Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Факультет інформатики Кафедра мережних технологій

РЕКОМЕНДАЦІЙНА СИСТЕМА ЗАСНОВАНА НА СИНГУЛЯРНОМУ РОЗКЛАДАННІ

Текстова частина до магістерської роботи (тези) за спеціальністю „Інженерія програмного забезпечення”

Виконав: студент 6-го року навчання
Титаренко Владислав Олександрович

Керівник Малашонок Г.І.,
доктор фізико-математичних наук, професор

Рецензент_____
(прізвище та ініціали)

Магістерська робота захищена
з оцінкою «_____»

Секретар ДЕК _____
«____» _____2021 р.

Київ 2021

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

ЗАТВЕРДЖУЮ
доц., канд. фіз.-мат. наук
_____С. С. Гороховський
(підпис)
8 листопада 2020 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на дипломну роботу

студенту Титаренку В.О. факультету інформатики 2 курсу МП
ТЕМА Рекомендаційна система заснована на сингулярному розкладанні

Зміст ТЧ до курсової роботи:

Індивідуальне завдання
Вступ
1. Огляд рекомендаційних систем
2. Алгоритм для рекомендацій заснований на методі SVD
3. Проектування та розробка рекомендаційної системи
Висновок
Список використаних джерел

Дата видачі 8 листопада 2020 р.

Керівник _____
(підпис)

Завдання отримав _____
(підпис)

Зміст

Анотація

Дипломна робота описує рекомендаційнй систему засновану на методі сингулярного розкладанні матриць, що надає можливість персоналізованих рекомендацій користувачам систем.

Перший розділ присвячений огляду в цілому рекомендаційним системам. В цьому розділі буде дано визначенню поняття рекомендаційної системи, а також розглянуто та описано медоти реалізації, за якийми розділяються рекомендаційні системи. Будуть наведені приклади сфер використання рекомендаційних систем.

В другому розділі буде описано безпосередньо метод сингулярного розкладання матриць, а також розглянуто алгоритм рекомендацій заснований на даному методі.

Третій розділ присвячено архітектурі системі рекомендацій. Будуть визначено та описано вимоги, яким система повинна задовольняти. В розділі буде запропонована архітектура системи з обгрунтуванням її доцільності. Також, розділ описує модуль для рекомендацій розробленого за допомогою метода сингулярного розкладання матриць, та інших компонентів системи таких, як: сервіс бізнес логіки та мобільний застосунок.

Вступ

На сьогоднішній день майже кожна людина використовує різні сервіси для прослуховування музики, перегляду фільмів, а також здійснює покупки в онлайн магазинах. Ці сервіси зацікавленні в правильній пропозиції свої продуктів чи послуг кожному їхньому користувачеві. Одним із найефективніших способів для вирішення цієї задачі є надання персональних рекомендацій кожному клієнту. Персональні рекомендації є дуже корисними, адже вони допомагають визначити, що потребують користувачі і як наслідок дозволяють збільшити продажі в онлайн магазині і, як наслідок покращити прибутковість бізнесу.

Але для реалізації такого виду рекомендацій потрібно імплементувати рекомендаційну систему, яка би враховувала вподобання кожного користувача. Причому висувається ряд специфічних вимог до такої системи, які стосуються зберігання оперативного надання рекомендацій.

Виходячи з зазначеного вище, за мету даної дипломної роботи було поставлено спроектувати та реалізувати систему рекомендацій для онлайн магазинів, яка би використовувала алгоритм сингулярного розкладання матриць та враховувала попередні покупки кожного користувача для забезпечення максимально персоналізованих рекомендацій товарів.

1. Огляд рекомендаційних систем

1.1 Поняття рекомендаційної системи

Протягом останніх кількох десятиліть, із зростанням Youtube, Amazon, Netflix та багатьох інших таких веб-сервісів, системи рекомендацій займають дедалі більше місця в нашому житті. Від реклами в Інтернеті до електронної комерції, рекомендаційні системи сьогодні присутні майже на всіх онлайн платформах і сервісах.

Рекомендаційні системи - це алгоритми, спрямовані на те, щоб запропонувати користувачам відповідні предмети. До прикладу, рекомендація товарів на сайтах онлайн рітейлерів, підбірка фільмів, що можуть сподобатись, або музики на стрімінгових сервісах.

Системи, що рекомендують, є критично важливими в деяких галузях, оскільки вони можуть приносити величезний дохід, коли вони ефективні, а також можуть бути конкурентною перевагою. Недарма кілька разів Netflix організував змагання «приз Netflix», метою яких було створити систему рекомендацій, яка працює ефективніше, ніж власний алгоритм Netflix з призом 1 мільйон доларів.

Метою рекомендаційної системи є пропозиції користувачам релевантних для них речей. Щоб досягти цього використовуються дві основні категорії методів: методи колаборативної фільтрації та методи на основі вмісту.

1.2 Рекомендаційні методи на основі колаборативної фільтрації

Колаборативна фільтрація - це методи, які базуються виключно на минулих взаємодіях, зафіксованих між користувачами та елементами, з метою генерації нових рекомендацій. Ці взаємодії зберігаються у так званій "матриці взаємодій між елементами користувача".

Основною ідеєю колаборативної фільтрації є виявлення схожих користувачів та подібних елементів на основі минулих взаємодій між

елементами та користувачами і прогнозування на основі цих характеристик близькості.

Клас алгоритмів колаборативної фільтрації поділяється на дві підкатегорії, які називаються підходами на основі пам'яті та на основі моделі. Підходи, засновані на пам'яті, безпосередньо працюють зі значеннями взаємодій, припускаючи відсутність моделі, і по суті засновані на пошуку найближчих сусідів (наприклад, знайти схожих користувачів до користувача та запропонувати йому найбільш популярні предмети цих сусідів). Підходи, що базуються на моделях, передбачають наявність узагальнюючої моделі, яка пояснює взаємодію між елементами і користувачами, намагається виявити її, щоб робити нові рекомендації.

Головною перевагою колаборативної фільтрації є необов’язковість інформації про користувачів або предмети, і тому їх можна використовувати в набагато більшій кількості ситуацій. Окрім того, чим більше користувачів взаємодіє з елементами, тим більше точні нові рекомендації (для фіксованого набору користувачів та елементів нові взаємодії, зафіксовані з часом, приносять нову інформацію та роблять систему все більш ефективною).

Так як колаборативна фільтрація враховує лише попередні взаємодії, з цього випливає що колаборативна фільтрація страждає від проблеми «холодного старту” - неможливість рекомендувати новим користувачам або рекомендувати новий елемент іншим користувачам, через те що багато користувачів або елементів мають занадто мало взаємодій.

Цей недолік можна вирішити по-різному:

- рекомендувати випадкові предмети новим користувачам або нові предмети випадковим користувачам,
- рекомендувати популярні предмети новим користувачам або нові предмети найбільш активним користувачам,

- рекомендувати набір різних предмети для нових користувачів або нові предмети для набору різних користувачів.

1.3 Рекомендаційні методи на основі вмісту

На відміну від методів колаборативної фільтрації, які базуються лише на взаємодії між елементами та користувачами, підходи на основі вмісту використовують додаткову інформацію про користувачів та елементи. Наприклад у системі рекомендації фільмів, цією додатковою інформацією може бути вік, стать, електронна пошта, країна та інша персональна інформація користувачів. Щодо елементів це може бути категорія фільму, актори, режисер, сценарист, тривалість та інше.

Головна ідея методів на основі вмісту побудові моделі на наявних характеристиках (features), що пояснюють взаємодії між елементами та користувачами. Наприклад жінки частіше надають перевагу фільмам за жанром драма, а чоловіки триллерам. В такому разі модель робитиме прогнози для користувача на основі інформації з його профілю і з цієї інформації робити рекомендації.

Методи на основі вмісту, страждають набагато менше від проблеми «холодного старту», ніж колаборативна фільтрація. Нових користувачів або елементи можна описати за їх характеристиками (features), тому для них можна зробити відповідні рекомендації. Цю проблему можуть мати лише нові користувачі або нові елементи.

1.4 Сингулярний розклад матриці (SVD) у рекомендаційних системах

Найпоширеніший методом в рекомендаційних системах є розглянутий вище метод колаборативної фільтрації, який рекомендую на основі попереднього набору даних про взаємодію користувачів та елементів. Двома популярними підходами колаборативної фільтрації є моделі прихованих

факторів, які визначають характеристики з матриць користувачів та елементів і модель сусідів, що знаходить схожість між продуктами чи користувачами.

Модель сусідства орієнтована на елементи підхід для виявлення переваг користувача на основі оцінок, які користувач отримує для подібних предметів. Модель прихованих факторів виділяє характеристики та кореляцію з матриці елементів та користувачів. У даній роботі розглядається модель сингулярного розкладання матриці (SVD), яка є видом моделі прихованих факторів.

2. Алгоритм для рекомендацій заснований на методі SVD

У рекомендаційних системах SVD використовується при колаборативній фільтрації. Дані представляються у матричній формі, де кожен рядок представляє користувача, а кожен стовпець - елемент, рейтинги, які присвоюються користувачам.

Факторизація матриці відбувається через її сингулярне розкладання. Спочатку знаходяться фактори матриць на основі факторизації матриці високого рівня, наприклад користувач-товар-рейтинг.

Сингулярне розкладання - це метод розкладання матриці на три інші матриці:

$$A = USV^T,$$

де A - це mxn матриця корисності, U - ортогональна ліва сингулярна матриця amxr, яка представляє взаємозв'язок між користувачами та прихованими факторами, S - діагональна матриця rxr, яка описує силу кожного прихованого фактора, а V - діагональ правої сингулярної матриці rxn, що вказує подібність між предметами та прихованими факторами. Прихованими факторами є характеристики елементів, наприклад, вид продукту. SVD зменшує розмірність матриці корисності A, витягуючи її приховані фактори та відображає кожного користувача та кожен елемент у r-мірний прихований простір. Це перетворення сприяє чіткому відображенню взаємозв'язків між користувачами та елементами.

Нехай кожен елемент буде представлений вектором x_i, а кожен користувач - вектором y_u. а очікувана оцінка користувачем товару r_{ui}, яка також може бути задана як $r_{ui} = x_i^T * y_u$

Таким чином r_{ui} є формою факторизації при сингулярному розкладанні. x_i та y_u можна отримати так, щоб різниця квадратних помилок між їх точковим добутком та очікуваною оцінкою в матриці елемента користувача була мінімальною. Це можна виразити так: $min(x, y) = \Sigma(r_{ui} - x_i^T y_u)^2$

Для того, щоб модель добре узагальнювала і не перенавчала дані, додається значення регуляризації до формули як «покарання» або «ціна»: $min(x, y) = \Sigma(r_{ui} - x_i^T y_u)^2 + \lambda(||x_i||^2 + ||y_u||^2)$

Для зменшення помилки між величиною, що передбачається моделлю, та фактичним значенням, алгоритм використовує bias значення. Нехай для пари користувач-елемент (u, i), μ - середня оцінка всіх елементів, b_i - середня оцінка елемента i - μ, а b_u - середня оцінка, дана користувачем u - μ, остаточне рівняння після додавання bias значення і значення регуляризації виглядає так:

$min(x, y, b_i, b_u) = \Sigma(r_{ui} - x_i^T y_u - \mu - b_i - b_u)^2 + \lambda(||x_i||^2 + ||y_u||^2 + b_i^2 + b_u^2)$

Це рівняння є основною складовою алгоритму рекомендації на основі сингулярного розкладу матриць.

2.1 Сингулярне розкладання матриці (SVD)

Сингулярне розкладання матриці (SVD), це класичний метод лінійної алгебри, який використовується в науці про дані як техніка зменшення розмірності. SVD є методом факторизації матриці, який зменшує кількість ознак(features) в даних. Це відбувається завдяки зменшенню розмірності простору з n-розміру до n*-розмірності, $n^* \in (1, n)$.

Нехай A - матриця m × n. Сингулярне розкладання (SVD) матриці A є: $A = U\Sigma V$, де U це m × m ортогональна матриця, V це n × n ортогональна матриця, а Σ -діагональна матриця m × n з невід'ємними діагональними значеннями: $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_p$, $p = min\{m, n\}$,також відомі як сингулярні значення A, цей розклад дає багато інформації про A, включаючи його діапазон, нульовий простір, ранг та число умов 2-норми.

Практичний алгоритм для обчислення SVD матриці A, за Голубом та Каханом:

Нехай U і V мають наступний розподіл стовпців $U = [u_1, \ldots, u_m]$, $V = [v_1, \ldots, v_n]$ З наступних відношень $Av_j = \sigma_j u_j$, $A^T u_j = \sigma_j u_j$, $j = 1, \ldots, p$. випливає, що $A^T A v_j = \sigma^2 v_j$. Тобто, квадрати сингулярних значень є власними значеннями $A^T A$, яка є симетричною матрицею.

З цього випливає, що одним із підходів до обчислення SVD матриці A є застосування симетричного алгоритму QR до $A^T A$ для отримання розкладу $A^T A = V\Sigma^T \Sigma V^T$. Тоді співвідношення $Av_j = \sigma_j u_j$, $j = 1, \ldots, p$. може використовуватися спільно з QR-факторизацією з поворотом стовпця для отримання U. Однак цей підхід не є найбільш практичним через витрати та втрату інформації від обчислень $A^T A$.

Натомість ми можемо неявно застосувати симетричний алгоритм QR до $A^T A$. Перший крок QR полягає у використанні перетворення Хаусхолдера для зменшення матриці до тридіагональної форми, щоб замість цього не зменшувати A до верхньої двобічної форми:

$$U_1^T A V_1 = B = \begin{bmatrix} d_1 & f_1 & & & \\ & d_2 & f_2 & & \\ & & \ddots & \ddots & \\ & & & d_{n-1} & f_{n-1} \\ & & & & d_n \end{bmatrix}$$

Звідси випливає, що $T = B^T B$є симетричною і тридіагональною матрицею. Тоді ми могли б застосувати симетричний алгоритм QR безпосередньо до T, але, щоб уникнути втрати інформації від обчислення T явно, можна неявно застосувати алгоритм QR до T, виконуючи наступні кроки під час кожної ітерації:

1. Визначити перший поворот рядка Гівенса G_1^T що застосовується до $T - \mu I$, де μ - зсув Уілкінсона від симетричного алгоритму QR. Для цього потрібне лише обчислення першого стовпця T, який має лише два ненульові записи $t_{11} = d_1^2$ і $t_{21} = d_1 f_1$.

2. Застосувати G_1 для обертання стовпців 1 і 2 B, щоб отримати $B_1 = BG_1$. Це вводить небажане ненульове значення у записі (2, 1).

3. Застосувати обертання рядка Гівенса H_1 до рядків 1 і 2 щоб занулити входження (2, 1) B_1, що дає $B_2 = H_1^T BG_1$. Тоді, B_2 має небажане ненульове значення у входженні (1, 3).

4. Застосувати обертання Гівенса стовпця до стовпців 2 і 3 B_2, що дає $B_3 = H_1^T BG_1 G_2$.Це вводить небажаний нуль у входжені (3, 2).

5. Продовжити чергування обертань рядків і стовпців, щоб переслідувати небажане ненульове значення по діагоналі B, поки остаточно B не відновиться до верхньої дводіагональної форми.

За теоремою Q-аналог, оскільки G_1- це обертання Гівена, яке застосовується до першого стовпця T, обертання стовпців, які допомагають відновити верхню бідіагональну форму, дорівнюють тим, які застосовувалися б до T, якби симетричний алгоритм QR застосовувався безпосередньо до T. Отже, симетричний QR-алгоритм правильно застосовується, неявно, до B.

2.2 Огляд підходів QR декомпозиції

Існує кілька методів фактичного обчислення QR декомпозиції або факторизації. Одним з таких методів є процес Грама-Шмідта. У 1907 р. Ерхард Шмідт опублікував алгоритм ортогоналізації, який став популярним та широко використовуваним.

Дано n лінійно незалежних елементів a_1, a_2, ..., a_nу внутрішньому просторі добутку. Алгоритм обчислює ортонормальний базис $q_{1,}q_{2,}$ ..., q_n, такий що:

$$a_k = r_{1k}q_1 + r_{2k}q_2 + \ \dots \ + \ r_{kk}q_k, \quad k \ = \ 1{:}\,n$$

Інтерпретація матриці полягає в тому, що A = (a_1, a_2, ..., a_n)враховується у добутку:

$$A = (q_1, q_2, ..., q_n) \begin{pmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ & r_{22} & \cdots & r_{2n} \\ & & \ddots & \vdots \\ & & & r_{nn} \end{pmatrix}$$

R - верхня трикутна матриця з позитивними діагональними записами. Ця факторизація є унікальною визначеною. Метод названий на честь Йоргена Педерсена Грама та Ерхарда Шмідта, проте П'єр-Саймон Лаплас працював над цим ще до Грама та Шмідта. Шмідт працював над тим, що зараз відоме як класичний процес Грама – Шмідта. [3]

Лаплас працював над тим, що зараз відоме як модифікований процес Грама – Шмідта. Рутісгаузер запропонував метод, заснований на реортогоналізації [1]. Цей метод не набув великої популярності, оскільки приблизно в той же час був запропонований новий спосіб обчислення декомпозиції з використанням елементарних ортогональних матриць Хаусхолдера. Реортогоналізація подвоює обчислювальні зусилля, із цієї причини метод Хаусхолдера є кращим [4].

Інший метод пошуку Q і R для факторизації матриці передбачає використання перетворень Хаусхолдера. Спочатку ідея була запропонована Елстоном Хаусхолдером у 1958 р. На практиці вона використовується частіше, ніж метод Грам-Шмідта. Він не рекомендується, оскільки іноді може спричинити неточність обчислень, що призводить до неортогональної Q-матриці, тоді як використання перетворень Хаусхолдера є, мабуть, найпростішим із чисельно стабільних алгоритмів декомпозиції QR завдяки використанню відбиття як механізм отримання нулів у матриці R.

Крім перетворень Хаусхолдера, розкладання може бути досягнуто методом повороту Гівенса, який був запроваджений Уоллесом Гівенсом у 1950-х роках. Розкладання QR можна обчислити за допомогою ряду обертань Гівенса. Кожне обертання обнуляє елемент у піддіагоналі матриці, утворюючи матрицю R. Об'єднання всіх обертань Гівенса утворює ортогональну матрицю Q [5].

2.2.1 Процес Грама - Шмідта

Процес Грама-Шмідта - це простий алгоритм, який використовується для пошуку ортогональної або ортонормальної основи для будь-якого ненульового підпростору R^n. Ми знаємо, що ортогональна матриця має вектори рядків і стовпців одиничної довжини:

$$|| a_n || = \sqrt{(a_n * a_n)} = \sqrt{(a_n^T * a_n)} = 1 ,$$

де a_n - лінійно незалежний вектор стовпця матриці. Вектори є також перпендикуляр в ортогональній основі. Процес Грама-Шмідта працює шляхом знаходження ортогональної проекції q_n для кожного вектора стовпця a_n, а потім віднімання його проекцій на попередні проекції (q_i). Отриманий вектор потім ділиться на довжину цього вектора, щоб отримати одиничний вектор.

Розглянемо матрицю A з n векторами стовпців такими, що: $A = [a_1 \mid a_2 \mid \ldots \mid a_n]$. І ми хочемо знайти ортонормальну проекцію u_n. З u_1 немає проблем: він може увійти напрямок a_1. Отже, ми знаходимо ортонормальну проекцію першого векторного стовпця a_1 наступним чином:

$$v_1 = a_1;\ u_1 = v_1 / ||\, v_1||$$

Справжня проблема починається з v_1- який повинен бути ортогональним v_2. Якщо другий вектор a_2 має будь-який компонент в напрямку v_1 (який є напрямком a_1), цей компонент потрібно відняти:

$$v_2 = a_2 - \text{проекція}_{v1}(a_2) = a_2 - (a_2 * u_1)u_1, \quad u_2 = v_2 / ||\, v_2||$$

Цей процес повторюється аж до n-го стовпцевого вектора, при цьому кожен наступний крок $k + 1$обчислюється як:

$$v_{k+1} = a_{k+1} - (a_{k+1} * u_1)u_1 - \ldots - (a_{k+1} * u_k)u_k, \qquad u_{k+1} = v_{k+1} / ||\, v_{k+1}||$$

[6]

Це і є ідея всього процесу Грама-Шмідта, потрібно відняти від кожного нового вектора його складові в напрямках, які вже врегульовані, і так повторювати знову і знову. [7]

Оскільки класичний метод Грама-Шмідта має дуже погані числові властивості в тому, що серед підрахованих q зазвичай спостерігається серйозна втрата ортогональності. Тому представимо реалізацію модифікованого Грам-Шмідта (MGS), що дає більш надійний результат.

При $A \in R^{mxn}$з рангомn ($rank(A) = n$), наступний алгоритм обчислює QR-факторизацію $A = Q_1 R_1$, де $Q_1 \in R^{mxn}$, $R_1 \in R^{nxn}$ −верхня трикутна матриця. [7]

$for\ k = 1 : n$

$R(k,\ k) = ||\, A(1{:}\, m,\ k)||_2$

$$Q(1{:}\,m,\ k) = A(1{:}\,m,\ k) \,/\, R(k,\ k)$$

$$for\ j = k + 1{:}\,n$$

$$R(k,\ j) = Q(1{:}\,m,\ k)^T * A(1{:}\,m,\ j)$$

$$A(1{:}\,m,\ j) = A(1{:}\,m,\ j) - Q(1{:}\,m,\ k)R(k,\ j)$$

$$end$$

2.2.2 Перетворення Хаусхолдера

Перетворення Хаусхолдера є іншим методом QR-декомпозиції, працює шляхом пошуку відповідних матриць Q (матриць Хаусхолдера) та множення їх зліва на вхідну матрицю A для побудови верхньої трикутної матриці R. На відміну від процесу Грама-Шмідта, підхід перетворень Хаусхолдера явно не формує матрицю Q. Однак її можна знайти, взявши точковий добуток кожної сформованої матриці Хаусхолдера.

$$Q = Q_1^T * Q_2^T * \ \dots\ * Q_n^T$$

Основна ідея полягає в тому, щоб спроектувати унітарні матриці Q_k так, щоб $Q_n \dots Q_2 Q_1 A$ була вертикальною. Матриця Q_k обрана для введення нулів нижче діагоналі в k-тому стовпці, зберігаючи всі введені раніше нулі. Приклад для матриці 4 на 3 наведений нижче:

$$\underset{A}{\begin{pmatrix} * & * & * \\ * & * & * \\ * & * & * \\ * & * & * \end{pmatrix}} \longrightarrow \underset{Q_1A}{\begin{pmatrix} * & * & * \\ 0 & * & * \\ 0 & * & * \\ 0 & * & * \end{pmatrix}} \longrightarrow \underset{Q_2Q_1A}{\begin{pmatrix} * & * & * \\ 0 & * & * \\ 0 & 0 & * \\ 0 & 0 & * \end{pmatrix}} \longrightarrow \underset{Q_3Q_2Q_1A}{\begin{pmatrix} * & * & * \\ 0 & * & * \\ 0 & 0 & * \\ 0 & 0 & 0 \end{pmatrix}}$$

На початку кроку k, у перших k-1 стовпцях цих рядків є блок нулів. Застосування Q_k утворює лінійні комбінації цих рядків, а лінійні комбінації нульових записів залишаються нульовими. Після n кроків усі записи нижче діагоналі стають нулями, і $Q_n \dots Q_2 Q_1 A$= R є верхньою трикутною матрицею.

І в результаті ми отримуємо QR факторизацію матриці A:

$$Q_n \dots Q_2 Q_1 A = \mathrm{R}$$

$$A = QR$$

Далі поданий спосіб обрахунку матрицю Q_k (матриць Хаусхолдера). Так як множення матриць має внести нулі в k-ий стовпець, матриці мають спричинити наступне відображення:

$$x = \begin{pmatrix} * \\ * \\ * \\ \vdots \\ * \end{pmatrix} \xrightarrow{Q_k} Q_k x = \begin{pmatrix} ||x|| \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} = ||x|| e_1$$

Перетворювач Q_k буде відображати простір вектора по гіперплощині H ортогональній $v = ||x|| e_1 - x$. Коли застосовується перетворювач, кожна точка на одній стороні гіперплощини H відображається в дзеркальному відображенні з іншого боку. Зокрема, x відображається в $||x|| e_1$. Формула відображення може бути виведеною з

$$Py = (I - \frac{vv^T}{v^T v})y = y - v(\frac{v^T y}{v^T v}).$$

Псевдокод алгоритму Хаусхолдера:

$$\begin{aligned} &for\ k = 1\ to\ n: \\ &\quad x = A_{k:m,k} \\ &\quad v_k = sign(x_1)||x||e_1 + x \\ &\quad v_k = v_k / ||v_k|| \\ &\quad A_{k:m,k:n} = A_{k:m,k:n} - 2v_k(A_{k:m,k:n}) \\ &end \end{aligned}$$

2.2.3 Поворот Гівенса

Поворот Гівена представлений матрицею наступного вигляду:

$$G(i,j,\theta) = \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & c & \cdots & -s & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & s & \cdots & c & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix}$$

де $c = cos\theta$ і $s = sin\theta$ з'являються на перетинах i-го і j-го рядка і стовпчика. Ненульовими елементами матриці Гівена є:

$g_{kk} = 1$ для $k \neq i, j$; $g_{kk} = c$ для $k = i, j$; $g_{ji} = -g_{ji} = -s$. Оскільки $c^2 + s^2 = cos^2\theta + sin^2\theta = 1$, то $G(i, j, \theta)$є ортогональною. Обертання Гівена оперує парою рядків, щоб ввести в матрицю одиничний нуль. Враховуючи скаляри a і b, наступна функція обчислює $c = cos\theta$ та $s = sin\theta$ так

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix}$$

Псевдокод для параметрів обертання [1]:

$function\ [c,\ s] = givens(a,\ b)$

$if\ b = 0$

$c = 1;\ s = 0$

$else$

$if\ |b| > |a|$

$r = -a/b;\ s = 1/\sqrt{(1 + r^2)};\ c = sr$

$else$

$$r = -\ b/a;\ \ s = 1/\sqrt{(1 + r^2)};\ \ c = cr$$

Алгоритм Гівенса QR використовується для представлення матриці у вигляді A = QR, де Q - ортогональна матриця, а R - верхня трикутна матриця. На кожному кроці алгоритму Гівенса обертаються два рядки матриці, і перетворюються. Враховуючи A ∈ R mxn з m> n, алгоритм замінює A на $Q^T A = R$, де R верхня трикутна матриця, а Q ортогональна матриця.

Псевдокод для QR-методу Givens:

$for\ j = 1\ to\ n:$
$\quad for\ i = m\ downto\ j + 1:$
$\quad\quad [c,\quad s] = givens(A(i-1, j), A(i, j)])$
$\quad\quad A(i-1 : i, j : n) = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T A(i-1 : i, j : n)$
end

Розкладання QR за допомогою обертання Гівенса є найбільш гнучким. Рядки, що беруть участь у кожному оновленні, а також порядок введення нулів. Суттєвою перевагою є те, що кожен новий нульовий елемент впливає лише на рядок з елементом, що обнуляється, і рядок вище. Це робить алгоритм обертання Гівенса ефективнішим та має більшу пропускну здатність, ніж метод відображення Хаусхолдера.

2.2.4 Порівняння ефективності методів QR декомпозиції

Після вивчення кожного методу QR декомпозиції у даному розділі наведене порівняння часу їх роботи над матрицями різного розміру. Для цього кожен із методів реалізований у R, і був оцінений час роботи на випадково згенерованих матрицях різної розмірності. Даний експеримент було проведено на комп'ютері з такими характеристиками: 2,6 GHz 6-Core Intel Core i7, RAM 16 ГБ, ОС MacOS 11.4.

Заміряємо час роботи кожного з методів для матриць розмірністю 5 x 5, 25 x 25 та 50 x 50. Результати вимірювань:

Givens method

5 x 5 matrix: 0.1019395 sec

25 x 25 matrix: 0.26893583 sec

50 x 50 matrix: 1.08134554 sec

Gram_Schmidt method

5 x 5 matrix: 0.2081259 sec

25 x 25 matrix: 0.27568024 sec

50 x 50 matrix: 0.28836764 sec

Householder method

5 x 5 matrix: 0.35198063 sec

25 x 25 matrix: 1.21347816 sec

50 x 50 matrix: 2.42531291 sec

Як можемо бачити метод Хаусхолдера має найбільший час виконання. я Метод Грама-Шмідта має найкращий час виконання для матриць великих розмірностей (50 x 50), в свою чергу метод Гівенса найкраще працює на метрицях менших розмірностей (5 x 5 та 25 x 25).

Для кращого порівняння було проведено 1250 експериментів для кожного з трьох методів, де вимірювався час роботи алгоритму для всіх варіантів матриць з розмірністю від 1 до 50 по вертикалі та з розмірністю від 1 до 50 по горизонталі. Всього було проведено 3750 експериментів.

Для всіх трьох методів та побудовано три теплові карти, які показують, як розмір матриць впливає на час виконання кожного із методів. Метод Грама-Шмідта є найшвидшим із усіх трьох і розмір матриць насправді не впливає на часову ефективність цього методу.

На діаграмі значення n по вертикалі, m по горизонталі відповідає за розмірність матриці, кожен сегмент відповідає за час роботи конкретного методу QR декомпозиції для конкретної розмірності матриці n x m.

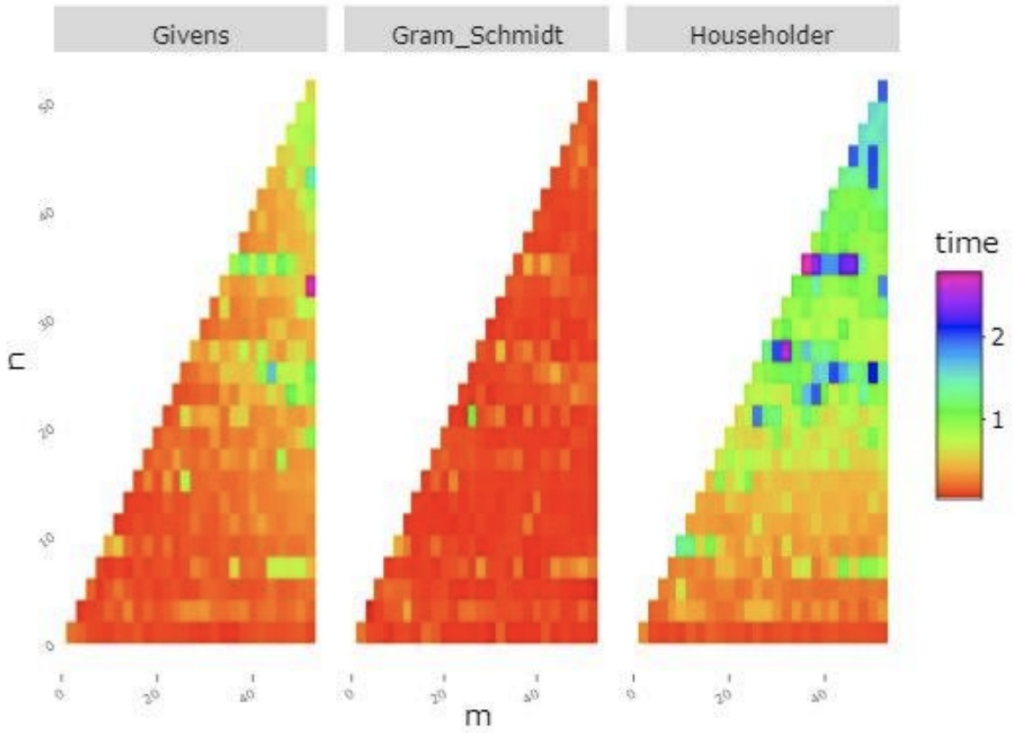

Рис.1 Порівняння ефективності методів

Дана діаграма була реалізована за допомогою нижче наведеного коду, даний код та реалізацію всіх трьох методів на R можна знайти в додатку А:

```
tvec = map2(m, n, ~QR_comp(matrix(runif(.x*.y), ncol = .y)))
plotly::ggplotly(
bind_rows(tvec) %>%
  gather("func","time") %>%
  mutate(n = rep(n, 3), m = rep(m, 3)) %>%
  ggplot(aes(m, n, fill = time)) +
  geom_tile() +
  facet_grid(.~func) +
  scale_fill_gradientn(colours = rainbow(9)) +
```

```
theme(panel.background = element_blank(),
      axis.ticks.y = element_blank(),
      axis.text.y = element_text(angle = 35, size = 5),
      axis.text.x = element_text(angle = 30, size = 5)), width = 550, heigh = 400)
```

Метод Гівена працює швидко на менших матриць, але стає є трохи повільнішим за метод Грама-Шмідта. Метод перетворень Хаусхолдера має найгірший час виконання і складність. На діаграмі ми можемо помітити, що він стає повільнішим набагато швидше, ніж інші методи, а для великих матриць він має найдовший час виконання.

Дані результати співпадають із попередніми пунктами, метод Хаусхолдера не ефективно використовувати для сингулярного розкладання матриці. Метод Грама-Шмідта є найшвидшим, але при цьому через втрату ортогональності є неточним. Відповідно метод Гівенса є оптимальним як за часом роботи так і за коректністю результату.

3. Проектування та розробка рекомендаційної системи

3.1 Функціонал та вимоги до системи

Як і для будь-якої системи, перед реалізацією системи для рекомендацій продуктів, потрібно визначитися з функціоналом данної системи. Визначимо набір функціональних можливостей виходячи з очікуваного сценарію використання даної системи.

Перш за все планується, що інтерфейсом користувача повинен бути мобільний додаток. В даній роботі буде реалізовано мобільний додаток для iOS, виходячи з того що, на теперішній час, це одна з двох найпопулярніших мобільних операційних систем. Очікуваний сценарій користування для цього додатку повинен складатися з декількох етапів. Спочатку користувач повинен авторизуватися в iOS додатку, далі, отримавши доступ до системи, він може здійснювати покупки товарів, обираючи їх, як і з загального каталогу так і з рекомендованих продуктів. Рекомендації повинні базуватися на вподобаннях користувача, тобто в нашому випадку - на раніше придбаних користувачем продуктів.

Виходячи з вищесказаного можна формалізувати на виділити основний функціонал та можливості системи:

- надання списку рекомендованих продуктів користувачу
- можливість придбання товарів через мобільний додаток
- можливість авторизації та аутентифікації користувача в системі
- перегляд історії покупок користувача
- врахування раніше придбаних продуктів в рекомендаціях

Визначившись з функціоналом, також потрібно задати вимоги до системи. Перш за все планується, що система буде складатися з трьох, так званих, модулів, а саме:

- модуль рекомендацій;
- серверна частина;

- мобільний додаток;

Виходячи з цього сформулюємо окремо вимоги до кожної частини системи.

Перш за все визначимо вимоги для модуля рекомендацій. Головний функціонал данного модуля це надання рекомендацій товарів, базуючись на даних з історії покупок користувача. Отже, основні вимоги:

- модуль повинен мати доступ до бази даних історії покупок користувача та до загального каталогу продукту;
- рекомендації товарів повинно базуватися на вподобаннях користувача;
- обрахування рекомендацій повинно займати до 5 секунд;
- для взаємодій з рекомендаційним модулем повинно використовуватися REST API;
- модуль повинен мати можливість легко горизонтально масштабуватися;

Тепер визначимо основні вимоги до серверної частини. Серверна частина буде відповідати за такий функціонал: взаємодія з рекомендаційним модулем, збереження інформації про користувача та його історії, надання списку каталогу продуктів, обробка операції покупки товару, ідентифікація, авторизація та аутентифікація користувача. Виходячи з цього визначемо основні вимогу до серверної частини:

- модуль повинен мати доступ до бази даних користувачі та історії їх покупок;
- для взаємодії з модулем повинно бути передбачено REST API
- модуль повинен мати можливість легко горизонтально масштабироваться;
- можливість надання доступу користувача тільки до своїх власних даних та історії покупок;

Наостанок, визначимо вимоги до мобільного додатку. Як було зазначено вище даний додаток буде розроблений для мобільної операційної системи iOS.

Додаток буде взаємодіяти з серверною частиною та буде відображати списки рекомендованих товарів, а також товари з каталогу. Отже, сформулюємо вимоги до мобільного додатку:

- додаток повинен бути сумісним с мобільною операційною системою iOS;
- додаток повинен мати можливість взаємодіяти з REST API серверної частини;
- додаток повинен відображати в формі списку товари з каталогу;
- додаток повинен відображати в формі списку рекомендовані товари;
- додаток повинен надавати форми для входу користувача;
- додаток повинен містити елементи інтерфейсу для спрощення покупки товарів;

3.2 Архітектура системи

Отож, після того, як основні вимоги були визначенні, можна почати будувати архітектуру системи. З усього вищезазначеного випливає, що в архітектурі рекомендаційної системи потрібно враховувати масштабованість системи. В сучасних подібних архітектурах використовуються хмарні обчислювальні платформи так, як вони можуть забезпечити автоматичне горизонтальне масштабування системи.

3.2.1 Огляд технологій та архітектурних рішень

3.2.1.1 Amazon Web Services

Для реалізації практичної частини даної курсової роботи було обрано хмарну платформу AWS, так як це найпопулярніша та найбільш функціонально наповнена платформа такого класу, яка задовольняє всім вище перечисленим вимогам.

AWS (Amazon Web Services) - це комплексна платформа хмарних обчислень, що надається Amazon, що включає поєднання інфраструктури як послуги (IaaS), платформи як послуги (PaaS) та програмного забезпечення як

послуги (SaaS). Служби AWS можуть запропонувати обчислювальні потужності, різні розподілені файлові системи та бази даних, а також забезпечує ефективне масштабування всіх своїх сервісів.

AWS була запущена в 2006 році з внутрішньої інфраструктури, яку Amazon.com створила для ведення своїх роздрібних операцій в Інтернеті. AWS була однією з перших компаній, яка запровадила хмарну обчислювальну модель, яка масштабується, щоб забезпечити користувачів обчисленнями, зберіганням або пропускною спроможністю за потреби.

AWS пропонує безліч різних інструментів та рішень для підприємств та розробників програмного забезпечення, які можна використовувати в центрах обробки даних у 190 країнах. Такі групи, як державні установи, навчальні заклади, некомерційні організації та приватні організації, можуть користуватися послугами AWS.

AWS розділений на різні сервіси; кожен з них може бути налаштована по-різному залежно від потреб користувача. Понад 100 сервісів наявні на даний час в AWS, включаючи сервіси для хмарних обчислень, бази даних, сервіси управління інфраструктурою, розробки додатків та безпеки. Ці сервіси розділенні за категоріями включають: хмарні обчислення, бази даних, сервіс управління даними, міграції, інструменти розробки, моніторинг, сервіси безпека, сервіси для обробки управління великих об'ємів даних, аналітика, штучний інтелект (ШІ), мобільна розробка, сервіси для повідомлень та сповіщень.

Amazon Web Services надає послуги з десятків центрів обробки даних, розподілених по зонах доступності (AZ) в регіонах по всьому світу. AZ - це місце, яке містить кілька фізичних центрів обробки даних. Регіон - це сукупність АЗ в географічній близькості, пов’язаних мережевими зв’язками з низькою затримкою.

Бізнес вибере одну чи кілька зон доступності з різних причин, таких як відповідність та близькість до кінцевих споживачів. Наприклад, клієнт AWS може обертати віртуальні машини (ВМ) і тиражувати дані в різних AZ для досягнення високо надійної інфраструктури, стійкої до збоїв окремих серверів або цілого центру обробки даних.

AWS також пропонує інструмент автоматичного масштабування для динамічного масштабування ємності для підтримки працездатності екземпляра та продуктивності.

Служба реляційних баз даних Amazon - яка включає параметри для Oracle, SQL Server, PostgreSQL, MySQL, MariaDB та власну високопродуктивну базу даних під назвою Amazon Aurora - забезпечує реляційну систему управління базами даних для користувачів AWS. AWS також пропонує керовані бази даних NoSQL через Amazon Redshift.

3.2.1.2 Amazon ECS та Docker

Служба Amazon Elastic Container Service (Amazon ECS) - це повністю керована служба організації контейнерів, яка допомагає легко розгортати, керувати та масштабувати контейнерні програми. Він глибоко інтегрується з рештою платформи AWS, щоб забезпечити безпечне та просте у використанні рішення для запуску контейнерних навантажень у хмарі та тепер у вашій інфраструктурі з Amazon ECS Anywhere.

Amazon ECS використовує без серверну технологію від AWS Fargate для забезпечення автономних контейнерних операцій, що зменшує час, витрачений на налаштування, виправлення та безпеку. Замість того, щоб турбуватися про управління площиною управління, надбудовами та вузлами, Amazon ECS дозволяє швидко створювати додатки та розвивати свій бізнес.

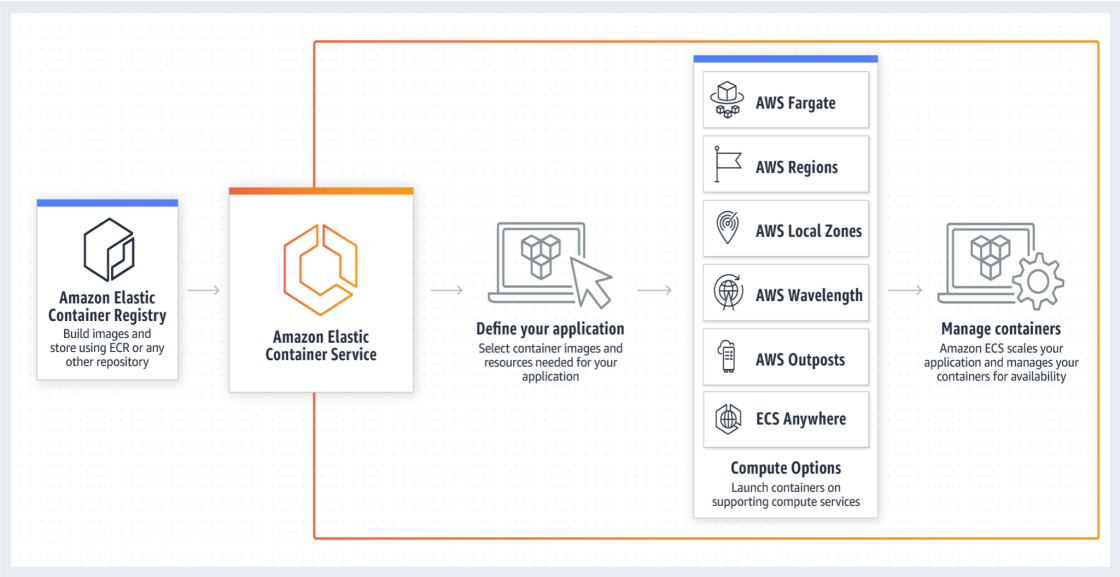

Рис.2 Принцип роботи Amazon ECS

Docker - це програмна платформа, яка дозволяє швидко створювати, тестувати та розгортати програми. Docker пакує програмне забезпечення у стандартизовані блоки, які називаються контейнерами, що містять усе необхідне для запуску програмне забезпечення, включаючи бібліотеки, системні інструменти, код та час роботи. За допомогою Docker ви можете швидко розгорнути та масштабувати програми в будь-якому середовищі та знати, що ваш код буде запущений.

Запуск Docker на AWS надає розробникам та адміністраторам надзвичайно надійний, недорогий спосіб створення, доставки та запуску розподілених програм у будь-якому масштабі.

Останні оголошення: Docker співпрацює з AWS, щоб допомогти розробникам пришвидшити доставку сучасних програм у хмару. Ця співпраця допомагає розробникам використовувати Docker Compose та Docker Desktop, щоб використати той самий локальний робочий процес, який вони використовують сьогодні для безперебійного розгортання програм на Amazon ECS та AWS Fargate. Читайте блог для отримання додаткової інформації.

Docker працює, забезпечуючи стандартний спосіб запуску вашого коду. Docker - це операційна система для контейнерів. Подібно до того, як віртуальна

машина віртуалізує (усуває необхідність безпосередньо керувати) серверним обладнанням, контейнери віртуалізують операційну систему сервера. Docker встановлюється на кожному сервері та надає прості команди, за допомогою яких можна створювати, запускати або зупиняти контейнери.
Сервіси AWS, такі як AWS Fargate, Amazon ECS, Amazon EKS та AWS Batch, дозволяють легко запускати та керувати контейнерами Docker у масштабі.

Використання Docker дозволяє швидше відправляти код, стандартизувати роботу додатків, плавно переміщувати код та економити гроші, покращуючи використання ресурсів. За допомогою Docker ви отримуєте один об’єкт, який може надійно працювати в будь-якому місці. Простий і зрозумілий синтаксис Docker дає вам повний контроль. Широке впровадження означає, що існує потужна екосистема інструментів та готових програм, готових до використання з Docker.

Ви можете використовувати контейнери Docker як основний будівельний матеріал для створення сучасних додатків та платформ. Docker дозволяє легко створювати і запускати архітектури розподілених мікропослуг, розгортати ваш код за допомогою стандартизованих конвеєрів безперервної інтеграції та доставки, будувати високо масштабовані системи обробки даних та створювати повністю керовані платформи для своїх розробників. Недавня співпраця між AWS та Docker полегшує вам розгортання артефактів Docker Compose на Amazon ECS.

3.2.2 Загальний опис архітектури системи

Розглянувши весь основні технології, які будуть використані в побудові архітектури, можемо приступити безпосередньо до її розробки. В даній системі використані наступні сервіси AWS, а саме: Amazon ECS, який буде використовувати для запуску Docker контейнерів серверної частини та модуля рекомендацій, що дозволяє легко та гнучко масштабувати систему і підлаштовуватися до її навантажень, а також сервіс реляційної бази даних

Amazon RDS для зберігання інформації про користувачів та сервіс для NoSQL бази даних - Amazon Redshift, для зберігання та ефективного доступу до історії покупок користувача.

В даній архітектурі запропоновано розділення на серверну та рекомендаційну систему з метою забезпечення гнучкого та зручного масштабування окремих компонентів системи, а також тому, що для оптимальної імплементації даних компонентів потрібні кардинально різні фреймворки, бібліотеки та мови програмування. До прикладу в серверна частина написана на мові програмування Java версії 11, в свою чергу рекомендаційний модуль реалізований на мові програмування Python версії 3.

Також, в архітектурному рішенні рекомендаційної системи використовується дві бази даних, різних типів та різних за призначенням. В запропонованому рішенні використовується реляційна база даних Amazon RDS, який відповідає за зберігання інформації користувачів, а саме ідентифікаційний номер користувачів, їх імена, номера телефонів, а також паролі в зашифрованому вигляді для входу в систему.

Але також існує ще інша база даних, яка використовується для зберігання та доступу до даних про історію покупок користувача. На основі цих даних рекомендацій буде надавати рекомендації, а тому очікується, що до такої бази даних будуть приходити запити одразу на велику кількість даних на всі або декілька колонок в таблиці. Виходячи з вищесказаного було обрано NoSQL базу даних колоноорієнтованого типу. В AWS підходящою за всіма параметрами - це Amazon Redshift.

На наступній діаграмі можна побачити реалізацію всієї вище описаної архітектури системи рекомендацій:

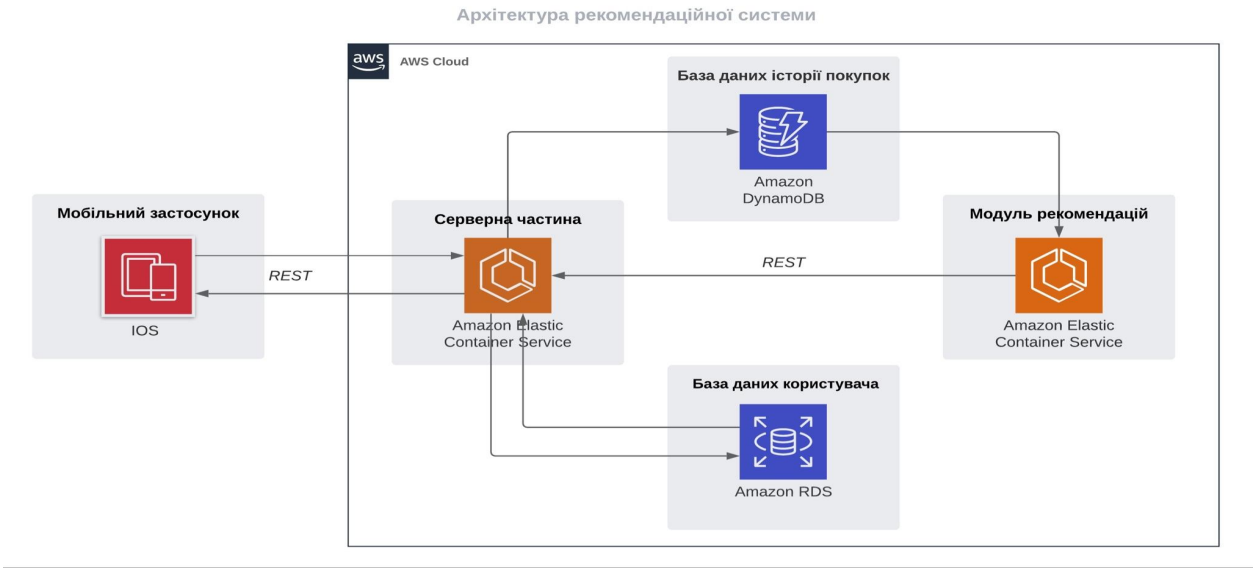

Рис.3 Архітектура рекомендаційної системи

Розглянемо декілька сценаріїв роботи даної системи. Перший сценарій - це здійснення покупки користувачем. В мобільному додатку авторизований покупець здійснює покупку товари, після цього мобільний додаток відправляє REST запит по HTTP протоколу до сервера. Сервер в свою чергу, перевіряє чи користувач авторизований за допомогою бази даних користувача і, якщо це так, зберігає інформацію про покупку до бази даних історії покупок.

Другий сценарій - це надання рекомендацій користувачеві мобільного додатка. В мобільному додатку авторизований покупець може переглядати товари, які йому рекомендує система. Для цього мобільний додаток робить запит до серверної частини, яка в свою чергу робить REST запит через HTTP протокол до модуля рекомендацій. Модуль рекомендацій за допомогою методу сингулярного розкладання матриць та даних про історію покупок даного покупця, повертає у відповідь серверу список в JSON форматі, який у свою чергу повертає результати рекомендацій назад, до мобільного додатку.

3.3 Модуль рекомендацій

3.3.1 Реалізація модуля рекомендацій

Базуючись на дослідженні зробленого в розділі 2 та архітектурі описанії в попередніх розділах реалізуємо модуль системи рекомендацій.

Для імплементації даного модуля було обрано мову програмування Python 3. Python - інтерпретована об'єктно-орієнтована мова програмування високого рівня зі строгою динамічною типізацією. Розроблена в 1990 році Гвідо ван Россумом. Структури даних високого рівня разом із динамічною семантикою та динамічним зв'язуванням роблять її привабливою для швидкої розробки програм, а також як засіб поєднання наявних компонентів. Python підтримує модулі та пакети модулів, що сприяє модульності та повторному використанню коду. Інтерпретатор Python та стандартні бібліотеки доступні як у скомпільованій, так і у вихідній формі на всіх основних платформах. В мові програмування Python підтримується кілька парадигм програмування, зокрема: об'єктно-орієнтована, процедурна, функціональна та аспектно-орієнтована. Дана мова програмування була обрана для реалізації даного модуця рекомендації тому, що вона підтримує велику кількість бібліотек для обробки та аналізу даних таких, як: Pandas, NumPy, SciPy та інші. Також варто зазначити, що мова програмування Python 3 сумісна з середовищеми розробки Jupiter Notebook, який суттєво полегшує створення прототипів програми.

Наявність таких бібліотек суттєво спрощує реалізацію різних алгоритмів в тому числі і рекомендаційних алгоритмів на базі сингулярного розкладання матриць. Розглянемо детальніше бібліотеки, які були використані в реалізації даного модуля рекомендацій.

SciPy - відкрита бібліотека високоякісних наукових інструментів для мови програмування Python. SciPy містить модулі для оптимізації, інтегрування, спеціальних функцій, обробки сигналів, обробки зображень, генетичних алгоритмів, розв'язування звичайних диференціальних рівнянь та інших задач,

які розв'язуються в науці і при інженерній розробці. Для візуалізації при використанні SciPy часто застосовують бібліотеку Matplotlib. В даний час SciPy поширюється під ліцензією BSD і його розробники спонсоруються Enthought. Дана бібліотека також містить алгоритм сингулярного розкладання матриці, який використаний в реалізації модуля. Приклад коду використання сингулярного розкладання матриці для рекомендаціїї:

```
from scipy.sparse.linalg import svds
U, sigma, Vt = svds(R_demeaned, k = 50)
```

Наступна використання бібліотека в реалізації модуля - це Pandas. Pandas - програмна бібліотека, написана для мови програмування Python для маніпулювання даними та їхнього аналізу. Вона, зокрема, пропонує структури даних та операції для маніпулювання чисельними таблицями та часовими рядами. pandas є вільним програмним забезпеченням, що випускається за трипунктовою ліцензією BSD. За допомогою цієї бібліотеки зручно здійснювати трансформацію даних, які були попередньо отримані з NoSQL бази даних історії покупок користувачі. Саме ці трансформовані дані, за допомогою Pandas, будуть використані в алгоритмі SVD від SciPy, описаного вище.

Також було використано бібліотеку NumPy. NumPy — розширення мови Python, що додає підтримку великих багатовимірних масивів і матриць, разом з великою бібліотекою високорівневих математичних функцій для операцій з цими масивами. Попередник Numpy, Numeric, був спочатку створений Jim Hugunin. Numpy — відкрите програмне забезпечення і має багато розробників. Ця бібліотека забезпечує до безлічі математичних операцій, які використовуються для надання рекомендацій товарів.

Для забезпечення доступу до модуля зі сторони сервісу бізнес логіки, було реалізовано REST API за допомогою бібліотеки Flask. Flask - мікрофреймворк для веб-додатків, створений з використанням Python. Даний

мікрофреймворк полегшує реалізацію REST сервісів.

Запит до сервісу рекомендацій виглядає так:

GET /api/v1/recommendations/{id користувача}

У відповідь рекомендаційний модуль повертає список товарів рекомендованих для користувача з ідентифікаційним номером переданим в URL запиту:

```
[
    {
        “sku”: “<унікальний ідентифікаційний номер продукту>”
        “productName”: “<назва товару>”,
        “price”: “<ціна>”
    },
    {...}
]
```

3.3.2 Опис даних для навчання

Для демонстрації роботи системи потрібні дані історії покупок користувачів. Так, як на даному етапі система ще немає доступу до реальних даних історії покупок, було використано сторонній відкритий датасет з сайту kaggle.com, який був туди завантажений компанією Instacart.

Набір даних собою реляційний набір файлів, що описують замовлення клієнтів з часом. Набір даних є анонімним та містить зразок понад 3 мільйонів замовлень на продукти від більш ніж 200 000 користувачів. Для кожного користувача наявні від 4 до 100 їхніх замовлень із послідовністю придбаних продуктів у кожному замовленні. Також наявні дані про тиждень та годину доби, коли було зроблено замовлення, та відносну міру часу між покупками. Кожен файл має однакову структуру та містить колонки: ідентифікаційний номер користувача, унікальний ідентифікатор продукту, який придбав користувач та час здійснення покупки. Ці дані були завантажені до Amazon Redshift - NoSQL бази історії покупок.

3.4 Сервіс бізнес логіки

Після визначення всіх компонентів архітектури можна описати кожен компонент системи більш детально. Почнемо з серверної частини, яка відповідає за всю бізнес логіку системи. Саме цей компонент взаємодіє з зв'язує всі без винятків модулі системи, а саме: рекомендаційний модуль, база даних користувачі, база даних їх історії покупок, а також мобільний додаток. Серверну частину можна вважати центральним компонентом в архітектурі системи.

Для реалізації даного сервісу було обрано одну з найпопулярніших об’єктно-орієнтованих мову програмування - Java 11. Java є високим рівнем, класовим, об'єктно-орієнтованим мовою програмування, яка розроблена якнайменше залежно від реалізації. Це мова програмування загального призначення, призначеного для того, щоб дозволити розробникам додатків писати один раз, запустити будь-де (Wora), що означає, що складений код Java може працювати на всіх платформах, які підтримують Java без необхідності перекомпіляції. Застосування Java, як правило, складаються до ByTecode, який може працювати на будь-якій віртуальній машині Java (JVM) незалежно від основної комп'ютерної архітектури. Синтаксис Java подібний до C і C ++, але має менше об'єктів низького рівня, ніж будь-який з них. Java Runtime надає динамічні можливості (такі як модифікація коду відображення та виконання коду), які, як правило, недоступні у традиційних складених мовах. Станом на 2019 рік Java був одним із найпопулярніших мов програмування, що використовуються відповідно до GitHub, особливо для веб-додатків клієнт-сервера, з повідомленими 9 мільйонами розробників.

Також для розробки на джаві даного веб-сервісу було обрано фреймворк Spring. Це найпоширеніший фреймворк для Java, який включає безліч бібліотек, які полегшують та прискорюють реалізацію веб-сервісів.

Для реалізації прикладних програмних інтерфейсів (Application Program Interface) в тому числі і REST API, який є головним інтерфейсом взаємодії між

даним сервісом, мобільним додатком та рекомендаційним модулем. Spring Framework робить будівництво веб-додатків швидким і без проблемним, надаючи сучасну модель веб-програмування.

Spring Framework допомагає підключати свої веб-додатки до багатьох джерела даних. Він підтримує реляційні та нереляційні бази даних та може легко інтегруватися с хмарними базами даних в тому числі і з Amazon RDS та Amazon Redshift.

В сервісі бізнес логіки використані такі компоненти Spring Framework:

Spring Core - забезпечує можливість реалізації шаблону програмування інверсії залежності, що являється стандартом для веб-сервісів реалізованих за допомогою Java;

Spring Boot - компонент полегшує конфігурацію всіх модулів Spring та пришвидшує розробку програмного забезпечення;

Spring MVC - забезпечує інструменти для реалізації тришарової архітектури сервісу;

Spring Data - дозволяє працювати з реляційними системами управління базами даних на платформі Java з використанням підключення до бази даних (JDBC), а також надає можливість інтеграції з NoSQL базами даних;

Spring Security - забезпечує ідентифікацію, аутентифікацію та авторизацію користувача в системі, а також покращує захищеність системи.

Для взаємодії з мобільним додатком реалізовано REST API, який включає в себе такі можливість обробки таких HTTP запитів:

POST /api/v1/auth/login - запит для логін користувача в систему. Приймає на вхід номер телефону користувача та пароль в форматі JSON:

{

“mobilePhoneNumber”: “<мобільний телефон користувача>”,

“password”: “< пароль для входу в систему>”

}

У відповідь повертає HTTP статус 200 OK, в разі успішного входу в систему та 401 Unauthorized в разі невдачі при вході.

GET /api/v1/recommendation/{id-користувача} - запит на отримання списку рекомендованих товарів для конкретного користувача за його ідентифікаційним номером.

У відповідь відправляє список рекомендованих товарів в форматі JSON:

```
[
    {
        “sku”: “<унікальний ідентифікаційний номер продукту>”
        “productName”: “<назва товару>”,
        “price”: “<ціна>”
    },
    {...}
]
```

GET /api/v1/products - запит на отримання списку продуктів з каталогу. У відповідь повертає аналогічний список у форматі JSON, як в попередньому запиті.

Для інтеграції з модулем рекомендацій в сервісі бізнес логіки було реалізовано REST клієнт за допомогою бібліотек Spring. Даний відправляв запити та оброблював відповіді від REST API модуля рекомендацій.

Запит до сервісу рекомендацій виглядає так:

GET http://<хост сервісу рекомендацій>/api/v1/recommendations/{id користувача}

У відповідь від рекомендаційного модуля, сервіс бізнес логіки отримує список товарів рекомендованих для користувача з ідентифікаційним номером переданим в URL запиту:

```
[
    {
```

“sku”: “<унікальний ідентифікаційний номер продукту>”

“productName”: “<назва товару>”,

“price”: “<ціна>”

},

{...}

]

Також, в даного сервісу бізнес логіки є інтеграція з реляційною базою даних Amazon RDS з інформацією користувачів, для запитів до якої використовується інтерфейс взаємодій JDBC та об'єктно-реляційне представлення, саме тому в коді системи присутні класи, які репрезентують таблиці в базі даних. В свою чергу для зв’язку з NoSQL базою даних Amazon Redshift, яка зберігає інформацію про покупки користувачів, використовуються пропрієтарний інтерфейс розроблений спеціально для цієї бази даних.

3.5 Мобільний додаток

Мобільний додаток дозволяє відслідковувати які товари наявні у магазині у даний час а також дозволяє подивитись більш детальну інформацію про той чи інший товар прямо у смартфоні. Також кожен користувач має спеціальну корзину, куди можна додати необхідну кількість товару. Для більшої безпеки, корзина розташована у backend частині проекту, тому користувач не має можливості змінювати у ній будь-які дані. Комунікація відбувається за допомогою архітектурного стилю REST. У якості клієнта виступає мобільний додаток, у якості серверу – back-end застосунок на Java.

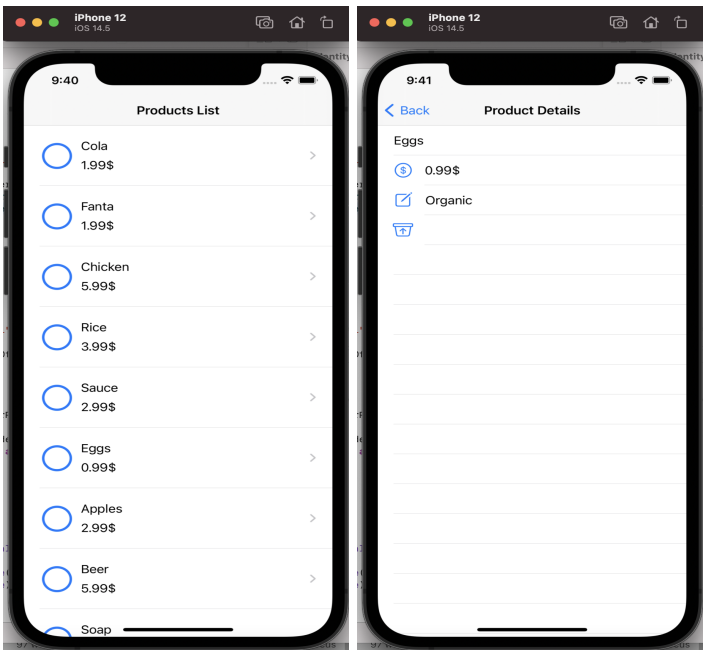

Рис. 4 Скріншоти мобільного додатку

Для додання товару у корзину використовується POST запит, який надсилає ID товару та кількість, для видалення використовується DELETE запис.

Для розробки мобільного застосунку було обрано платформу iOS. Основна причина – компанія Apple сама розробляє і засоби розробки і самі смартфони, через це процес розробки значно спрощується. Також компанія постійно випускає оновлення для операційної системи, і ці оновлення є сумісними з доволі старими смартфонами. Для даного мобільного додатку мінімальною версією є iOS 14 під управлінням якої працює 86% усіх мобільних девайсів вироблених компанією Apple. Під управлінням операційної системи Android працює занадто багато смартфонів, до того ж існує кілька паралельних версій даної ОС, наприклад для китайських смартфонів такий як Meizu або Xiaomi. Необхідність враховувати всі ці фактори значно ускладнює процес розробки. Розробка під iOS платформу ведеться мовою програмування Swift (C-подібна мова програмування розроблена компанією Apple). Swift має два основних фреймворк для розробки під iOS: UIKit та SwiftUI. Swift UI з’явився лише нещодавно і ще має багато недоліків, тож для розробки даного мобільного

застосунку було обрано фреймворк UIKit. За основу для UIKit взято дизайн паттерн MVC. UIKit надає багато основних об'єктів для застосунків і їх можна використовувати майже без модифікацій, що істотно спрощує розробку.

На рисунку нижче зображена структура типового додатку який використовує UIKit. Об'єкти моделей управляють бізнес-логікою та зберігають дані застосунку. Дані об'єкти треба писати самостійно. Також самостійно пишуться об'єкти Controller зв'язують відображення та моделі та переносять дані між ними коли це необхідно. Натомість об'єкти View які відповідають за візуальне відображення даних надаються UIKit. Усі компоненти які знаходяться на екрані наслідування від класу UIView. Також часто використовується Foundation framework який надає базові типи для моделей, такі як строки, числа, масиви та інші типи даних.

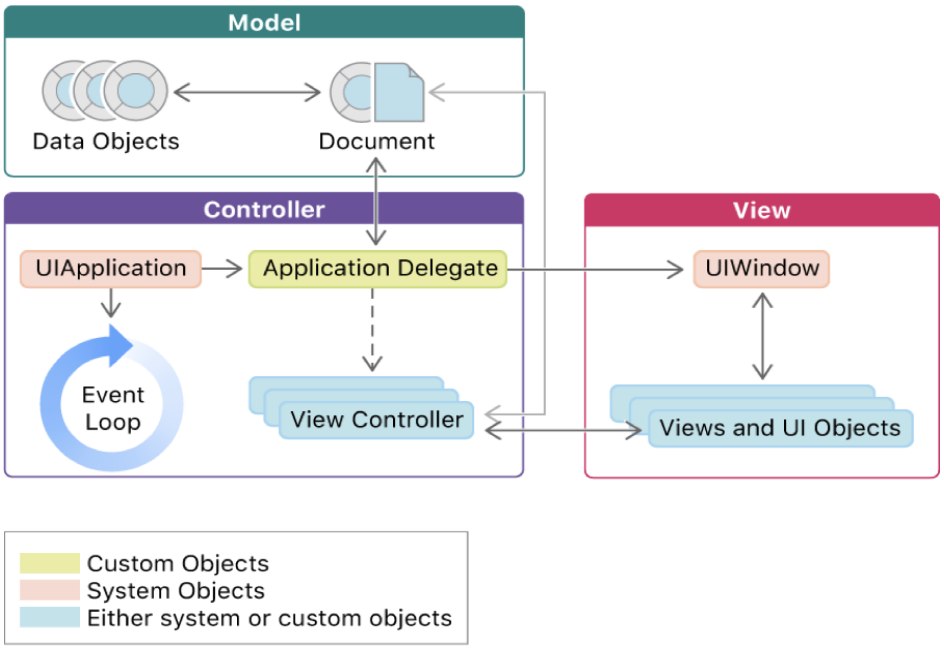

Рис. 5 Архітектура типового UIKit застосунку

Також часто використовується Foundation framework який надає базові типи для моделей, такі як строки, числа, масиви та інші типи даних. Основним об'єктом у застосунку є UIApplication у якому знаходиться основний цикл подій і який управляє всім життєвим циклом мобільного додатку.

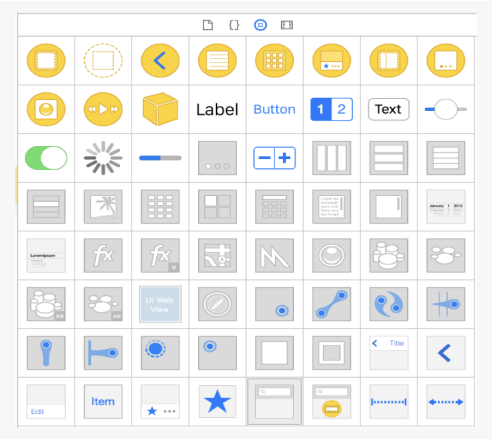

Рис. 6 Стандартні елементи інтерфейсу користувача у UIKit

Висновок

Підчас виконання даної дипломної роботи було опновано алгоритм рекомендації, який заснований методі сингулярного розкладання матриць. А також було досліджено та виявлені найефективніші методи його реалізації. Даний алгоритм було використано в реалізації системи для рекомендації продуктів.

Основним результатом роботи є розроблена система для рекомендацій продуктів, яка базується на історії покупок користувача та використовує метод сингулярного розкладання матриць. Дана система відповідає специфічним вимогам, які випливають з особливостей зберігання та обробки даних історії покупок користувачів, які використовуються для надання рекомендацій користувачу. Завдяки розробленій архітектурі, система забезпечує надійне зберігання великих об’ємів даних та швидкий доступ до них за допомогою сервісів хмарного провайдера AWS, що є важливим для оперативної роботи рекомендаційного модуля. Для зручності взаємодії з системою було розроблено мобільний iOS додаток, який забезпечує доступ до всього функціоналу системи.

Також, практичною значимістю розробленої системи, є її відносна простота горизонтального масштабування, яка забезпечує можливість роботи системи під великим навантаженням та можливість доповнення системи іншими модулями рекомендацій та реалізації додаткового функціоналу.

Саме тому подальшим логічним розвитком системи рекомендацій може бути її доповнення іншими рекомендаційними модулями, які базуються на інших алгоритмах роботи, для покращення вже наявних рекомендацій.

Список використаних джерел

[1] H. Rutishauser. Wilkinson-Reinsch, Linear Algebra, Springer Verlag. 1971.

[2] David C. Lay. Linear Algebra and Its Applications, 4th Edition.

[3] Gram–Schmidt Orthogonalization: 100 Years and More.

[4] W. Gander. Algorithms for the QR-Decomposition, Eidgen¨ossische Technische Hochschule. 1980.

[5] Gene H. Golub and Charles F. Van Loan. Matrix Computations, 3rd edition. Johns Hopkins University Press, 1996.

[6] QR Decomposition with the Gram-Schmidt Procedure.

[7] Gilbert Strang. Linear Algebra and Its Applications, 4th Edition. 1968.

[8] David Bau Lloyd N. Trefethen. Numerical Linear Algebra. 1997.

Додаток А

```
#Comparing time efficiency of each method
qrgivens <- function(A){
  m = nrow(A)
  n = ncol(A)
  Q = eye(m)
  R = A
  for (j in 1:n){
    for (i in seq(m, j + 1)){
      G = eye(m)
      res = givensrotation(R[i-1,j], R[i,j])
      c = res[1]
      s = res[2]
      mm = matrix(c(c, -s, s, c), nrow=2, ncol=2, byrow = TRUE)
      G[c(i-1, i),c(i-1, i)] = mm
      R = t(G) %*% R
      Q = Q %*% G
    }
  }
}

givensrotation <- function(a,b) {
  if (b == 0){
    c = 1
    s = 0
  }else {
    if (abs(b) > abs(a)){
      r = a/b
      s = 1 / sqrt(1 + r*r);
      c = s*r;
    } else{
      r = b/a
      c = 1 / sqrt(1 + r*r);
      s = c*r;
    }
  }
  return(c(c, s))
}

gramschmidt <- function(x) {
  #x <- as.matrix(x)
  # Get the number of rows and columns of the matrix
  n <- ncol(x)
  m <- nrow(x)

  # Initialize the Q and R matrices
  q <- matrix(0, m, n)
  r <- matrix(0, n, n)

  for (j in 1:n) {
    v = x[,j] # Step 1 of the Gram-Schmidt process v1 = a1
    # Skip the first column
    if (j > 1) {
      for (i in 1:(j-1)) {
        r[i,j] <- t(q[,i]) %*% x[,j] # Find the inner product (noted to be q^T a earlier)
        # Subtract the projection from v which causes v to become perpendicular to all columns of Q
        v <- v - r[i,j] * q[,i]
      }
```

```
    }
    # Find the L2 norm of the jth diagonal of R
    r[j,j] <- sqrt(sum(v^2))
    # The orthogonalized result is found and stored in the ith column of Q.
    q[,j] <- v / r[j,j]
  }
}

HouseholderRefl <- function(A, y) {
  n <- ncol(A)
  m <- nrow(A)
  H <- list() #to store the H_k matrices
  R <- as.matrix(A)

  if (m > n) {
    l <- n
  }
  else {
    l <- m
  }
  for (k in 1:l) {
    x <- R[k:m, k]
    e <- as.matrix(c(1, rep(0, length(x)-1)))
    v_k <- sign(x[1]) * sqrt(sum(x^2)) * e + x

    H_k <- diag(length(x)) - 2 * as.vector(v_k %*% t(v_k)) / (t(v_k) %*% v_k)
    if (k > 1) {
      H_k <- bdiag(diag(k-1), H_k)
    }
    H[[k]] <- H_k
    R <- H_k %*% R
  }

  Q <- Reduce("%*%", H)
}

QR_comp = function(A){
  t0 = Sys.time()
  qrgivens(A)
  Givens = Sys.time() - t0

  t0 = Sys.time()
  gramschmidt(A)
  Gram_Schmidt = Sys.time() - t0


  t0 = Sys.time()
  HouseholderRefl(A)
  Householder = Sys.time() - t0

    return(data.frame(Givens = as.numeric(Givens), Gram_Schmidt = as.numeric(Gram_Schmidt), Householder =
as.numeric(Householder)))
}

n <- c()
m <- c()

for (n1 in seq(1,52, by=2)){
  for (m1 in seq((n1+1),53, by=2)){
```

```
    n <- append(n, n1)
    m <- append(m, m1)

  }
}

tvec = map2(m, n, ~QR_comp(matrix(runif(.x*.y), ncol = .y)))

plotly::ggplotly(
bind_rows(tvec) %>%
  gather("func","time") %>%
  mutate(n = rep(n, 3), m = rep(m, 3)) %>%
  ggplot(aes(m, n, fill = time)) +
  geom_tile() +
  facet_grid(.~func) +
  scale_fill_gradientn(colours = rainbow(9)) +
  theme(panel.background = element_blank(),
        axis.ticks.y = element_blank(),
        axis.text.y = element_text(angle = 35, size = 5),
        axis.text.x = element_text(angle = 30, size = 5)), width = 550, heigh = 400)
```