МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ”

Інститут прикладного системного аналізу
(назва факультету, інституту)

Кафедра системного проектування
(назва кафедри)

До захисту допущено

Завідувач кафедри

_____ А.І. Петренко
(підпис) (ініціали, прізвище)

“___”_____2007 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проекту (роботи) освітньо-кваліфікаційного рівня “ **спеціаліст** ”
(назва ОКР)

з напряму підготовки (спеціальності)_____
(код та назва напряму підготовки або спеціальності)

7.05010103 «Системне проектування»

на тему: Рекомендаційна система з використанням алгоритмів штучного інтелекту на базі серверлес архітектури._____
_____

Студент групи **ДА-52с** **Сливка Тарас Петрович** _____
(шифр групи) (прізвище, ім’я, по батькові) (підпис)

Керівник проекту **доцент Булах Б. В.** _____
(вчені ступінь та звання, прізвище, ініціали) (підпис)

Консультанти:

з технічних вимог_____ к.т.н., доц. Харченко К.В. _____
(назва розділу ДП (ДР)) (вчені ступінь та звання, прізвище, ініціали) (підпис)

нормоконтроль_____ к.т.н., доц. Стіканов В.Ю. _____
(назва розділу ДП (ДР)) (вчені ступінь та звання, прізвище, ініціали) (підпис)

Київ – 2016

Національний технічний університет України

“Київський політехнічний інститут”

Факультет (інститут) **Інститут прикладного системного аналізу**
(повна назва)

Кафедра **Системного проектування**
(повна назва)

Напрям підготовки **050101, Комп’ютерні науки**
(код, назва)

Спеціальність **7.05010103, «Системне проектування»..**
(код, назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри
_____ А.І. Петренко
(підпис) (ініціали, прізвище)

“___”_____ 2017 р.

ЗАВДАННЯ
на дипломний проект (роботу) освітньо-кваліфікаційного рівня
“спеціаліст”
(назва рівня)

студенту Сливці Тарасу Петровичу
(прізвище, ім’я, по батькові)

1. **Тема проекту** (роботи) Рекомендаційна система з використанням алгоритмів штучного інтелекту на базі серверлес архітектури.
затверджена наказом по університету від «20» 09 2016р. № 89/1-СТ

2. **Термін здачі** студентом закінченого проекту (роботи) “15” 01 2017р.

3. **Вихідні дані до проекту** (роботи)
Програмна архітектура: серверлес, здатність до швидкого розгортання сервісу, відповідь системи на запит в межах 5 сек., при кількості об'єктів для аналізу порядку 10^5.

4. **Перелік питань, які мають бути розроблені**
- Сформувати постановку задачі
- Розробити теоретичні варіанти вирішення задачі
- Проаналізувати та обрати інструменти вирішення

- Розробити практичне рішення
- Розробити управляння термінами

4. **Перелік графічного (ілюстративного) матеріалу**
Архітектурна схема PC (креслення)
Основні типи архітектур програмних сервісів(креслення)
Архітектурна схема на базі AWS (1-ий плакат)
Метод колаборативної фільтрації (2-ий плакат)
ALS (3-ій плакат)
Основні положення (4-ий плакат)

6. **Консультанти**
з технічних вимог: доц.к.т.н . Харченко К.В

7. **Дата видачі завдання** " 10 " 09 2016 р.

Керівник дипломного проекту (роботи) _____ Булах Б.В.
(підпис) (ініціали, прізвище)

Завдання прийняв до виконання _____ Сливка Т.П.
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

До дипломного проекту Сливки Тараса Петровича

на тему «Рекомендаційна система з використанням алгоритмів штучного інтелекту на базі серверлес архітектури.»

Даний дипломний проект присвячений розробці рекомендаційної системи на основі серверлес архітектури.

Об’єкт дослідження : рекомендаційні системи.

Предмет дослідження: методи оптимізації архітектурного впровадження, алгоритми машинного навчання для розробки рекомендаційної системи.

Цілі і задачі дослідження: головною метою даного дослідження є побудова та реалізація, практичне втілення РС з використанням серверлес архітектури. З’ясувати актуальність теми, її теоретичне вирішення, спробувати реалізувати його на практиці та зробити відповідні висновки.

Метод дослідження: наукове прикладне дослідження з використанням відомих архітектур, методів, алгоритмів.

Актуальність теми: робота тісно пов’язана з проблемою оптимізації кількості необхідного серверного часу для обчислень великих даних, у зв’язку із зростанням електронної комерції у світі.

Наукова новизна: в даній роботі була розроблена рекомендаційна система з використанням алгоритмів штучного інтелекту на базі серверлес архітектури, а саме - без використання постійних серверів. Реалізовано на AWS стеку компонентів.

Загальний об'єм: 71 сторінок, з яких 4 - додаток обсягом 6 сторінок, 10 рисунків, 2 таблиці.

Кількість посилань: 13.

Ключові слова: РЕКОМЕНДАЦІЙНА СИСТЕМА, СЕРВЕРЛЕС, SERVERLESS, AWS, SPARK, КОЛАБОРАЦІЙНА ФІЛЬТРАЦІЯ.

To the diploma thesis by Slyvka Taras Petrovych

The topic is «Recommender system with using artificial intelligence algorithms based serverless architecture»

This thesis is devoted to the development recommender system with using artificial intelligence algorithms based serverless architecture.

Object of research: recommender system.

Subject of research: optimization methods of architecture implementation, machine learning algorithms to develop recommendation system.

Goals and objectives of the study: the main purpose of this study is to construct and implement recommender system on a serverless architecture. Find out the relevance of the topic and its theoretical solution, try to put it into practice in the form of a computer program and draw appropriate conclusions

Method of research: scientific applied research on the use of the known architectures, methods and algorithms.

Relevance of the topic: work closely linked to the optimization process of minimizing server resources for processing big data, due to the growth of electronic commerce in the world.

Scientific novelty: in this study developed recommender system using artificial intelligence algorithms based on serverless architecture - namely, without using regular servers. Implemented in AWS stack components.

Total capacity: 71 pages, of which 4 application of 6 page, 10 figures, 2 tables.

Number of links: 13.

Tags: RECOMMENDER SYSTEM, SERVERLESS, AWS, SPARK, COLLABORATIVE FILTERING.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ

РС – рекомендаційна система

КФ – колаборативна фільтрація

AWS – з англ. Amazon Web Services

ALS – з англ. Alternating Least Squares

SVD – з англ. Singular Value Decomposition

HTTP – з англ. Hypertext Transfer Protocol

S3 – з англ. Simple Storage Service

EMR – з англ. Elastic MapReduce

ВСТУП

Ми живемо в цікаву епоху, де все швидко росте та змінюється. Це стосується й інформації. Її кількість з кожним роком все більше і більше зростає. Сюди входять різні дані, аудіо та медіа контент, тексти.

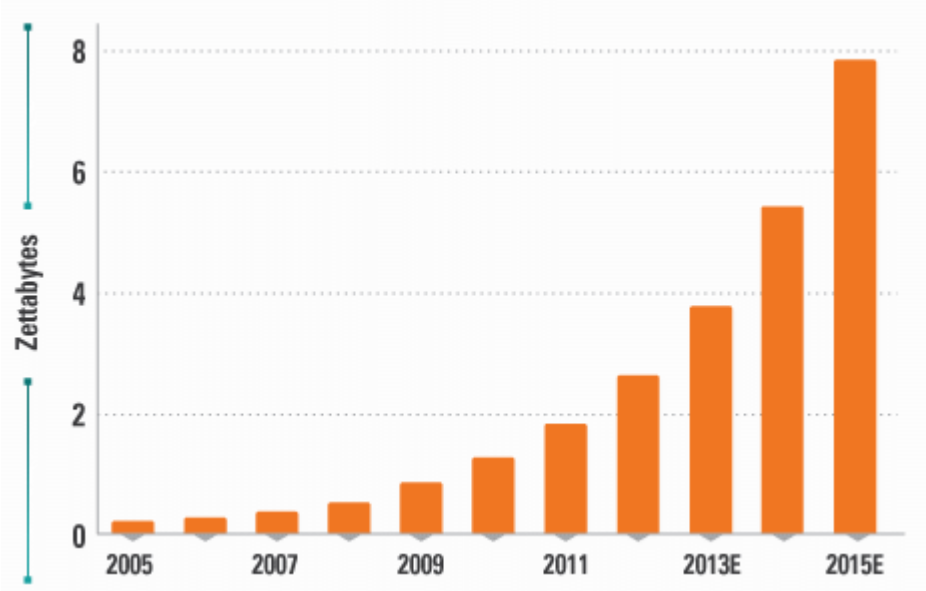


Рисунок 1.1 - Графік росту інформації [1]

Як наслідок це призводить до збільшення ентропії інформації. Що породжує проблему структуризації, підбору потрібного нам рішення, необхідної покупки і т.д. Тобто нас все важче й важче віднайти необхідно інформацію.

Пригадуючи одну з найкращих історій А. Азімова “Останнє запитання” (“Last question”). Де красиво розкрите питання збільшення ентропії. Над цією задачею працює найбільший комп’ютер у світі. Змінюються покоління, планети, сонячні системи, а робота над пошуком рішення, “Як розвернути процес збільшення ентропії у зворотному напрямку?” триває. За текстом задачу вирішать, хоча, на мій погляд, за надто пізно. Але, не забуваймо, що це - все ж таки книга з розділу фантастики. Ми живемо в реальному світі і маємо практичні потреба та шанс змінити розв’язку книги.

Дедалі більше значення Інтернету, як середовища для електронних і ділових операцій, що служить в якості рушійної сили для розвитку технології рекомендаційних систем. Важливим каталізатором в цьому відношенні є легкість, з якою Web дозволяє користувачам забезпечити зворотний зв'язок про свої симпатії чи антипатії. Наприклад, розглянемо сценарій Netflix. Користувачі цього сервісу можуть легко проявити свою симпатію простим натисненням кнопки миші. Типова методика для забезпечення зворотного зв'язку у вигляді рейтингів, в яких користувачі вибирають числові значення з системи конкретної оцінки (наприклад, п'ятизіркову систем рейтингу), якою визначають свої симпатії і антипатії різних предметів.

Існують й інші форми зворотного зв'язку не настільки явні, але ще легші у способі збору особливо, у Web середовищі. Наприклад, простий акт покупки або перегляду елемента може розглядатися як схвалення для цього об’єкта. Такі форми зворотного зв'язку, як правило, використовується онлайн продавці, відомим прикладом може слугувати компанія Amazon. Такий збір не потребує додаткових втручань споживача, що призводить до більш масштабного збору даних. Основна ідея рекомендаційних системи є використання різних джерел даних для розуміння інтересів клієнтів.

Особа, до якої дана рекомендація надається є користувачем, а рекомендований продукт є елементом. Таким чином, рекомендаційний аналіз часто заснований на попередній взаємодії між користувачами та елементами, так, як інтереси проявлені в минулому часто є хорошими індикаторами того, що буде цікавий користувачу під час майбутніх виборів. Винятком є інший тип РС (Knowledge-based recommender system), в яких рекомендації пропонуються на основі заданих користувачем вимог, а не його історії в минулому.

Отже, що є основним принципом, який лежить в основі алгоритмів роботи рекомендаційної системи? Основний принцип полягає в тому, що існують суттєві залежності між користувачами та їх діями по відношенню до цільових елементів, їхньої діяльності. Наприклад, користувач, який

зацікавлений в історично-документальних фільмах, більш імовірно, буде зацікавлений в інших історично-документальних або освітніх програмах, а не в бойовику. У багатьох випадках різні категорії елементів можуть показати значущі кореляції, що можуть бути використані для покращення точності рекомендації.

Чим більше число номінальних елементів, які доступні для користувача, тим легше зробити надійні прогнози щодо його майбутньої поведінки. Багато різних моделей навчання може бути використано для виконання цього завдання. Наприклад, колективна покупка або оцінка поведінки різних користувачів можуть бути використані для створення когорти подібних користувачів, які зацікавлені в аналогічних продуктах. Інтереси і дії цих когорт в свою чергу можуть бути використані, щоб зробити рекомендації для окремих членів цих когорт.

В даній роботі мова йтиме про рекомендаційні системи(РС). Ви дізнаєтесь, що це таке і для чого потрібно. Історія виникнення, розвиток, проблеми сьогодення. Які існують види та де їх краще використовувати. Як реалізувати РС у реальних умовах та які існують підводні камені. Ціллю даної роботи є знаходження відповіді на запитання: “Якою має бути сучасна рекомендаційна система?”.

1 ПОСТАНОВКА ЗАДАЧІ

В даному розділі мова йтиме про те що таке РС, звідки виникла, яким чином змінювалась з часом, як використовується в сучасному світі. Розглянемо існуючі рішення. Дамо відповідь на запитання, чому задача з більш ніж 20-ти річною історією має актуальність в сьогоденні. На основі отриманої інформації сформуємо вимоги до задачі, поставимо цілі та підіб'ємо підсумки.

1.1 Що таке рекомендаційна система

Заходимо на сайт пошукової системи google.com, вводимо запит: "Що таке рекомендаційна система" (українською мовою). Отримуємо близько 49 тис. результатів, орієнтовно за 0.5 секунди. Пошукова система ранжує результати таким чином, що спершу йдуть більш релевантні. Тобто бажану відповідь ми швидше за все знайдемо на перші сторінці. Але на превеликий жаль її там немає. Ні, це не говорить про погано роботу алгоритму пошуку. Це свідчить про малу кількість релевантного контенту українською мовою. Однією з другорядних цілей моєї робити є спроба змінити цю ситуацію.

Змінивши запит я все ж знаходжу визначення. "Рекомендаційні системи являють собою програмні інструменти і методи, які забезпечують пропозицію елементів, що будуть корисними для користувача. Пропозиції відносяться до різних процесів прийняття рішень, наприклад: які товари купувати, яку музику слухати, або які новини читати."[2]

Більшість з нас зустрічається в побуті з РС. Читаючи статті на medium.com нам порекомендують ще декілька, серед яких знаходиться та, яка нас зацікавить.

Найбільша у світі соціальна мережа facebook.com пронизана РС, де стрічка новин формується на основі ваших вподобань проявлених раніше тим чи іншим способом. Нам порекомендують людей, яких ми могли б знати, і ми їх знаємо.

Купуючи товари на Amazon ми отримуємо підбірку інших, серед яких найдеться щось, що нас зацікавить. За цим всім стоїть РС. Так це виглядає зі сторони користувача, для математика: РС - це підклас системи фільтрації інформації, яка будує рейтинговий перелік об'єктів. А для інженера це все ускладнюється великою кількість особливостей під час практичної реалізації.

Тобто РС - це чорний магічний ящик, який на основі попередніх спостережень чи властивостей елементів може видати схожі чи то супутні елементи. Що в середині та як воно працює розглянемо далі.

1.1.1 Історія РС

Рекомендація системи бере своє коріння в "Usenet," всесвітня розподілена система обговорення, що існувала в університеті Герцога в кінці 1970-х років. Usenet працював в форматі клієнт/сервер, що дозволяв каталогізувати введені користувачем повідомлення, та відносити їх до певної групи новин. У Usenet, пости, зроблені користувачами поділялися на ці групи новин, а також за необхідності розділялись на підкатегорії.

Інформаційна Фільтрація (IF) є способом просіювання, у зв’язку з надмірною кількістю даних в Інтернеті. Так, як інформація росла в геометричній прогресії, то адміністратори не справлялись з менедженням відповідних даних. Ось деякі з ранніх рішень для просіювання даних включають в себе:

Tapestry - розроблений Xerox, вони придумали фразу "колаборативна фільтрація".

Lotus Notes - компонент цього програмного забезпечення був побудований, як елемент колаборативної фільтрації.

GroupLens - все почалося в 1992 році, цей проект є Open Source. Був побудований з посилу **Tapestry**, з метою спрощення Usenet даних за допомогою розподілених мереж. Що робив поради на основі рейтингів.

Pattie Maes була, в першу чергу, відповідальною за колаборативну фільтрацію з появою за допомогою її зусиль в MIT системи під назвою Firefly. Система рекомендацій для любителів музики. Firefly пізніше був куплений Microsoft приблизно за 40 мільйонів доларів.

1.1.2 Види РС

На даний момент РС можна класифікувати за їхніми методами вирішення, розділивши на 3-ри види: *фільтрація вмісту*, *колаборативна фільтрація* та їх поєднання - *гібридна система*. У кожного з яких є свої недоліки та переваги про це більш детальніше йтиме мова у розділі **2.1**.

Фільтрація вмісту - методи побудовані на основі властивостях елементу та профайлу користувача, його уподобань. У **фільтрації вмісту** РС ключові слова використовуються для опису елементів та профайлів коростувача, на основі яких будуються передбачення щодо типу елементів, які можуть сподобатись. Іншими словами цей тип РС намагається порекомендувати елементи, що схожі, до тих, які користувач вподобав у минулому.

Колаборативна фільтрація - це метод, який використовує відомі переваги (оцінки) групи користувачів для прогнозування невідомих переваг іншого користувача. Основне припущення колаборативної фільтрації полягає в наступному: ті, хто однаково оцінювали будь-які предмети в минулому, схильні давати схожі оцінки інших предметів і в майбутньому.

Гібридна система - це комбінація фільтрації вмісту та колаборативної.

1.1.3 Приклади використання

На даний момент сформувалась група світових лідерів, які вправно та ефективно використовують РС на практиці.

Amazon - один із перших інтернет-сервісів, орієнтованих на продаж реальних товарів масового попиту. РС компанії Амазон набула популярності у всьому світі, практично кожен, хто пише статтю чи книгу пов’язану з РС

згадую компанію. Її ставлять у приклад, як потрібно використовувати РС в провідних університетах світу, як з інженерної так, і з бізнес сторони.

Історія почалась, коли 1997 року 24-річний Грег Лінден відклав роботу над докторською дисертацією у Вашингтонському університеті та був нанятий як інженер.

На той момент у компанії існував окремий редакторський відділ, який займався рекомендаціями книг. Співробітників якого, журнал Wall Street Journal назвав одними з самих впливових літературних критиків США, це була гордість компанії.

Вже 1998 році Лінден подав заявку на патент методу колаборативної фільтрації “предмет-предмет” РС. Цей алгоритм дозволив виконувати попередні обрахунки завчасно та надавати швидкі рекомендації в режимі онлайн. Після інтеграції дослідження показали, що впровадження РС призвело до збільшення продажів практично у 100 разів. Після чого редакторський відділ розпустили, а в компанії ходив жарт, про те, що на сайті достатньо показувати одну книгу - ту, яку ви купите. На даний момент вважається, що третину своїх продаж компанія завдячує РС. [4]

Netflix - американський провайдер інтернет-служби "на вимогу" потокового мультимедіа у 243 країнах або територіях . Ця компанія також відома своєю крутою РС. Особливо цікавою є історія про “The Netflix Prize” вже з далекого 2009 року. Все почалось 2-го жовтня 2006 року, коли компанія об’явила початок публічного конкурсу на вдосконалення їхньої РС. Серед умов: вдосконалити алгоритм РС більше ніж на 10% на тестових даних. Переможець отримує $ 1 000 000. Задачу вирішили, і таких знайшлось аж дві команди, що спромоглись підвищити точність прогнозування на 10.06%, правда одна з команд на 20 хв швидше завершила роботу і отримала приз. Конкурс тривав три роки та досягнув результату. [3]

LinkedIn - бізнес орієнтована соціальна мережа, яка формує рекомендації для людей, яких ви могли б знати, роботу яку ви могли б любити, групу, в яку

ви можливо захотіти б вступити, або компанію, яка могла б бути зацікавлена у вас.

1.2 Чому ця задача актуальна

Вирішення проблеми РС було новизною понад 20-ть років тому, але незважаючи на це задача не втратила своєї актуальності. За цей час змінились потреби до системи, умови її використання. Надзвичайно сильно збільшився обсяг даних. Якщо раніше мова йшла про 10^5, то зараз актуальними стали 10^8 кількість користувачів. А це призвело до збільшився обсяг даних, яку не просто обробити, передати. Також з'являються нові вимоги до РС, такі, як час розробки, що корелює з фінансовими затратами. Вартість підтримки системи. Тому сама задача з часом зазнала змін і зараз вже не достатньо розробити алгоритм, а потрібно врахувати великий обсяг даних, та пов'язані з ним наслідки. Ключовий момент відіграє архітектура системи.

1.3 Вимоги проектування

Проаналізувавши, дослідивши питання ми приходимо до висновку про те, що задача на розробку РС є актуальною та важливою. Але її вимоги з часом зазнають змін. А саме на даний момент нам потрібно проаналізувати та запропонувати сучасну архітектуру РС, що відповідатиме вимогам бізнесу.

- **Мінімізувати час розробки** - максимально скоротити необхідний час для практичної реалізації системи.
- **Мінімізувати вартість роботи** - максимально зменшити фінансові затрати при роботі системи на одиницю часу (місяць).
- **Роботу в реальному часі** - система повинна видавати рекомендації в режимі реального часу, з попередніми обрахунками.
- **Робота з великими даними** - система повинна витримувати навантаження при обробці великих даних (big data).

1.4 Висновки

В даному розділі я ознайомився з поняттям РС. Це певною мірою чорний ящик, що здатний формувати певні залежності між тим, які люди і що купують. Та на основі цих залежностей можна отримати передбачення цікавого/потрібного продукту для користувача. Усі РС можна поділити на три види: фільтрація вмісту, колаборативна фільтрація та їх поєднання - гібридна. Історично практичне використання РС прийшло у наше життя не так і давно.

На даний момент існує декілька компаній світового рівня, що слугують прикладом ефективного використання РС, серед яких Amazon, Netflix, LinkedIn. Їх ставлять у приклад, використовують, як зразок в університетах світу.

Не зважаючи на поширеність та велику кількість алгоритмів, літератури, способів вирішення ця задача залишається актуальною. В основній мірі це пов’язано зі зростанням кількості даних та необхідності адаптації під це алгоритмів, архітектурних підходів та оптимізації коштів на затрату, як при розробці, так і при підтримці та утриманні системи.

2 ТЕОРЕТИЧНІ ВАРІАНТИ ВИРІШЕННЯ ЗАДАЧІ

2.1 Вибір методу РС

При розробці РС зазвичай стикаються з рядом проблем прогнозування:

• **Розрідженість даних** - більшість користувачів не ставить оцінки товарам, отже дані з попередніми оцінками являють собою розріджену матрицю.

• **Холодний старт** - робота з новими користувачами або товарами.

• **Синонімія** - проблема розпізнавання схожих товарів з різними назвами.

• **Шахрайство** - цілеспрямоване завищення рейтингів певних товарів їх власниками.

• **Розмаїття** - при великій вибірці нові або маловідомі товари мають низькі позиції в рейтинговому списку.

• **Білі ворони** - унікальні користувачі, смаки яких дуже важко обробити, оскільки вони не співпадають зі смаками відокремлених типів.

2.1.1 Фільтрація вмісту

Цей тип алгоритмів прогнозування базується на моделі об’єкту, оцінки якого будуть прогнозуватися. Для кожного об’єкту буде побудовано математичну модель з використанням конкретних характеристик товару (параметри моделі). Рекомендації будуть базуватися на порівнянні характеристик поточного товару та власне характеристик користувача (інформація, яка міститься в профілі користувача). Для прикладу візьмемо сайт з онлайн кінотеатром. Нехай маємо користувача, який переглянув наступні фільми: «Міцний горішок» (бойовик), «Скайфолл» (бойовик, трилер). Висувається гіпотеза, що цьому користувачу подобаються фільми з жанром бойовик, тому логічно будуть створені рекомендації фільмів жанру бойовик.

Переваги: більш точний результат; немає проблеми холодного старту, оскільки рекомендації базуються на моделі об’єкта, а не на попередніх оцінках користувачів.

Недоліки: «затратне» створення моделі (її побудо- ва досить складна), невисока швидкодія алгоритмів (багато обчислень); втрата точності при скороченні параметрів моделі.

2.2.2 Колаборативна фільтрація

Це один з методів прогнозу в рекомендаційних системах, який використовує відомі переваги (оцінки) групи користувачів для прогнозування невідомих переваг (оцінок) іншого користувача. За допомогою цього алгоритму будується певна таблиця користувачів, які групуються за схожістю, та прогнозуються результати для інших користувачів.

Наприклад, маємо декількох користувачів порталу з музикою. Всіх користувачів можна поділити на групи за їх смаками (одним подобається джаз, іншим — рок). За цією інформацію в середині кожної групи можна виділити най популярніші хіти, які користувачі слухають більше всього. Отже, кожному учаснику певної групи будуть порекомендовані популярні композиції, які ним не були ще прослухані. [5]

Переваги: швидка робота алгоритмів (K-based та ін.) — мала кількість ітерацій; прості в реалізації.

Недоліки: холодний старт; нема що порекомендувати новим або нетиповим користувачам; розріджені матриці оцінок (іноді неможливо зробити прогноз); шахрайство.

2.2.3 Гібридна система

Даний тип алгоритмів поєднує в собі підходи колаборативної та content-based фільтрації. Цей підхід найбільш популярний при розробці рекомендаційних систем для комерційних сайтів, так як його було створено

щоб подолати проблеми колаборативної фільтрації, а також покращити якість прогнозування оцінок конкретної моделі.

Переваги: велика швидкодія; кращі результати.

Недоліки: дуже дорога розробка рекомендаційної системи, оскільки реалізація цього типу алгоритмів дуже складна; важко підтримувати, оскільки навіть незначні зміни в роботі призводять до змін роботи алгоритму.

2.2 Вибір алгоритму РС

На даний момент розроблено велику кількість алгоритмів та їх модифікацій для методу колаборативна фільтрація.

Одним з основних відмінностей є тип даних, які ми маємо в розпорядженні. Грубо кажучи вона поділяється на явних і неявних даних.

Приклади явного збору даних включають в себе наступне:

- Попросити користувача оцінити елемент за динамічною шкалою.
- Попросити користувача оцінити колекцію елементів від менш до більш улюблених.
- Представити користувачу два елементи і попросити вибрати кращий з них.
- Попросити користувача сформувати список улюблених елементів.

Приклади неявного збору даних включають в себе наступне:

- Перегляди сторінок на онлайн ресурсі.
- Покупки онлайн.

Зазвичай компанії, які хочуть вирішити цю проблему мають велику кількість неявних даних. Я представлю, як метод найменших квадратів (ALS) може бути використаний для явних даних, а також для неявних.

2.2.1 Перемінно найменших квадратів (ALS)

З анг. Alternating Least Squares - перемінно найменших квадратів. Ідея алгоритму, як і більшості інших КФ полягає в декомпозиції матриці.

Основна ідея будь-якого методу матричної факторизації, щоб знайти менший набір прихованих чинників, які можуть бути використані для прогнозування відсутніх записів.

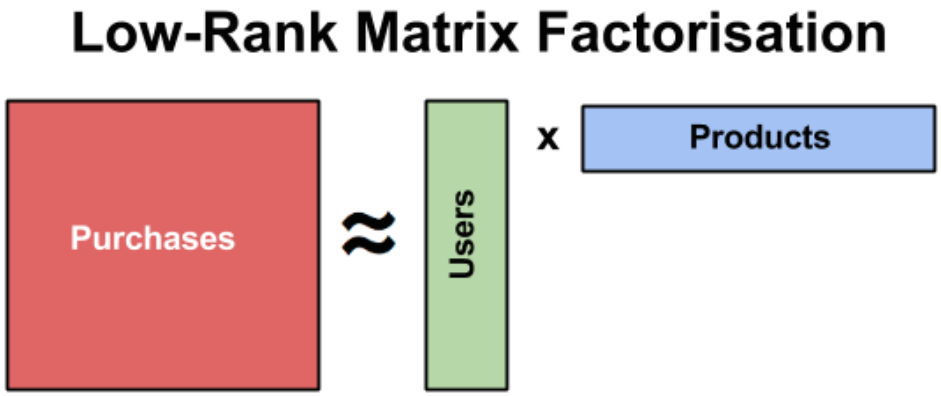


Рисунок 2.1 - Схематична схема декомпозиції матриці

У наведеному вище прикладі, ми хочемо розкласти матрицю покупок в набір призначених для користувача латентних факторів і набір продуктів латентних факторів.

Як правило, ми не в змозі виконати цю факторизацию, використовуючи прямі методи (наприклад, SVD) в якості вхідної матриці містить в основному відсутні значення. Для вирішення цієї проблеми ми розглядаємо приховані чинники, як параметри, які можна отримати й обробити факторизацію як завдання оптимізації.

Давайте припустимо, що у нас є деяка кількість користувачів, а також деяка кількість елементів. Ми звертаємось до користувачів за допомогою синтаксису u та v, а також до елементів i та j.

Користувачам і елементи пов'язані через $r_{u,i}$, які ми будемо називати спостереженням. Для явних наборів даних це - рейтинги, що користувач дав елементу. Більші числа означають більш сильну перевагу. Для неявних наборів даних це буде означати дії користувача. Наприклад, це може бути скільки разів користувач придбав товар або переглянути елемент в Інтернеті.

Ми будемо асоціювати кожного користувача u з фактор-вектором x_u, відповідно кожен елемент з Y_i фактор-вектором. Наша мета полягає в тому, щоб передбачити наші спостереження від прихованих чинників:

$$\hat{r}_{u,i} = x_u^\top y_i$$

Ми хочемо, щоб наші передбачення, щоб були якомога ближче до істини, наскільки це можливо. Тому, щоб знайти приховані вектори формуємо задачу як оптимізації використовуючи стандартну похибку квадратів з регуляризацією.

$$\min_{x,y} \sum_{r_{u,i} \text{is known}} (r_{u,i} - x_u^\top y_i)^2 + \lambda (\sum_u ||x_u||^2 + \sum_i ||y_i||^2)$$

Давайте згрупуємо всі фактор-вектори рокистувача в $m \times f$ матрицю X та відповідно всі фактор-вектори елементів в $n \times f$ матрицю Y. Якщо ми припустимо, якийсь з векторів є фіксованим, то це виглядатиме так само, як проблема регуляризації найменших квадратів. Отже, спочатку давайте припустимо, що Y є фіксованою і вирішимо для x_u:

$$x_u = (Y^\top Y + \lambda I)^{-1} Y^\top r_u$$

Аналогічно, якщо ми припускаємо, що X фіксоване, і вирішимо для y_i:

$$y_u = (X^\top X + \lambda I)^{-1} X^\top r_i$$

Ідея полягаю в тому, щоб ітерувати цих два кроки до моменту виконання критерію зупинки. Кількість ітерацій, як правило, в районі 10-ти.

Неявні дані. Безпосереднє використання сирих спостережень показало не таку ефективність, як неявних. Тому замість цього ми визначаємо набір бінарних змінних:

$$p_{u,i} = \begin{cases} 1 & \text{if } r_{u,i} > 0 \\ 0 & \text{otherwise} \end{cases}$$

Тепер ми хочемо визначити деякі рівні довіри для кожного $p_{u,i}$. Коли $r_{u,i} = 0$ ми маємо низький рівень довіри, це може бути користувач ніколи не мав справи з цим елементом, або сам елемент може бути недоступний в цей час.

Коли $r_{u,i}$ є малим, це також можна пояснити, як покупку подарунок для когось, отже, ми все одно маємо низький рівень довіри. Коли $r_{u,i}$ є великим, ми маємо набагато більше впевненості.

Ми маємо повну свободу, як ми визначаємо довіру. Початкова пропозиція:

$$c_{u,i} = 1 + \alpha r_{u,i}$$

У нас є певний рівень довіри для кожного $p_{u,i}$. Параметр α є налаштовується, до набору даних на етапі крос-перевірки.

Наша нова функція оптимізації:

$$\min_{x,y} \sum_{r_{u,i} \text{is known}} c_{u,i}(p_{u,i} - x_u^\top y_i)^2 + \lambda(\sum_u ||x_u||^2 + \sum_i ||y_i||^2)$$

Визначимо діагональну матрицю C^i, де $C^i_{u,u} = c_{u,i}$. Знаходження параметрів є таким же, як при стандартним методом найменших квадратів:

$$y_u = (X^\top C^i X + \lambda I)^{-1} X^\top C^i p_i$$

$$x_u = (Y^\top C^u Y + \lambda I)^{-1} Y^\top C^u p_u$$

Ви можете запитати, навіщо вдаватись до ваг і чому б не масштабувати рейтинги натомість? Проблема в тому, що ви включаєте дуже рідкісну проблему в дуже щільною проблеми, і ви будете мати проблеми навіть чисельно обчислень МНК на щільною проблеми.

Давайте ще раз подивимося на цінову функцію:

$$\mathcal{L} = \min_{x,y} \sum_{r_{u,i} \text{is known}} (r_{u,i} - x_u^\top y_i)^2 + \lambda(\sum_u ||x_u||^2 + \sum_i ||y_i||^2)$$

Коли ми припускаємо, що *Y* фіксоване, *L* стає опуклою в *X*. Кожен крок наближає нас до мінімуму, але не обов'язково глобального мінімуму.

Це буде означати, що кожного разу алгоритм отримуватиме інший результат в залежності від того, як ініціалізованих *X* та *Y*. Варто запускати алгоритм кілька разів при тестуванні різних значень гіперпараметрів.

Також зверніть увагу, якщо ви зробите λ досить великою, алгоритм буде сходитися до того ж рішення кожен раз. Це тому, що стримуватиметься ваш простір рішень.

Ініціалізація. Оскільки функція не є опуклою, то ініціалізація параметрів може допомогти сходяться до вирішення швидше .

Одна рекомендований підхід, що в даний час використовується вбіліотеці Spark: Виберіть одиничний вектор рівномірно випадковим чином з одиничної сфери, але з "першого квадранта", де всі елементи невід'ємні. Це може бути зроблено шляхом вибору елементів, розподілених в звичайному режимі (0,1) і приймаючи абсолютне значення, а потім нормалізувати.

Важливо зауважити, що краща ініціалізація може швидше привести до глобального мінімуму.

Гіперпараметри. Точні значення λ та *a* є значеннями, що напряму залежать від даних і повинні визначатися на етапі перехресної перевірки.

Критерій зупинки. Коли зупинити алгоритм, як правило, визначається:

- Якщо максимальне число ітерацій виконано
- Якщо різниця між виміряним MSE поточної ітерації і попередньої ітерації стає нижче деякого епселон.

Функції довіри. Ви можете використовувати будь-яку функцію для ваги довіри, яку захочете. Безпосередня лінійне масштабування не може бути тим, що вам ротрібно. Наприклад, в сценарії електронної комерції, ви можете використовувати ваги уздовж ліній:

- 3 для перегляду сторінки продукту
- 10 для додавання в кошик
- 40 для покупки продукту [5]

2.2.2 Сингулярний розклад матриці (SVD)

Сингулярний розклад матриці (SVD) - з англ. Singular Value Decomposition. Теорема про сингулярний розклад стверджує, що у будь-якої матриці *A* розміру n на m існує розклад в добуток трьох матриць: *U,* □ та V^t :

$$\underset{n \times m}{A} = \underset{n \times n}{U} \times \underset{n \times m}{\Sigma} \times \underset{m \times m}{V^T}$$

Матриці U і V - ортогональні, а □ - діагональна (хоч і не квадратна).

$$UU^T = I_n, VV^T = I_m$$

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

Лямбди в формулі розташовані за спаданням. Доведення теореми буде опущене.

Окрім звичайного розкладання існує ще усічене, коли ми з усіх лямбд залишаємо лише перші ***d*** чисел, а інші вважаємо рівними нулю.

$$\lambda_1, \dots, \lambda_{\min(n,m):=0}$$

Виявляється, що на практиці нова матриця A’ дуже добре наближає вихідну матрицю A та, тим більше, є найкращим наближенням. [7]

Спростимо трохи отриману формулу, вважаючи добуток перших двох матриць за одну:

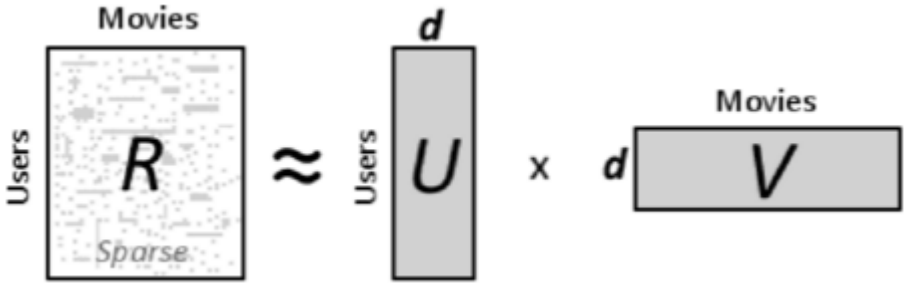


Рисунок 2.2 - Схематичне зображення SVD розкладу

Тобто отримано наступний алгоритм: щоб передбачити оцінку користувача *U* для фільму *V*, ми беремо деякий вектор ***p*** (набір параметрів

користувача) та вектор $\boldsymbol{q}$ (набір параметрів фільму). Їх скалярний добуток і буде необхідним прогнозування.

Наприклад, у векторі користувача на першому місці буде стояти параметр, який відповідає за стать користувача (хлопчик чи дівчинка), а на другому - його вік. У фільмів на аналогічному першому місці буде стояти параметр, який вказує на те, чи цей фільм подобається більше хлопчикам/дівчатам, а інший - для якої вікової категорії цей фільм більше підходить. Тому ми бачимо, що цей алгоритм дозволяє не тільки прогнозувати оцінки. Також ми можемо прогнозувати скриті інтереси користувачів та параметри об’єктів.

Але все виявляється не так просто. По-перше, нам не відома повністю матриця оцінок $\boldsymbol{R}$, а, по-друге, розклад матриці на добуток не єдиний, тому не факт, що на першому місці вектора $\boldsymbol{p}$ буде стояти параметр, що відповідає за стать.

2.3 Вибір архітектурного стилю РС

Архітектурний стиль - це шаблон програмного забезпечення, що являє собою звіт «добрих практик» (good practices) вирішення архітектурних проблем розробки програмного забезпечення. Архітектурні шаблони виражають фундаментальну схему структурної організації певної програмної системи. Така схема складається із визначених наперед підсистем, а також точно визначає їхні сфери відповідальності та взаємовідносини.

На мій погляд реалізація РС в контексті поставленої здачі може бути втілена в *монолітному*, *мікросервісному* чи *серверлес* (Serverless) стилях.

2.3.1 Монолітний стиль

Монолітний — архітектурний стиль за якого єдиний застосунок будується як сукупність невеличких модулів кожен з яких працює у спільному процесі і комунікує з рештою на пряму, не використовуючи додаткових

сервісів. Все реалізовується на основі бази одного коду. Також деплой відбувається одночасно всього проекту.

Рисунок 2.3 - Візуалізація монолітного стилю архітектури

Переваги:

- Можливість швидко деплоїти проект, так, як все знаходиться в одному місці.
- Легко повторно використовувати код.

Недоліки:

- При потребі редеплою якогось з модумів, потрібно перезапускати весь проект.
- Важко замінити окремий модуль через його глибоку інтеграцію у вже існуючий проект.
- Відсутність можливості комбінувати різні, сумісні та не сумісні технології чи їх версії в одному проекті.
- Відсутність можливості збільшувати ресурсоємність конкретного модуля.

2.3.2 Мікросервісний стиль

Мікросервіси — архітектурний стиль за якого єдиний застосунок будується як сукупність невеличких сервісів кожен з яких працює у своєму власному процесі і комунікує з рештою використовуючи легковагові механізми, зазвичай HTTP. Ці сервіси будуються навколо бізнес-потреб і розгортаються незалежно з використанням зазвичай повністю автоматизованого середовища. Існує абсолютний мінімум централізованого керування цими сервісами. Самі по собі вони можуть бути написані з використанням різних мов і технологій зберігання даних.

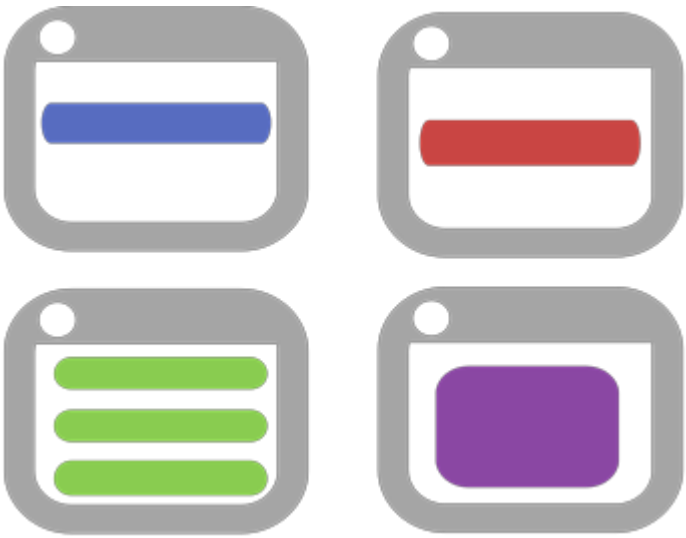


Рисунок. 2.4 - Візуалізація мікросервісного стилю архітектури

Основні властивості:

- Високий рівень незалежності.
- Простота заміни однієї реалізації сервісу іншою.
- Сервіси організовані відносно бізнес логіки яку вони виконують.
- Кожен сервіс незалежно від інших може бути реалізований за допомогою будь-якої мови програмування, СБД, та ін.

Філософія мікросервісного підходу схожа на філософію Unix «Роби одну річ і роби її якісно». Сервіси невеликі, розбиті на виконання єдиної функції.

Організаційна культура повинна охоплювати автоматизацію розгортання та тестування. Культура і принципи проектування повинні охоплювати опрацювання відмов і дефектів. Кожен сервіс гнучкий, стійкий до відмов, має легку здатність компонутися з іншими сервісами, функціонально мінімальний та закінчений.[5]

Переваги:

- Перевагами мікросервісного стилю є, вже вище перелічені недоліки монолітного.

Недоліки:

- Проблемність в оркеструванні

2.3.3 Серверлес стиль

Серверлес — архітектурний стиль за якого єдиний застосунок будується як сукупність невеличких сервісів кожен з яких представлений у вигляді функції, працює у своєму власному процесі і комунікує з рештою використовуючи легковагові механізми, зазвичай HTTP.

Кожна з функцій представлена у вигляді коду, і тільки за умови звернення до неї ініціалізується середовище, виконується код та повертається результат. Таким чином необхідність у постійній підтримці сервера відсутня, що здешевлює підтримку не часто вживаного функціоналу чи то проектів.

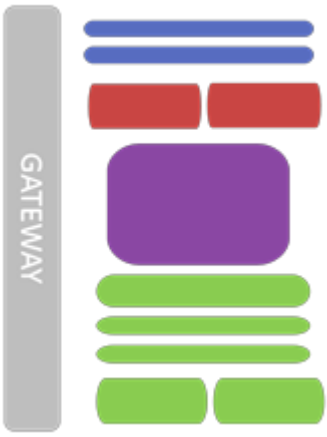


Рисунок 2.5 - Візуалізація серверлес стилю архітектури

Переваги:

- Велика гранулярність, що дозволяє легко заміняти/змінювати частини проекту.
- Дешевизна серверів.
- Легко масштабуються.

Недоліки:

- Велика гранулярність, через яку, при деплої проекту виникає проблема запуску та налаштування величезної кількості функцій.
- Важкість в оркеструванні.

2.3.3 Порівняльна характеристика

Проаналізувавши запропоновані стилі, їх недоліки та переваги. Також врахувавши вимоги до задачі приходимо до висновку, що найбільш оптимальним є *серверлес* стиль архітектури. Його переваги дозволяють нам:

- Йти в ногу з новими технологіями.
- Отримати фінансово вигоду.
- Зручно керувати проектом, налагодженням та деплоєм.
- Легко масштабувати РС.

2.4 Розробка архітектурної схеми

Для реалізації РС в архітектурному стилі серверлес нам необхідно наступні архітектурні компоненти:

- **Сховище даних** - це швидкий, повністю керований, масштабоване сховища даних, що робить його простим і економічно ефективним, щоб проаналізувати всі дані, використовуючи існуючі інструменти бізнес аналітики.
- **Швидкісна шина** - це сервіс, що допомагає надійно і з заданими інтервалами обробляти дані і переміщати їх між різними

обчислювальними сервісами і сервісами сховищ, а також локальними джерелами даних.

- **MapReduce кластер** - надає керовану інфраструктуру Hadoop чи іншу (Apache Spark, HBase, Presto і Flink), яка здатна ефективно, швидко та економічно обробляти великі обсяги даних на динамічно масштабованих серверах.
- **Нереляційна БД** - база даних, що забезпечує механізм для зберігання та видобування даних, що організовані в інакший спосіб, ніж звичний підхід таблиць-відношень в реляційних базах даних.
- **FaaS** (англ. Function as a Service) - категорія послуг хмарних обчислень, яка забезпечує платформу, що дозволяє розробляти, запускати і керувати функціональними можливостями програми без складності будівництва і підтримки інфраструктури, що пов'язані з розробкою і запуском програми.

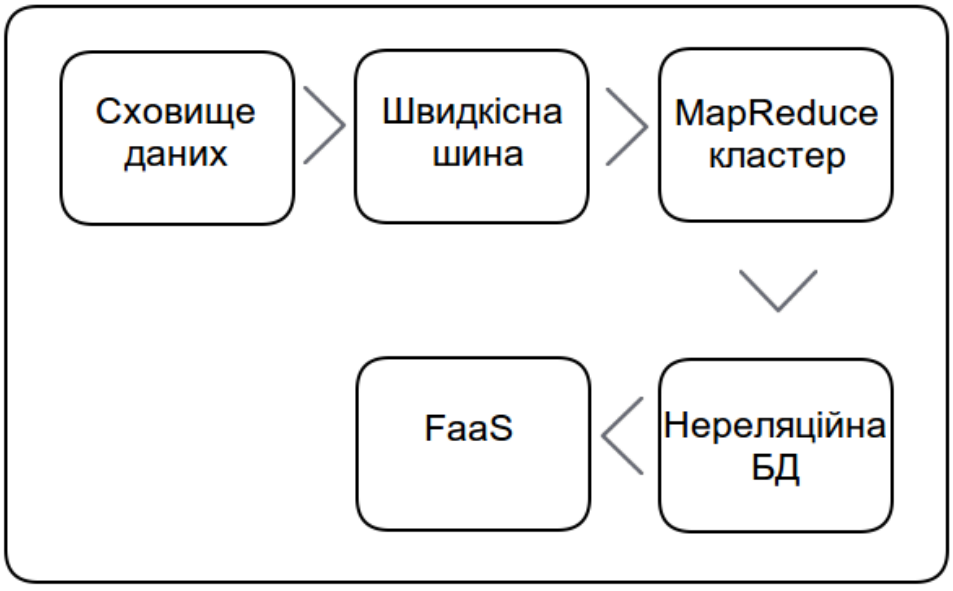


Рисунок 2.6 - Архітектурна схема РС

На першому кроці дані збираються в сховищі. Далі в перед обрахуваннями за допомогою швидкісної шини потрапляють на MapReduce кластер, другий крок. Обробка на кластері відбувається періодично, (як правило

раз в добу), крок третій. Транспортування є необхідним тому, що вартість кластера значно вища за вартість сховища даних з транспортуванням. На четвертому, після проведений обчислень результати записуються в нереляційну БД. Нереляційна БД дозволяє отримати можливість високочастотного читання у великій БД. На п’ятому кроці ми підключаємо FaaS для надання результатів роботи РС по запиту від користувача.

2.5 Висновки

Даний розділ присвячено теоретичній складовій мого рішення. А саме мені вдалось побудувати всю теоретичну складову РС, включаючи тип архітектури, метод РС, та алгоритм. Провести теоретичні дослідження, надати порівняльні характеристики та зробити відповідні висновки.

Я проаналізував та дослідив методи реалізації РС: фільтрація вмісту, колаборативна фільтрація, гібридна система. Прийшов до висновку, що в сучасних реаліях та особливостями поставлених вимог раціональним є використання методу колаборативної фільтрації. Важливий вплив на це рішення внесла особливість вхідних даних, дії користувачів по відношенню до продуктів (елементів).

Після пошуку до розгляду взяв алгоритми ALS та SVD. Їх порівняння показало, що в практичних умовах доцільно використовувати ALS.

Останнє, а можливо й найважливіше рішення прийняте в цьому розділі щодо вибору типу архітектури. Незважаючи на широку популярність та зручність в розробці монолітної архітектури, гнучкість мікросервісної я все ж зосередився на серверлес.

3 ВИБІР ІНСТРУМЕНТІВ ВИРІШЕННЯ

3.1 Вибір бібліотеки для алгоритму

Для реалізації алгоритму КФ необхідно проаналізувати існуючі на ринку рішення та прийти до оптимального вибору. В даному розділі ми проаналізуємо існуючі бібліотеки, що дозволяють виконувати КФ на велики даних.

3.1.1 Spark

Apache Spark - це швидкий і універсальний кластер обчислювальної системи. Він надає API-інтерфейси високого рівня на Java, Scala, Python і R, а також є оптимізованим двигуном, який підтримує загальні графіки виконання. Він також підтримує багатий набір інструментів вищого рівня, в тому числі Spark SQL для SQL і структурованої обробки даних, MLlib для машинного навчання, Graphx для обробки графів, і Spark Streaming.

На високому рівні, кожен додаток Spark складається з керуючої програми, яка запускає основну функцію користувача і виконує різні паралельні операції в кластері. Основна абстракція Spark являє собою пружний розподілений набір даних (ПРНД), який в свою чергу являє собою сукупність елементів розподілених між вузлами кластера, що можуть працювати паралельно. ПРНД створюється починаючи з файлу в файлової системі Hadoop (або будь-який інший Hadoop підтримуваній файловій системі), або в колекціях Scala, і перетворюючи їх. Користувачі також можуть попросити Spark зберігати ПРНД в пам'яті, що дозволяє ефективне повторне використання у паралельних операціях. Також ПРНД має властивість автоматичного відновлення після збоїв.

Другою абстракцією в Spark є колективні змінні, що можуть бути використані в паралельних операціях. За замовчуванням, коли Spark запускає функцію паралельно, як набір завдань на різних вузлах, він поставляє копію

кожної змінної, що використовується в функції для кожного завдання. Іноді, змінна повинна бути розділена між завданнями програмою драйвера. Spark підтримує два типи колективних змінних: *широкомовні* змінні, які можуть бути використані для кешування значення в пам'яті на всіх вузлах, і *акумулятори*, які є змінними типу лічильники.[9]

З компаній, котрі використовують Spark, відзначаються Alibaba, Cloudera, Databricks, IBM, Intel, Yahoo, Cisco Systems.

У жовтні 2014 року Apache Spark встановив світовий рекорд при сортуванні 100 терабайт даних.[10]

Згідно опитування O'Reilly у 2015 році 17% дослідників даних використовують Apache Spark.

3.1.2 Hadoob

Hadoop є відкритим вихідним кодом Apache написаний на Java, що дозволяє розподілену обробку великих масивів даних по кластерах комп'ютерів за допомогою простих моделей програмування. Hadoop додаток працює в середовищі, яка забезпечує розподілене зберігання і обчислення по кластерах комп'ютерів. Hadoop призначений для масштабування в з одного сервера на тисячі машин, кожен з яких пропонує локальні обчислення і зберігання.

Фреймворк Hadoop включає в себе наступні чотири модулі:

- Hadoop Common
- Hadoop YARN
- Hadoop Distributed File System (HDFS ™)
- Hadoop MapReduce

Ми можемо використовувати наступну діаграму, щоб зобразити ці чотири компоненти, доступні в рамках Hadoop.

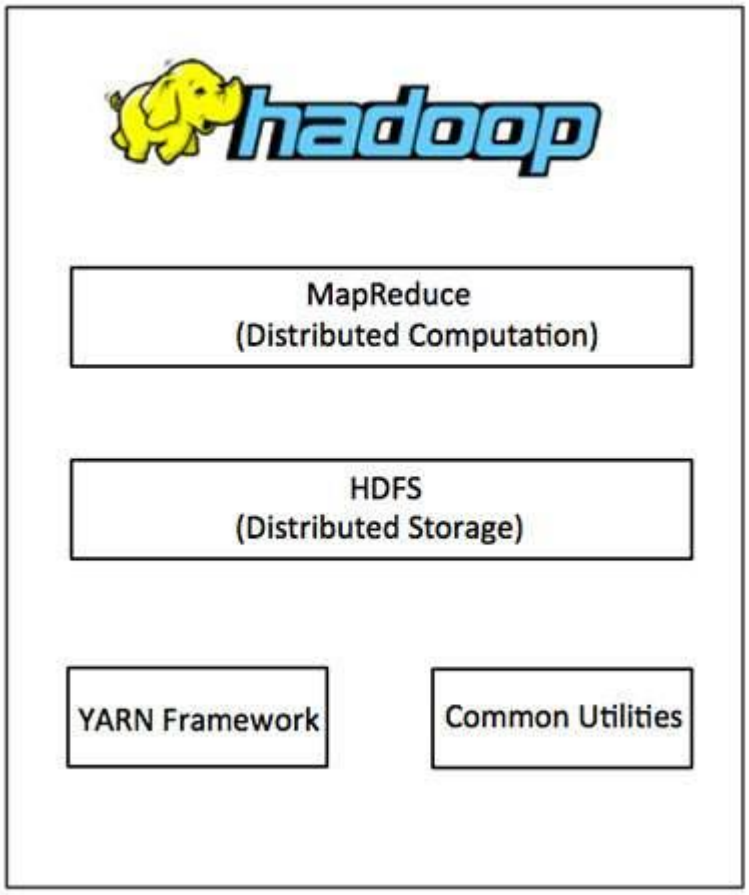


Рисунок 3.1 - Схематична модель Hadoop

На даний момент нас цікавить останні з них. Hadoop MapReduce є основою програмного забезпечення для створення ПЗ, що обробляють великі обсяги даних паралельно, на великих кластерах (тисяч вузлів), із забезпеченням відмовостійкості.

Термін MapReduce відноситься до таких двох завдань, які програми виконують на Hadoop:

- Map задача: це перше завдання, яке приймає вхідні дані і перетворює його в набір даних, де окремі елементи розбиті на кортежі (ключ / значення).
- Reduce задача: це завдання вимагає вихідних даних із завдання task в якості вхідних даних і об'єднує ці кортежі даних в менший набір кортежів. Завдання зменшення завжди виконується після завдання task.

Зазвичай і вхідні дані і вихідні зберігаються в файловій системі. Фреймворк піклується про заплановані завдання, їх моніторинг і повторне виконання невдалих завдання.

Структура MapReduce складається з одного ведучого JobTracker і одного відомого TaskTracker на вузлі кластера. Майстер несе відповідальність за управління ресурсами, відстеження споживання ресурсів / доступності та планування завдань, компонентів для підлеглих, їх моніторинг і повторного виконання невдалих завдань. Підлеглі TaskTracker виконують завдання в відповідності до вказівок майстра і періодично надають інформацію про стан завдання до майстра.

JobTracker є єдиною точкою відмови для служби Hadoop MapReduce, що означає, якщо JobTracker припиняє роботу, то всі запущені задачі зупиняються.[11]

3.2 Вибір платформи реалізації

На даний момент на ринку є невелика кількість компаній, що пропонують хмарні рішення як сервіс. Розглянемо найбільш поширені, особливо варто звернути увагу на потрібні нам елементи та їх особливості.

3.2.1 Google Cloud Platform (GCP)

GCP - це сервіс хмарних обчислень від Google, який пропонує хостинг на тій же інфраструктурі, що Google використовує внутрішньо для кінцевих користувачів таких продуктів, як Google Search і YouTube.

Кількість доступних регіонів - 6.

Дискретність часу при оплаті - хвилини.

Таблиця 3.1 - Підбір GCP відповідників

Назва елемента	GCP відповідник	Посилання
Сховище даних	BigQuery	https://cloud.google.com/bigquery/
Швидкісна шина	Dataflow	https://cloud.google.com/dataflow/
MapReduce кластер	Dataproc	https://cloud.google.com/dataproc/
Нереляційна БД	Datastore	https://cloud.google.com/datastore/
FaaS	Functions	https://cloud.google.com/functions/

Як бачимо у GCP сервісу є всі необхідні компоненти. На даний момент сервіс *Functions* знаходиться на стадії *альфа*.

3.2.2 Amazon Web Services (AWS)

AWS — структура онлайнових хмарних сервісів, що розробляється компанією Amazon з 2006 року.

Кількість доступних регіонів - 16.

Дискретність часу при оплаті - години.

Таблиця. 3.2 - Підбір AWS відповідників

Назва елемента	AWS відповідник	Посилання
Сховище даних	Redshift	https://aws.amazon.com/redshift/
Швидкісна шина	Data Pipeline	https://aws.amazon.com/datapipeline/
MapReduce кластер	EMR	https://aws.amazon.com/emr/
Нереляційна БД	DynamoDB	https://aws.amazon.com/dynamodb/
FaaS	Lambda	https://aws.amazon.com/lambda/

Як бачимо у AWS сервісу є всі необхідні компоненти. Також потрібно відзначити зручність панелі користувача.

3.2.3 Microsoft Azure (MA)

MA - Це хмарна платформа та інфраструктура корпорації Microsoft, призначена для розробників застосунків хмарних обчислень (англ. cloud computing) і покликана спростити процес створення онлайнових застосунків.

http://www.tomsitpro.com/articles/azure-vs-aws-cloud-comparison,2-870-2.html

3.3 Архітектурна схема в технологіях Amazon

У зв'язку з вибором платформи AWS для реалізації потрібно перенести вище розроблену архітектуру на відповідну компонентну базу.

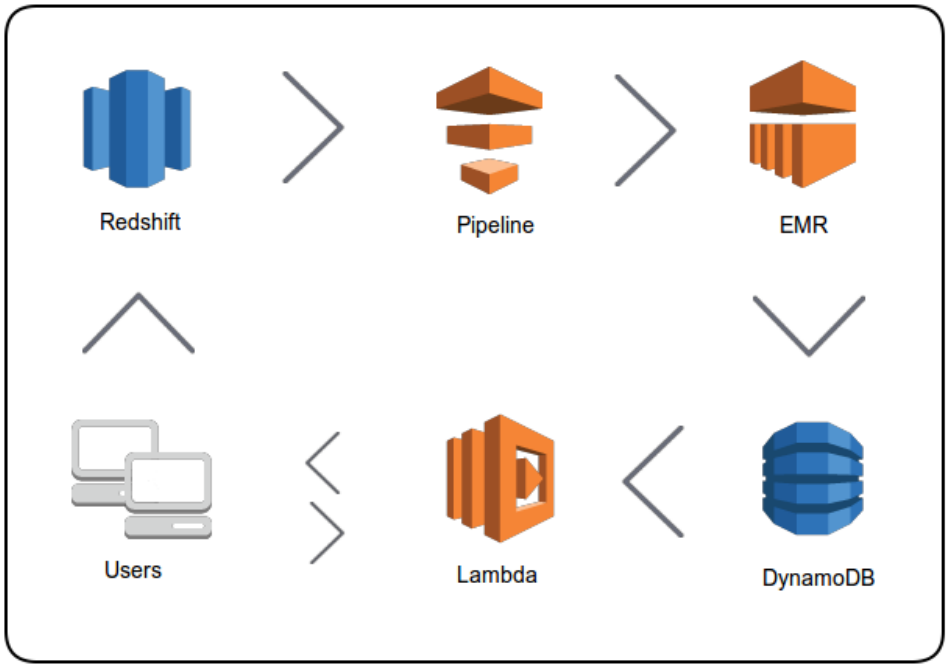


Рисунок 3.2 - Архітектурна схема РС на базі компонентів AWS

3.4 Висновки

В даному розділі я дослідив питання вибору бібліотеки для реалізації РС, провів аналіз та прийшов до висновку, що бібліотекою, яка відповідає нашим вимогам, легка у використанні, здатна легко масштабуватись є Spark. Також дослідив необхідну для мене реалізацію ALS декомпозиції.

Хмарні технології набули великої поширеності у сучасному світі. Я проаналізував сервіси Amazon AWS, Google Cloud, Microsoft Azure, провів порівняння, моїм фаворитом став Amazon AWS. Також побудував відповідник, до розробленої раніше архітектури на AWS компонентах.

4 ПРАКТИЧНЕ РІШЕННЯ

4.1 Побудова та налаштовування середовища

Обраний нами стек технологій для реалізації РС в архітектурному стилі серверлес - AWS Amazon. Метафорично це великий та сучасний космічний корабель, а космічні кораблі, особливо їх налаштовування не тривіальна річ.

4.1.1 Як побудований AWS Amazon

Region. Одним із базових понять є *регіон*, їх налічується 14. Це локації серверних, що розміщені на різних континентах для забезпечення швидкої доступності до ресурсів. Важливо обрати правильний регіон, найбільш вигідним для України є *eu-central-1*, що розташований у Франкфурті, Німеччина. Для вибору *регіону* заходимо в консоль розробника. З права в горі у випадаючому списку обираємо Frankfurt, по замовчуванню Oregon.

Security Group. Для забезпечення налаштувань доступності існує поняття *група безпеки*. Її об’єкт створюється практично при ініціалізації будь якого сервісу.

Virtual Private Cloud (VPC). Це логічно ізольований розділ хмари AWS, в якому можна запускати ресурси AWS в побудованій вами віртуальної мережі. Таким чином можна повністю контролювати середовище віртуальної мережі, в тому числі вибирати власний діапазон IP-адрес, створювати підмережі, а також налаштовувати таблиці маршрутизації і мережеві шлюзи. Для забезпечення надійного і зручного доступу до ресурсів і додатків в VPC можна використовувати як IPv4, так і IPv6.

Для переходу між сервісами з ліва в горі, натиснувши на розділ Services, в пошуку вводимо потрібну нам назву, до прикладу “Lambda”та обираємо з запропонованих нам сервісів. Таким чином ми потрапляємо в ту частину

консолі розробника, яка відноситься тільки до обраного сервісу та всі налаштування проходитимуть тут.

Для того щоб створити один з сервісів AWS нам потрібно обдумати де його використовуватимуть, тобто обрати *регіон*. Також відповісти на запитання хто матиме право доступу до цього ресурсі та звідки, відповідно налаштувавши *групу безпеки* і *VPC*.

4.1.2 Аккаунт AWS

Для створення акаунта переходимо за посиланням https://aws.amazon.com, покроково слідуємо та заповняємо відповідні поля. Нас попросять прив'язати банківську карту, з якої по мірі використання ресурсів зніматимуться кошти. Якщо ви вперше використовуєте AWS, то отримуєте безкоштовне використання сервісів, з певними обмеженнями, на період 12-ти місяців. В цей пакет входить:

- 750 годин/місяць серверного часу (EC2)
- 5 Гб на сховищі S3
- 750 годин/місяць використання бази даних (RDS)
- 1 млн. запитів/місяць для серверлес (Lambda)
- 1 Гб для аналітичного застосунку (QuickSight)

4.1.3 Lambda

На даний момент лямбда потрібна для періодичного запуску обчислень та діставання результатів з БД.

В консолі розробника переходимо в розділ "AWS Lambda" та натискаємо кнопку "Get Started Now". Із запропонованих нам шаблонів обираємо "Blank Function". Далі "Next". Серед запропонованих полів вводимо назву (Name), середовище (Runtime) обираємо Python 2.7, вказуємо роль (Role), обираємо створити нову (Create new role from template), де (Policy templates) обираємо (AMI read-only permissions). Двічі нажимаємо далі "Next".

Для проміжного тестування нажимаємо кнопку “Test”. За умови отримання негативного результату ретельніше повторюємо вище перелічені кроки. За умовив успішного тестування йдемо далі.

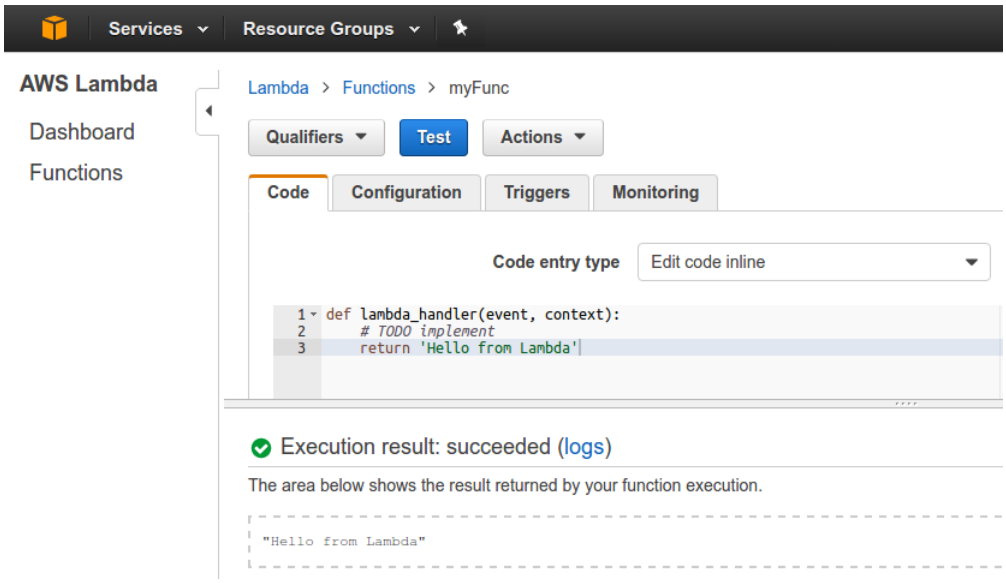


Рисунок 4.1 - Вид лямбда-функції на панелі користувача

Для можливості доступу із зовні нам потрібно налаштувати API GateWay. Переходимо у відповідний розділ консолі, тиснемо кнопку “GetStarted”. З опцій “New API”, далі вказуємо ім’я. Тиснемо кнопку “Create”. В підрозділі “Resources” обираємо “Action”, “Create Method”. Для прикладу підійде GET. Вказуємо регіон, такий же, в якому створена наша лямбда-функція. Обираємо саму функцію. Наступний кроком нам потрібно зробити це API публічним. Для цього в меню “Action” обираємо “Deploy API”. Заповнивши необхідні поля. Переходимо за автоматично згенеруваним посиланням “Invoke URL”. Очікувана відповідь "Hello from Lambda".

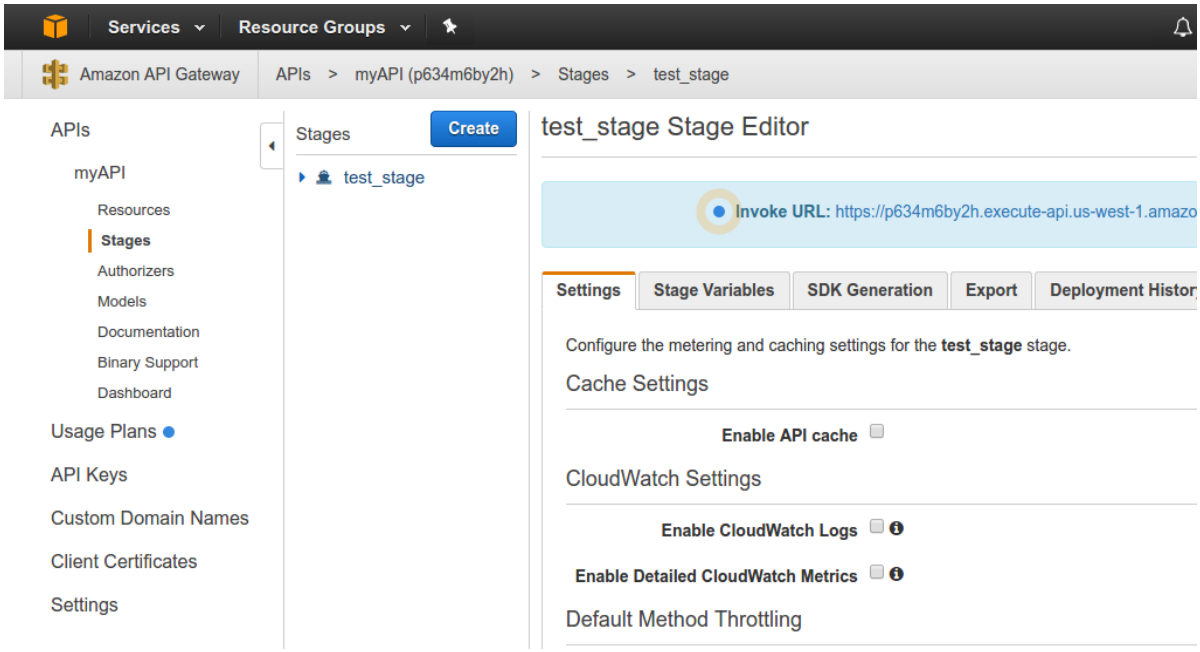


Рисунок 4.2 - API GateWay на панелі користувача

На даний момент пророблено шлях створення та налаштування лямбда-функції. Також підлкючено та налаштовано API GateWay.

Для додаткового налаштування та можливості створення EMR сервера з коду лямбда-функції необхідно додати відповіді права, описані в Додатку А.

4.1.4 Amazon S3

На даний момент S3 використовуєтеся для зберігання виконуваних скриптів для EMR, для зберігання голів.

Для налаштування переходимо в розділ S3 консолі розробника та нажимаємо кнопку “Create Bucket”. Особливістю S3 сховище є відсутність приналежності до регіону та необхідність унікального імені.

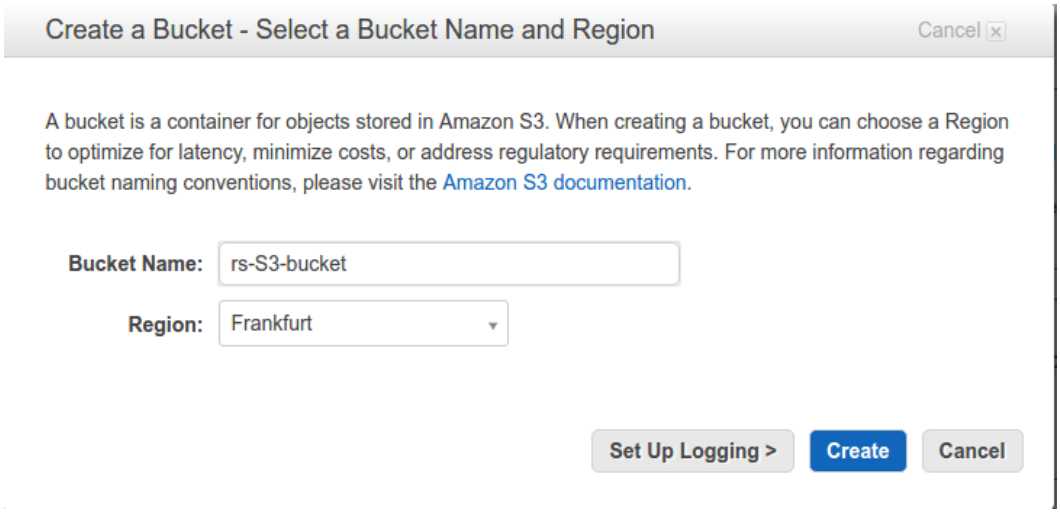


Рисунок 4.3 - Вікно створення S3

Після створення нам доступні такі операції, як створення папки, завантаження файлу. Їх можна проводити, як і в ручному, через веб консоль, так і програмно, за допомогою коду.

4.1.5 EMR

EMR ми використовуємо для обчислень над великими даними. Варто зазначити, що EMR являє собою групу серверів об'єднаних між собою. Що особливо вирізняється характеристиками: об'ємом оперативної, фізичної пам'яті, процесорною потужністю. Все формується комбінацією EC2 серверів. Виконання задач на EMR представлене у вигляді кроків "step". За допомогою яких ми формуємо необхідну для нас послідовність дій. Для створення потрібно перейти у розділ EMR консолі розробника та натиснути кнопку "Create Cluster". Далі ми потрапляємо на вікно створення.

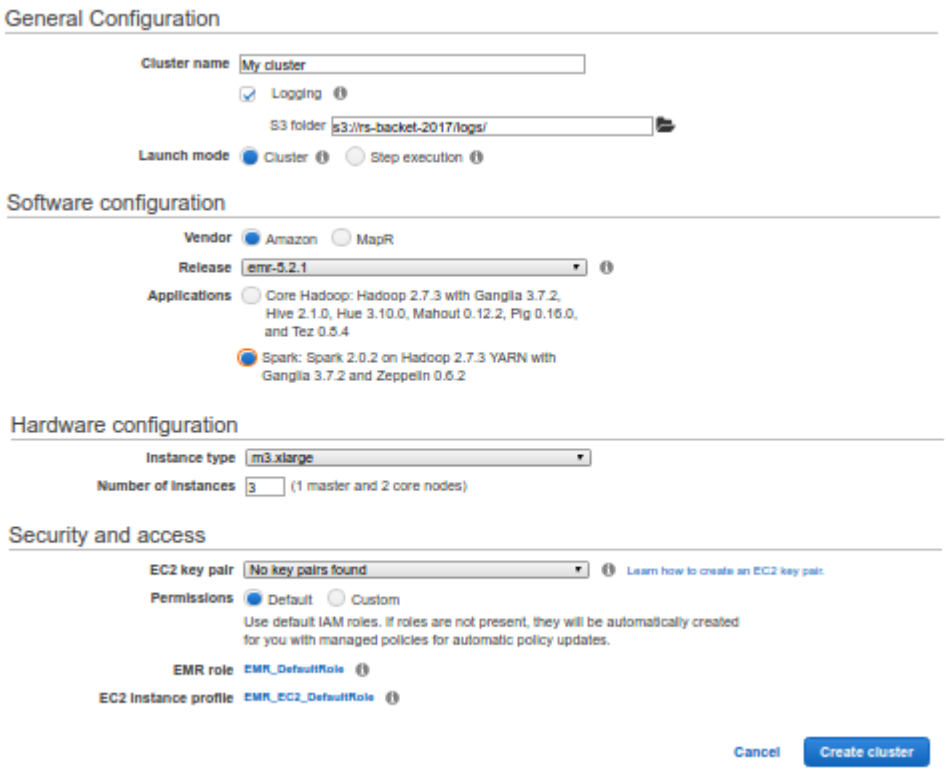


Рисунок 4.4 - Вікно створення EMR в консолі користувача.

Вказуємо шлях до створеного нами bucket в S3, попередньо створивши папку logs. В розділі “Software configuration” обираємо Spark. За потреби доступу до сервера через SSH у розділі “Security and access” створюємо пару ключів. Решту залишаємо без змін. Нажавши на кнопку “Create cluster” відбудеться ініціалізація. Це найлегший спосіб створення, але для нашої задачі, для більшої зручності використаємо програмний. Ідея полягає в тому, що ми ініціалізуємо сервер EMR з потрібними нам конфігураціями, завантажимо на нього відповідні скрипти, а також додамо кроки “step” для обчислень. Для цього використаємо бібліотеку boto3. Необхідний функціонал виконуватиметься на лямбда функції, що у свою чергу викликатиметься періодично, раз в добу. Код лямбда-функції представлений в Додатку Б. Для запуску програми рекомендацій, потрібно попередньо підготувати середовище, що робиться за допомогою *sh* скрипта, представленого в Додатку В.

4.2 Програмна реалізація

У попередніх розділах ми прийшли до висновку про реалізацію РС використовуючи алгоритм колаборативної фільтрації, за допомогою бібліотеки Spark з її програмною обгорткою pySpark. Потрібні нам методи знаходяться в бібліотеці Machine Learning Library (MLlib). Для попередньої обробки та трансформації даних використаємо бібліотеку *pandas*.

У зв'язку з тим, що кожен запуск на EMR несе за собою стягування коштів, тому доцільно відлагодити роботу скрипту на локальній машині. Код представлений в Додатку Г, а вже потім тестувати на сервері.

4.2.1 MLlib (Machine Learning Library)

MLlib є дистрибутивним фреймворком з машинного навчання на базі Spark, що в значній мірі є більш ніж, як в дев'ять разів швидше, ніж реалізації на основі жорстких дисків, що використовуються Apache Mahout. [13] В ньому об'єднуються поширені алгоритми машинного навчання і статистичні, що спрощує навчання навчання та розробку складніших алгоритмів та структур.

Найбільш важливим, у контексті даної задачі є метод побудови рекомендації на базі ALS факторизації матриці, *trainImplicit* - тренує факторизаційну модель, на основі неявних даних користувачів по відношенню до множини продуктів. Для забезпечення цього ALS рухається ітеративно з певним рівнем паралелізму.

Параметри методу *trainImplicit*:

- ratings – RDD рейтингу або триплет (userID, productID, rating).
- rank – ранг матриці особливостей, кількість особливостей.
- iterations – кількість ітерацій для ALS. (по замовчуванню: 5)
- lambda – регулюючий параметр. Один із гіпер параметрів. (по замовчуванню: 0.01)

- blocks – Кількість блоків використаних для паралельних обчислень. Значення -1 використовується, як автоматичний підбір (по замовчуванню: -1)
- alpha – Константа, що використовується при обчисленні довіри. Один із гіпер параметрів. (по замовчуванню: 0.01)
- nonnegative – Значення True вирішиться методом найменших квадратів з невід'ємним обмеженням. (по замовчуванню: False)
- seed – Випадкове зерно для ініціалізації матричної факторизації. [12] Значення None буде сприйняте, як те, щоб використовувати системний час, як зерно. (по замовчуванню: None)

4.2.2 Pandas

Pandas — вільна бібліотека Python, створена для проведення зручних маніпуляцій з даними та аналізу. Зокрема, дана бібліотека пропонує можливості роботи з числовими таблицями та часовими рядами. Назву бібліотека отримала від терміну з економетрії «panel data» — багатовимірного струкрутованого набору данних.

Pandos ми використаємо для попередньої підготовки та обробки даних.

4.3 Висновки

В даному розділі я зібрав докупи напрацьовані вище результати та реалізував архітектурну складову РС на базі AWS компонентів у серверлес стилі. Під час розробки було використано наступні компоненти: S3, EMR, Lambda, Pipeline, DynamoDB. Варто відзначити велику кількість особливостей налаштування AWS, що зв'язані з широкими можливостями хмарного сервісу AWS.

Реалізовано алгоритмічну складову на базі pySpark - пітоніської обгортки бібліотеки Spark.

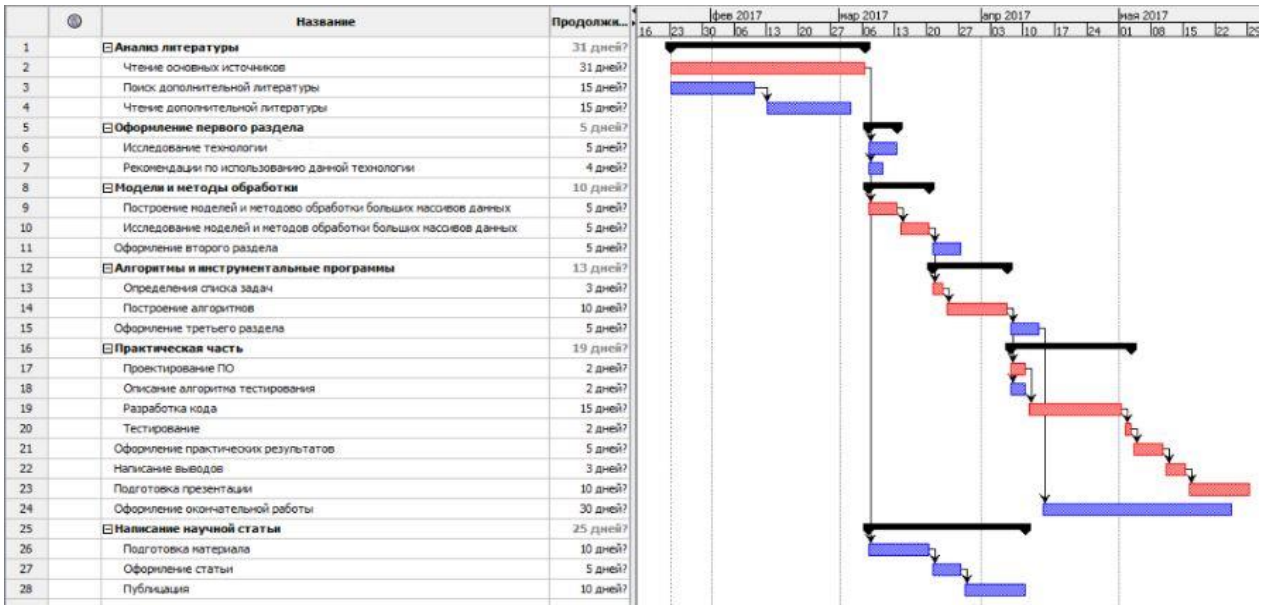
5 УПРАВЛІННЯ ТЕРМІНАМИ

5.1 Діаграма Ганта

Діаграма Ганта - це популярний тип діаграм, який використовується для ілюстрації плану, графіка робіт за будь-яким проектом. Є одним з методів планування та управління проектами. Перший формат діаграми був розроблений Генрі Л. Гантом у 1910 році [8].

Діаграма Ганта являє собою відрізки (графічні плашки), розміщені на горизонтальній шкалі часу. Кожен відрізок відповідає окремому завданню або підзадачі. Завдання і підзадачі, складові плану, розміщуються по вертикалі. Початок, кінець і довжина відрізка на шкалі часу відповідають початку, кінцю і тривалості завдання.

Діаграма може використовуватися для представлення поточного стану виконання робіт: частина прямокутника, що відповідає завданню, заштриховується, відзначаючи відсоток виконання завдання; показується вертикальна лінія, що відповідає моменту «сьогодні» (рис 6.1).



	Название	Продолжи...
1	**Анализ литературы**	31 дней?
2	Чтение основных источников	31 дней?
3	Поиск дополнительной литературы	15 дней?
4	Чтение дополнительной литературы	15 дней?
5	**Оформление первого раздела**	5 дней?
6	Исследование технологии	5 дней?
7	Рекомендации по использованию данной технологии	4 дней?
8	**Модели и методы обработки**	10 дней?
9	Построение моделей и методово обработки больших массивов данных	5 дней?
10	Исследование моделей и методов обработки больших массивов данных	5 дней?
11	Оформление второго раздела	5 дней?
12	**Алгоритмы и инструментальные программы**	13 дней?
13	Определения списка задач	3 дней?
14	Построение алгоритмов	10 дней?
15	Оформление третьего раздела	5 дней?
16	**Практическая часть**	19 дней?
17	Проектирование ПО	2 дней?
18	Описание алгоритма тестирования	2 дней?
19	Разработка кода	15 дней?
20	Тестирование	2 дней?
21	Оформление практических результатов	5 дней?
22	Написание выводов	3 дней?
23	Подготовка презентации	10 дней?
24	Оформление окончательной работы	30 дней?
25	**Написание научной статьи**	25 дней?
26	Подготовка материала	10 дней?
27	Оформление статьи	5 дней?
28	Публикация	10 дней?

Рисунок 5.1 - Діаграма Гантта

5.2 Критичний шлях

Критичний шлях і час виконання (песимістичний - реалістичний — оптимістичний):

1. Читання літературних джерел - 31 - 25 - 20 днів
2. Аналіз предметної області - 30 - 25 - 20 днів
3. Вивчення існуючих робіт - 6 - 5 - 2 дні
4. Створення моделі алгоритму розв'язання задачі 23 - 20 - 15 днів
5. Проектування програмного рішення - 23 - 15 - 13 днів
6. Розробка коду - 18 - 11 - 9 днів
7. Тестування - 6 - 5 - 2 дні

Разом 112 днів (реалістична оцінка)

Відповідно до таблиці 6.1, з ймовірністю 0.4443 проект буде завершений за 112 днів.

Таблиця. 5.1 – Розрахунок ймовірності виконання проекту

№	Песимістич	Реально	Оптимістично	Дисперсія	Відхилення	Те
1	31	25	20	37.5	6.123724	31.83
2	30	25	20	25	5	25
3	6	5	2	4.3333	2.081666	4.6
4	22	19	14	16.3333	4.04	19.66
5	20	14	12	28	5.291503	16
6	13	10	9	22.33333	4.7258	11.83
7	6	5	2	4.333333	2.08166	4.667
Сума	127	96	87	137.832	22.54625	113.67
		Z	-0.14196			
		F(Z)	0.543			

5.3 Управління ризиками

Управління ризиками - це процес прийняття та виконання управлінських рішень, спрямованих на зниження ймовірності виникнення несприятливого результату і мінімізацію можливих втрат, викликаних його реалізацією. В рамках управління ризиками здійснюється кількісна та якісна оцінка ймовірності досягнення передбачуваного результату, невдачі і відхилення від мети.

Ризик являє собою можливу небезпеку несприятливого результату. Поняття ризику поєднує в собі оцінку ймовірності і наслідки настання несприятливої події.

У ситуації ризику відповідальна особа опиняється перед необхідністю розробки альтернативних варіантів рішення і подальшого вибору найбільш прийнятного з них. При цьому якщо дієслово "ризикувати" асоціюється з діями всупереч існуючим небезпекам, зневагою до них, то управління ризиком передбачає аналіз причин, джерел та факторів ризику, реалістичну оцінку небезпеки на шляху до наміченої мети, оцінку ефективності різних методів ризик-менеджменту і в той же час відхід від непотрібного ризику і невиправданих втрат. Вибір варіанту, максимально знижує ризик, часто веде до невисоких результатів.

Результати проведення класифікації ризиків структуризовано та продемонстровано в таблиці 6.2.

Таблиця 5.2 – Класифікація ризиків

Ризик	Заходи	Імовірн.	Вплив
Несподівана зміна умов використання використовуваного в роботі ПО	Розгляд альтернативних програмних рішень для виконання роботи заздалегідь	низька	сильний

Таблиця 5.2 (продовження)

Несподівана втрата продуктивності і ефективності в силу озчарування тим що нічого не працює.	Розгляд можливих поширених проблем Oracle SOA Suite, детальне вивчення документації.	висока	сильний
Вихід з ладу основного домену розгортки веб-сервісів - localhost	Збереження проміжних результатів, резервне розгортання на менш надійних ресурсах (Google App Engine, etc)	середня	сильний
Невиконання робіт в термін через несподівані додаткових завдань.	Більш якісне планування робіт по виконанню диплома.	середня	сильний
Труднощі в розумінні архітектури майбутньої системи і того як все це повинно бути влаштовано	Витратити додатковий час на вивчення документації і принципів роботи.	середня	середній
Неповний обсяг необхідної інформації в відібраних джерелах	Витратити додатковий час на вивчення всіх додаткових джерел	низька	середній
Втрата актуальності роботи	Постановка завдання та опис актуальності в універсальному вигляді.	низька	Вище середнього

5.4 Висновок

Методи управління термінами дозволяють краще зрозуміти часові рамки виконання проекту, його важливі задачі, які можуть зупинити процес при неправильному розподілені часу та розрахувати ризики та заздалегідь запобігти їх появі.

ВИСНОВКИ

В даній роботі я ознайомився з поняттям РС. Це певною мірою чорний ящик, що здатний формувати залежності між тим, які люди і що купують. Та на основі цих залежностей можна отримати передбачення цікавого/потрібного продукту для користувача. Усі РС можна поділити на три види: фільтрація вмісту, колаборативна фільтрація та їх поєднання - гібридна. Історично практичне використання РС прийшло у наше життя не так і давно.

На даний момент існує декілька компаній світового рівня, що слугують прикладом ефективного використання РС, серед яких Amazon, Netflix, LinkedIn. Їх ставлять у приклад, використовують, як зразок в різних університетах світу.

Важливе місце виділено під дослідження теоретичної складовій нашого рішення. А саме мені вдалось побудувати схематично РС, включаючи тип архітектури, метод РС, та алгоритм. Прийшов до висновку, що в сучасних реаліях та особливостями поставлених вимог раціональним є використання методу *колаборативної фільтрації*.

Для вибору алгоритму, після пошуку до розгляду обрав ALS та SVD. Їх порівняння показало, що в практичних умовах доцільно використовувати ALS.

Незважаючи на широку популярність та зручність в розробці монолітної архітектури, гнучкість мікросервісної я все ж зосередився на серверлес.

Дослідив питання вибору бібліотеки, провів аналіз та прийшов до висновку, що бібліотекою, яка відповідає нашим вимогам є Spark. Досліджено необхідну для нас реалізацію ALS декомпозиції.

Проаналізовано хмарні сервіси Amazon AWS, Google Cloud, Microsoft Azure, проведено порівняння, моїм фаворитом став Amazon AWS. Для проміжного результату побудував схематичну РС в AWS компонентах.

Кульмінацією стала практична реалізація напрацьованих результатів - реалізація архітектурної складової РС на базі AWS компонентів у серверлес стилі. Під час розробки було використано наступні компоненти AWS: S3, EMR,

Lambda, Pipeline, DynamoDB. Варто відзначити велику кількість особливостей налаштування AWS, що зв’язані з широкими можливостями хмарного сервісу AWS. Алгоритмічну складову реалізовано на базі pySpark - пітоніської обгортки бібліотеки Spark.

Використання методів управління термінами дозволили краще зрозуміти часові рамки виконання проекту, його важливі задачі, які можуть зупинити процес при неправильному розподілені часу та розрахувати ризики та заздалегідь запобігти їх появі.

В даній роботі досліджене питання РС. Сформовано задачу, поставлено відповідні цілі. Знайдено теоретичне рішення, реалізовано його практичне відображення на AWS компонентах. Зроблено відповідні висновки.

ПЕРЕЛІК ПОСИЛАНЬ

1. Інформаційно новинний ресурс . – Режим доступу: https://infograph.venngage .com/p/89148/the-impact-of-technology-on-unemployment – Дата доступу: 01.10.2016

2. Recommender Systems Handbook[Текст] / Ricci, F.; Rokach, L.; Shapira, B.; Kantor, P.B : LLC, 2011. — 1 с.

3. BIG DATA[Текст] / Mayer-Schönberger V.; Kenneth Cukier, 2014 – 57 с.

4. Конкурс PC від Netflix. – Режим доступу: http://www.netflixprize.com/ – Дата доступу: 12.12.2016

5. The Philosophy of Microservice Architecture [Текст] / Lucas Kraus, 2015. – 57 с.

6. Стаття про КФ. - Режим доступу: http://danielnee.com/2016/09/collaborative-filtering-using-alternating-least-squares/ – Дата доступу: 15.12.2016

7. ПОКРАЩЕННЯ РЕЗУЛЬТАТІВ РОБОТИ РЕКОМЕНДАЦІЙНИХ СИСТЕМ ЗА ДОПОМОГОЮ АЛГОРИТМУ SVD [Текст], Мазурик О.Ю., // Международный научный журнал // № 9, 2015

8. William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery (1992). Numerical Recipes in C / Singular Value Decomposition. 2nd ed. New York: Cambridge University Press. p59-71

9. Офіційний портал бібліотеки Spark. . – Режим доступу: http://spark.apache.org/docs/latest/programming-guide.html – Дата доступу: 17.12.2016

10. Офіційний портал. – Режим доступу: https://opensource.com/business/15/1/apache-spark-new-world-record – Дата доступу: 18.12.2016

11. Офіційний портал Hadoop. – Режим доступу: http://www.uk.w3eacademy .com/hadoop/hadoop_introduction.htm

12. Вікіпедія. – Режим доступа: https://uk.wikipedia.org - Дата доступа: 05.01.2017.

13. http://spark.apache.org/docs/latest/api/python/pyspark.mllib.html #pyspark.mllib.recommendation.ALS Дата доступа: 07.01.2017.

ДОДАТОК А. НАЛАШТУВАННЯ ПРАВ AWS LAMBDA

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "logs:CreateLogGroup",
            "logs:CreateLogStream",
            "logs:PutLogEvents"
         ],
         "Resource": "arn:aws:logs:*:*:*"
      },
      {
         "Effect": "Allow",
         "Action": [
            "elasticmapreduce:RunJobFlow",
            "iam:PassRole"
         ],
         "Resource": "*"
      }
    ]
   }
```

ДОДАТОК Б. ЛІСТИНГ LAMBDA ФУНКЦІЇ

```
import boto3
def lambda_handler(json_input, context):
   client = boto3.client('emr', region_name='eu-central-1')
```

```
client.run_job_flow(
    Name='YourApp',
    ReleaseLabel='emr-5.2.1',
    Instances={
        'MasterInstanceType': 'm3.xlarge',
        'SlaveInstanceType': 'm3.xlarge',
        'InstanceCount': 21,
        'Ec2KeyName': 'rs-keypair',
        'KeepJobFlowAliveWhenNoSteps': False,
        'TerminationProtected': False,
        'Ec2SubnetId': 'subnet-9a73ecf2'
    },
    Steps=[
        {
            'Name': 'YourStep',
            'ActionOnFailure': 'TERMINATE_CLUSTER',
            'HadoopJarStep': {
                'Jar': 'command-runner.jar',
                'Args': [
                    'spark-submit',
                    '--driver-memory','10G',
                    '--executor-memory','4G',
                    '--executor-cores','4',
                    '--num-executors','20',
                    '/home/hadoop/process_data.py'
                ]
            }
        },
```

```
],
BootstrapActions=[
   {
      'Name': 'cluster_setup',
      'ScriptBootstrapAction': {
         'Path': 's3://rs-backet-2017/lambda/setup.sh',
         'Args': []
      }
   }
],
Applications=[
   {
      'Name': 'Spark'
   },
],
Configurations=[
   {
      "Classification": "spark-env",
      "Properties": {

      },
      "Configurations": [
         {
            "Classification": "export",
            "Properties": {
               "PYSPARK_PYTHON": "/usr/bin/python2.7",
               "PYSPARK_DRIVER_PYTHON": "/usr/bin/python2.7"
            },
            "Configurations": [
```

```
                ]
            }
        ]
    },
    {
        "Classification": "spark-defaults",
        "Properties": {
            "spark.akka.frameSize": "2047"
        }
    }
],
VisibleToAllUsers=True,
JobFlowRole='EMR_EC2_DefaultRole',
ServiceRole='EMR_DefaultRole'
)
```

ДОДАТОК В. НАЛАШТУВАННЯ СЕРЕДОВИЩА

```
#!/bin/bash
# Install our dependencies - replace these libraries with your needs!

sudo yum install -y gcc python-setuptools python-devel postgresql-devel
sudo python2.7 -m pip install SQLAlchemy==1.0.8
sudo python2.7 -m pip install boto==2.38.0
sudo python2.7 -m pip install funcsigs==0.4
sudo python2.7 -m pip install pbr==1.8.1
sudo python2.7 -m pip install psycopg2==2.6.1
sudo python2.7 -m pip install six==1.10.0
sudo python2.7 -m pip install wsgiref==0.1.2
```

```
sudo python2.7 -m pip install requests[security]

# Download code from S3 and set up cluster
aws s3 cp s3://your-bucket/spark_app.zip /home/hadoop/spark_app.zip
aws s3 cp s3://rs-backet-2017/lambda/process_data.py /home/hadoop/process_data.py
```

ДОДАТОК Г. ЛІСТИНГ ПРОГРАМИ

```
import pandas as pd

# pass in column names for each CSV and read them using pandas.
# Column names available in the readme file

#Reading users file:
u_cols = ['user_id', 'age', 'sex', 'occupation', 'zip_code']
users = pd.read_csv('ml-100k/u.user', sep='|', names=u_cols,
 encoding='latin-1')

#Reading ratings file:
r_cols = ['user_id', 'movie_id', 'rating', 'unix_timestamp']
ratings = pd.read_csv('ml-100k/u.data', sep='\t', names=r_cols,
 encoding='latin-1')

#Reading items file:
i_cols = ['movie id', 'movie title' ,'release date','video release date', 'IMDb URL',
'unknown', 'Action', 'Adventure',
 'Animation', 'Children\'s', 'Comedy', 'Crime', 'Documentary', 'Drama', 'Fantasy',
 'Film-Noir', 'Horror', 'Musical', 'Mystery', 'Romance', 'Sci-Fi', 'Thriller', 'War',
'Western']
```

```
items = pd.read_csv('ml-100k/u.item', sep='|', names=i_cols,
 encoding='latin-1')
users.head()
ratings.head()
items.head()
r_cols = ['user_id', 'movie_id', 'rating', 'unix_timestamp']
ratings_base = pd.read_csv('ml-100k/ua.base', sep='\t', names=r_cols, encoding='latin-1')
ratings_test = pd.read_csv('ml-100k/ua.test', sep='\t', names=r_cols, encoding='latin-1')
ratings_base.shape, ratings_test.shape

import graphlab
train_data = graphlab.SFrame(ratings_base)
test_data = graphlab.SFrame(ratings_test)

#Train Model
item_sim_model = graphlab.item_similarity_recommender.create(train_data, user_id='user_id', item_id='movie_id', target='rating', similarity_type='pearson')

#Make Recommendations:
item_sim_recomm = item_sim_model.recommend(users=range(1,6) ,k=5)
item_sim_recomm.print_rows(num_rows=25)
```