Optimizing Amazon's Recommendation System Using Collaborative Filtering and SVD

Muhammad Rizain Firdaus - 13523164
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13523164@std.stei.itb.ac.id, icon.firdaus@gmail.com

***Abstract*— This paper presents optimizing amazon's recommendation system using collaborative filtering and SVD. The study focuses on addressing challenges in e-commerce recommendation systems, such as data sparsity and scalability, by leveraging matrix factorization techniques. By applying SVD to the user-item interaction matrix, latent factors representing user preferences and item characteristics are extracted, enabling the system to predict missing interactions and generate personalized recommendations. Using Amazon's product review dataset, the paper demonstrates how SVD improves the accuracy and efficiency of recommendations by reducing dimensionality and uncovering hidden patterns in the data. The findings highlight the critical role of CF and SVD in delivering scalable, accurate, and personalized user experiences, ultimately driving customer satisfaction and business growth.**

***Keywords*— Collaborative Filtering, Singular Value Decomposition, Recommendation System, E-commerce, Amazon**

I. Introduction

In the competitive landscape of e-commerce, delivering personalized user experiences has become a critical factor for success. Recommender systems play a central role in achieving this, as they help businesses suggest relevant products to customers based on their preferences and behavior. Amazon, one of the world's largest e-commerce platforms, exemplifies the power of such systems by utilizing them to boost user engagement and maximize sales. A key component of these systems is the application of **Singular Value Decomposition (SVD)**, a matrix factorization technique from linear algebra. SVD is particularly effective in collaborative filtering, where user-item interactions are modeled using large-scale matrices. By decomposing these matrices into latent factors, SVD identifies patterns that are not explicitly visible, such as similarities between users with seemingly different preferences or hidden relationships between products. Collaborative filtering is one of the most widely adopted and successful recommendation approaches. Unlike approaches based on intrinsic consumer and product characteristics, CF characterizes consumers and products implicitly by their previous interactions. The simplest example is to recommend the most popular products to all consumers. Researchers are advancing CF technologies in such areas as algorithm design, human-computer interaction design, consumer incentive analysis, and privacy protection [1].

Took an example from big e-commerce company such as Amazon's recommendation system, SVD works by analyzing massive datasets of user-item interactions, such as purchase history, product ratings, and browsing activity. For example:

- Matrix Representation: The data is represented as a sparse matrix, where rows correspond to users, columns correspond to products, and entries represent interactions (e.g., ratings or purchase counts).
- Decomposition: SVD decomposes this matrix into three components: user features, item features, and a diagonal matrix of singular values, reducing dimensionality while preserving key information.
- Prediction: These latent features are then used to predict a user's preference for unseen products, allowing the system to recommend items they are likely to engage with.

This paper delves into the mathematical foundation of SVD, its application in collaborative filtering, and its integration into one of the famous e-commerce recommendation architectures which the writer chooses Amazon as a sample. Furthermore, it examines how SVD addresses practical challenges in e-commerce systems, such as data sparsity and scalability, and highlights its impact on customer satisfaction and business outcomes.

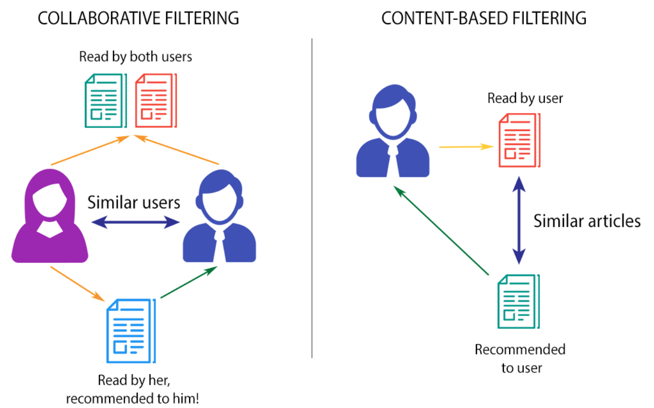


Fig 1.2 Illustration for collaborative filtering and content-based filtering scenario

Source: LinkedIn

II. Theoretical Basis

A. Fundamental of Matrix

1. Introduction to Matrices

Matrix is a very important data representation [2]. and is widely used in many fields of engineering including informatics. A matrix is a two-dimensional array of numbers arranged in rows and columns, often denoted as A. Mathematically, a matrix of size $m \times n$ has m rows and n columns, where each entry A_{ij} represents a scalar value located at the i-th row and j-th column.

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} & \cdots & A_{1n} \\ A_{21} & A_{22} & A_{23} & \cdots & A_{2n} \\ A_{31} & A_{32} & A_{33} & \cdots & A_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A_{m1} & A_{m2} & A_{m3} & \cdots & A_{mn} \end{bmatrix}$$

Fig 2.1 Matrix with the size of $m \times n$.

Source: Dummies

Matrices are fundamental in linear algebra and are used to represent and manipulate data in various domains, including physics, engineering, and computer science. Their versatility stems from their ability to capture relationships between entities and perform operations such as addition, multiplication, and transformation, which are essential in complex computations and analyses. For example, in e-commerce systems like Amazon, matrices can encode interactions between users and products, enabling structured analysis and recommendation generation.

A square matrix has an equal number of rows and columns ($m = n$). For example, a 3×3 matrix is square. Conversely, a rectangle matrix has a matrix that has a non-equal number of rows and columns ($m \neq n$). There is also a diagonal matrix, which is a special case of a square matrix where all non-diagonal elements are zero and the diagonal elements are any values. For example, as the matrix below the elements of a and b are any values except zero so it is known to be as matrix diagonal.

$$\begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}$$

$$a \neq 0,\ b \neq 0$$

Fig 2.2 Matrix diagonal with the size of 2×2 *which also a square matrix.*

Source: Dummies

Matrices also have an operational system that includes addition, subtraction, multiplication, division, scalar multiplication, and scalar division. Just like regular operational system for basic operations, matrices are also valid to use arithmetic rules. There are also special types of matrices, such as transpose matrices and identity matrices. An identity matrix, often denoted as I_n is a diagonal matrix with all diagonal entries being one. It acts as the multiplicative identity in matrix multiplication.

$$[1],\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},\ \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix},\ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Fig 2.3 Identity matrices

Transpose Matrix is a fundamental operation in linear algebra that involves flipping the matrix over its diagonal, effectively switching its rows and columns. For a given matrix A, the transpose is denoted as A^T or sometimes A'. The element in the i-th row and j-th column of A becomes the element in the j-th row and i-th column of A^T. Formally, this can be expressed as:

$$(A^T)_{ij} = A_{ji}$$

The result of transpose matrix not only the position of the values is change but possibly if the matrix is a rectangle then the size is also change (from $m \times n$ to $n \times m$).

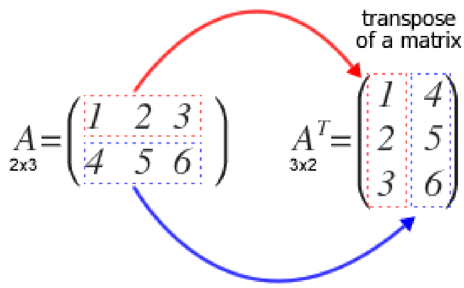


Fig 2.4 Transpose matrix example for matrix 2×3

Source: andreaminini

2. Matrix as Representation of Data

Matrix representation is a method for storing and manipulating data in a rectangular array of numbers, or functions, in rows and columns. Matrices are used to represent datasets, where each row corresponds to a sample or observation, and each column represents a feature or attribute of that sample [3]. In collaborative filtering-based recommendation systems, data is typically represented as a matrix to model the relationship between users and items (products). This matrix, commonly referred to as the **user-item matrix**, serves as the foundation for analysis and recommendation generation.

The user-item matrix is a two-dimensional representation, where rows correspond to users and columns correspond to items available on the e-commerce platform. Each element of the matrix (R_{ij}) shows the interaction or preference of user i for item j. These interactions can take the form of explicit ratings, for example: Numerical values provided by users, such as a score from 1 to 5 or star ratings.

$$R = \begin{bmatrix} 4 & 0 & 3 & 5 & 0 \\ 0 & 2 & 0 & 4 & 3 \\ 1 & 5 & 0 & 0 & 0 \end{bmatrix}$$

Fig 2.5 Example of the matrix that contains explicit user ratings from 1-5 for each item.

In this matrix R :

- The first row represents a user who has given explicit ratings to certain items (4 for item 1, 3 for item 3, and 5 for item 4).
- Zero entries indicate missing interactions, meaning that the user has not interacted with or rated those items.

B. Fundamentals of Singular Value Decomposition (SVD)

1. Introduction to Eigenvalues and Eigenvectors

Eigenvalues and eigenvectors are fundamental concepts in linear algebra, widely applied in areas such as systems of linear equations, transformations, machine learning, and matrix factorizations like Singular Value Decomposition (SVD). Given a square matrix A of size $m \times m$ an eigenvector v and an eigenvalue λ satisfy the following equation:

$$Av = \lambda v$$

An eigenvector represents the direction that remains unchanged under the transformation of a matrix A, although its magnitude may be scaled. The scaling factor associated with an eigenvector is called the eigenvalue. Eigenvalues are determined by solving the characteristic equation, given as:

$$\det(A - \lambda I) = 0$$

Where I is the identity matrix and λ represents the solutions or roots of the equation. If the matrix A is diagonalizable, it can be expressed in the form:

$$A = PDP^{-1}$$

Where P is a matrix whose columns are the eigenvectors of A, and D is a diagonal matrix containing the eigenvalues of A.

2. Singular Value Decomposition

SVD factors a $m \times n$ matrix A into matrices U, Σ, and Given a matrix A :

$$A = U\Sigma V^T$$

Here, U is an $m \times m$ orthogonal matrix whose columns are the left singular vectors, V^T is the transpose of an $n \times n$ orthogonal matrix whose rows are the right singular vectors, and Σ is an $m \times n$ diagonal matrix containing the singular values of A. These singular values are the square roots of the eigenvalues of A^TA or AA^T, and they represent the magnitudes of the matrix's transformation in each principal direction. SVD provides a way to understand the geometry of a matrix by identifying its key components, making it fundamental in applications such as dimensionality reduction, image compression, and solving linear systems.

C. Collaborative Filtering

1. Introduction to Collaborative Filtering

Collaborative Filtering (CF) is an effective method utilized in recommendation systems for e-commerce platforms such as Amazon. This method leverages user preferences or behavioral data, including purchases, searches, or product ratings, to provide personalized product recommendations [1]. In the context of e-commerce, CF plays a crucial role in enhancing user experience by suggesting relevant products, thereby driving increased sales and improving customer satisfaction [2].

Collaborative Filtering (CF) consists of two types: Item-Based Collaborative Filtering (IBCF), which computes similarities between items, and User-Based Collaborative Filtering (UBCF), which computes similarities between users. IBCF is generally more efficient than UBCF, as typical applications involve far more users than items, making the similarity matrix for IBCF more compact. Additionally, item similarity estimates are more likely to converge over time and can be precomputed and cached, unlike user similarities, which require dynamic computation at regular intervals. However, IBCF recommendations tend to be more conservative compared to UBCF.

2. Implementation of Collaborative Filtering to Amazon Recommendation System

Amazon implements Collaborative Filtering by collecting extensive data on user-product interactions. These interactions include activities such as viewing, purchasing, rating, or adding products to the shopping cart. Based on this data, CF operates through two primary approaches. The first approach, User-Based

CF, identifies users with similar preferences and recommends products that these similar users have purchased or liked. The second approach, Item-Based CF, analyzes the similarity between products based on patterns of co-purchases or co-ratings [7]. For instance, if multiple users tend to buy or rate two products together, the system identifies these products as related and recommends them accordingly [8].

To manage the vast and sparse user-product matrix—where most users interact with only a small subset of available products—Amazon employs Singular Value Decomposition (SVD). SVD reduces the dimensionality of the matrix, making it computationally feasible to uncover latent patterns such as product categories or shared user preferences [9]. For example, when a user purchases a specific book, SVD can reveal latent patterns indicating that other users with similar interests also purchased books in the same category, allowing the system to recommend those books effectively.

In practice, if a user buys a product like "Gaming Laptop X," the Item-Based CF system identifies that other users who purchased "Gaming Laptop X" also bought complementary items such as "Gaming Mouse Y" and "Gaming Headset Z." These products are then recommended to the user. Similarly, through User-Based CF, if a user's shopping history aligns with another user who purchased "Mechanical Keyboard W," the system recommends the keyboard to the first user [7].

The advantages of CF in Amazon's recommendation system are significant. This method provides highly personalized suggestions by analyzing individual user preferences based on historical data [5]. Furthermore, with the application of SVD, Amazon can efficiently handle millions of users and products without compromising the accuracy of its recommendations [9]. Collaborative Filtering also boosts sales by encouraging cross-selling and upselling through relevant product suggestions, such as items frequently bought together. Additionally, CF enhances user engagement by providing tailored recommendations that keep customers interested and satisfied with the platform [8].

Despite its effectiveness, Collaborative Filtering faces certain challenges. One of these is sparsity, as the user-product matrix is often highly sparse due to the limited interactions of users with the majority of products [6]. To overcome this, SVD is employed to reduce the matrix's dimensionality and extract latent features. Another challenge is the cold start problem, where new users or products lack sufficient data for accurate recommendations. This issue can be mitigated by combining CF with content-based methods or by incorporating metadata about the products [7].

In conclusion, Collaborative Filtering, particularly when augmented with Singular Value Decomposition (SVD), forms the backbone of recommendation systems in e-commerce platforms like Amazon. This approach enables the delivery of highly accurate and personalized recommendations, efficiently handles large-scale data, and significantly enhances customer experience while driving platform revenue [5][9].

III. Developing Product Recommendation System Based Using Amazon Datasets

A. Collecting Datasets

To build a collaborative filtering-based product recommendation system using Singular Value Decomposition (SVD), it is crucial to collect relevant datasets that capture user interactions with products. In this study, the Amazon Product Review Dataset, available from publicly accessible sources such as Kaggle and Amazon's open data portal, will be used. This dataset includes detailed information such as user ratings, product metadata, and review text, which are essential for generating personalized product recommendations through collaborative filtering.

For this project datasets that will be chosen to be a sample is an open source datasets gained from an open source datasets provider called Kaggle. The dataset itself has over 7824482 rows and 4 column (User Id, Product Id, Rating, and Timestamp).

```python
import kagglehub

# Download latest version
path = kagglehub.dataset_download("
saurav9786/amazon-product-reviews")

print("Path to dataset files:", path)
```

Fig 3.1 Importing datasets
Source: Author's documents

```
AKM1MP6P0OYPR,0132793040,5.0,1365811200
A2CX7LUOHB2NDG,0321732944,5.0,1341100800
A2NWSAGRHCP8N5,0439886341,1.0,1367193600
A2WNBOD3WNDNKT,0439886341,3.0,1374451200
A1GI0U4ZRJA8WN,0439886341,1.0,1334707200
A1QGNMC6O1VW39,0511189877,5.0,1397433600
A3J3BRHTDRFJ2G,0511189877,2.0,1397433600
A2TY0BTJOTENPG,0511189877,5.0,1395878400
A34ATBPOK6HCHY,0511189877,5.0,1395532800
A89DO69P0XZ27,0511189877,5.0,1395446400
...
```

Fig 3.2 Datasets.csv snippets contain User ID, Product ID, Rating, and Timestamp.
Source: Author's documents

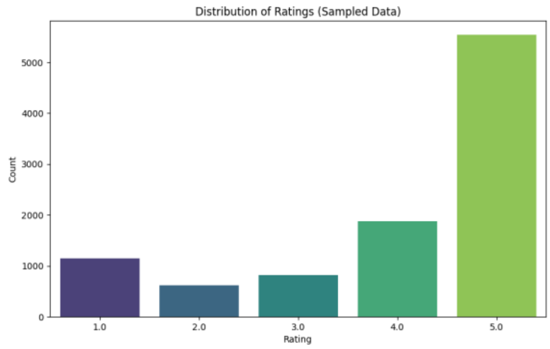


Fig 3.3 Datasets distribution based on user product review (before datasets preprocessing)
Source: Author's documents

B. Preprocessing Dataset

Load Datasets

```python
# Utilities
import math, random, warnings
from time import time
from datetime import datetime
from collections import defaultdict
from IPython.core.interactiveshell import InteractiveShell

# Mathematical calculation
import numpy as np
from scipy.sparse.linalg import svds
from sklearn import model_selection
from sklearn.metrics.pairwise import cosine_similarity

# Data handling
import pandas as pd

# Data Visualization
import matplotlib.pyplot as plt
import seaborn as sns

# scikit-surprise recommender package
from surprise import SVD, KNNWithMeans
from surprise import Dataset, Reader, accuracy
from surprise.model_selection import train_test_split, GridSearchCV
from surprise.prediction_algorithms.baseline_only import BaselineOnly
```

Fig 3.4 Import libraries needed to process datasets and developing recommendations
Source: Author's documents

```python
# Load the dataset into a Pandas dataframe called ratingsand skip any lines that return
an error
ratings = pd.read_csv('ratings_Electronics.csv',
                      names=['userId', 'productId', 'rating', 'timestamp'],
                      error_bad_lines=False,
                      warn_bad_lines=False)

# Save an original copy of the dataframe
ratings_original = ratings.copy(deep=True)
```

Fig 3.5 Preprocessing datasets
Source: Author's documents

	userId	productId	rating	timestamp
0	AKM1MP6P0OYPR	0132793040	5.000	1365811200
1	A2CX7LUOHB2NDG	0321732944	5.000	1341100800
2	A2NWSAGRHCP8N5	0439886341	1.000	1367193600
3	A2WNBOD3WNDNKT	0439886341	3.000	1374451200
4	A1GI0U4ZRJA8WN	0439886341	1.000	1334707200

Fig 3.6 4 head datasets after preprocessed
Source: Author's documents

1. Drop The Duplicates and Make The Timestamp Readable

Duplicate entries are among the most pervasive issues in raw datasets. They may arise from various sources, including system errors, manual data entry mistakes, or the integration of multiple datasets.

```python
# Get a dataframe consisting only of ratings that are duplicated
rating_combination = ['userId', 'productId']
ratings[ratings.duplicated(subset=rating_combination, keep=False)].
sort_values(rating_combination).head()

# ratings.drop_duplicates(subset=['userId','productId','rating'],inplace=Tru
e)
```

Fig 3.7 Drop duplicates process
Source: Author's documents

```python
# Convert the timestamp column to a readable date time format
ratings['timestamp'] = ratings.timestamp.apply(lambda ts: datetime.
utcfromtimestamp(ts).strftime('%Y-%m-%d %H:%M:%S'))
# Convert the datatype to datetime
ratings['timestamp'] = pd.to_datetime(ratings['timestamp'])
ratings.head()
ratings.info()
```

Fig 3.8 Make the timestamp from distorted metrics error into a readable timestamp
Source: Author's documents

	userId	productId	rating	timestamp
0	AKM1MP6P0OYPR	0132793040	5.000	2013-04-13
1	A2CX7LUOHB2NDG	0321732944	5.000	2012-07-01
2	A2NWSAGRHCP8N5	0439886341	1.000	2013-04-29
3	A2WNBOD3WNDNKT	0439886341	3.000	2013-07-22
4	A1GI0U4ZRJA8WN	0439886341	1.000	2012-04-18

Fig 3.9 4 heads dataset after preprocessed
Source: Author's documents

After the datasets are preprocessed, the datasets are now ready to be used as a part of developing a recommendation system using a collaborative filtering technique.

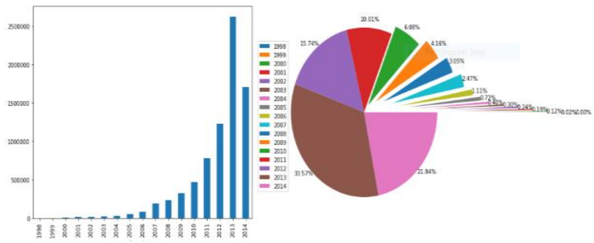


Fig 3.9 Visualization for datasets after being preprocessed
Source: Author's documents

Build Collaborative Filtering Model

1. Creating the Sparse Matrix
   Collaborative filtering leverages user-item interaction data to predict user preferences. SVD is applied to decompose the user-item matrix, capturing latent factors that represent user and item characteristics. These latent factors are then used to predict missing entries in the matrix.

```python
# Create the User-Item sparse matrix
user_item = ratings.pivot(index='userId', columns='productId', values='
rating').fillna(0)
print('Shape of User-Item sparse matrix:', user_item.shape)
user_item.head()
```

Shape of User-Item sparse matrix: (1540, 48190)

Out[33]:

productId	0594451647	0594481813	0970407998	0972683275	1400501466	1400501520	1400501776	1400532620	140
userId									
A100UD67AHFODS	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	
A100WO06OQR8BQ	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	
A105S56ODHGJEK	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	
A105TOJ6LTVMBG	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	
A10AFVU66A79Y1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	

5 rows × 48190 columns

Fig 3.10 Creating the sparse matrix

Source: Author's documents

2. Calculate user-user similarity and item-item similarity
   – **User-User Similarity**: Helps identify users with similar preferences or behaviors (e.g., users who rate similar items in a similar way). Recommendations for a target user are then derived from items preferred by these similar users.
   – **Item-Item Similarity**: Identifies items that are rated or interacted with similarly by users. If a user likes or interacts with an item, they are likely to prefer similar items based on this similarity.

```python
# Calculate the user-user similarity
user_similarity = cosine_similarity(user_item)
np.fill_diagonal(user_similarity, 0)
user_similarity_df = pd.DataFrame(user_similarity,index=user_item.index,
columns=user_item.index)
user_similarity_df.head()
```

```python
# Calculate the item-item similarity
item_similarity = cosine_similarity(item_user)
np.fill_diagonal(item_similarity, 0)
item_similarity_df = pd.DataFrame(item_similarity, index=item_user.index, columns=item_user.index)
item_similarity_df.head()
```

userId	A100UD67AHFODS	A100WO06OQR8BQ	A105S56ODHGJEK	A105TOJ6LTVMBG	A10AFVU66A79Y1	A10H24TDLK2VDP	A10NMELR4
userId							
A100UD67AHFODS	0.000	0.011	0.000	0.015	0.026	0.000	
A100WO06OQR8BQ	0.011	0.000	0.013	0.016	0.009	0.010	
A105S56ODHGJEK	0.000	0.013	0.000	0.000	0.022	0.065	
A105TOJ6LTVMBG	0.015	0.016	0.000	0.000	0.000	0.000	
A10AFVU66A79Y1	0.026	0.009	0.022	0.000	0.000	0.021	

5 rows × 1540 columns

productId	0594451647	0594481813	0970407998	0972683275	1400501466	1400501520	1400501776	1400532620	1400532655	140053271X	...
productId											
0594451647	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	...
0594481813	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	...
0970407998	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	...
0972683275	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.261	0.000	...
1400501466	0.000	0.000	0.000	0.000	0.000	0.539	0.421	0.000	0.000	0.523	...

5 rows × 48190 columns

Fig 3.11 Calculating user-user similarity and item-item similarity (with the results)

Source: Author's documents

3. Calculate the n-neighborhood based on the user-user and item-item similarity
   An *n-neighborhood* (or k-nearest neighbors) refers to selecting the *n* most similar users or items for making recommendations.

```python
# Method to find top N neighbors
def find_n_neighbors(df,n):
    order = np.argsort(df.values, axis=1)[:, :n]
    df = df.apply(axis=1, func=lambda x: pd.Series(x.sort_values(ascending=False).iloc[:n].index,
                                          index=['top{}'.format(i) for i in range(1, n+1)]))
    return df

# Find 10 neighbors of each user
user_10_neighbors = find_n_neighbors(user_similarity_df, 10)
user_10_neighbors.head(10)

# Find 10 neighbors of each item
item_10_neighbors = find_n_neighbors(item_similarity_df, 10)
item_10_neighbors.head(10)
```

	top1	top2	top3	top4	top5	top6	top7	top8	top9	
productId										
0594451647	B005B47AIU	B005OIB714	B0076AUCKU	B0058FN6V2	B0073V1NPO	B000MYOJMY	B00AV1UWWY	B005J8OBJE	B004EHSP3Y	B0
0594481813	B00D30UXBI	B0009YFS4A	B00B5E79KQ	B00C80UFJQ	B00CBDUD60	B009WESJEY	B007C2IKQU	B007NNTLK2	B001QSDSGQ	B0
0970407998	B00009R8D3	B00078Y24K	B001E9BIU2	B001EYU6Q4	B0015IOUK2	B000XALL8W	B001ADUCTU	B005JY5XK4	B001CT00AS	B0
0972683275	B007J4BOWI	B008U25LI6	B003ICXCZM	B004HW67MW	B003GZD81Q	B00022P00M	B001TLEXES	B00681RUR6	B00009UH9J	B0
1400501466	B0049QV65O	B00BC4J5O0	B0098JS19G	B002LITT3S	B000GT6HAM	B000I46TFW	B002WV6LA8	B00462RU60	B004P1ITV2	B0(
1400501520	B0030LKGAU	B0053EVW1Y	B004HO58NO	B002YIG8BQ	B004P1ITV2	B00462RU60	B004P1ITU8	B000GT6HAM	B00009W3U9	B0
1400501776	B00462RU60	B003D2BS4E	B0053EVVTW	B004P1ITV2	B004P1ITU8	B002YIG8BQ	B005KHFF2Q	B000FSILBM	B000GT6HAM	
1400532620	B00BGTN06A	B006K59QRC	B002NFZSL6	B007AHQMTO	B00004WHFL	B008J91VEA	B0010KWCWI	B00AEFFLJQ	B00GQTDOC0	B(
1400532655	B005JHF5RM	B004NTO2I0	B0034XDTF8	B000E39V9E	B0003009E4	B002V92X9Y	B000MNA082	B007STXHTM	B001S0RTXM	B0
140053271X	B000EXWJ66	B000GT6HAM	B008I6672I	B0044R1VY4	B002LITT3S	B005KHFF2Q	B0030LKGAU	B0053EVVTW	B0053EVW1Y	B0

top3	top4	top5	top6	top7	top8	top9	to
A3CG93783LP0FO	A2NOW4U7W3F7RI	A2LB4FURZG6CUV	A2DG63DN704LOI	AT53ZTTO707MB	A2XX2A4OJCDNLZ	A33XMCG1UFDSY2	A2YW6XOID7G\
A2Y3WWPUKIJ59I	A298GL2D0BHGKZ	A1CST2WUA32GP0	A15X02BEGGFEOW	A3D822N1K2IAQD	A2L4ZGN7GZJ95T	A18S2VGUH96CV5	AXP8GO5Q3A
ANTN61S4L7WG9	A2LF16F0KX9L7P	A2UKE7GIVC7XFW	AJX9A4Q19UTJU	A2R1HUYHXV7H18	A3J3ZHGDUDPCFL	A1WXG10FJ353K0	A20L63B48RPS
\2XXBZPQT5EXHV	A1NZLRAZJGD99W	A1D27BCSYV7VWH	AVTJEWTCJ3PBE	AAK6SOEJY30YG	A2GKMXRLI7KLFP	A2VCX0AWCPQ13P	AKSFZ4G1AX
2PMR2PIGWKCQ9	A4H4KYSM2KQ85	A2TVH2OBNXYXHV	A84FLVDK2R5JF	A2UOHALGF2X77Q	A24RCBRDXRXR0Y	A2IFKH3TJ10387	A3JLO0XFM75Q
A3V2EZ6MA32FF6	A3SP7T2PZ3HSDE	AO09RWV40IX78	A3J3ZHGDUDPCFL	A3QNQQKJTL76H0	A1RL1OTV8UMVD	A19W47CXJJP1MI	A3LDLV1GWQJL
A3F7USIDJBR8WU	A3GX0FAMEXV6FB	A32HSNCNPRUMTR	A1L5TA5UQF7EWE	A3223W5IROMYTY	A1ZPUEHF4AES86	A17W0GMB0YY83M	A2HV76MYH7UI
AZ8XSDMIX04VJ	AN9CP6J4JF91X	AJEOFP6ZWY0MK	A2HRHF83I3NDGT	A24P4E3RJ6AX94	A28621TSIRSEOG	A1VLE2SH9J8WYS	A2A4X545ERLS
2MJ8OL2FYN7CW	A1CMD08Z49PGKQ	A2BLFCOPSMBOZ9	A2CWIYIETNBAK3	A1MCH5RXDOH87H	A28X0LT2100RL1	A1PI8VBCXXSGC7	AJX9A4Q19U
1TQBAHI3M4Z8Q	A87N6UTYA6NO8	AZ8XKUH4AIW3X	A3HCMJOGR8SAF2	A3IRA0BHI9NE9U	A19N3S7CBSU6O7	A1L1N3J6XNABO2	AYNAH993VD

Fig 3.12 Similarity correlation using n-neighborhood (or k-nearest neighbors)

Source: Author's documents

The combination of similarity search and the *n-neighborhood* approach ensures the recommendations are both computationally efficient and personalized, catering to the user's preferences effectively.

4. Generate the recommendation system using CF model

   Once the similarity computations between users or items are completed and the *n-neighborhood* is identified, Collaborative Filtering (CF) employs these relationships to generate personalized recommendations. This process involves aggregating the preferences of similar users (User-Based CF) or drawing inferences from similar items (Item-Based CF) to predict and recommend items that align with the target user's interests.

```python
# Method to Recommend the items with the highest predicted ratings
def recommend_items(userId, orig_df, preds_df, top_n):
    # Get and sort the user's ratings
    sorted_user_ratings = orig_df.loc[userId].sort_values(ascending=False) #sorted_user_ratings
    sorted_user_predictions = preds_df.loc[userId].sort_values(ascending=False) #sorted_user_predictions

    # Prepare recommendations
    recommedations = pd.concat([sorted_user_ratings, sorted_user_predictions], axis=1)
    recommedations.index.name = 'Recommended Items'
    recommedations.columns = ['user_ratings', 'user_predictions']

    # Take the products which user has NOT rated
    recommedations = recommedations.loc[recommedations.user_ratings == 0]
    recommedations = recommedations.sort_values('user_predictions', ascending=False)
    return recommedations.head(top_n)
```

```python
# Find recommendation for couple of users using UBCF
find_recom = {'A100UD67AHFODS': 6}   # This list is user, top_n recommendat
for user in find_recom:
    print("Top %d recommendations for the userId: %s" %(find_recom[user],use
    recommend_items(user, user_item, user_prediction, find_recom[user])
    print("\n")
```

Fig 3.13 Product recommendation based on CF model
Source: Author's documents

Based on the testing for the user_id given ("A100UD67AHFODS"), the result are shown as below:

Top 6 recommendations for the userId: A100UD67AHFODS

Recommended Items	user_ratings	user_predictions
B003ES5ZUU	0.000	0.873
B007WTAJTO	0.000	0.684
B0088CJT4U	0.000	0.509
B00G4UQ6U8	0.000	0.485
B002V88HFE	0.000	0.463
B00829THK0	0.000	0.462

Fig 3.14 Product recommendations results
Source: Author's documents

5. Generate the recommendation system using the SVD model

The implementation of a recommendation system using the Singular Value Decomposition (SVD) model involves leveraging matrix factorization to predict user preferences for items. The process begins with decomposing the user-item interaction matrix into three components: U, representing user-specific latent features; Σ, a diagonal matrix of singular values indicating the importance of latent features; and V^T, capturing item-specific latent features. By reconstructing the matrix through $U.\Sigma.V^T$, the system can estimate missing values, effectively predicting user-item interactions. These predictions enable the system to recommend items by identifying those with the highest predicted ratings for each user that they have not interacted with. Additionally, evaluating the accuracy of these predictions using metrics such as Root Mean Square Error (RMSE) ensures the model's reliability. The SVD-based recommendation system efficiently captures latent patterns within the data, providing scalable and personalized suggestions tailored to user preferences.

```python
# Trying out different latent factors
svd_list = [svds(user_item, k=k) for k in [100,250,500,750,1000]]
pred_list = [pd.DataFrame(np.dot(np.dot(svd[0], np.diag(svd[1])), svd[2]),
                          index=user_item.index,
                          columns=user_item.columns) for svd in svd_list]
RMSE_list = [round((((user_item.mean() - pred.mean()) ** 2).mean() ** 0.5), 5) for
 pred in pred_list]
RMSE_list

# Singular Value Decomposition
U, sigma, Vt = svds(user_item, k=50)
# Construct diagonal array in SVD
sigma = np.diag(sigma)

# Print the shape of the decomposed matrices
print('Shape of the Left Singular matrix           :', U.shape)
print('Shape of the Latent Factor Diagonal matrix :', sigma.shape)
print('Shape of the Right Singular matrix          :', Vt.shape)
U.shape, sigma.shape, Vt.shape

# Predicted ratings
svd_prediction = pd.DataFrame(np.dot(np.dot(U, sigma), Vt), index=user_item.index,
columns=user_item.columns)
svd_prediction.head()

# Find recommendation for couple of users
find_recom = {'A100UD67AHFODS': 6}   # This list is user, top_n recommendation dict.
for user in find_recom:
    print("Top %d recommendations for the userId: %s" %(find_recom[user],user))
    recommend_items(user, user_item, svd_prediction, find_recom[user])
    print("\n")
```

Fig 3.15 Product recommendations based on SVD model
Source: Author's documents

```python
# Method to calculate RMSE for different model
def calculate_rmse(orig_df, preds_df):
    rmse_df = pd.concat([orig_df.mean(), preds_df.mean()], axis=1)
    rmse_df.columns = ['Avg_actual_ratings', 'Avg_predicted_ratings']
    RMSE = round(np.sqrt(((rmse_df.Avg_actual_ratings - rmse_df.
Avg_predicted_ratings) ** 2).mean()), 5)
    print('\nRMSE for this recommender model = {}\n'.format(RMSE))
    return rmse_df.head()

calculate_rmse(user_item, user_prediction)

calculate_rmse(user_item, item_prediction)

calculate_rmse(user_item, svd_prediction)
```

Fig 3.16 RMSE calculation
Source: Author's documents

Based on the SVD models given the results are shown as below:

```python
# Find recommendation for couple of users
find_recom = {'A100UD67AHFODS': 6}
for user in find_recom:
    print("Top %d recommendations for the userId: %s" %(find_recom[user],user))
    recommend_items(user, user_item, svd_prediction, find_recom[user])
    print("\n")
```

Top 6 recommendations for the userId: A100UD67AHFODS

Recommended Items	user_ratings	user_predictions
B0019EHU8G	0.000	1.407
B003ES5ZUU	0.000	1.097
B007OY5V68	0.000	0.987
B000JMJWV2	0.000	0.946
B009SYZ8OC	0.000	0.848
B00DTZYHX4	0.000	0.745

Fig 3.17 Product recommendations results
Source: Author's documents

V. CONCLUSION

This paper explores the use of Collaborative Filtering (CF) enhanced with Singular Value Decomposition (SVD) in developing a recommendation system for Amazon's e-commerce platform. By leveraging SVD, the recommendation system efficiently addresses challenges such as data sparsity and scalability, extracting latent factors to uncover hidden relationships in user-item interactions. The system demonstrates the ability to predict user preferences accurately, providing personalized and relevant recommendations. Through the application of SVD, the study emphasizes the importance of matrix factorization techniques in handling large-scale datasets and delivering impactful customer experiences. The integration of CF and SVD not only enhances user engagement but also drives revenue growth by encouraging cross-selling and upselling opportunities. Future work could explore hybrid models combining CF with other techniques to overcome limitations like the cold start problem and further improve recommendation accuracy.

VI. Appendix

For the GitHub source code you can access this link.

VII. Acknowledgment

First and foremost, I would like to express my deepest gratitude to God Almighty for His endless blessings, guidance, and strength throughout the process of completing this paper. Without His grace, this work would not have been possible. I am profoundly thankful to Mr. Dr. Ir. Rinaldi Munir, M.T., and Mr. Arrival Dwi Sentosa, S.Kom., M.T., for their invaluable guidance, support, and expertise. Their insightful feedback and encouragement have been instrumental in shaping this work and enriching my understanding of the subject matter.

Additionally, I extend my heartfelt appreciation to my friends and colleagues who have provided unwavering support, constructive suggestions, and collaborative spirit during this journey. Their encouragement and shared dedication have been a source of inspiration and motivation. Thank you all for contributing to the successful completion of this paper.

References

[1] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "A comparison of collaborative-filtering recommendation algorithms for e-commerce," *Proceedings of the 1st ACM Conference on Electronic Commerce*, 2000.

[2] R. Munir, "Aljabar Geometri: Review Matriks," *Institut Teknologi Bandung (ITB)*, 2023. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2023-2024/Algeo-01-Review-Matriks-2023.pdf. [Accessed: Dec. 31, 2024].

[3] CK-12 Foundation, "Matrices to represent data," in *CK-12 Precalculus Concepts*, 2nd ed., [Online]. Available: https://flexbooks.ck12.org/cbook/ck-12-precalculus-concepts-2.0/section/8.3/primary/lesson/matrices-to-represent-data-pcalc/. [Accessed: Dec. 31, 2024].

[4] R. Munir, "Singular Value Decomposition: Bagian 1," 2023. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2023-2024/Algeo-21-Singular-value-decomposition-Bagian1-2023.pdf. [Accessed: Dec. 31, 2024].

[5] S. Khan, J. Ali, and S. Ullah, "Recommendation System for E-Commerce Using Collaborative Filtering and Machine Learning," *International Journal of Advanced Computer Science and Applications*, vol. 12, no. 5, pp. 560-567, 2021.

[6] C. Maklin, "Model Based Collaborative Filtering — SVD," *Medium*, 2019. [Online]. Available: https://medium.com/@corymaklin/model-based-collaborative-filtering-svd-19859c764cee. [Accessed: Dec. 31, 2024].

[7] A. Tam, "Using Singular Value Decomposition to Build a Recommender System," *Machine Learning Mastery*, 2021. [Online]. Available: https://machinelearningmastery.com/using-singular-value-decomposition-to-build-a-recommender-system/. [Accessed: Dec. 31, 2024].

[8] R. Gupta, "How Singular Value Decomposition (SVD) is used in Recommendation Systems — Clearly Explained," *Medium*, 2020. [Online]. Available: https://medium.com/@ritik_gupta/how-singular-value-decomposition-svd-is-used-in-recommendation-systems-clearly-explained-201b24e175db. [Accessed: Dec. 31, 2024].

[9] D. A. Galron, Y. M. Brovman, J. Chung, M. Wieja, and P. Wang, "Deep Item-based Collaborative Filtering for Sparse Implicit Feedback," *arXiv preprint arXiv:1812.10546*, 2018. [Online]. Available: https://arxiv.org/abs/1812.10546. [Accessed: Dec. 31, 2024].

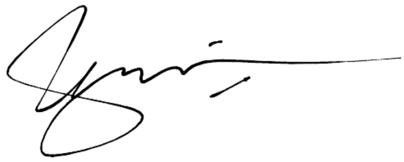
[10] S. Kumar, "Amazon Product Reviews," Kaggle, 2024. [Online]. Available: https://www.kaggle.com/datasets/saurav9786/amazon-product-reviews?resource=download. [Accessed: Dec. 31, 2024].

.

Statement

Hereby, I declare that the paper I have written is my own work, not an adaptation or translation of someone else's paper, and not plagiarism.

Bandung, January 1 2025



Muhammad Rizain Firdaus (13523164)