алгоритми під цю особливість. Для класичних послідовних рекомендаційних архітектур, можна було згрупувати кожного користувача по своєму ідентифікатору (це буде аналог сесії) та отримати по його історії взаємодій однакові по довжині послідовності. У випадку session-based моделей (рис. 2.10) для кожної сесії без авторизованих користувачів, довжина подій, що відбулися за її час, буде різнитися. Тому треба

перевизначити спосіб генерації послідовностей для кожної сесії, щоб натренувати рекурентну мережу.

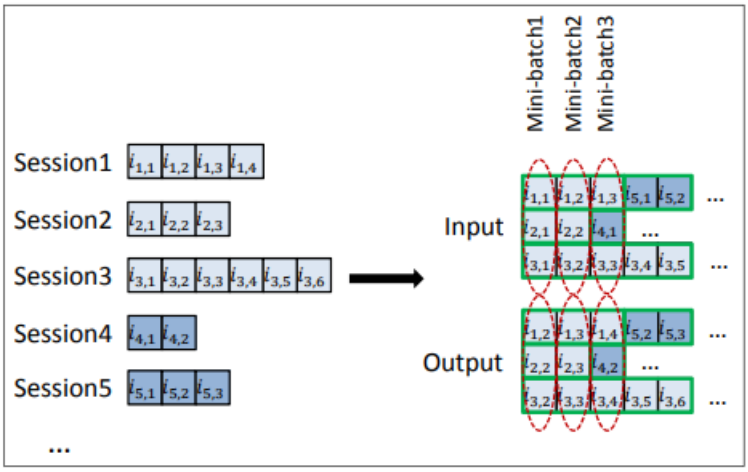


Рис. 2.10 Спосіб генерації тестової вибірки для послідовної рекомендаційної архітектури GRU4Rec для вирішення задачі session-based recommendation

Сесії можна скомбінувати так, в відповідному полі вектору output буде наступна подія (верхня права клітинка в блоці input та верхня права в блоці output) (рис. 2.10). Коли сесія закінчується, тобто відбувається остання подія у сесії, ми вставляємо події наступної сесії на вільні місця і продовжуємо алгоритм. Внаслідок такого підходу отримуємо послідовності однакової довжини на вхід блоку GRU. Також в методі GRU4Rec відмінні від класичного GRU функції похибок, вони намагається вирішити проблему «затухання» градієнту в процесі тренування для рекурентної нейронної мережі, тому модель може тренуватися трохи довше, ніж схожа архітектура, реалізована без вказаних особливостей (розділ 3.2.2 та 3.3.5). Рекурентні нейронні мережі на момент свого створення не призначалися для створення рекомендацій, тому основна задача складних послідовних рекомендаційних архітектур – максимально адаптувати модель під цю задачу [27].

Архітектура Caser [28] також пов’язана із кодуванням послідовностей. На відміну від GRU4Rec ми передаємо не закодовані представлення послідовності в наступний рекурентний шар GRU, а складаємо їх вертикально, утворюючи 2D матрицю, аналог картинки. Далі використовуємо паралельно горизонтальну та вертикальну згортку на утворену картинку; на отриманих згорткових шарах використовуємо метод MaxPooling [33] (беремо максимальний елемент по вісі), виконуємо конкатенацію [33] векторів, що отримали після пулінгу, об’єднуючи їх методом конкатенації разом із представленням користувача. На отриманий вектор «навішуємо» звичайний повнозв’язний шар нейронної мережі [33] (рис.2.11):

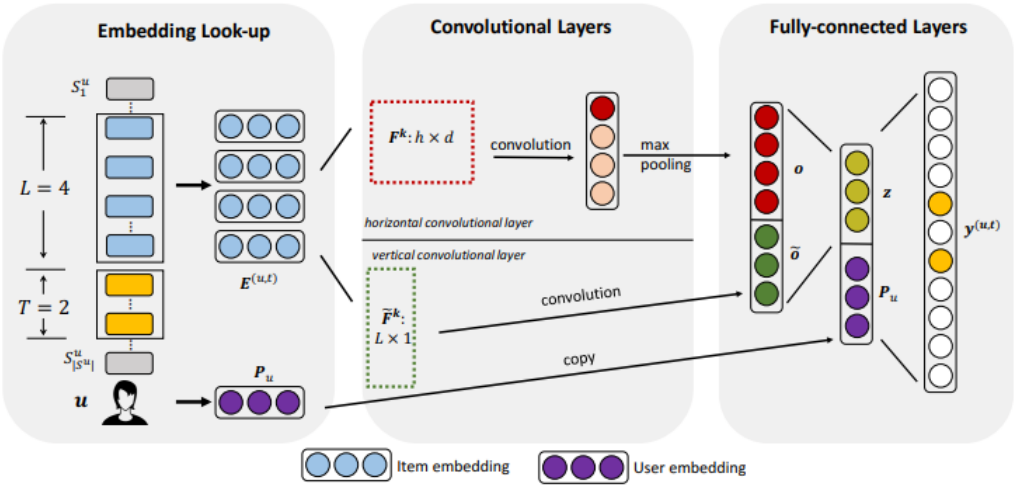


Рис.2.11 Представлення складної послідовної рекомендаційної архітектури Caser [28], що кодує послідовність взаємодій користувача, використовуючи підхід згорткових мереж [33]

Ця ідея працює, оскільки в згенерованій картинці в нас не просто закодовані об’єкти, а послідовності і частина наступної послідовності міститься в попередній, а отже і в її представленні. Горизонтальний фільтр потрібен, що віднайти важливі сигнали – події всередині послідовностей по часу, які будуть повторюватися в послідовних послідовностях, вертикальний фільтр – та сама

ідея, але тепер частинами проходимося по представленням. У випадку вертикальних згорток розглядаємо частини представлень, але, на відміну від горизонтальних, за весь час.

2.4 Метрики оцінювання

Класифікаційні:

- $Precision@K = \frac{number\ of\ recommendations\ that\ are\ relevant}{K}$; (2.12)
- $Recall@K = \frac{number\ of\ recommendations\ that\ are\ relevant}{number\ of\ all\ the\ possible\ relevant\ items}$; (2.13)

Ранжувальні:

- $MAP@K = \frac{1}{number\ of\ users}\sum_{i=1}^{number\ of\ users} AP@K$; (2.14)

mean average precision for k recommendations;

$$AP@K = \frac{1}{N(K)}\sum_{i=1}^{K}\frac{TP_{seen}(i)}{i}$$

average precision for k recommendations (рис. 2.12)*;*

$$N(k) = \min(k,\ TP_{total})$$

$$TP_{seen} = \begin{cases} 0: & i^{th} = True \\ True\ Positives\ seen\ till\ i\ position: & i^{th} = False \end{cases}$$

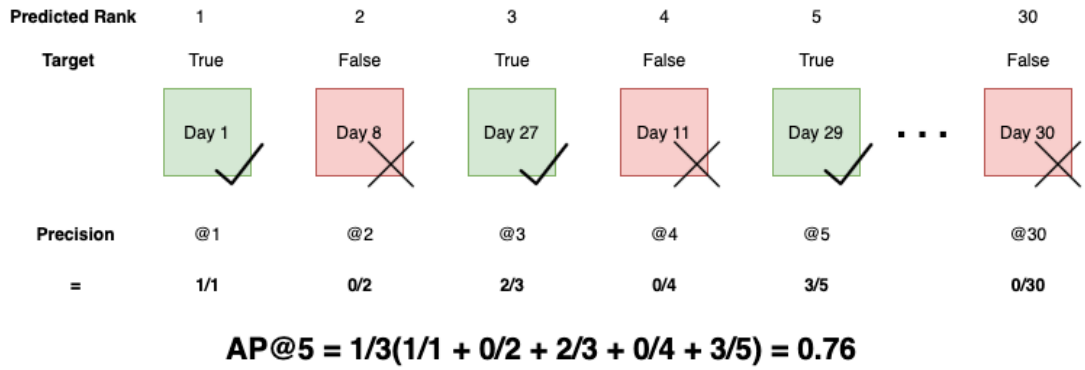


Рис.2.12 Спосіб розрахунку ранжувальної метрики якості average precision для випадку рекомендацій вибірки довжиною 5

- $MRR = \frac{1}{Q}\sum_{q=1}^{Q}\frac{1}{rank_q}$; (2.15)

де $rank_q$ – ранг першого елемента, який був релевантний, в рекомендації q; Q – кількість всіх рекомендацій

Висновки до розділу 2

Існують два основних підходи до побудови рекомендаційних систем – це експертні системи або контентна фільтрація та колаборативна фільтрація. Методи, що поєднують ці два підходи, називаються гібридними та широко використовуються в реальних застосунках; створені вони, щоб зменшити вплив недоліків одночасно контентної та колаборативної фільтрації, використавши при цьому сильні сторони обох підходів.

До метод класичної колаборативної фільтрації можна віднести задачу матричної факторизації, або певні модифікації методу (розділ 2.2.3). Нейронні мережі можуть бути дуже гнучкими та ефективними як рекомендаційні системи, адже допомагають узагальнити процес матричної факторизації (рис.

2.4) та додати нові особливості: тепер можна ускладнювати функцію-міру близькості між векторами (розділ 2.3.1), понижувати розмірність вхідних даних та передавати метадані для представлення користувача чи об’єкта (розділ 2.3.4). Принциповою особливістю сучасних архітектур став перехід від задачі регресії [35] до задачі ранжування (розділ 2.2.4).

Крім цього нейронні мережі не дозволяють існування окремих моделей-представлень всередині основної моделі (розділ 2.3.3) для представлення об’єкта чи користувача. На вхід таким моделям-представленням можна подавати будь-яку інформацію: відео, аудіо, текст, модель опрацює дані та згенерує вектор-представлення [20]. Крім цього, для рекомендацій можна використовувати рекурентні або послідовні нейронні мережі, що реалізують механізм кодування послідовностей, що передають на вхід або користувачі, або об’єкти (розділ 3.2.2).

3 АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

3.1 Опис даних

В процесі роботи ми будемо використовувати два набори даних: класичний для задач рекомендацій MovieLens 1M Dataset [37] та більш складний Google Local Review Data 2021 для штату Аляска, США [38].

3.1.1 MovieLens 1M Dataset

Оригінальний набір даних містить 3 файли: таблицю із взаємодією, таблицю з метаданими для користувача та таблицю з метаданими для об’єктів – фільмів (табл. 3.1).

Таблиця 3.1 Вміст таблиць у вхідному наборі даних MovieLens

	Взаємодія	Користувачі	Об’єкти
Кількість	1000209	6040	3883

Таблиця взаємодій буде містити інформацію про взаємодію користувач-об’єкт, виставлену оцінку та час взаємодії у форматі timestamp (табл. 3.2); метадані користувача та об’єкта міститимуть інформацію про наступні характеристики (табл. 3.3 та 3.4 відповідно):

Таблиця 3.2 Приклад запису у таблиці взаємодій для набору даних MovieLens

	gmap_id	user_id	rating	time
0	1	1193	5	978300760

Таблиця 3.3 Приклад запису у таблиці метаданих користувача для набору даних MovieLens

	user_id	gender	age	role	time
0	1	F	1	10	48067

Таблиця 3.4 Приклад запису у таблиці метаданих користувача для набору даних MovieLens

	gmap_id	title	genre
0	1	Toy Story (1995)	Animation|Children's|Comedy

Проведемо додаткову трансформацію даних. Для користувача проведемо бінаризацію колонки «age». Після цього виконаємо one-hot кодування [36] всіх особливостей. Користувач матиме 29 особливостей після кодування (табл. 3.5):

Таблиця 3.5 Фрагмент закодованого запису у таблиці метаданих користувача для набору даних MovieLens

	uid	gender_F	gender_M	age bin_(0, 10]	age bin_(10, 20]	age bin_(20, 30]	age bin_(30, 40]	age bin_(40, 50]	ag bin_(5 6
0	40	1	0	1	0	0	0	0	

Для об’єкта додамо колонки «belongs_to_collection», «language», «budget», «language», «revenue», «runtime», «country». Колонки «budget», «revenue», «runtime» бінаризуємо. Далі всі особливості по аналогії з матрицею

для користувачів закодуємо за допомогою one-hot кодування. Об’єкт матиме 132 особливості після кодування (табл. 3.6):

Таблиця 3.6 Фрагмент закодованого запису у таблиці метаданих об’єкта для набору даних MovieLens

	iid	belongs_to_collection	year bin_(1918, 1928]	year bin_(1928, 1938]	year bin_(1938, 1948]	year bin_(1948, 1958]	year bin_(1958, 1968]	year bin_(1968, 1978]	bin_(1 1
0	0	1	0	0	0	0	0	0	

Проаналізуємо середню взаємодію між користувачем та об’єктом. Для цього виведемо графік розподілу кількості взаємодій для користувача та об’єкта (рис 3.1 та 3.2 відповідно):

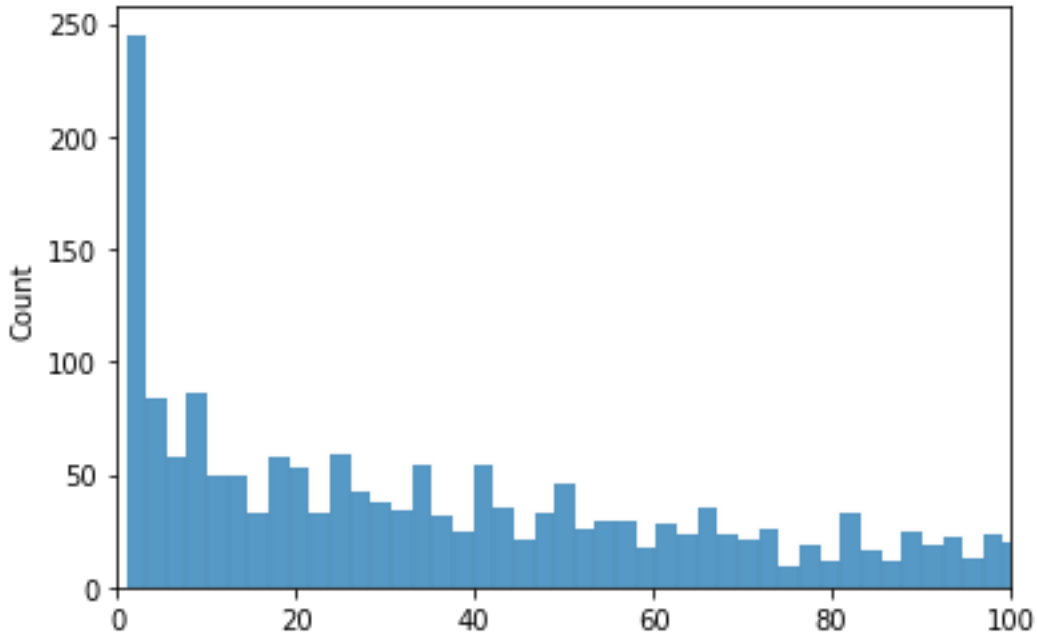


Рис. 3.1 Розподіл кількості взаємодій для користувачів в наборі даних MovieLens

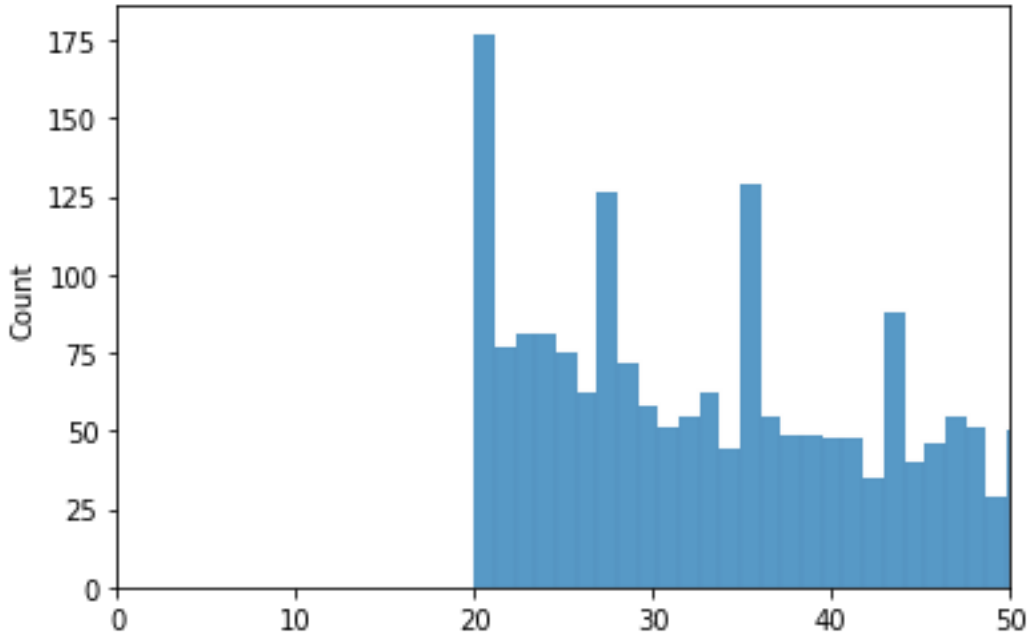


Рис. 3.2 Розподіл кількості оцінок для об’єктів в наборі даних MovieLens

Для зменшення навантаження обчислень залишимо взаємодії лише тих користувачів, що мали більше або рівне 10 взаємодій. Після видалення користувачів, що не потрапили під введене правило, та їх взаємодій із матриці взаємодій, а також відповідних ним об’єктів, маємо наступний вміст таблиць (табл. 3.7):

Таблиця 3.7 Вміст таблиць у трансформованому наборі даних MovieLens

	Взаємодія	Користувачі	Об’єкти
Кількість	633945	3260	3829

Отриману матрицю взаємодій поділимо на тренувальну та тестову вибірку у відношенні 90:10 по часу (табл. 3.8):

Таблиця 3.8 Спосіб розбиття трансформованих даних MovieLens на тренувальну та тестову вибірку у відношенні 90:10 по часу

Вміст тренувальної вибірки (рядки)	570550
Вміст тестової вибірки (рядки)	63395
Поріг по часу	2001-01-13 18:40:51

3.1.2 Google Local Review Data 2021

Оригінальний набір даних містить 2 файли: таблицю із взаємодією і таблицю з метаданими для об’єктів – локацій; кількість користувачів порахуємо з таблиці взаємодій (табл. 3.9):

Таблиця 3.9 Вміст таблиць у вхідному наборі даних Google Local Review

	Взаємодія	Користувачі	Об’єкти
Кількість	1051246	278695	12774

Таблиця взаємодій буде містити інформацію про взаємодію користувач-об’єкт, виставлену оцінку, час взаємодії у форматі timestamp та додаткову інформацію: коментар та фотографія (якщо присутні). Відфільтруємо таблицю так, щоб в ній залишилися лише ідентифікатор користувача, об’єкта, рейтинг та час взаємодії (табл. 3.10); метадані об’єкта міститимуть інформацію про наступні характеристики (табл. 3.11):

Таблиця 3.10 Приклад запису у таблиці взаємодій для набору даних Google Local Review

	user_id	gmap_id	rating	time
0	1.091298e+20	0x56b646ed2220b77f:0xd8975e316de80952	5.0	1566331951619

Таблиця 3.11 Приклад фрагменту запису у таблиці метаданих об’єкта для набору даних Google Local Review

	name	address	gmap_id	description	latitude	longitude	category	avg_rating	num_of_revie
0	Bear Creek Cabins & RV Park	Bear Creek Cabins & RV Park, 3181 Richardson H...	0x56b646ed2220b77f:0xd8975e316de80952	None	61.100644	-146.214552	[RV park, Cabin rental agency, Campground]	4.5	

Проведена додаткова трансформація даних. Закодована географічна особливість об’єкта, за допомогою характеристик «longitude» та «latitude». Для цього стандартизуємо ці особливості та бінаризуємо їх, поділивши район на квадрати (рис. 3.3):

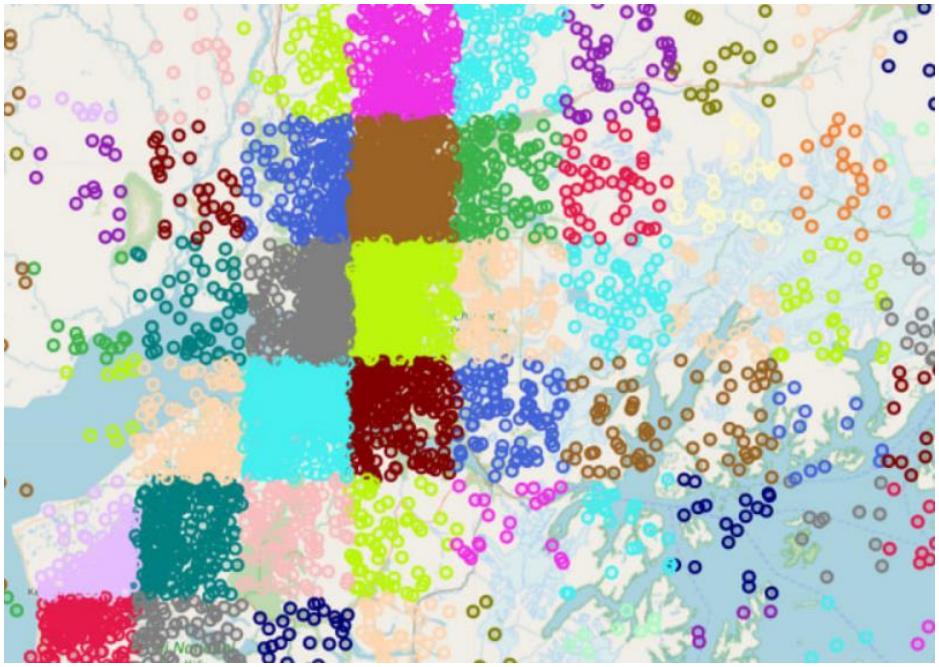


Рис. 3.3 Приклад поділу регіону на квадрати однакової площі, використовуючи бінаризацію стандартизованої довготи та широти для набору даних Google Local Review

Кожному квадрату присвоєно окрема категорія, всього – 229, проведено one-hot кодування [36]. Зазначу, що даний алгоритм географічного кодування показав себе найкраще, також було проведено експерименти з використанням кластеризаційних алгоритмів KMeans [29] та DBSCAN [30]. Вони показували себе гірше, часто класифікуючи декілька різних субрегіонів як один, або виділяючи один великий кластер серед усіх об’єктів, якого інтуїтивно існувати не повинно (рис 3.4, зелені точки).

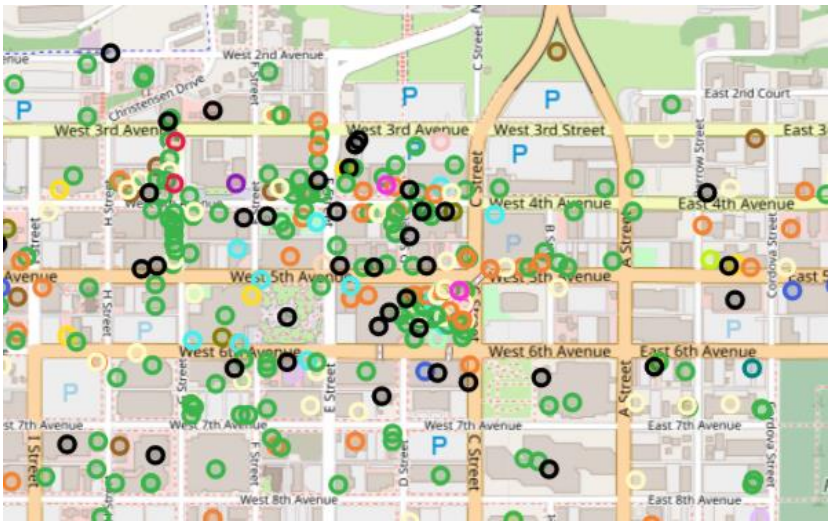


Рис. 3.4 Приклад кластеризації об’єктів, використовуючи стандартизовану довготу та широту, методом DBSCAN для набору даних Google Local Review

Далі була закодована категоріальна особливість; складність кодування для цієї особливості полягала в наступному:

- одна локація могла належати до декількох категорій (до 11 категорій одночасно);
- кількість усіх можливих категорій була значною (2100);
- об’єкт міг належати до нечисельних категорій, які можна було б згрупувати в більш чисельні та близькі по своєму сенсу.

Було прийнято рішення представити категоріальну особливість як послідовність слів в тексті [22] (рис 3.5):

- розглядалося лише 5 перших категорій для позначення об’єкта;
- кожній категорії було присвоєно чисельний ідентифікатор;
- для кожної такої послідовності було застосовано операції паддінгу [33].

	0	1	2	3	4
0	0	0	1857	898	271

Рис. 3.5 Приклад представлення категоріальної особливості {*RV Park*, *Cabin rental agency*, *Campground*} за допомогою аналізу перших 5 категорій, конвертації кожної категорії в числений ідентифікатор та використання операції паддінгу для набору даних Google Local Review

Також для метаданих об’єкта було створено особливості:

- popularity (*середній рейтинг локації * кількість відгуків для цієї локації*);
- індикатор is_small_cluster для розуміння чи знаходиться в гео-кластері більше ніж одна локація;
- час з моменту першого відгуку та час з моменту останнього відгуку (порівнюючи з останнім записом по часу в таблиці взаємодії).

Всі чисельні змінні, окрім представлення категорії, було стандартизовано. Всього метадані об’єкта матимуть 14 особливостей (рис. 3.6):

	iid	GEO	0	1	2	3	4	avg_rating	num_of_reviews	latitude scaled	longitude scaled	popularity	days from first inte
0	0	179	0	0	1857	898	271	0.252917	-0.308556	-0.062606	0.139879	-0.300087	0

Рис. 3.6 Приклад фрагменту запису трансформованої таблиці метаданих об’єкта для набору даних Google Local Review

Метадані користувачів було створено з таблиці взаємодій, оскільки в оригінальному наборі ми не мали таблицю для них. Для цього було пораховано середню довготу та широту по кожному користувачу з їх подальшою стандартизацією; була порахована середня оцінка та загальна кількість оцінок, та аналогічно таблиці метаданих об’єктів – час з моменту першої та останньої взаємодії. Всі особливості було стандартизовано. Кількість особливостей користувача – 6 (рис. 3.7):

	uid	latitude scaled	longitude scaled	number reviews	average rating	days from first interactions	days from last interactions
0	0.0	-2.522356	-2.637646	-0.324851	0.238959	0.391864	0.163209

Рис. 3.7 Приклад фрагменту запису створеної таблиці метаданих користувача для набору даних Google Local Review

Проаналізуємо середню взаємодію між користувачем та об’єктом. Для цього виведемо графік розподілу кількості взаємодій для користувача та об’єкта (рис 3.8 та 3.9 відповідно). Користувачів маємо набагато більше ніж об’єктів, це сильно ускладнить процес обчислень та створення рекомендацій, оскільки історія взаємодій буде не інформативною. Тому встановлюємо «жорсткий» поріг входжень для користувача – 10 взаємодій, для об’єкта – 1 взаємодія.

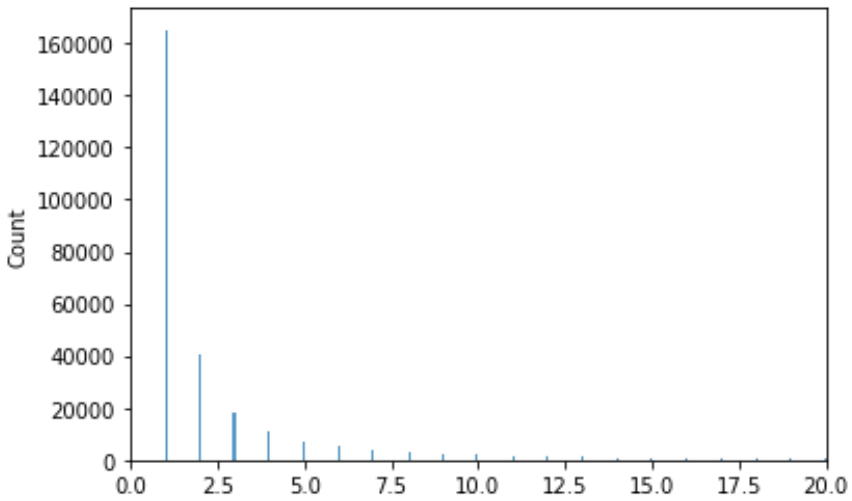


Рис. 3.8 Розподіл кількості взаємодій для користувачів в наборі даних Google Local Review

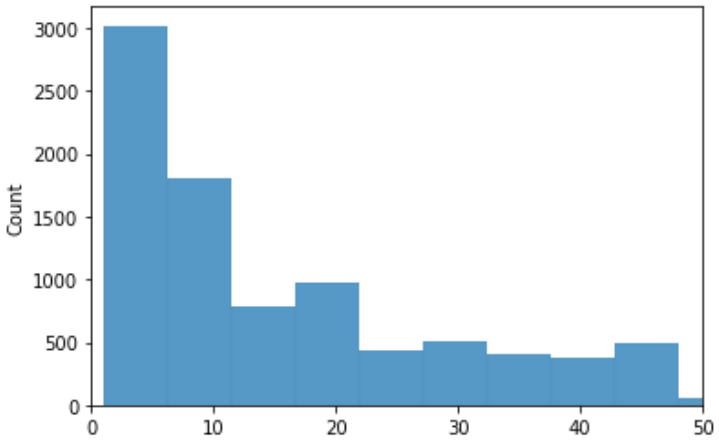


Рис. 3.9 Розподіл кількості оцінок для об’єктів в наборі даних Google Local Review

Після видалення користувачів та об’єктів, що не потрапили під введене правило, та їх взаємодій із матриці взаємодій, маємо наступний вміст таблиць (табл. 3.12):

Таблиця 3.12 Вміст таблиць у трансформованому наборі даних Google Local Review

	Взаємодія	Користувачі	Об’єкти
Кількість	544551	20748	12100

Отриману матрицю взаємодій поділимо на тренувальну та тестову вибірку у відношенні 90:10 по часу (табл. 3.13):

Таблиця 3.13 Спосіб розбиття трансформованих даних Google Local Review на тренувальну та тестову вибірку у відношенні 90:10 по часу

Вміст тренувальної вибірки (рядки)	490095
Вміст тестової вибірки (рядки)	54456
Поріг по часу	2020-10-04 23:05:59

3.2 Огляд розробленого продукту

В роботі була реалізована архітектура DSSM та власна інтерпретація архітектури GRU, а також важливі допоміжні функції, що використовуються для всіх методів.

3.2.1 Реалізація архітектури DSSM

Для реалізації методу DSSM (рис. 3.10) ми маємо реалізувати наступні важливі компоненти:

- ранжувальну функцію похибок;
- генератор триплетів (анхор-позитивний-негативний), що реалізує задану логіку генерації;
- функції трансформації об’єкта (рис. 3.10) та користувача для трансформації вхідного вектору у інший векторний простір;
- архітектуру основної моделі, що поєднуватиме одночасно функцію трансформації об’єкта та користувача в одну.

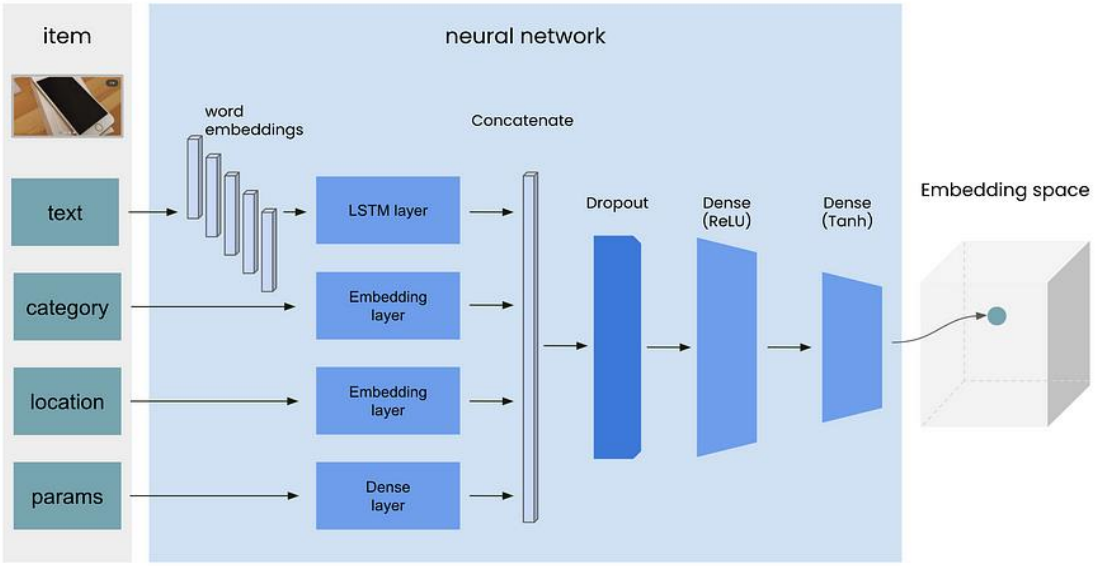


Рис. 3.10 Типова реалізація функції трансформації об’єкта (user2vec) в архітектурі DSSM

Реалізація ранжувальної функції помилок – функція triplet_loss (рис. 3.11). На вхід ми приймаємо вектор-представлення анхора, позитивного та негативного елемента, об’єднані в один вектор за допомогою операції конкатенації [33] після приведення їх функціями трансформацій item2vec та user2vec у інший векторний простір (розділ 2.3.3). Була реалізована запропонована архітектура (рис 2.3) та відповідна функція похибок. Дана функція помилок означається як основна; її значення виводиться в процесі тренування (розділ. 3.3.4).

```
def triplet_loss(y_true, y_pred, n_dims=50, alpha=0.4):


    anchor = y_pred[:, 0:n_dims]
    positive = y_pred[:, n_dims:n_dims*2]
    negative = y_pred[:, n_dims*2:n_dims*3]

    pos_dist = K.sum(K.square(anchor - positive), axis=1)
    neg_dist = K.sum(K.square(anchor - negative), axis=1)

    basic_loss = pos_dist - neg_dist + alpha
    loss = K.maximum(basic_loss, 0.0)

    return loss
```

Рис. 3.11 Реалізація ранжувальної функції похибок методу DSSM – функція triplet_loss

Реалізація генератору триплетів – функція generaror (рис. 3.12). На вхід передаємо трансформовані таблиці метаданих користувачів та об’єктів, таблицю взаємодій (як правило передаємо лише її тренувальну частину) та всі ідентифікатори користувачів та об’єктів. Розмір вибірки по замовчуванню встановляємо рівним 1024. Функція реалізує наступний алгоритм:

- випадково отримуємо ідентифікатор користувача – це буде наш анхор;
- для вибраного користувача отримуємо всі його взаємодії;
- серед вибраного списку обираємо взаємодію випадковим чином, надаючи перевагу тим, що матимуть вищий рейтинг – об’єкт, що прийматиме участь у взаємодії, буде позитивним елементом;
- серед списку всіх можливих об’єктів випадково вибираємо об’єкт – це буде негативний елемент.

Для кожного триплету вставляємо його частини у відповідні списки. Після формування повної вибірки передаємо триплети у вигляді списку списків.

При цьому зауважимо, що формально функція повертає два параметра, але вся інформація буде міститься в першому, тобто другий параметр буде фіктивним. Такий спосіб реалізації був вибраний через недолік узагальнення модуля TensorFlow – для тренування ми завжди маємо незалежні та залежні змінні. Дана реалізація передбачає:

- автоматичну генерацію вибірки в ході тренування;
- адаптацію для підходу ранжування, коли ми не маємо вектор залежних змінних.

Щоб бути впевненим, що ми тренуємося лише на тренувальній вибірці, в генератор можемо передавати лише тренувальну частину таблиці всіх взаємодій.

```
def generator(users, items, interactions, all_users, all_items,
              batch_size=1024):

    while True:
        anchor = []
        pos = []
        neg = []

        for _ in range(batch_size):

            uid_i = np.random.choice(all_users)

            uid_i_df = interactions[interactions.uid == uid_i]
            uid_i_df = uid_i_df.reset_index()

            pos_i = np.random.choice(uid_i_df.iid.values,
                                     p=uid_i_df.rating.values / uid_i_df.rating.values.sum())
            neg_i = np.random.choice(all_items)

            anchor.append(users.iloc[int(uid_i), 1:].values)
            pos.append(items.iloc[int(pos_i), 1:].values)
            neg.append(items.iloc[int(neg_i), 1:].values)

        yield [np.array(anchor), np.array(pos), np.array(neg)],
         [np.array(['x']*np.array(anchor).shape[0]), np.array(['x']*np.array(anchor).shape[0])]
```

Рис. 3.12 Реалізація генератора триплетів методу DSSM – функція generator

Означимо функції трансформації векторів користувача та об’єкта (рис. 3.13 та 3.14 відповідно). При цьому зауважу, що для різних даних була

реалізована різна архітектура, враховуючи відмінність в даних. У випадку даних MovieLens для функції користувача була використана Resnet [34] архітектура, що дозволяє градієнтам на великій кількості ітерацій не перетворюватися в нуль; для об'єкта – звичайна MLP [33] архітектура, враховуючи більшу кількість вхідних параметрів. На виході як для користувача, так і для об'єкта отримуємо вектор-представлення розміром 50. В даному блоці можемо експериментувати розбиваючи вхідний вектор (наприклад для об'єкта), індивідуально застосовуючи свої архітектури для частин розбиття (рис. 3.15)

```
def user_model():

  inp_meta = tf.keras.layers.Input(shape=29)

  meta_1 = tf.keras.layers.Dense(50, activation='elu')(inp_meta)
  meta_2 = tf.keras.layers.Dense(50, activation='elu')(meta_1)
  add_meta = tf.keras.layers.Add()([meta_1, meta_2])

  return tf.keras.models.Model(inp_meta, add_meta)
```

Рис. 3.13 Реалізація функції трансформації вектору користувача для методу DSSM в даних MovieLens – функція user_model

```
def item_model():

    inp = tf.keras.layers.Input(shape=132)

    dense1_cat = tf.keras.layers.Dense(150, activation='elu')(inp)
    dense2_cat = tf.keras.layers.Dense(50, activation='elu')(dense1_cat)

    return tf.keras.models.Model(inp, dense2_cat)
```

Рис. 3.14 Реалізація функції трансформації вектору об'єкта для методу DSSM в даних MovieLens – функція item_model

```
def item_model():

    inp = tf.keras.layers.Input(shape=14)
    inp_category, inp_geo, inp_dense = inp[:, :5], inp[:, 5:6], inp[:, 6:]


    emb_geo = tf.keras.layers.Embedding(vocab_geo+1, embedding_size_geo)(inp_geo)
    add_geo = tf.math.reduce_sum(emb_geo, axis=1)

    dense1_cat = tf.keras.layers.Dense(50, activation='elu')(inp_category)
    dense2_cat = tf.keras.layers.Dense(50, activation='elu')(dense1_cat)
    add_cat = tf.keras.layers.Add()([dense1_cat, dense2_cat])

    dense1 = tf.keras.layers.Dense(50, activation='elu')(inp_dense)
    dense2 = tf.keras.layers.Dense(50, activation='elu')(dense1)
    add_dense = tf.keras.layers.Add()([dense1, dense2])

    concat = tf.keras.layers.concatenate([add_geo, add_cat, add_dense])
    out = tf.keras.layers.Dense(50, activation='linear')(concat)

    return tf.keras.models.Model(inp, out)
```

Рис. 3.15 Реалізація функції трансформації вектору об’єкта для методу DSSM в даних Google Local Review – функція item_model

Основна модель – поєднує user_model та item_model (рис. 3.16):

```
u2v = user_model()
i2v = item_model()

anchor_in = tf.keras.layers.Input(shape=29)
pos_in = tf.keras.layers.Input(shape=132)
neg_in = tf.keras.layers.Input(shape=132)

anchor = u2v(anchor_in)
pos = i2v(pos_in)
neg = i2v(neg_in)

res = tf.keras.layers.Concatenate(name="concat_ancor_pos_neg")([anchor, pos, neg])
```

Рис. 3.16 Приклад означення основної моделі для методу DSSM в даних MovieLens

```
Model: "model_29"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_58 (InputLayer)          [(None, 29)]         0           []

 input_59 (InputLayer)          [(None, 132)]        0           []

 input_60 (InputLayer)          [(None, 132)]        0           []

 model_27 (Functional)          (None, 50)           4050        ['input_58[0][0]']

 model_28 (Functional)          (None, 50)           9200        ['input_59[0][0]',
                                                                  'input_60[0][0]']

 concat_ancor_pos_neg (Concaten  (None, 150)         0           ['model_27[0][0]',
 ate)                                                             'model_28[0][0]',
                                                                  'model_28[1][0]']

==================================================================================================
Total params: 13,250
Trainable params: 13,250
Non-trainable params: 0
```

Рис. 3.17 Приклад означення основної моделі для методу DSSM в даних MovieLens

3.2.2 Реалізація архітектури GRU

Основні блоки, використовані в даній архітектурі – генератор та функції трансформації послідовності і об’єкта.

В даному випадку перед нами не стоїть задача генерувати триплети, тепер нам необхідно отримати N послідовних взаємодій для користувача та наступний об’єкт, з яким відбулася взаємодія. Відразу накладаємо обмеження для користувачів: ми маємо вибирати лише серед тих, які мають більше ніж N взаємодій. Отримавши такий список ми беремо всі можливі пари: беремо всі послідовності та наступні елементи починаючи з позиції 0 та закінчуючи позицією $m - N$ де m – загальна кількість взаємодії для користувача. Цю логіку реалізує допоміжна функція f (рис. 3.19). По аналогії з генератором для методу DSSM (рис. 3.12), щоб впевнитися, що ми не тренуємося на тестовій вибірці – передаємо в якості параметра interactions лише тренувальну частину таблиці

взаємодій. Тренувальну вибірку було сформовано для послідовностей з $N = 10$ взаємодіями. Реалізація генератору (рис. 3.18):

```
def generator_gru(interactions, SEQUENCE_LENGTH, f):

  temp = interactions.groupby('user_id').count()
  temp_idx = temp[temp.gmap_id > SEQUENCE_LENGTH].index.unique()

  users_with_more_than_N_interactions_df = interactions[
        interactions.user_id.isin(temp_idx)
       ]

  lst_seq, lst_item = [], []
  for user_id in tqdm(users_with_more_than_N_interactions_df.user_id.unique()):

    user_df = users_with_more_than_N_interactions_df[
        users_with_more_than_N_interactions_df.user_id == user_id
        ]
    arr_seq, arr_item = f(user_df['iid'].values)

    lst_seq += arr_seq
    lst_item += arr_item
```

Рис. 3.18 Реалізація генератора даних для методу GRU– функція generator_gru

```
def f(lst, N=10):

  # dummy function for fetching all possible sequences & items

  lst1 = []
  lst2 = []

  for _ in range(0, len(lst)-N):

    seq = lst[_:_+N]
    item = lst[_+N]

    lst1.append(seq)
    lst2.append(item)

  return lst1, lst2
```

Рис. 3.19 Реалізація допоміжної функції f

Реалізація функцій трансформації для послідовності та об’єкта наступні (рис. 3.20). Архітектура для обох функцій схожа – спочатку ми отримуємо певне приховане семантичне представлення за допомогою Embedding layer [22];

механізм представлення схожий на метод Word2Vec [22]. Єдина відмінність функцій-трансформацій у наявності кодування для послідовності методом GRU [25]. Архітектура рекурентних мереж дозволяє запам’ятовувати послідовно введену інформацію, представлену у векторах. Для кодування представлення послідовності було взято $N = 10$ блоків GRU (рис. 3.20).

```
s2v = tf.keras.Sequential([
          tf.keras.layers.StringLookup(vocabulary=unique_item_ids, mask_token=None),
          tf.keras.layers.Embedding(len(unique_item_ids) + 1, embedding_dimension),
          tf.keras.layers.GRU(embedding_dimension),
          tf.keras.layers.Dense(64, activation="linear")])

i2v = tf.keras.Sequential([
          tf.keras.layers.StringLookup(vocabulary=unique_item_ids, mask_token=None),
          tf.keras.layers.Embedding(len(unique_item_ids) + 1, embedding_dimension),
          tf.keras.layers.Dense(64, activation="linear")
          ])
```

Рис. 3.20 Реалізація функції-трансформації послідовності об’єктів (s2v) та наступного об’єкта (i2v)

Архітектура генератора, функцій-трансформацій є однаковою для кожної вибірки даних.

3.2.3 Реалізація інших допоміжних модулів

Крім фрагментів програм, що відносяться до архітектур DSSM та GRU було використано універсальні допоміжні функції – generate_{*model_name*}_recs_mapper (рис. 3.21) для створення рекомендацій, враховуючи ранг, та compute_metrics (рис. 3.22) – для обчислення значень метрик якості по згенерованим рекомендаціям.

Для кожного методу ми по-своєму визначаємо спосіб створення рекомендацій, але в кінці маємо отримати вектор recs, що представляє собою

міри близькості вибраного вектору користувача до всіх об’єктів. Далі реалізація збігається для всіх методів.

```
def generate_gru_recs_mapper(s2v, i2v,
                             train_,
                             known_items,
                             N, unique_item_ids,
                             items_enc, num_threads=4):
    def _recs_mapper(user):

        seq = train_[train_.uid== user].iid.values[-N:]
        seq_enc = s2v.predict(seq.reshape(-1, N))
        dists = ED(seq_enc, items_enc)
        recs = dists.reshape(-1, )

        additional_N = len(known_items[user]) if user in known_items else 0
        total_N = N + additional_N
        top_cols = np.argpartition(recs, -np.arange(total_N))[-total_N:][::-1]

        final_recs = [item for item in unique_item_ids[top_cols]]
        if additional_N > 0:
            filter_items = known_items[user]
            final_recs = [item for item in final_recs if item not in filter_items]
        return final_recs[:N]
    return _recs_mapper
```

Рис. 3.21 Реалізація функції generate_gru_recs_mapper для створення топ N рекомендацій для архітектури GRU

```
def compute_metrics(df_true, df_pred, top_N):
    result = {}
    test_recs = df_true.set_index(['uid', 'iid']).join(df_pred.set_index(['uid', 'iid']))
    test_recs = test_recs.sort_values(by=['uid', 'rank'])

    test_recs['users_item_count'] = test_recs.groupby(level='uid')['rank'].transform(np.size)
    test_recs['reciprocal_rank'] = (1 / test_recs['rank']).fillna(0)
    test_recs['cumulative_rank'] = test_recs.groupby(level='uid').cumcount() + 1
    test_recs['cumulative_rank'] = test_recs['cumulative_rank'] / test_recs['rank']

    users_count = test_recs.index.get_level_values('uid').nunique()
    for k in range(1, top_N + 1):
        hit_k = f'hit@{k}'
        test_recs[hit_k] = test_recs['rank'] <= k
        result[f'Precision@{k}'] = (test_recs[hit_k] / k).sum() / users_count
        result[f'Recall@{k}'] = (test_recs[hit_k] / test_recs['users_item_count']).sum() / users_count

    result[f'MAP@{top_N}'] = (test_recs["cumulative_rank"] / test_recs["users_item_count"]).sum() / users_count
    result[f'MRR'] = test_recs.groupby(level='uid')['reciprocal_rank'].max().mean()
    return pd.Series(result)
```

Рис. 3.22 Реалізація функції compute_metrics для обчислень метрик якості

3.3 Результати

3.3.1 Результати для алгоритму SVD

Для задачі класичної матричної факторизації надалі будемо обраховувати класифікаційні метрики – $Precision@K$ та $Recall@K$ (табл. 3.13):

Таблиця 3.13 Значення метрик Precision@K та Recall@K для методу SVD на даних MovieLens для різної довжини рекомендованої вибірки (k) та різної кількості врахованих найбільших сингулярних чисел

	Precision@k			Recall@k		
	кількість врахованих найбільших сингулярних чисел			кількість врахованих найбільши врахованих сингулярних чисел		
k	25	50	100	25	50	100
1	0.065965	0.082688	0.079591	0.001769	0.002802	0.003181
2	0.066894	0.074636	0.0703	0.003846	0.005032	0.005602
3	0.066481	0.069991	0.065036	0.006462	0.007262	0.008008
4	0.06519	0.066197	0.061397	0.009177	0.009743	0.010565
5	0.065345	0.064726	0.059151	0.011509	0.012184	0.012761
6	0.064158	0.062868	0.05848	0.01417	0.014399	0.015381
7	0.062735	0.060788	0.056984	0.016435	0.016468	0.017368
8	0.061126	0.060158	0.056519	0.018223	0.018806	0.020135
9	0.059771	0.059874	0.055297	0.019814	0.020966	0.021888
10	0.059151	0.058253	0.053856	0.02186	0.022976	0.023603

Також порахуємо точність самого наближення – регресійну метрику RMSE. Оцінимо наскільки точно ми змогли наблизити тренувальну матрицю взаємодій (табл. 3.14):

Таблиця 3.14 Значення метрики RMSE для методу SVD на даних MovieLens для різної кількості врахованих найбільших сингулярних чисел

RMSE		
кількість врахованих найбільших сингулярних чисел		
25	50	100
5.69	4.97	4.09

Обчислювальна складність алгоритму SVD занадто висока для даних Google Review Data, тому обчислювати метрики за допомогою даного алгоритму для даної вибірки ми не будемо. Скористаємося фактом, що результати LightFM, лише використовуючи матрицю взаємодії, будуть співставними з результатами стандартної матричної факторизації [7].

3.3.2 Результати для матричної факторизації методом ALS

Скористаємося готовою програмною імплементацією – класом ExplicitMF [8] (табл. 3.15):

Таблиця 3.15 Значення метрик Precision@K та Recall@K для матричної факторизації методом ALS на даних MovieLens для різної довжини рекомендованої вибірки (k)

k	Precision@k	Recall@k
1	0.084546	0.003405

Продовження таблиці 3.15

k	Precision@k	Recall@k
2	0.079436	0.006585
3	0.074326	0.009473
4	0.071694	0.011913
5	0.069309	0.014535
6	0.066274	0.01667
7	0.064505	0.01936
8	0.062868	0.021568
9	0.061767	0.02349
10	0.061041	0.025628

По аналогії з попереднім методом порахуємо якість наближення тренувальної матриці:

$$RMSE \approx 5.25$$

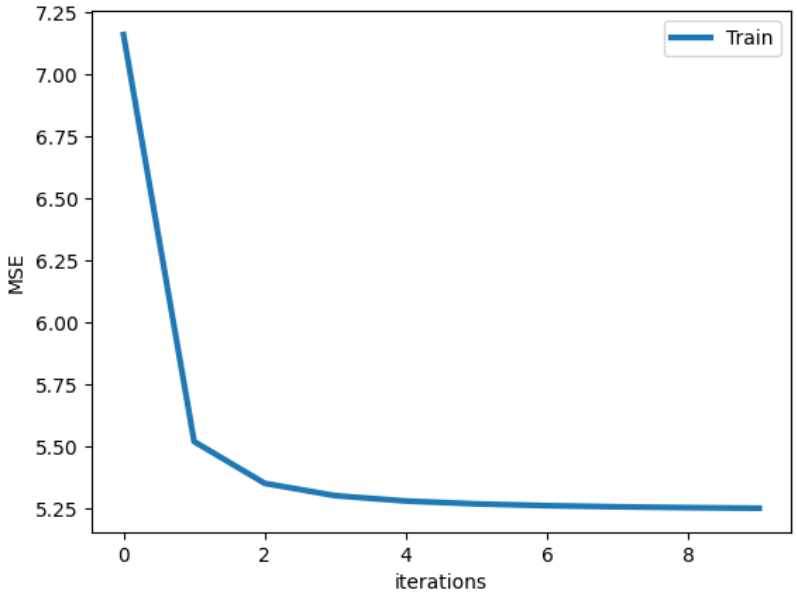


Рис. 3.23 Процес мінімізації тренувальної функції похибок для матричної факторизації методом ALS на даних MovieLens

По аналогії з методом SVD для даних Google Review Data обчислювальна складність алгоритму занадто висока, тому обчислювати метрики за допомогою даного алгоритму для даної вибірки ми також не будемо.

3.3.3 Результати для методу LightFM

Спочатку реалізуємо метод лише на індикаторних особливостях: результати мають бути співставні з результатами попередніх методів, отримані для стандартної матричною факторизацією [7]. Порахуємо як класифікаторні $Precision@K$ та $Recall@K$, так і ранжувальні метрики $MAP@10$ та MRR, оскільки в даному методі вирішуємо вже іншу задачу – ранжування (табл. 3.16):

Таблиця 3.16 Значення метрик Precision@K та Recall@K для матричної факторизації методом LightFM, використовуючи лише індикаторні особливості, на даних MovieLens для різної довжини рекомендованої вибірки (k)

k	Precision@k	Recall@k
1	0.048043	0.002058
2	0.046452	0.004402
3	0.045286	0.006535
4	0.043509	0.008478
5	0.043398	0.010625
6	0.042581	0.012444
7	0.042089	0.014305
8	0.041958	0.016295
9	0.041114	0.017763
10	0.040598	0.019374

$$MAP@10 \approx 0.0124$$

$$MRR \approx 0.1097$$

Спробуємо використати метадані користувача та об’єкта (табл. 3.17):

Таблиця 3.17 Значення метрик Precision@K та Recall@K для матричної факторизації методом LightFM, використовуючи метадані, на даних MovieLens для різної довжини рекомендованої вибірки (k)

k	Precision@k	Recall@k
1	0.022272	0.000617
2	0.030862	0.002642
3	0.037013	0.006071
4	0.035953	0.008566
5	0.033598	0.009496
6	0.033673	0.01098
7	0.033544	0.012147
8	0.032851	0.013055
9	0.032135	0.013749
10	0.031785	0.014487

$$MAP@10 \approx 0.0087$$

$$MRR \approx 0.0805$$

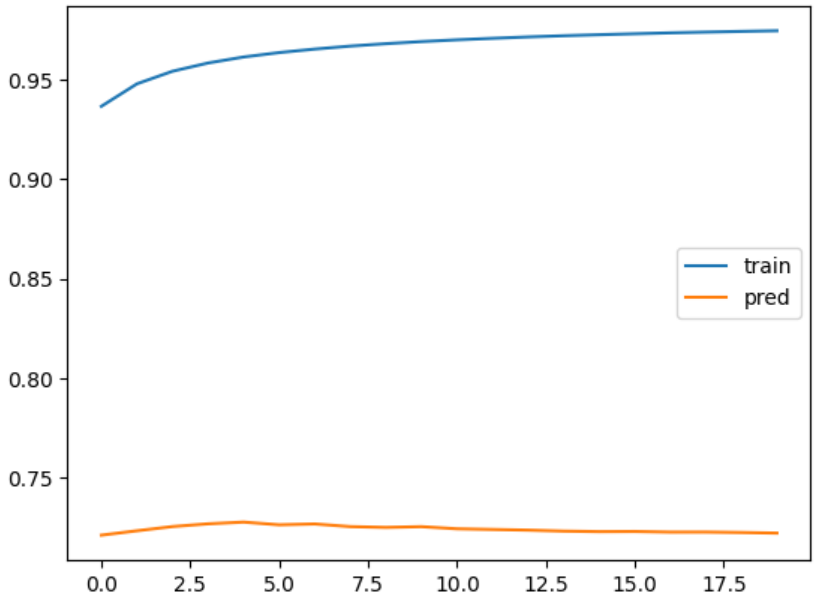


Рис. 3.24 Значення коефіцієнта AUC для методу LightFM лише на індикаторних особливостях на даних MovieLens

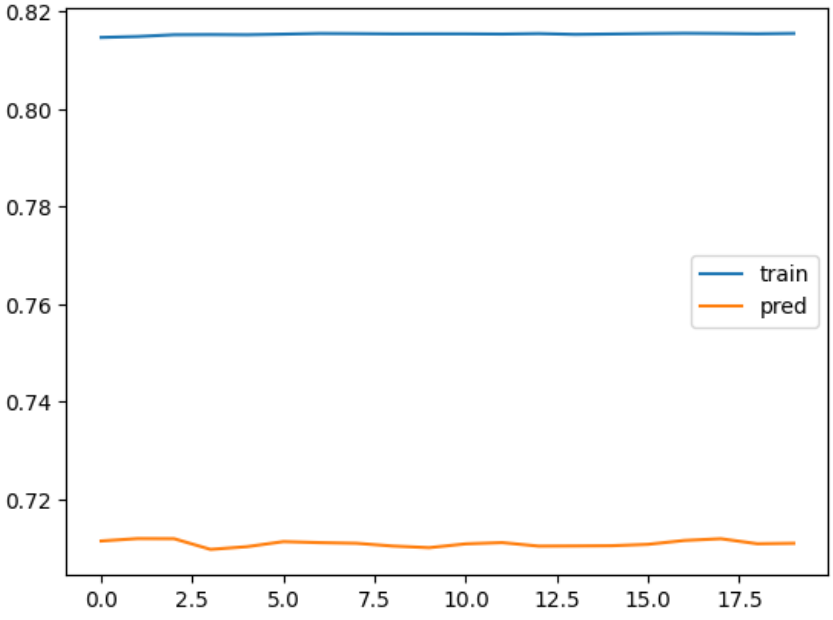


Рис. 3.25 Значення коефіцієнта AUC для методу LightFM на індикаторних особливостях та метаданих користувача і об’єкта на даних MovieLens

Аналогічні розрахунки виконаємо і для іншої вибірки – Google Review Data (табл. 3.18 та 3.19):

Таблиця 3.18 Значення метрик Precision@K та Recall@K для матричної факторизації методом LightFM, використовуючи лише індикаторні особливості, на даних Google Review Data для різної довжини рекомендованої вибірки (k)

k	Precision@k	Recall@k
1	0.0361	0.004272
2	0.031238	0.008054
3	0.028329	0.010958
4	0.027408	0.014338
5	0.026562	0.017435
6	0.025732	0.020137
7	0.02491	0.02251
8	0.024227	0.024747
9	0.023608	0.026963
10	0.022992	0.028935

$$MAP@10 \approx 0.014$$

$$MRR \approx 0.067$$

Таблиця 3.19 Значення метрик Precision@K та Recall@K для матричної факторизації методом LightFM, використовуючи метадані, на даних Google Review Data для різної довжини рекомендованої вибірки (k)

k	Precision@k	Recall@k
1	0.025976	0.002608
2	0.023578	0.005412

Продовження таблиці 3.19

k	Precision@k	Recall@k
3	0.022601	0.007567
4	0.02168	0.009634
5	0.02118	0.011971
6	0.020647	0.01394
7	0.019848	0.015409
8	0.019199	0.016811
9	0.018457	0.018515
10	0.01809	0.020048

$$MAP@10 \approx 0.0096$$

$$MRR \approx 0.0524$$

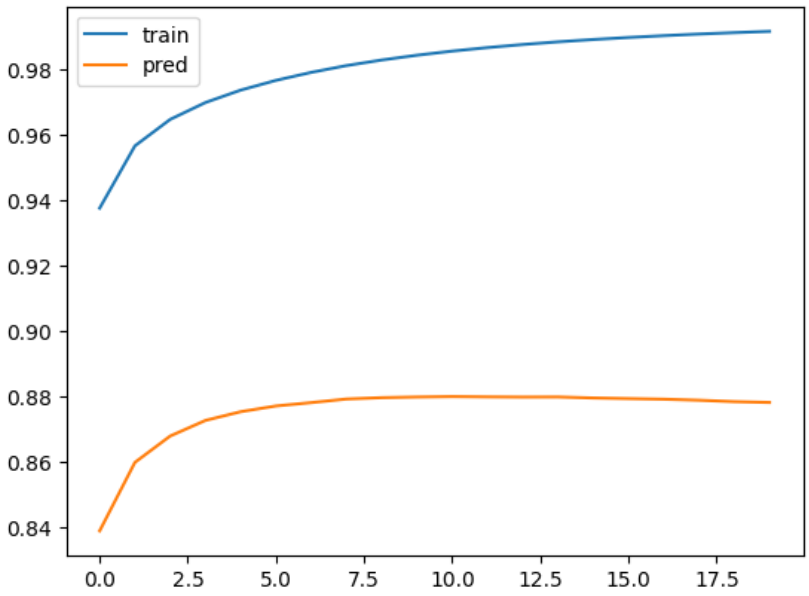


Рис. 3.26 Значення коефіцієнта AUC для методу LightFM лише на індикаторних особливостях на даних Google Review Data

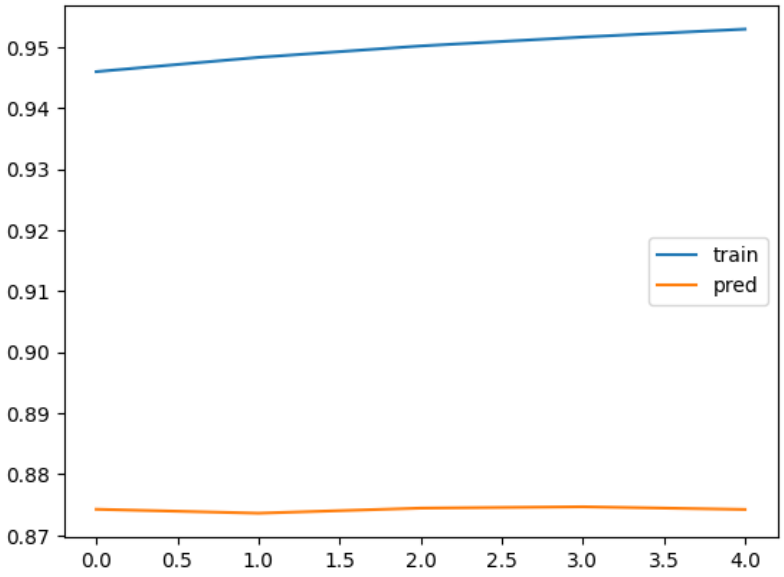


Рис. 3.27 Значення коефіцієнта AUC для методу LightFM на індикаторних особливостях та метаданих користувача і об’єкта на даних Google Review Data

3.3.4 Результати для методу DSSM

Даний метод передбачає використання власних особливостей користувача та об’єкта, без представлення їх у індикаторному вигляді. Реалізуємо підхід user2vec – для користувача будемо рекомендувати об’єкти (табл. 3.20 та 3.21):

Таблиця 3.20 Значення метрик Precision@K та Recall@K для методу DSSM на даних MovieLens для різної довжини рекомендованої вибірки (k)

k	Precision@k	Recall@k
1	0.00928	0.00004
2	0.01072	0.000087
3	0.012053	0.000126

Продовження таблиці 3.20

k	Precision@k	Recall@k
4	0.0132	0.000195
5	0.014208	0.000262
6	0.014453	0.000324
7	0.01536	0.000411
8	0.01572	0.000494
9	0.01568	0.000538
10	0.015904	0.000588

$$MAP@10 \approx 0.00016$$

$$MRR \approx 0.033$$

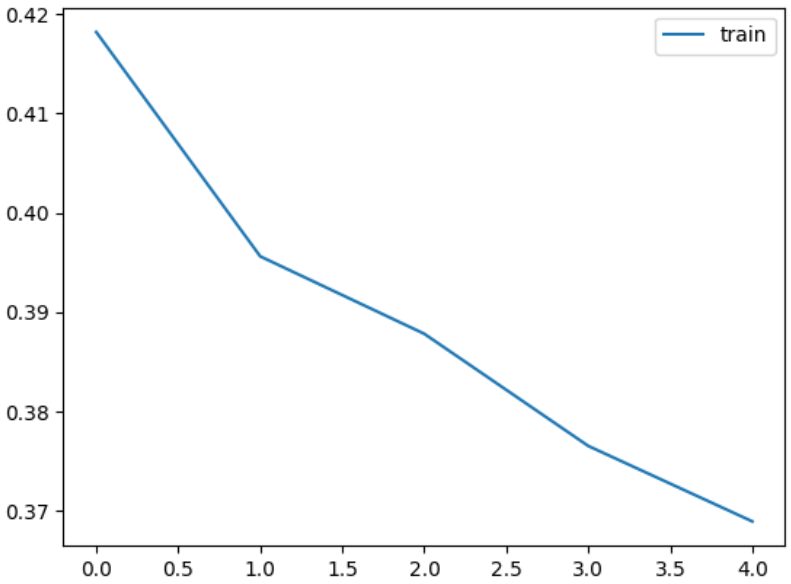


Рис. 3.28 Процес мінімізації тренувальної функції похибок для методу DSSM на даних MovieLens

Таблиця 3.21 Значення метрик Precision@K та Recall@K для методу DSSM на даних Google Review Data для різної довжини рекомендованої вибірки (k)

k	Precision@k	Recall@k
1	0.001394	0.000026
2	0.001390	0.000049
3	0.0015347	0.000067
4	0.0014741	0.000092
5	0.0015347	0.000117
6	0.0016589	0.000139
7	0.0016048	0.000156
8	0.0018522	0.000185
9	0.0020743	0.000202
10	0.0017741	0.000219

$$MAP@10 \approx 0.0052$$

$$MRR \approx 0.0089$$

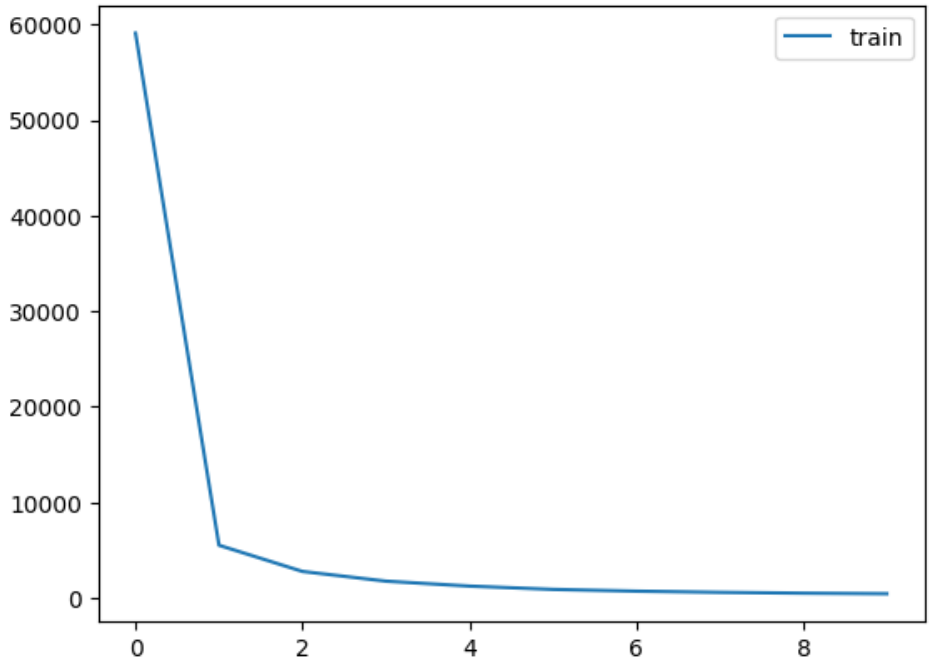


Рис. 3.29 Процес мінімізації тренувальної функції похибок для методу DSSM на даних Google Review Data

3.3.5 Результати для методів Sequential

Для реалізованої архітектури GRU маємо наступні результати (табл. 3.22 та 3.23):

Таблиця 3.22 Значення метрик Precision@K та Recall@K для реалізованого методу GRU на даних MovieLens для різної довжини рекомендованої вибірки (k)

k	Precision@k	Recall@k
1	0.007433	0.000149
2	0.006968	0.000307
3	0.0064	0.000404

Продовження таблиці 3.22

k	Precision@k	Recall@k
4	0.006116	0.000454
5	0.006132	0.000531
6	0.006658	0.000728
7	0.006813	0.0008
8	0.007278	0.00098
9	0.007157	0.001068
10	0.007278	0.001185

$$MAP@10 \approx 0.00038$$

$$MRR \approx 0.019$$

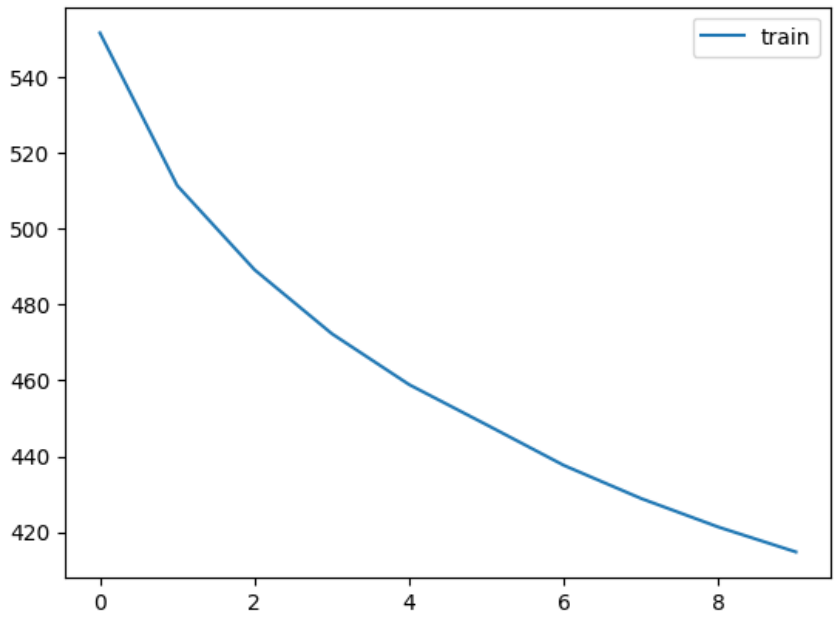


Рис. 3.30 Процес мінімізації тренувальної функції похибок для реалізованого методу GRU на даних MovieLens

Таблиця 3.23 Значення метрик Precision@K та Recall@K для реалізованого методу GRU на даних Google Review Data для різної довжини рекомендованої вибірки (k)

k	Precision@k	Recall@k
1	0.001867	0.000189
2	0.002230	0.000385
3	0.002212	0.000515
4	0.002152	0.000696
5	0.002198	0.000974
6	0.002039	0.001044
7	0.002059	0.001212
8	0.001996	0.001507
9	0.001936	0.001736
10	0.001898	0.001997

$$MAP@10 \approx 0.00056$$

$$MRR \approx 0.00571$$

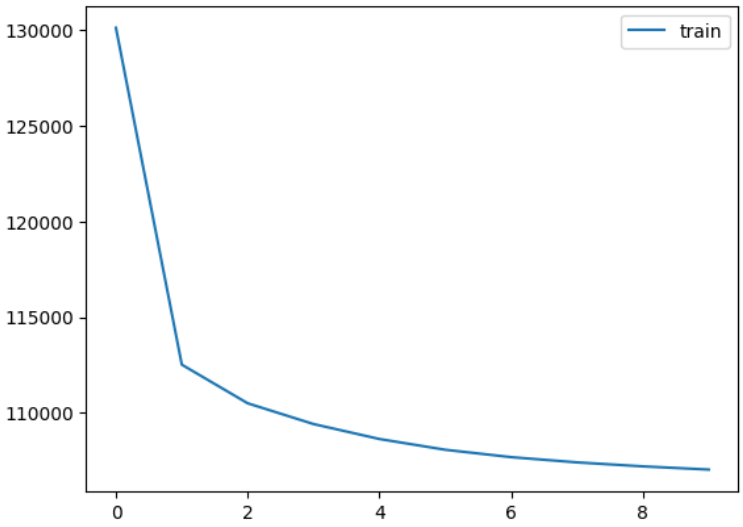


Рис. 3.31 Процес мінімізації тренувальної функції похибок для реалізованого методу GRU на даних Google Review Data

Використаємо готову програмну реалізацію бібліотеки Recbole схожого алгоритму – GRU4Rec [14, 17] (табл. 3.24) та іншу архітектуру Recbole – Caser [28, 40] (табл. 3.25):

Таблиця 3.24 Результати метрик якості для методу GRU4Rec після 10 ітерації на даних MovieLens

	Recall@10	MRR
	0.244	0.0949

Таблиця 3.25 Результати метрик якості для методу Caser після 10 ітерації на даних MovieLens

	Recall@10	MRR
	0.241	0.092

Порахуємо результати GRU4Rec та Caser для вибірки Google Data Reviews (табл. 3.26 та 3.27):

Таблиця 3.26 Результати метрик якості для методу GRU4Rec після 10 ітерації на даних Google Data Reviews

	Recall@10	MRR
	0.0948	0.0346

Таблиця 3.27 Результати метрик якості для методу Caser після 10 ітерації на даних Google Data Reviews

	Recall@10	MRR
	0.0786	0.0264

Висновки до розділу 3

В ході реалізації програмного продукту на двох вибірках даних – MovieLens [37] та Google Local Review [38] було пораховано метрики якості. Дані MovieLens є класичними для задачі рекомендацій, оскільки мають значну кількість взаємодій, як у таблиці взаємодії, так і на користувача або об’єкта, якщо порахувати середню кількість; крім того природа даних є сприятливою для використання у задачах рекомендацій: в історії переглядів користувача, дійсно, може зберігатися інформація про вподобання, а вибрати наступний фільм для перегляду на онлайн-платформі – достатньо просто. Дані Google Local Review мають схоже наповнення по кількості взаємодій, але меншу середню кількість взаємодій на користувача, це призводить до ряду проблем: з одного боку нам складніше буде зробити висновок про вподобання людини, оскільки взаємодій буде менше, з іншого боку, в методах, де безпосередньо використовується матрична факторизація – складність алгоритму кратно зростає і обчислення значно ускладнюються. В даних, так чи інакше, зберігається інформація про вподобання, але на відміну від попередньої вибірки, «сила» такого вподобання є меншою: відгук на наступну локацію може бути ніяк не пов’язаний з попередніми відгуками і може бути зумовлений випадковістю. Проведено попередню трансформацію даних для обох вибірок.

В роботі програмно реалізовано два методи – DSSM та архітектуру GRU. Для методу DSSM було створено важливі частини: ранжувальну функцію помилок, генератор для даних, моделі-трансформації представлення для користувача та об’єкта, всі частини було об’єднану в основну модель. Для різних вибірок даних архітектура моделей-трансформацій відрізнялася, враховуючи особливості даних. Для методу GRU був створений свій генератор даних, означено моделі-трансформації послідовності та об’єкта. Крім цього

було реалізовано додаткові модулі для створення рекомендацій по рангу та модулі обчислення якості рекомендацій.

Найкращі результати по метрикам якості для обох вибірок показали методи послідовної рекомендації – GRU4Rec [25] та Caser [28], проте час на тренування таких методів – значний. Найшвидшими по часу роботи виявилися методу класичної матричної факторизації – SVD та ALS [7], для цих алгоритмів важливим фактором виявалася кількість необхідної пам’яті для обчислень. Ця причина унеможливила розрахунки для вибірки Google Data Review. Метод LightFM також можна віднести до класу швидких методів, але на відміну від SVD та ALS, він дає змогу провести обчислення на значних об’ємах даних, врахувавши в обчисленнях метадані. Реалізовані методи DSSM та метод GRU по результатам показали себе найгірше, але дали змогу провести експерименти з різними архітектурами всередині основної моделі. У випадку DSSM, таку архітектуру реалізуючи всі сучасні продукти, пов’язані з рекомендаціями [20, 21], використовуючи при цьому будь-яку наявну інформацію: відео, фото, аудіо, текст та гео-дані. В роботі було реалізовано схожу архітектуру, але з меншою кількістю вхідних даних. У випадку методу GRU ми отримали гірші результати, оскільки архітектура послідовної мережі не передбачає застосування її для створення рекомендацій, тому її треба адаптувати до такої задачі [27]. Це стосується як і способу формування тренувальної вибірки, так і використання іншої функції похибок.

4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

В заданому розділі буде проведено оцінювання основних характеристик для майбутнього програмного продукту, що спеціалізується на генерації рекомендацій для користувача.

Дана реалізація буде сприяти проведенню усіх необхідних досліджень, що дасть змогу якісно дослідити питання ефективної імплементації оглянутих алгоритмів.

Також в даному дослідженні показано різні варіанти реалізації для забезпечення найбільш коректної та оптимальної стратегії вибору чинників, що мають вплив на різні фактори та сумісність з майбутнім програмним продуктом. Для цього застосовувався апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) передбачає собою технологію, що дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. ФВА проводиться з метою виявлення резервів зниження витрат за рахунок ефективніших варіантів виробництва, кращого співвідношення між споживчою вартістю виробу та витратами на його виготовлення. Для проведення аналізу використовується економічна, технічна та конструкторська інформація.

Алгоритм функціонально-вартісного аналізу включає в себе визначення послідовності етапів розробки продукту, визначення повних витрат (річних) та кількості робочих часів, визначення джерел витрат та кінцевий розрахунок вартості програмного продукту.

4.1 Постановка завдання

У роботі застосовується метод ФВА для проведення техніко-економічного аналізу розробки системи створення рекомендацій. Оскільки рішення стосовно проектування та реалізації компонентів, що розробляється, впливають на всю систему, кожна окрема підсистема має її задовольняти. Тому фактичний аналіз представляє собою аналіз функцій програмного продукту, призначеного для збору, обробки та проведення аналізу даних по компанії.

Технічні вимоги до програмного продукту є наступні:

- функціонування на персональних комп'ютерах із стандартним набором компонентів;
- зручність та зрозумілість для користувача;
- швидкість обробки даних та доступ до інформації в реальному часі;
- можливість зручного масштабування та обслуговування;
- мінімальні витрати на впровадження програмного продукту.

4.2 Обґрунтування функцій програмного продукту

Головна функція F_0 – розробка можливого програмного продукту, яка дозволяє аналізувати різні характеристики, що безпосередньо впливають на стійкість підприємства. Беручи за основу цю функцію, можна виділити наступні:

- F_1 – вибір мови програмування;
- F_2 – вибір сервісу для розробки ПЗ.

Кожна з цих функцій має декілька варіантів реалізації:

Функція F_1:

а) Python;

б) R;

Функція F_2:

а) Jupyter Notebook;

б) Google Colaboratory;

в) Kaggle Notebook.

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 4.1).

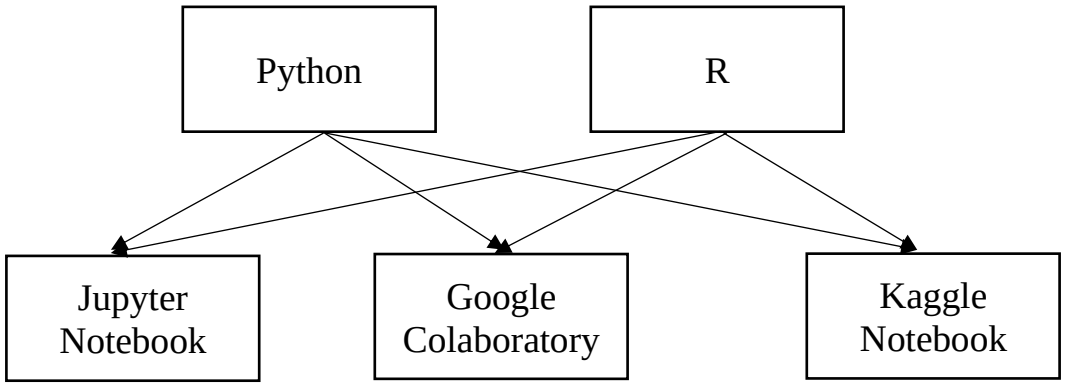


Рис. 4.1 – Морфологічна карта

Морфологічна карта відображає множину всіх можливих варіантів основних функцій. Позитивно-негативна матриця показана в таблиці 4.1.

Таблиця 4.1 - Позитивно-негативна матриця

Функції	Варіанти реалізації	Переваги	Недоліки
F_1	*А*	Мова програмування, створена для написання програм; чимала кількість бібліотек, що містять готову реалізацію методів, які будемо розглядати	Не знайшов
	Б	Добре підходить для проведення статистичного пост-аналізу, після отримання результатів; на більшій кількості ітерацій працює швидше	Відсутність необхідних бібліотек для роботи, шукати помилки складніше, синтаксис є складнішим

Продовження таблиці 4.1

Функції	Варіанти реалізації	Переваги	Недоліки
F_2	*А*	Чудово підходит для тестових запусків та розвідувального аналізу даних; зручний інтерфейс	Повільна швидкість виконання програми
	Б	Більш швидка робота програми, можливість додати обчислювальні одиниці та читат файли з cloud-диску; багато модулів вже встановлено на сервері	Інтерфейс є незручним саме для написання програм
	В	Робота програми швидка; можливість додати додаткові дані з інших зада машинного навчання швидко	Інтерфейс не є зручним; проблема з доступом через зареєстрований акаунт

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому, що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

- Функція F_1:

Для розробки нам необхідна мова програмування, що має зрозумілий синтаксис, щоб реалізовувати програму було просто. Окремим плюсом буде наявність модулів, в рамках яких вже реалізовано математичні методи, що будуть використовуватися. Мова програмування R – краще підходить під інші задачі, більш аналітичного характеру, тоді як Python – оптимальний варіант. Тому обираємо А.

- Функція F_2:

В процесі роботи я використовував всі середовища розробки. Варіант А – в самому початку, коли проводив аналіз вхідних даних чи тестував виконання окремих фрагментів програми. Надалі я більше використовував варіанти Б та В. Google Colaboratory використовував як основний інструмент, який працює швидко і має багато вже встановлених бібліотек; в ньому я робив основні запуски і обчислював метрики якості. В Kaggle Notebook я запускав на виконання дві послідовні моделі; вони виводили табло з інформацією про тренування на відміну від варіанту Б і працювали швидко. Обираємо варіанти Б та В.

Тоді наявні наступні комбінації:

- F_1а – F_2б;
- F_1а – F_2в.

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

4.3 Обґрунтування системи параметрів ПП

На основі даних, розглянутих вище, визначаються основні параметри вибору, які будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

- *X1* – швидкодія мови програмування;
- *X2* – об’єм пам’яті для обчислень та збереження даних;
- *X3* – час навчання даних;
- *X4* – потенційний об’єм програмного коду.

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію програмного продукту, як показано у таблиці 4.2.

Таблиця 4.2 - Основні параметри програмного продукту

Назва параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Швидкодія мови програмування	1	оп/мс	50	75	100
Об’єм пам’яті	2	Мб	32	16	8
Час попередньої обробки даних	3	мс	400	260	50
Потенційний об’єм програмного коду	4	кількість рядків коду	3000	1000	500

За даними таблиці 4.2 будуються графічні характеристики параметрів – рис. 4.2 – рис. 4.5.

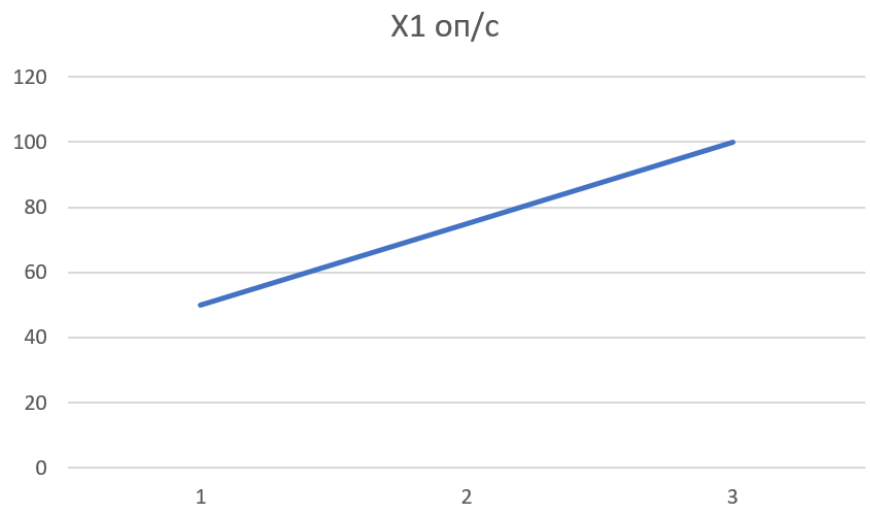


Рисунок 4.2 – X1, швидкодія мови програмування

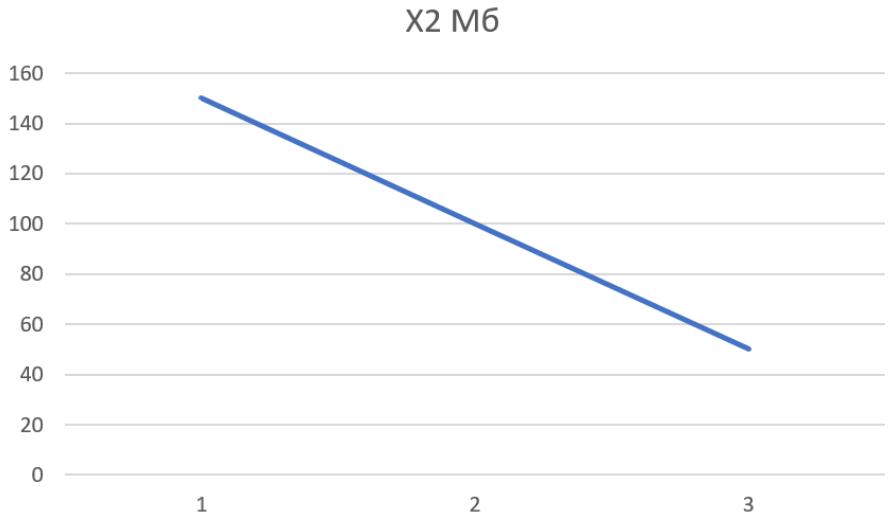


Рисунок 4.3 – X2, об’єм пам’яті

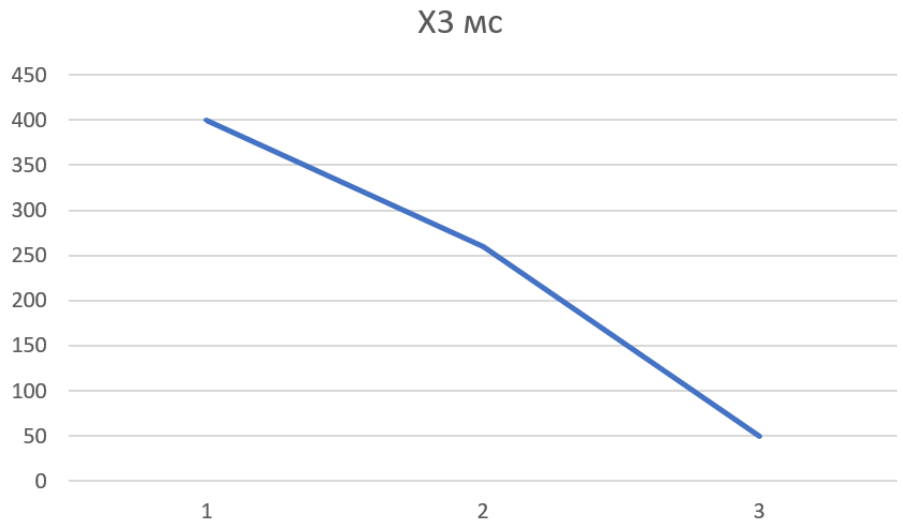


Рисунок 4.4 – X3, час попередньої обробки даних

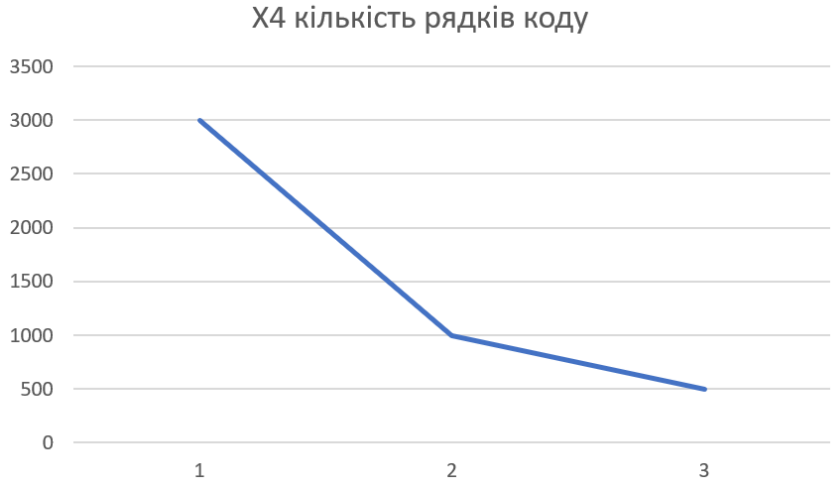


Рисунок 4.5 – X4, потенційний об’єм програмного коду

4.4 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який дає найбільш точні рекомендації.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

- визначення рівня значимості параметра шляхом присвоєння різних рангів;
- перевірку придатності експертних оцінок для подальшого використання;
- визначення оцінки попарного пріоритету параметрів;
- обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 4.3.

Таблиця 4.3 Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангі	Відхи-лення	i^2
			1	2	3	4	5	6	7			
X1	Швидкодія мови програмування	оп/мс	1	2	2	1	1	2	1	10	-7.5	56.25
X2	Об’єм пам’яті	Мб	2	1	1	2	2	1	2	11	-6.5	42.25
X3	Час попередньої обробки даних	мс	4	4	3	4	3	3	4	25	7.5	56.25

Продовження таблиці 4.3

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангі	Відхи-лення	$_i^2$
			1	2	3	4	5	6	7			
X4	Потенційний об'єм програмного коду	кількість рядків коду	3	3	4	3	4	4	3	24	6.5	42.25
	Разом		10	10	10	10	10	10	10	70	0	197

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70, \quad (4.1)$$

N – число експертів,

n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17{,}5 \quad (4.2)$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T. \quad (4.3)$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г) загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 197. \quad (4.4)$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 179}{7^2(4^3 - 4)} = 0{,}8 > W_k = 0{,}67. \quad (4.5)$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 4.4.

Таблиця 4.4 - Попарне порівняння параметрів

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	<	>	>	<	<	>	<	<	0,5
X1 і X3	<	<	<	<	<	<	<	<	0,5
X1 і X4	<	<	<	<	<	<	<	<	0,5
X2 і X3	<	<	<	<	<	<	<	<	0,5
X2 і X4	<	<	<	<	<	<	<	<	0,5
X3 і X4	>	>	<	>	<	<	>	>	1,5

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, a_{ij} визначається по формулі:

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases}. \quad (4.6)$$

З отриманих числових оцінок переваги складемо матрицю A=$\|a_{ij}\|$. Для кожного параметра зробимо розрахунок вагомості $K_{вi}$ за наступними формулами:

$$K_{\text{вi}} = \frac{b_i}{\sum_{i=1}^{n} b_i} \tag{4.7}$$

$$b_i = \sum_{i=1}^{N} a_{ij} \tag{4.8}$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{\text{вi}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}, \tag{4.9}$$

$$b_i' = \sum_{i=1}^{N} a_{ij} b_j \tag{4.10}$$

Як видно з таблиці 4.5, різниця значень коефіцієнтів вагомості не перевищує 2% (до округлення), тому більшої кількості ітерацій не потрібно.

Таблиця 4.5 - Розрахунок вагомості параметрів

Параметри x_i	Параметри x_j				Перша ітерація		Друга ітерація	
	X1	X2	X3	X4	b_i	$K_{\text{вi}}$	b_i^1	$K_{\text{вi}}^1$
X1	1	0,5	1,5	1,5	4,5	0,28	16,25	0,27
X2	1,5	1	1,5	1,5	5,5	0,34	21,25	0,36

Продовження таблиці 4.5

Параметри x_i	Параметри x_j				Перша ітерація		Друга ітерація	
	X1	X2	X3	X4	b_i	$K_{\text{ві}}$	b_i^1	$K_{\text{ві}}^1$
X3	0,5	0,5	1	1,5	3,5	0,22	12,25	0,21
X4	0,5	0,5	0,5	1	2,5	0,16	9,25	0,16
Всього:					16	1	59	1

4.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів *X2* (Об'єм пам'яті), *X3* (час попередньої обробки даних) та *X4* (потенційний об'єм програмного коду) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра *X1* (швидкість роботи мови програмування) обрано не найгіршим.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 4.6):

$$K_K(j) = \sum_{i=1}^{n} K_{вi,j} B_{i,j}\,, \qquad (4.11)$$

де n – кількість параметрів;

$K_{вi}$– коефіцієнт вагомості i–го параметра;

B_i – оцінка i–го параметра в балах.

Таблиця 4.6 Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	Python	*X1*	50	45	0,27	12.15
		X4	250	15	0.16	2.4
F2	Google Colaboratory	*X2*	100	25	0,36	9
	Kaggle Notebook	*X3*	290	15	0,21	3.15

За даними з таблиці 4.6 за формулою:

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}], \quad (4.12)$$

визначаємо рівень якості кожного з варіантів:

K_1 = 12.15 + 2.4 + 9 = 23.55

K_2 = 12.15 + 2.4 + 3.15 = 17.7

Як видно з розрахунків, кращим є 1 варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.6 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Розробка проекту програмного продукту;

2. Розробка програмної оболонки;

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Загальна трудомісткість обчислюється як:

$$T_O = T_P \cdot K_П \cdot K_{СК} \cdot K_M \cdot K_{СТ} \cdot K_{СТ.М}, \quad (4.13)$$

де T_P – трудомісткість розробки ПП;

$K_П$ – поправочний коефіцієнт;

$K_{СК}$ – коефіцієнт на складність вхідної інформації;

K_M – коефіцієнт рівня мови програмування;

$K_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$K_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: T_P =40 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $K_П$ = 1.8.

Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $К_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $К_{СТ}$ = 0.8. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$Т_1 = 40 \cdot 1.8 \cdot 0.8 = 48 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $Т_Р$ =25 людино-днів, $К_П$ =0.9, $К_{СК}$ = 1, $К_{СТ}$ =0.8:

$$Т_2 = 25 \cdot 0.9 \cdot 0.8 = 18 \text{ людино-днів.}$$

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$$Т_I = (48 + 18 + 4.8 + 18) \cdot 8 = 710.04 \text{ людино-годин.}$$
$$Т_{II} = (48 + 18 + 6.91 + 18) \cdot 8 = 727.28 \text{ людино-годин.}$$

Найбільш високу трудомісткість має варіант II.

В розробці беруть участь два програмісти з окладом 30000 грн., один аналітик в області даних з окладом 25000 грн. Визначимо середню зарплату за годину за формулою:

$$С_Ч = \frac{М}{T_m \cdot t} \text{ грн.,} \qquad (4.14)$$

де М – місячний оклад працівників;

T_m – кількість робочих днів тиждень;

t – кількість робочих годин в день.

$$С_Ч = \frac{30000 + 30000 + 25000}{3 \cdot 21 \cdot 8} = 168{,}65 \text{ грн.} \quad (4.15)$$

Тоді, розрахуємо заробітну плату за формулою:

$$С_{ЗП} = С_ч \cdot Т_i \cdot К_Д, \quad (4.16)$$

де $С_ч$– величина погодинної оплати праці програміста;

$Т_i$ – трудомісткість відповідного завдання;

$К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $С_{ЗП} = 168{,}65 \cdot 710.04 \cdot 1.2 = 143697{,}89$ грн.

II. $С_{ЗП} = 168{,}65 \cdot 727.28 \cdot 1.2 = 147186{,}92$ грн.

Відрахування на єдиний соціальний внесок становить 22%:

I. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 143697{,}89 \cdot 0.22 = 31613{,}53$ грн.

II. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 147186{,}92 \cdot 0.22 = 32381{,}12$ грн.

Тепер визначимо витрати на оплату однієї машино-години. ($С_М$)

Так як одна ЕОМ обслуговує одного програміста з окладом 30000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 30000 \cdot 0,2 = 72000$ грн.

З урахуванням додаткової заробітної плати:

$С_{ЗП} = С_Г \cdot (1 + К_З) = 72000 \cdot (1 + 0.2) = 86400$ грн.

Відрахування на соціальний внесок:

$С_{ВІД} = С_{ЗП} \cdot 0.22 = 86400 \cdot 0,22 = 15840$ грн.

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 15000 грн.

$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.4 \cdot 0.25 \cdot 15000 = 5250$ грн.,

де $К_{ТМ}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

$К_А$– річна норма амортизації;

$Ц_{ПР}$– договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.3 \cdot 15000 \cdot 0.08 = 1560$ грн.

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{ЕФ} = (Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 12 – 16) \cdot 8 \cdot 0.9 =$$

$$= 1677,6 \text{ годин},$$

де $Д_К$ – календарна кількість днів у році;

$Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів;

$Д_Р$ – кількість днів планових ремонтів устаткування;

t –кількість робочих годин в день;

$К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1677,6 \cdot 0,25 \cdot 0,4 \cdot 4.87 = 817 \text{ грн.},$$

де $N_С$ – середньо-споживча потужність приладу;

$К_З$– коефіцієнтом зайнятості приладу;

$Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$С_Н = Ц_{ПР} \cdot 0.67 = 15000 \cdot 0,67 = 10050 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н, \qquad (4.17)$$

$$С_{ЕКС} = 86400 + 15840 + 5250 + 1560 + 817 + 10050 = 119917 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$С_{М\text{-}Г} = С_{ЕКС} / Т_{ЕФ} = 119917 / 1677,6 = 71,48$ грн/год.

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$С_М = С_{М-Г} \cdot Т, \quad (4.18)$$

I. $С_М = 71.48 \cdot 710,04 = 50753,66$ грн.

II. $С_М = 71,48 \cdot 727,28 = 51985,97$ грн.

Накладні витрати складають 67% від заробітної плати:

$$С_Н = С_{ЗП} \cdot 0,67, \quad (4.19)$$

I. $С_Н = 50753,66 \cdot 0,67 = 34004,95$ грн.

II. $С_Н = 51985,97 \cdot 0,67 = 34830,6$ грн.

Отже, вартість розробки ПП за варіантами становить:

$$С_{ПП} = С_{ЗП} + С_{ВІД} + С_М + С_Н, \quad (4.20)$$

I. $С_{ПП} = 143697,89 + 31613,53 + 50753,66 + 34004,95 = 260070,03$ грн.

II. $С_{ПП} = 147186,92 + 32381,12 + 51985,97 + 34830,6 = 266384,61$ грн.

4.7 Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕР\,j} = К_{Кj} / С_{Фj}, \quad (4.21)$$

$К_1 = 23,55 / 260070,03 = 9,05 \cdot 10^{-5}$,

$К_2 = 17,7 / 266384,61 = 6.64 \cdot 10^{-5}$.

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_1 = 9,05 \cdot 10^{-5}$.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості К = $9,05 \cdot 10^{-5}$. Цей варіант реалізації програмного продукту має такі параметри:

- вибір мови програмування – Python;
- вибір середовища програмування – Google Colaboratory.

Даний варіант виконання програмного комплексу є оптимальним для заданої задачі.

Висновки до розділу 4

У даній частині було проведено повний функціонально-вартісний аналіз програмного продукту. Також було знайдено оцінку основних функцій програмного продукту.

В результаті виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, було визначено та проведено оцінку основних функцій програмного продукту, а також знайдено параметри, які його характеризують. На основі аналізу вибрано варіант реалізації програмного продукту.

ВИСНОВКИ

В роботі було проаналізовано сучасні підходи для побудови рекомендаційних систем. Вони є актуальними для людей, можуть приносити користь для бізнесу, а ринок продуктів, що використовують рекомендаційні системи, активно розвивається.

Будь-яка людина матиме свої унікальні вподобання, тому якщо ми хочемо створити якісну систему рекомендацій, наша задача – зрозуміти своїх користувачів, використовувати дані їх взаємодії з максимальною ефективністю. Це не єдиний спосіб покращити роботу системи: поведінка користувачів буде підпорядковуватися соціальним нормам та визначеним правилам, а також особливостям людського вибору. Аналізуючи цю інформацію, ми можемо знаходити слабкі місця в запропонованій нами архітектурі рекомендацій, постійно отримуючи зворотній зв’язок від користувачів явно чи неявно.

Два принципових підходи в рекомендаціях: персоніфікований підхід або підхід експертної системи, коли ми генеруємо лише схожі об’єкти, на ті з якими користувач взаємодіяв, припускаючи, що він точно знає, що хоче побачити далі, адже його інтереси повністю закодовані в його взаємодії з іншими об’єктами, та колаборативний підхід – ми орієнтуємося на поведінку групи людей, яка буде схожа на користувача, дотримуючись принципу: «схожа група – схожі інтереси». Аналізуючи ці підходи окремо, можемо зрозуміти, що вони матимуть ряд суттєвих недоліків, хоч і пропонують цікаві та корисні ідеї для рекомендацій. Найкращий варіант – поєднати сильні сторони обох методів та зменшити вплив негативних факторів; це визначення гібридного методу рекомендації, такі методи активно використовуються в сучасних задачах.

Класичні алгоритми створення рекомендацій були методами колаборативного підходу, або колаборативної фільтрації. Вони являли собою

розклад матриці взаємодії на менші матриці з метою прогнозування всіх пропущених значень взаємодій (користувач-об’єкт). Сучасне різноманіття вибору, зробило даний метод – матричну факторизацію, обчислювально складною, хоча вона і досі показує якісні результати на даних менших об’ємів. Цей метод є базовим, з нього потрібно починати створення рекомендацій, нові методи варто порівнювати з ним. Метод легко адаптується до нового підходу в рекомендаціях – переходу до ранжування та підтримує деякі модифікації алгоритму.

Представлення суб’єкта та об’єкта рекомендацій за допомогою нейронних мереж – сутність методу DSSM. По своїй суті, це схожий на матричну факторизацію алгоритм, але тут спосіб представлення ми можемо означати самостійно, додаючи характеристики користувачів, об’єктів, їх взаємодій. Вхідне представлення представляється в такому векторному просторі, де схожі елементи будуть поруч. Такий підхід є унікальним, адже він дозволяє комбінувати будь-які архітектури, а отже кодувати будь-які дані.

Для представлень послідовних даних найкраще підходять послідовні моделі. Основна задача для них – адаптувати такий вид нейронних мереж під задачу рекомендацій. Існує безліч вже реалізованих послідовних методів, які реалізують цікаві архітектури та враховують обмеження для послідовних нейронних мереж. Складні системи дають кращі результати, але займають більше часу на тренування.

В процесі виконання роботи було розглянуто дві вибірки даних – класичні дані для рекомендації фільмів та реальні дані, зібрані для групи людей в рамках одної географічної локації. Для класичної вибірки ми отримали кращі результати для всіх методів, враховуючи її кращу «інформативність» - середню кількість взаємодій на одного користувача. Це було проблемою у вибірці для локацій, враховуючи також несприятливу «природу» рекомендацій для об’єктів – локацій, оскільки вибір відгуків на них, часто зумовлений не самими

вподобаннями, а іншими випадковими факторами. Проте це приклад нестандартних даних, в яких можна знаходити цікаві особливості та їх кодувати: наприклад географічне представлення чи текстова інформація.

Реалізовані методи DSSM та GRU, порівняно з іншими методами, дали найгірші результати, але це є очікуваним та, на мою думку, некритичним результатом. У випадку DSSM – основною задачею була програмна реалізація самого методу, було розроблено таку архітектуру, де можна використовувати моделі для трансформації будь-якої введеної інформації. Якості рекомендацій можна досягнути, додавши додаткові характеристики та провівши експерименти з архітектурами внутрішніх моделей. Слабкі результати є причиною використання недостатньої для якісної рекомендації кількості параметрів для обох вибірок. У випадку GRU слабкі результати є очікуваними, оскільки існують вже схожі реалізовані архітектури, що видають кращі рекомендації, оскільки враховують певні особливості архітектур, передбачаючи програмне представлення компонентів такої архітектури у іншому вигляді. Основною задачею тут було порівняння якості з уже реалізованим методом та часу, необхідного на тренування.

Аналіз всіх інших методів показав, що складні архітектури послідовних моделей дають найкращі результати, проте для тренування даних архітектур необхідно чимало часу. Методи матричної факторизації видали хороші результати, але на значній вибірці даних не змогли працювати, окрім методу LightFM, який завжди працював трохи гірше, але завжди швидше, був гнучким у використанні навіть на великих вибірках.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Sapolsky, Robert, The Biology of Humans at Our Best and Worst, 2018. 800 p.
2. Silvia Milano, Mariarosaria Taddeo, Luciano Floridi, Recommender systems and their ethical challenges, 2020. 11 p.
3. Luca Bertuzzi, YouTube's algorithm fuelling harmful content, study says. URL: https://www.euractiv.com/section/disinformation/news/youtubes-algorithm-fuelling-harmful-content-study-says/
4. Jeffrey Dastin, Amazon scraps secret AI recruiting tool that showed bias against women. URL: https://www.reuters.com/article/us-amazon-com-jobs-automation-insight-idUSKCN1MK08G
5. Recommendation Engine Market Size, Share & Trends Analysis Report By Type. URL: https://www.grandviewresearch.com/industry-analysis/recommendation-engine-market-report
6. Вержбицкий, В.М., *Вычислительная линейная алгебра: Учеб. Пособие для вузов, 2009. 351 с.*
7. Gábor Takács, Domonkos Tikk, Alternating Least Squares for Personalized Ranking, 2012. 10 p.
8. Alternating Least Squares With Weighted Regularization. URL: http://ethen8181.github.io/machine-learning/recsys/1_ALSWR.html
9. Xiangnan He, Hanwang Zhang, Min-Yen Kan, Tat-Seng Chua, Fast Matrix Factorization for Online Recommendation with Implicit Feedback, 2017. 10 p.
10. Maciej Kula, Metadata Embeddings for User and Item Cold-start Recommendations, 2015. 8 p.
11. Aayush Agrawal, Solving business usecases by recommender system using LightFM. URL: https://towardsdatascience.com/solving-business-usecases-by-recommender-system-using-lightfm-4ba7b3ac8e62

12. Tie-Yan Liu, Learning to Rank for Information Retrieval, 2009. 109 p.
13. Florian Schroff, Dmitry Kalenichenko, James Philbin, FaceNet: A Unified Embedding for Face Recognition and Clustering, 2015. 10 p.
14. Jason Weston, Samy Bengio, Nicolas Usunier, WSABIE: Scaling Up To Large Vocabulary Image Annotation, 2011. 7 p.
15. MTC. Your first recsys. URL: https://ods.ai/tracks/mts-recsys-df2020
16. Your Second RecSys. URL: https://ods.ai/tracks/recsys-course2021
17. Xiangnan He, Lizi Liao, Hanwang Zhang, Neural Collaborative Filtering, 2017. 10 p.
18. Chieh-Yuan Tsai, Yi-Fan Chiu, Yu-Jen Chen, A Two-Stage Neural Network-Based Cold Start Item, 2021. 18 p.
19. Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, Larry Heck, Learning Deep Structured Semantic Models for Web Search using Clickthrough Data, 2013. 8 p.
20. Item2Vec: Neural Item Embeddings to enhance recommendations. URL: https://tech.olx.com/item2vec-neural-item-embeddings-to-enhance-recommendations-1fd948a6f293
21. Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, Hemal Shah, Wide & Deep Learning for Recommender Systems, 2016. 4 p.
22. Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean, Efficient Estimation of Word Representations in Vector Space, 2013. 12 p.
23. Tomas Mikolov, Ilya Sutskever, Kai, Greg Corrado, Jeffrey Dean, Distributed Representations of Words and Phrases and their Compositionality, 2013. 9 p.

24. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin, Attention Is All You Need, 2017. 15 p.
25. Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, Yoshua Bengio, Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, Yoshua Bengio, Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling, 2014. 9 p.
26. Sepp Hochreiter, Jürgen Schmidhuber, Long Short-term Memory, 1997. 33 p.
27. Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, Domonkos Tikk, Session-based Recommendations with Recurrent Neural Networks, 2015. 10 p.
28. Jiaxi Tang, Ke Wang, Personalized Top-N Sequential Recommendation via Convolutional Sequence Embedding, 2018. 9 p.
29. Xin Jin, Jiawei Han, K-Means Clustering. URL: https://link.springer.com/referenceworkentry/10.1007/978-0-387-30164-8_425
30. Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise, 1996. 6 p.
31. Jerome H. Friedman, Greedy Function Approximation: A Gradient Boosting Machine, 2001. 44 p.
32. Robin M. Schmidt, Recurrent Neural Networks (RNNs): A gentle Introduction and Overview - Robin M. Schmidt, 2019. 16 p.
33. Keiron O'Shea, Ryan Nash, An Introduction to Convolutional Neural Networks, 2015. 11 p.
34. Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, Deep Residual Learning for Image Recognition, 2015. 12 p.
35. Netflix Prize. URL: https://en.wikipedia.org/wiki/Netflix_Prize
36. One-hot Encoding. URL: https://en.wikipedia.org/wiki/One-hot
37. MovieLens 1M Dataset. URL: https://grouplens.org/datasets/movielens/1m/
38. Google Local Data. URL: https://jiachengli1995.github.io/google/index.html

39. GRU4Rec — RecBole 1.1.1 documentation. URL: https://recbole.io/docs/user_guide/model/sequential/gru4rec.html
40. Caser — RecBole 1.1.1 documentation. URL: https://recbole.io/docs/user_guide/model/sequential/caser.html

Додаток А. Лістинг програмного модулю

```
 import pandas as pd
 import numpy as np
 from datetime import datetime
 import math
 from tqdm import tqdm

 # from sklearn.cluster import DBSCAN, KMeans
 from sklearn.preprocessing import StandardScaler
 from sklearn.preprocessing import LabelEncoder
 from sklearn.metrics.pairwise import euclidean_distances as ED

 import tensorflow as tf
 import tensorflow.keras.backend as K


 from tensorflow.keras.preprocessing.text import one_hot
 from tensorflow.keras.utils import pad_sequences

      interactions = pd.read_csv('/content/drive/MyDrive/Data/ratings.dat',
sep='::', header=None)
      items = pd.read_csv('/content/drive/MyDrive/Data/movies.dat', sep='::',
encoding='latin-1', header=None)
      users = pd.read_csv('/content/drive/MyDrive/Data/users.dat', sep='::',
header=None)

      # interactions_backup = interactions.copy()
      # items_backup = items.copy()

      interactions = interactions_backup.copy()
      items = items_backup.copy()

      interactions = interactions.rename({0: 'gmap_id', 1:'user_id',
2:'rating', 3:'time'}, axis=1)
      items = items.rename({0: 'gmap_id', 1:'title', 2:'genre'}, axis=1)
      users = users.rename({0:'user_id', 1:'gender', 2:'age', 3:'role',
4:'time'}, axis=1)

      """## Preprocessing

      ### Interactions Dataset Preprocessing
      """

      interactions = interactions[~interactions.user_id.isnull()]

      interactions = interactions[['user_id', 'gmap_id', 'rating', 'time']]

      interactions = interactions.drop_duplicates()

      print(interactions.shape)
```

```
      #
sns.histplot(interactions.groupby('user_id').count()['gmap_id'].values)
      # plt.xlim(0, 20)

      users_grouped = interactions.groupby('user_id').count()['gmap_id']
      items_grouped = interactions.groupby('gmap_id').count()['user_id']

      # Setting threshold for Users & Items

      users_threshold, items_threshold = 10, 5

      users_above_threshold = users_grouped[users_grouped >=
users_threshold].index
      items_above_threshold = items_grouped[items_grouped >=
items_threshold].index

      print(len(items_above_threshold))
      print(len(users_above_threshold))

      print(interactions.shape)

      interactions =
interactions[interactions.user_id.isin(users_above_threshold)]
      interactions =
interactions[interactions.gmap_id.isin(items_above_threshold)]

      print(interactions.shape)

      items_to_keep = set(interactions.gmap_id.unique())
      users_to_keep = set(interactions.user_id.unique())

      # обработка даты

      interactions['time'] = interactions['time'].apply(lambda x:
datetime.fromtimestamp(x))
      interactions['time'] = pd.to_datetime(interactions['time'])

      """### Item Dataset Preprocessing"""

      items.head(1)

      print(items.shape)
      items = items[items.gmap_id.isin(items_to_keep)]
      print(items.shape)

      # Duplicates drop

      print("Items dataset shape", items.shape)

      items_duplicates_index = items[items.gmap_id.duplicated()].index
      items = items.drop(items_duplicates_index)

      print("Items dataset shape after deleting duplicates", items.shape)
```

```
      items['genre'] = items.genre.str.split('|')
      all_genres =
pd.get_dummies(items['genre'].explode()).groupby(level=0).sum().columns

      pd.get_dummies(items['genre'].explode()).groupby(level=0).sum().head(1)

      items['year'] = items['title'].apply(lambda x: x[-5:-1])
      items['year'] = pd.to_numeric(items['year'])

      items['year bin'] = pd.cut(items['year'], bins=np.arange(1918, 2019,
10))

      link = pd.read_csv('/content/drive/MyDrive/Data/links.csv')
      link = link.rename({'movieId':'gmap_id'}, axis=1)

      items = items.merge(link, how='left')

      items = items[~items.tmdbId.isnull()]

      items = items[~items['title'].str.startswith('Criminals')]

      import requests

      def get_info(movie_id):

      url = "https://api.themoviedb.org/3/movie/"+str(movie_id)

      headers = {
      "accept": "application/json",
      "Authorization": "Bearer
eyJhbGciOiJIUzI1NiJ9.eyJhdWQiOiIzOWM3MjU1ODNmZTEwZmM5YjUyMjYzMTk3N2FlMTQ5NCIsI
nN1YiI6IjY0N2Q5MGRkMGUyOWEyMmJkZmVjNjhhNyIsInNjb3BlcyI6WyJhcGlfcmVhZCJdLCJ2ZXJ
zaW9uIjoxfQ.D-r6U0O1lYwCQDCbUQnScm4c4qF7TLz8ZuulIK4vETQ"
      }
      responce = requests.get(url, headers=headers).json()

      return True if responce['belongs_to_collection'] else False,
responce['budget'], responce['original_language'], responce['popularity'],
responce['production_companies'][0]['origin_country'] if
responce['production_companies']!=[] else 'N/A', responce['revenue'],
responce['runtime']

      tqdm.pandas()
      a, b, c, d, e, f, g = zip(*items['tmdbId'].progress_apply(get_info))

      items['belongs_to_collection'] = a
      items['budget'] = b
      items['language'] = c
      items['popularity'] = d
      items['country'] = e
      items['revenue'] = f
      items['runtime'] = g

      items['belongs_to_collection'] =
items['belongs_to_collection'].map({True:1, False:0})
```

```
      cat_feats_new = pd.get_dummies(items[['belongs_to_collection',
'language', 'country']])
      cat_feats_new

      items['budget'] = pd.to_numeric(items['budget'])
      items['popularity'] = pd.to_numeric(items['popularity'])
      items['revenue'] = pd.to_numeric(items['revenue'])
      items['runtime'] = pd.to_numeric(items['runtime'])

      items['budget bins'] = pd.cut(items['budget'], bins=np.arange(-1,
220000000, 20000000.0))
      items['popularity bins'] = pd.qcut(items['popularity'], 10)
      items['revenue bins'] = pd.cut(items['revenue'], bins=np.arange(-1,
2464162353, 226416235.3))
      items['runtime bins'] = pd.qcut(items['runtime'], 10)

      items = items[~items['revenue bins'].isnull()]

      items = pd.get_dummies(items[['gmap_id', 'year bin',
'belongs_to_collection', 'language', 'country', 'popularity bins', 'budget
bins', 'revenue bins', 'runtime bins']])
      items.head(1)

      items = items.reset_index(drop=True)
      items = items.reset_index()
      items = items.rename({'index': 'iid'}, axis=1)

      itemid_to_iid = {k:v for k, v in items[['gmap_id', 'iid']].values}
      iid_to_itemid = {v: k for k, v in itemid_to_iid.items()}

      items.head(1)

      """### Post Interactions & Users Dataset Preprocessing"""

      users_id = pd.DataFrame(interactions['user_id'].unique())
      users_id = users_id.reset_index()
      users_id = users_id.rename({'index': 'uid', 0:'user_id'}, axis=1)

      userid_to_uid = {k:v for k, v in users_id[['user_id', 'uid']].values}
      uid_to_userid = {v: k for k, v in userid_to_uid.items()}

      interactions =
interactions[interactions.gmap_id.isin(items.gmap_id.unique())]

      interactions['iid'] = interactions['gmap_id'].apply(lambda x:
itemid_to_iid[x])
      interactions['uid'] = interactions['user_id'].apply(lambda x:
userid_to_uid[x])

      """User dataset part start here"""

      users = users[users['user_id'].isin(users_to_keep)]

      users['uid'] = users['user_id'].apply(lambda x: userid_to_uid[x])
```

```
      users['age bin'] = pd.cut(users['age'], bins=np.arange(0, 70, 10,))

      users_ = pd.concat([pd.get_dummies(users[['uid', 'gender', 'age bin']]),
      pd.get_dummies(users['role'].explode(),
prefix='role').groupby(level=0).sum()
      ], axis=1)

      users_.columns

      users

      users = users_.copy()
      del users_

      """## Data Split"""

      date_threshold = interactions['time'].quantile(q=0.9,
interpolation='nearest')
      date_threshold

      train = interactions[(interactions['time'] < date_threshold)]
      pred = interactions[(interactions['time'] >= date_threshold)]

      print(f"train: {train.shape}")
      print(f"pred: {pred.shape}")

      # Оставляем только наявных в тренировочных данных пользователей
      pred = pred[pred['user_id'].isin(train['user_id'].unique())]

      print(f"pred: {pred.shape}")

      """## Model

      Interaction Matrix Build
      """

      matrix = pd.pivot_table(interactions, index='uid', columns='iid',
values='rating')

      matrix = matrix.values

      matrix = np.nan_to_num(matrix, 0)

      matrix.shape[1]

      def triplet_loss(y_true, y_pred, n_dims=50, alpha=0.4):


        anchor = y_pred[:, 0:n_dims]
        positive = y_pred[:, n_dims:n_dims*2]
        negative = y_pred[:, n_dims*2:n_dims*3]

        pos_dist = K.sum(K.square(anchor - positive), axis=1)
        neg_dist = K.sum(K.square(anchor - negative), axis=1)
```

```
        basic_loss = pos_dist - neg_dist + alpha
        loss = K.maximum(basic_loss, 0.0)

        return loss

    def generator(users, items, interactions, all_users, all_items,
    batch_size=1024):

        while True:
        anchor = []
        pos = []
        neg = []

        for _ in range(batch_size):

        uid_i = np.random.choice(all_users)

        uid_i_df = interactions[interactions.uid == uid_i]
        uid_i_df = uid_i_df.reset_index()

        pos_i = np.random.choice(uid_i_df.iid.values,
        p=uid_i_df.rating.values / uid_i_df.rating.values.sum())
        neg_i = np.random.choice(all_items)

        anchor.append(users.iloc[int(uid_i), 1:].values)
        pos.append(items.iloc[int(pos_i), 1:].values)
        neg.append(items.iloc[int(neg_i), 1:].values)

        yield [np.array(anchor), np.array(pos), np.array(neg)],
        [np.array(['x']*np.array(anchor).shape[0]),
np.array(['x']*np.array(anchor).shape[0])]

    temp = train.groupby('uid').count()
    uid_with_more_than_one_interaction = temp[temp.rating > 1].index


    def user_model():

        inp_meta = tf.keras.layers.Input(shape=29)

        meta_1 = tf.keras.layers.Dense(50, activation='elu')(inp_meta)
        meta_2 = tf.keras.layers.Dense(50, activation='elu')(meta_1)
        add_meta = tf.keras.layers.Add()([meta_1, meta_2])

        return tf.keras.models.Model(inp_meta, add_meta)

    def item_model():

        inp = tf.keras.layers.Input(shape=132)

        dense1_cat = tf.keras.layers.Dense(150, activation='elu')(inp)
        dense2_cat = tf.keras.layers.Dense(50, activation='elu')(dense1_cat)

        return tf.keras.models.Model(inp, dense2_cat)
```

```
        u2v = user_model()
        i2v = item_model()

        anchor_in = tf.keras.layers.Input(shape=29)
        pos_in = tf.keras.layers.Input(shape=132)
        neg_in = tf.keras.layers.Input(shape=132)

        anchor = u2v(anchor_in)
        pos = i2v(pos_in)
        neg = i2v(neg_in)

        res = tf.keras.layers.Concatenate(name="concat_ancor_pos_neg")([anchor,
pos, neg])

        model = tf.keras.models.Model([anchor_in, pos_in, neg_in], res)

        opt = tf.keras.optimizers.Adagrad(learning_rate=0.05)
        model.compile(loss=triplet_loss, optimizer=opt)

        # user_model().summary()

        # item_model().summary()

        model.summary()

        items.drop(['gmap_id'], axis=1).head(1)

        len(users.columns)

        next(generator(items=items.drop(['gmap_id'], axis=1),
        users=users,
        interactions=train,
        interactions_matrix=matrix,
        all_items=train.iid.unique(),
        all_users=train.uid.unique()
        ))

        log = model.fit(generator(items=items.drop(['gmap_id'], axis=1),
        users=users,
        interactions=train,
        interactions_matrix=matrix,
        all_items=train.iid.unique(),
        all_users=train.uid.unique()
        ),
        batch_size=256,
        steps_per_epoch=100,
        epochs=10,
        initial_epoch=0,
        )
        log


        def generate_dssm_recs_mapper(u2v, users, matrix, known_items,
items_vec, N, num_threads=4):
```

```
      def _recs_mapper(uid):

       print(uid)

       uid = int(uid)

       user_meta_feats = users.iloc[uid, 1:].values

       user_vec = u2v.predict(np.array(user_meta_feats).reshape(1, -1))

       dists = ED(user_vec, items_vec)

       recs = dists.reshape(-1, )


       additional_N = len(known_items[uid]) if uid in known_items else 0
       total_N = N + additional_N
       top_cols = np.argpartition(recs, -np.arange(total_N))[-total_N:][::-1]

       final_recs = [item for item in top_cols]
       if additional_N > 0:
       filter_items = known_items[uid]
       final_recs = [item for item in final_recs if item not in filter_items]
       return final_recs[:N]
       return _recs_mapper

      from tqdm import tqdm

      tqdm.pandas()

      top_N = 10

      user2item_prediction = pd.DataFrame({
      'uid': pred['uid'].unique()
      })

      known_items = train.groupby('uid')['iid'].apply(list).to_dict()
      known_items_ = {k:[] for k, v in known_items.items()}

      dssm_mapper = generate_dssm_recs_mapper(
      u2v,
      users=users,
      matrix=matrix,
      known_items=known_items_,
      N=top_N,
      items_vec=i2v.predict(items.drop(['gmap_id'], axis=1).iloc[:,
1:].to_numpy()),
      num_threads=20
      )

      user2item_prediction['iid'] =
user2item_prediction['uid'].progress_map(dssm_mapper)
      user2item_prediction =
user2item_prediction.explode('iid').reset_index(drop=True)
```

```
      user2item_prediction['rank'] =
user2item_prediction.groupby('uid').cumcount() + 1

      def compute_metrics(df_true, df_pred, top_N):
        result = {}
        test_recs = df_true.set_index(['uid',
 'iid']).join(df_pred.set_index(['uid', 'iid']))
        test_recs = test_recs.sort_values(by=['uid', 'rank'])

        test_recs['users_item_count'] =
 test_recs.groupby(level='uid')['rank'].transform(np.size)
        test_recs['reciprocal_rank'] = (1 / test_recs['rank']).fillna(0)
        test_recs['cumulative_rank'] =
 test_recs.groupby(level='uid').cumcount() + 1
        test_recs['cumulative_rank'] = test_recs['cumulative_rank'] /
 test_recs['rank']

        users_count = test_recs.index.get_level_values('uid').nunique()
        for k in range(1, top_N + 1):
        hit_k = f'hit@{k}'
        test_recs[hit_k] = test_recs['rank'] <= k
        result[f'Precision@{k}'] = (test_recs[hit_k] / k).sum() / users_count
        result[f'Recall@{k}'] = (test_recs[hit_k] /
 test_recs['users_item_count']).sum() / users_count

        result[f'MAP@{top_N}'] = (test_recs["cumulative_rank"] /
 test_recs["users_item_count"]).sum() / users_count
        result[f'MRR'] =
 test_recs.groupby(level='uid')['reciprocal_rank'].max().mean()
        return pd.Series(result)

      compute_metrics(train[['uid', 'iid']],
      user2item_prediction,
top_N=10)




      temp = train.groupby('user_id').count()
      temp_idx = temp[temp.gmap_id > 10].index.unique()

      users_with_more_than_N_interactions_df                                =
train[train.user_id.isin(temp_idx)]
      users_with_more_than_N_interactions_df


      def f(lst, N=10):

        # dummy function for fetching all possible sequences & items

        lst1 = []
        lst2 = []

        for _ in range(0, len(lst)-N):
```

```
          seq = lst[_:_+N]
          item = lst[_+N]

          lst1.append(seq)
          lst2.append(item)

        return lst1, lst2

      lst_seq, lst_item = [], []
      for                                user_id                                in
tqdm(users_with_more_than_N_interactions_df.user_id.unique()):

        user_df                                                                 =
users_with_more_than_N_interactions_df[users_with_more_than_N_interactions_df.
user_id == user_id]
        arr_seq, arr_item = f(user_df['gmap_id'].values)

        lst_seq += arr_seq
        lst_item += arr_item

      df = pd.DataFrame([lst_seq, lst_item]).T
      df = df.rename({0:'sequence', 1:'item'}, axis=1)
      df

      train_df = tf.data.Dataset.from_tensor_slices(df.to_dict(orient="list"))

      s2v = tf.keras.Sequential([
                # tf.keras.layers.Input(shape=(10)),
                tf.keras.layers.StringLookup(vocabulary=unique_item_ids,
mask_token=None),
                tf.keras.layers.Embedding(len(unique_item_ids)       +       1,
embedding_dimension),
                tf.keras.layers.GRU(embedding_dimension),
                tf.keras.layers.Dense(32, activation="linear")])

      i2v = tf.keras.Sequential([
                # tf.keras.layers.Input(shape=1),
                tf.keras.layers.StringLookup(vocabulary=unique_item_ids,
mask_token=None),
                tf.keras.layers.Embedding(len(unique_item_ids)       +       1,
embedding_dimension),
                tf.keras.layers.Dense(32, activation="linear")
                ])

      gmap_id_tf_df                                                             =
tf.data.Dataset.from_tensor_slices(dict(items[['gmap_id']]))
      gmap_id_tf_df = gmap_id_tf_df.map(lambda x: x["gmap_id"])

      metrics = tfrs.metrics.FactorizedTopK(
        candidates=gmap_id_tf_df.batch(50).map(i2v)
      )

      task = tfrs.tasks.Retrieval(
        metrics=metrics
      )
```

```
      class GmapRecommenderModel(tfrs.Model):

          def __init__(self, user_model, movie_model):
            super().__init__()
            self.sequence_model = s2v
            self.item_model = i2v

            self.task: tf.keras.layers.Layer = task

          def compute_loss(self, features, training=True):

            watch_history = features['sequence']
            watch_next_label = features['item']

            history_embedding = self.sequence_model(watch_history)
            label_embedding = self.item_model(watch_next_label)

            return        self.task(history_embedding,        label_embedding,
compute_metrics=True)

      model = GmapRecommenderModel(s2v, i2v)
      model.compile(optimizer=tf.keras.optimizers.Adagrad(learning_rate=0.025)
)

      BATCH_SIZE = 12800

      def scheduler(epoch, lr):
        if epoch > 5:
          return lr
        else:
          return lr * tf.math.exp(-0.1)

      # Callbacks
      exp_increase = tf.keras.callbacks.LearningRateScheduler(scheduler)
      decay = tf.keras.callbacks.ReduceLROnPlateau(monitor='loss', patience=2,
factor=0.8, verbose=1)

      cached_train = train_df.batch(BATCH_SIZE).cache()

      model.fit(cached_train, epochs=10,
                callbacks=[decay, exp_decay]
                )

      """### Predict"""

      users_for_predict = set(pred.user_id) & set(temp[temp.time>10].index)

      def generate_gru_recs_mapper(s2v, i2v, known_items, N, unique_item_ids,
items_enc, num_threads=4):
          def _recs_mapper(user):

              # получаем 10 последних из трейна
              seq = train[train.user_id== user].gmap_id.values[-10:]
```

```
                # кодируем последовательность
                seq_enc = s2v.predict(seq.reshape(-1, 10))

                # расчитываем "близость" и выбираем индексы N ближайших
                dists = ED(seq_enc, items_enc)

                # делаем reshape
                recs = dists.reshape(-1, )

                additional_N = len(known_items[user]) if user in known_items
else 0
                total_N = N + additional_N
                top_cols    =    np.argpartition(recs,    -np.arange(total_N))[-
total_N:][::-1]

                final_recs = [item for item in unique_item_ids[top_cols]]
                if additional_N > 0:
                    filter_items = known_items[user]
                    final_recs = [item for item in final_recs if item not in
filter_items]
                return final_recs[:N]
            return _recs_mapper

        all_cols = unique_item_ids


        tqdm.pandas()

        top_N = 10

        gru_prediction = pd.DataFrame({
            'user_id': list(users_for_predict)
        })

        known_items = train.groupby('user_id')['gmap_id'].apply(list).to_dict()

        mapper = generate_gru_recs_mapper(
            s2v,
            i2v,
            known_items=known_items,
            unique_item_ids=unique_item_ids,
            N=top_N,
            items_enc=i2v.predict(unique_item_ids.reshape(-1, 1)),
            num_threads=20
        )

        gru_prediction['gmap_id']                                              =
gru_prediction['user_id'].progress_map(mapper)
        gru_prediction                                                         =
gru_prediction.explode('gmap_id').reset_index(drop=True)
        gru_prediction['rank'] = gru_prediction.groupby('user_id').cumcount() +
1

        def compute_metrics(df_true, df_pred, top_N):
            result = {}
```

```
      test_recs = df_true.set_index(['user_id',
'gmap_id']).join(df_pred.set_index(['user_id', 'gmap_id']))
      test_recs = test_recs.sort_values(by=['user_id', 'rank'])

      test_recs['users_item_count'] =
test_recs.groupby(level='user_id')['rank'].transform(np.size)
      test_recs['reciprocal_rank'] = (1 / test_recs['rank']).fillna(0)
      test_recs['cumulative_rank'] =
test_recs.groupby(level='user_id').cumcount() + 1
      test_recs['cumulative_rank'] = test_recs['cumulative_rank'] /
test_recs['rank']

      users_count = test_recs.index.get_level_values('user_id').nunique()
      for k in range(1, top_N + 1):
      hit_k = f'hit@{k}'
      test_recs[hit_k] = test_recs['rank'] <= k
      result[f'Precision@{k}'] = (test_recs[hit_k] / k).sum() / users_count
      result[f'Recall@{k}'] = (test_recs[hit_k] /
test_recs['users_item_count']).sum() / users_count

      result[f'MAP@{top_N}'] = (test_recs["cumulative_rank"] /
test_recs["users_item_count"]).sum() / users_count
      result[f'MRR'] =
test_recs.groupby(level='user_id')['reciprocal_rank'].max().mean()
      return pd.Series(result)

      # from tools import compute_metrics

      gru_metrics = compute_metrics(pred[['user_id', 'gmap_id']],
      gru_prediction,
top_N=10)
```

Додаток Б. Презентація

Система надання рекомендацій методами нейронних мереж

Виконав:

Панаско Віталій Євгенович

Керівник: професор, докт. техн. наук

Недашківська Надія Іванівна

Поведінкові особливості людини

- Маємо вподобання, певний оптимальний вибір у декількох заданих категоріях
- Мислимо, як соціальна група, до якої належимо, маємо схожі інтереси з нею
- Прагнемо різноманітності у виборі, у випадку невизначеності з оптимальним вибором
- З часом змінюємо свої інтереси

Які проблема намагається вирішити людина?

- Необхідність тримати чимало варіантів в голові
- Значний час на пошук оптимального варіанту
- Інколи сама не знає, що хоче знайти
- Бажання пробувати нове

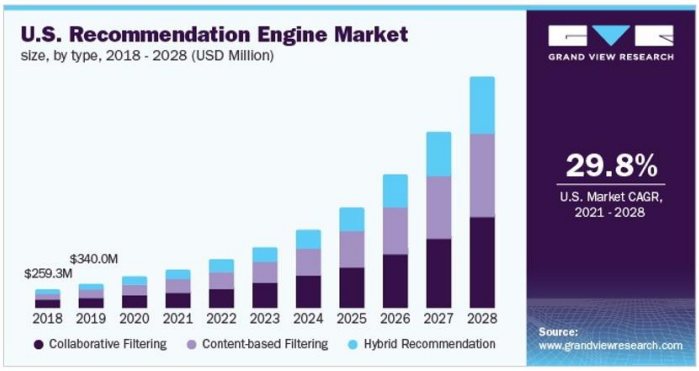
Як бачимо, питання рекомендаційних систем для людини є **актуальним.**

Актуальність для бізнесу

- Ринок продуктів, що використовують рекомендаційні системи, активно розвивається, демонструючи експоненційний ріст
- Можливість детальніше аналізувати соціальні групи, що користуються продуктом через рекомендації для них
- Можливість ефективніше налаштовувати рекламу на веб-сервісах

Після початку пандемії COVID-19 частка покупок через електронні сервіси зросла на **50%** та продовжує рости. Актуальність для бізнесу є **дуже високою**.

Експоненційний ріст ринку продуктів-рекомендаційних систем



Постановка задачі

Задача полягає у:

- прогнозуванні незаповнених клітинок r_{ui} в матриці взаємодії
- формуванні списку рекомендацій для u та для i, використовуючи міру близькості p

U – множина користувачів, або інших суб'єктів рекомендації

I – множина об'єктів, які будемо рекомендувати

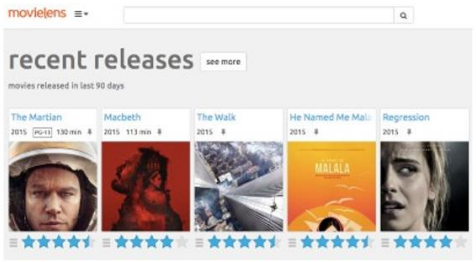
$R = \|r_{ui}\|$ – матриця взаємодії розміру $|U| \times |I|$

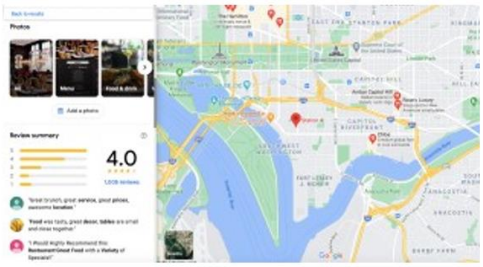
	Фільм 1	Фільм 2	Фільм 3
Користувач 1	4	?	?
Користувач 2	4	5	?
Користувач 3	?	5	?
Користувач 4	5	4	?

Дані

Маємо дві вибірки:

- Класична для задач рекомендації **MovieLens**, представляє собою відгуки користувачів на фільми обсягом 1 мільйон взаємодій
- Вибірка з відгуками користувачів на локації **Google Maps** обсягом 1 мільйон взаємодій





MovieLens

	Взаємодія	Користувачі	Об'єкти
Кількість	633945	3260	3829

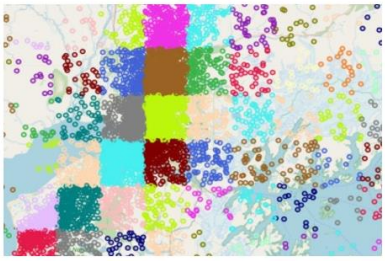
- Висока середня кількість взаємодій на одного користувача та на один об'єкт
- В оригінальній вибірці містять таблицю з взаємодіями та характеристики користувачів і об'єктів
- Всі змінні кодуються як категоріальні

	iid	belongs_to_collection	year bin_(1918, 1928]	year bin_(1928, 1938]	year bin_(1938, 1948]	year bin_(1948, 1958]	year bin_(1958, 1968]	year bin_(1968, 1978]	year bin_(1978, 1988]	year bin_(1988, 1998]	...
0	0	1	0	0	0	0	0	0	0	1	...

Google Review Data

	Взаємодія	Користувачі	Об'єкти
Кількість	544551	20748	12100

- Для кодування географічних даних всі локації спочатку було розбито на квадрати, крім цього були проведені експерименти з кластеризацією методом DBSCAN
- Приналежність до категорії було закодовано методом one-hot encoding



Регресійні та класифікаційні метрики якості

- Регресійні:

$$RMSE^2 = \frac{1}{|obs|} \sum_{(u,i) \in obs} (r_{ui} - \hat{r}_{ui})^2$$

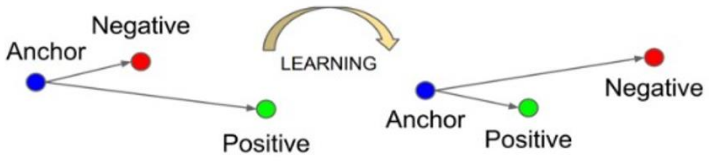
- Класифікаційні:

$$Precision@K = \frac{number\ of\ recommendations\ that\ are\ relevant}{K}$$

$$Recall@K = \frac{number\ of\ recommendations\ that\ are\ relevant}{number\ of\ all\ the\ possible\ relevant\ items}$$

Перехід до ранжування

Основна задача полягає не в точності наближення оцінок, а в приведенні векторного простору представлень користувача чи об'єкта таким чином, щоб по відношенню по головного елемента позитивні елементи знаходилися ближче, а негативні – якомога далі.



Ранжувальна метрика MAP@K

$$MAP@K = \frac{1}{number\ of\ users} \sum_{i=1}^{number\ of\ users} AP@K$$

$$AP@K = \frac{1}{N(K)} \sum_{i=1}^{K} \frac{TP_{seen}(i)}{i}$$

$$N(k) = \min(k, TP_{total})$$

$$TP_{seen} = \begin{cases} 0: & i^{th} = True \\ True\ Positives\ seen\ till\ i\ position: & i^{th} = False \end{cases}$$

Ранжувальна метрика MRR

$$MRR = \frac{1}{Q}\sum_{q=1}^{Q}\frac{1}{rank_q}$$

$rank_q$ - ранг першого елемента, який був релевантний в рекомендації q;

Q – кількість всіх рекомендацій;

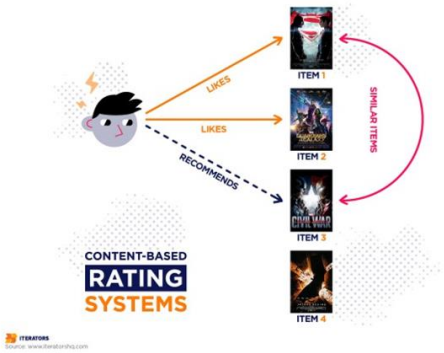
Основні підходи при побудові рекомендаційних систем

- Контентна фільтрація
- Колаборативна фільтрація
- Гібридні моделі рекомендацій (об'єднання двох попередніх методів)

Задача гібридних методів рекомендацій – об'єднати сильні сторони методів, при цьому мінімізувавши їх недоліки.

Контентна фільтрація або експертні системи

- Не враховуємо зв'язок з групою користувачів
- Будується певне середнє представлення про вподобання
- Робить припущення, що користувач гарантованно знатиме, що йому подобається



Недоліки методів контентної фільтрації

- «Холодний» старт – нові користувачі, нові характеристики об'єктів, незрозуміло як їх додавати в модель
- Нехтування взаємодіями інших користувачів – особливість підходу
- Одноманітні рекомендації, крім випадку, коли це саме те, що нам необхідно
- Неврахування часової залежності

Колаборативна фільтрація

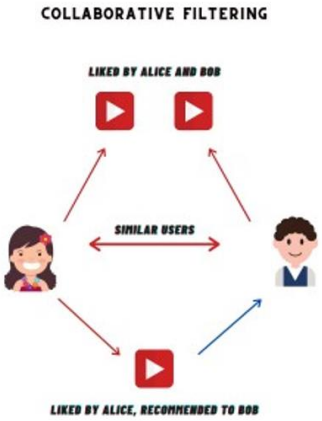
- Враховує зв'язки між користувачами
- Не запам'ятовує інформацію

про вподобання

- Вважає, що дані про персональні

вподобання можна виявити в поведінці

інших користувачів



Різні підходи в колаборативній фільтрації

Основна сутність – матриця взаємодій. Питання як саме ми будемо з нею працювати:

- Прогнозувати всі клітинки матриці – ML-based підхід
- Визначати «схожих» користувачів та об'єкти по метриці близькості та аналізувати тільки їх взаємодію – Memory-based підхід

$$S(u,v) = \frac{|I(u) \cap I(v)|}{|I(u) \cup I(v)|}$$

Недоліки методів колаборативної фільтрації

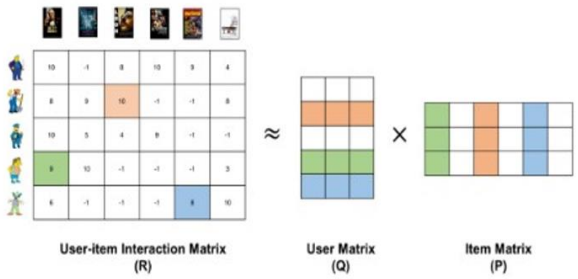
- Відсутність можливості давати персоналізовані рекомендації
- Що робити з новим користувачем, який не схожий ні на одну групу по своїй взаємодії?
- Часова залежність також не враховується

Матрична факторизація

Найвідоміший ML-based підхід в колаборативній фільтрації. Вважається класичним для задачі рекомендацій. Свою значимість підтвердив, внаслідок Netflix recommendation competition (2006-2009).

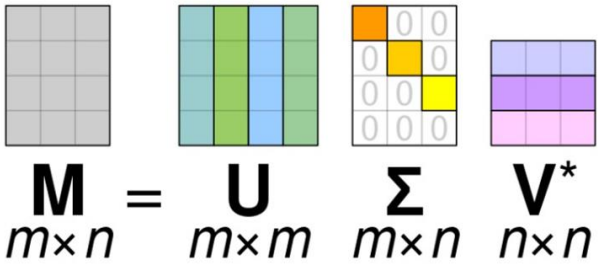
До основних методів належать:

- SVD
- ALS
- Модифікації



Сутність методу SVD

Розклад матриці на добуток трьох; щоб отримати наближення, враховуємо лише N найбільших сингулярних чисел матриці Σ



Сутність методу SVD

- Нехай $A = U\Sigma V^T$, а розміри матриць:

$$A: n \times m;\ U: n \times n;\ \Sigma: n \times m;\ V: m \times m$$

- Матриця Σ:

$$\Sigma = diag(\lambda_1, \lambda_2, \dots, \lambda_{min\,(m,n)})$$
$$\lambda_1 \geq \lambda_2 \geq \cdots \lambda_{min\,(m,n)} \geq 0$$

- Візьмемо d найбільших сингулярних чисел:

$$A \approx A' = U'\Sigma'V'^T$$

  - $A': n \times m; U': n \times d;\ \Sigma': d \times d;\ V': d \times m$

Результати, отримані методом SVD для вибірок MovieLens та Google Review Data

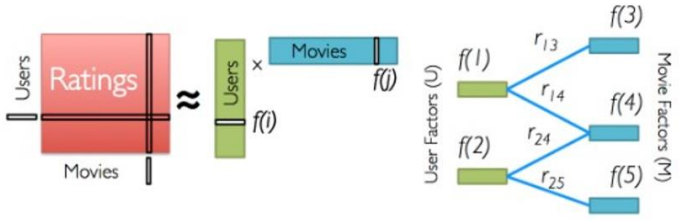
	Precision@k			Recall@k		
	кількість врахованих найбільших сингулярних чисел			кількість врахованих найбільших врахованих сингулярних чисел		
k	25	50	100	25	50	100
1	0.066	0.083	0.080	0.002	0.003	0.003
2	0.067	0.075	0.070	0.004	0.005	0.006
3	0.066	0.070	0.065	0.006	0.007	0.008
4	0.065	0.066	0.061	0.009	0.010	0.011
5	0.065	0.065	0.059	0.012	0.012	0.013
6	0.064	0.063	0.058	0.014	0.014	0.015
7	0.063	0.061	0.057	0.016	0.016	0.017
8	0.061	0.060	0.057	0.018	0.019	0.020
9	0.060	0.060	0.055	0.020	0.021	0.022
10	0.059	0.058	0.054	0.022	0.023	0.024

RMSE		
кількість врахованих найбільших сингулярних чисел		
25	50	100
5.69	4.97	4.09

Для вибірки Google Review Data обчислення виявилися занадто складними – матриця взаємодій є завеликою.

Сутність методу ALS

- Alternative least squares – модифікація алгоритму Least Squares. Полягає в роботі не з одним параметром, по якому хочемо мінімізувати, а відразу по двох
- В термінах задачі рекомендацій метод полягає в тому, щоб почергово робити ітерацію для матриці користувачів, вважаючи всі значення для об'єктів константними, а потім навпаки



Сутність методу ALS

- Припустимо метрика близько між користувачем та об'єктом – косинусна близькість:

$$L = \sum_{u,i\epsilon S} (r_{ui} - x_u {y_i}^T)^2 + \lambda \left(\sum_u \|x_u\|^2 + \sum_i \|y_i\|^2 \right),$$

- Робимо ітерацію для x, зафіксувавши y:

$$\frac{\partial L}{\partial x_u} = 0 \Rightarrow -2 \sum_i (r_{ui} - x_u {y_i}^T)\, y_i + 2\lambda x_u = 0$$

$$\Rightarrow x_u^* = Y(Y^T Y + \lambda I)^{-1}$$

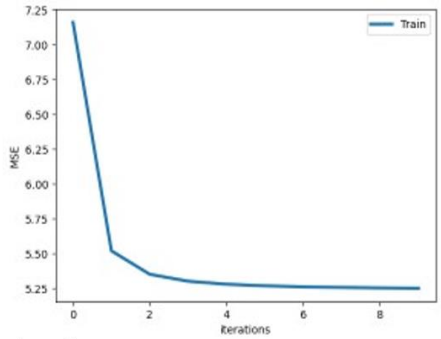
- Аналогічні дії виконуємо для y, зафіксувавши x:

$$\frac{\partial L}{\partial y_i} = 0 \Rightarrow y_i^* = r_i X (X^T X + \lambda I)^{-1}$$

Результати, отримані методом ALS для вибірок MovieLens та Google Review Data

k	Precision@k	Recall@k
1	0.085	0.003
2	0.079	0.007
3	0.074	0.009
4	0.072	0.012
5	0.069	0.015
6	0.066	0.017
7	0.065	0.019
8	0.063	0.022
9	0.062	0.023
10	0.061	0.026

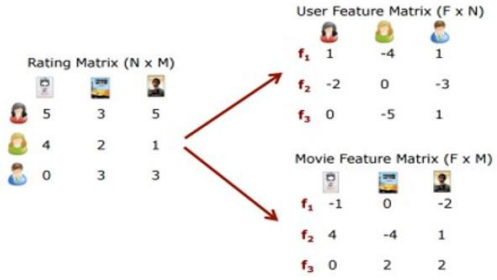
$RMSE \approx 5.25$



Аналогічно методу SVD для вибірки Google Review Data обчислення виявилися також занадто складними – матриця взаємодій є завеликою.

Модифікації матричної факторизації – метод LightFM

- Основна відмінність у переході до ранжувальної задачі
- Користувачів та об'єкти можна відразу задавати через власні характеристики – на відміну від методів SVD та ALS. Такий підхід дозволяє вирішити проблему холодного старту



Математична сутність ранжування LightFM

Алгоритм показує найкращі результати, внаслідок мінімізації WARP Loss – схожої архітектури на раніше згаданий Triplet Loss:

- Вибираємо користувача (u);
- Вибираємо об'єкт, з яким користувач взаємодіяв (i);
- Вибираємо випадковий об'єкт, з яким не було взаємодії (j);
- Обраховуємо скалярні добутки $p = (u, i)$ та $n = (u, j)$;
- Якщо $p > n$, повертаємося на крок 2;
- Якщо $p \leq n$, обраховуємо похибку, оновлюємо ваги та повертаємося на крок 3;

$$L\big(rank(u,i)\big) = \sum_{j \notin Pos} \log\left(\frac{|Neg|}{Sampled}\right) \big|1 - \langle p_u, q_i \rangle - \langle p_u, q_j \rangle\big|_+;$$

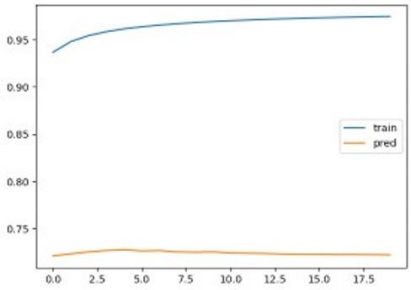
$$[\cdot]_+ = Max(0, x)$$

Результати отримані методом LightFM для вибірки MovieLens

k	Precision@k	Recall@k
1	0.048	0.002
2	0.046	0.004
3	0.045	0.007
4	0.044	0.008
5	0.043	0.011
6	0.043	0.012
7	0.042	0.014
8	0.042	0.016
9	0.041	0.018
10	0.041	0.019

$MAP@10 \approx 0.0124$

$MRR \approx 0.1097$

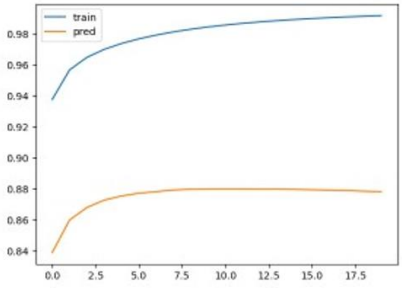


Результати отримані методом LightFM для вибірки Google Review Data

k	Precision@k	Recall@k
1	0.036	0.004
2	0.031	0.008
3	0.028	0.011
4	0.027	0.014
5	0.027	0.017
6	0.026	0.02
7	0.025	0.023
8	0.024	0.025
9	0.024	0.027
10	0.023	0.029

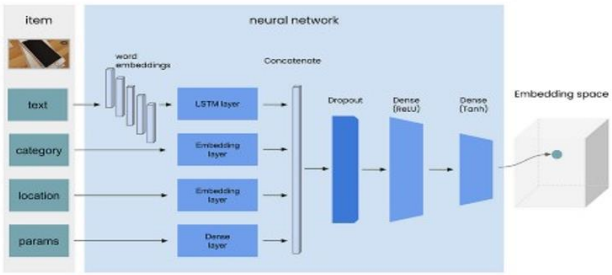
$MAP@10 \approx 0.014$

$MRR \approx 0.067$



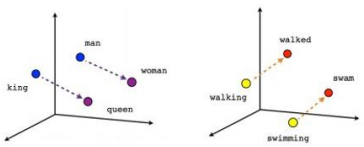
Сутність гібридного методу рекомендацій DSSM

- Основна ідея – надати користувачу або об'єкту таке представлення, щоб схожі сутності знаходилися «близько» у векторному просторі
- При цьому на вхід можуть передаватися дані будь-якої природи, в основну модель дозволяється включення різних внутрішніх архітектур



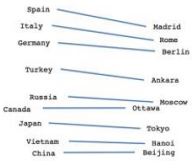
Сутність латентних семантичних представлень

- Дані можна кодувати та передавати в модель безпосередньо
- Інший спосіб – дати моделі внаслідок тренування створити власне представлення (embeddings). Цей підхід запозичили зі сфери Natural Language Processing, коли вчили представляти слова через їх положення в тексті (метод Word2vec):

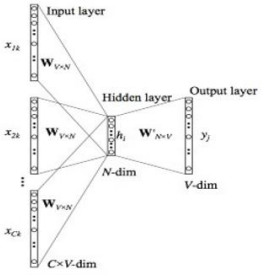


Male-Female

Verb tense



Country-Capital

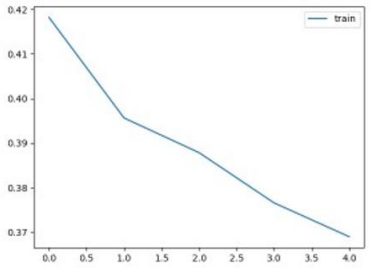


Результати, отримані методом DSSM для вибірки MovieLens

k	Precision@k	Recall@k
1	0.009	4E-05
2	0.011	9E-05
3	0.012	1E-04
4	0.013	2E-04
5	0.014	3E-04
6	0.014	3E-04
7	0.015	4E-04
8	0.016	5E-04
9	0.016	5E-04
10	0.016	6E-04

$MAP@10 \approx 0.00016$

$MRR \approx 0.033$

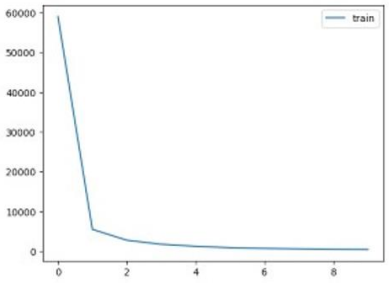


Результати, отримані методом DSSM для вибірки Google Review Data

k	Precision@k	Recall@k
1	0.001	3E-05
2	0.001	5E-05
3	0.002	7E-05
4	0.001	9E-05
5	0.002	1E-04
6	0.002	1E-04
7	0.002	2E-04
8	0.002	2E-04
9	0.002	2E-04
10	0.002	2E-04

$MAP@10 \approx 0.0052$

$MRR \approx 0.0089$



Сутність гібридних методів рекомендацій Sequential

- Кодувати ми можемо і послідовну інформацію. Наприклад послідовність взаємодій
- В такому випадку користувач буде представлятися через послідовність N його минулих взаємодій, а рекомендувати ми будемо «наступний» об'єкт – по такому правилу ми сформуємо тренувальну вибірку
- Текстову інформацію, або інформацію про приналежність до категорії, також можна закодувати як послідовну

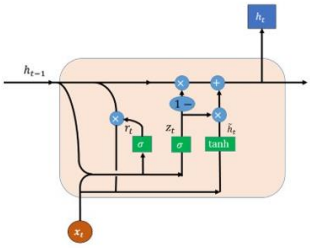
{RV Park, Cabin rental agency, Campground}

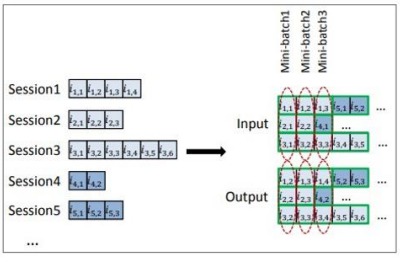
⇒

	0	1	2	3	4
0	0	0	1857	898	271

Сутність гібридних Sequential методів GRU та GRU4Rec

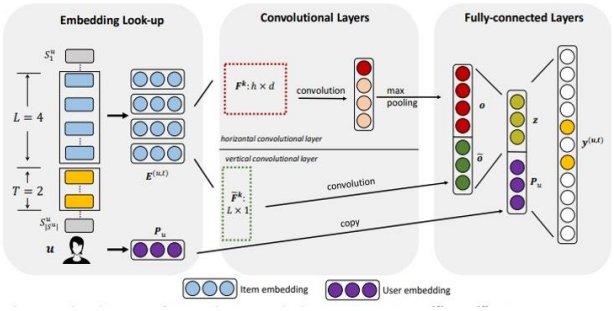
- Sequential моделі використовують рекурентні нейронні мереж (RNN). Дві найвідоміші архітектури – LSTM та GRU
- Послідовні архітектури не задумувались для надання рекомендацій, тому основна задача – адаптувати їх під цю задачу. Архітектура GRU4Rec є довершенням класичної GRU архітектури для Session-based задачі.





Сутність гібридного Sequential методу Caser

- При кодуванні послідовностей використовує підхід згортки зі сфери машинного навчання Computer Vision
- Нашим зображенням в даному випадку є послідовність послідовностей

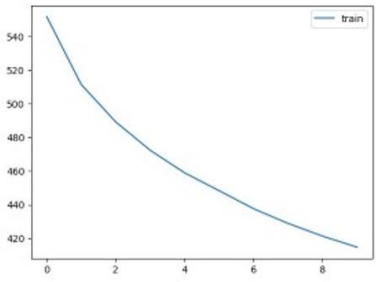


Результати, отримані методом GRU для вибірки MovieLens

k	Precision@k	Recall@k
1	0.006	1E-04
2	0.005	3E-04
3	0.006	4E-04
4	0.006	5E-04
5	0.006	5E-04
6	0.007	7E-04
7	0.007	8E-04
8	0.007	1E-03
9	0.007	0.001
10	0.007	0.001

$MAP@10 \approx 0.00038$

$MRR \approx 0.019$

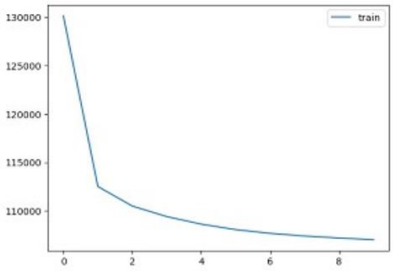


Результати, отримані методом GRU для вибірки Google Local Review

k	Precision@k	Recall@k
1	0.002	2E-04
2	0.002	4E-04
3	0.002	5E-04
4	0.002	7E-04
5	0.002	1E-03
6	0.002	0.001
7	0.002	0.001
8	0.002	0.002
9	0.002	0.002
10	0.002	0.002

$MAP@10 \approx 0.00056$

$MRR \approx 0.00571$

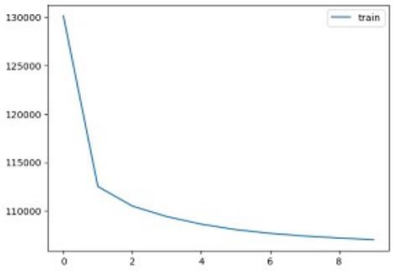


Результати, отримані методом GRU для вибірки Google Local Review

k	Precision@k	Recall@k
1	0.002	2E-04
2	0.002	4E-04
3	0.002	5E-04
4	0.002	7E-04
5	0.002	1E-03
6	0.002	0.001
7	0.002	0.001
8	0.002	0.002
9	0.002	0.002
10	0.002	0.002

$MAP@10 \approx 0.00056$

$MRR \approx 0.00571$



Висновки по результатам метрик якості

- Найкращі результати по метрикам якості для обох вибірок показали методи послідовної рекомендації – GRU4Rec та Caser, проте час на тренування таких методів – значний
- Найшвидшими по часу роботи виявилися методи класичної матричної факторизації – SVD та ALS, для цих алгоритмів важливим фактором виявився об'єм вхідної вибірки: в деяких випадках обрахунки неможливі
- Проте цю проблему може вирішити ранжувальний метод LightFM – як і попередні факторизаційні методи, він видає швидкі та доволі точні результати
- Реалізовані методи DSSM та GRU показали гірші результати, але їх архітектура дозволяє передавати дані різної природи та самостійно визначати спосіб трансформації