A SMART HYBRID ENHANCED RECOMMENDATION AND PERSONALIZATION ALGORITHM USING MACHINE LEARNING

A Project

Presented to the

Faculty of

California State University,

San Bernardino

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Computer Science

---

by

Aswin Kumar Nalluri

May 2024

A SMART HYBRID ENHANCED RECOMMENDATION AND PERSONALIZATION ALGORITHM USING MACHINE LEARNING

A Project

Presented to the

Faculty of

California State University,

San Bernardino

---

by

Aswin Kumar Nalluri

May 2024

Approved by:

Dr. Yan Zhang, Advisor, Computer Science and Engineering

Dr. Yunfei Hou, Committee Member

Dr. Jennifer Jin, Committee Member

ABSTRACT

In today's age of streaming services, the effectiveness and precision of recommendation systems are crucial in improving user satisfaction. This project introduces the Smart Hybrid Enhanced Recommendation and Personalization Algorithm (SHERPA) a cutting-edge machine learning approach aimed at transforming how movie suggestions are made. By combining Term Frequency Inverse Document Frequency (TF-IDF) for content based filtering and Alternating Squares (ALS) with Weighted Regularization for filtering SHERPA offers a sophisticated method for delivering tailored recommendations.

The algorithm underwent evaluation using a dataset that included over 50 million ratings from 480,000 Netflix users encompassing 17,000 movie titles. The performance of SHERPA was meticulously compared to traditional hybrid models demonstrating a 70% enhancement in prediction accuracy based on Root Mean Square Error (RMSE) metrics during training, testing and validation phases.

These findings highlight SHERPAs capability to understand and cater to users' subtle preferences representing an advancement in personalized recommendation systems.

ACKNOWLEDGEMENTS

I owe a heartfelt thanks to my advisor Dr. Yan Zhang for her wisdom, continuous support, and encouragement for my project "Smart Hybrid Enhanced Recommendation and Personalization Algorithm (SHERPA)".

I am sincerely thankful to Dr. Yunfei Hou and Dr. Jennifer Jin for agreeing to be part of my committee and for believing in me for the successful completion of my project. Your insights have helped me, and your willingness to share your expertise has deeply enriched my work.

I am also thankful to High Performance Computing Program team at California State University San Bernardino for utilizing the HPC resources to finalize my results.

A special note of gratitude to my family and friends, whose encouragement has been my source of strength and motivation throughout the project.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER ONE

INTRODUCTION

Background

In recent years personalized recommendation systems have become really popular because of the increasing presence of online shopping platforms, social networks and streaming services. Think about the last time you tried to pick a movie on a streaming site. Tough, right? That's because the engines behind those "Recommended for You" lists have a tough job. They mostly just look at what you've already watched (that's collaborative filtering) or suggest stuff based on movie genres you seem to like (content-based filtering) [13]. But often, they end up showing you more of the same, making it hard to stumble upon something new and exciting. Here's where we need a smarter approach, one that really gets what you're in the mood for by blending different tech tricks from the world of machine learning and introduce us to new stuff we'll actually like.

Significance

In the realm of streaming platforms, the key to success hinges on engaging and delighting audiences. A vital ingredient in achieving this is providing movie recommendations that captivate viewers like a touch of magic. Getting recommendations right can make users stick around longer and even recommend the service to friends. It's big business, with the power to shape the

streaming wars. That’s why nailing those suggestions by understanding what viewers really want to watch next, not just what an algorithm thinks they should be crucial. It’s about turning casual watchers into superfans who can’t wait to see what they’ll discover next [5].

Purpose

This project introduces the Smart Hybrid Enhanced Recommendation and Personalization Algorithm (SHERPA) with the goal of revolutionizing how movie suggestions are made. SHERPA combines filtering, content based filtering and advanced machine learning to provide tailored accurate personalized content recommendations [11]. Our aim is to simplify the process of discovering your film. We're blending techniques to align movies, with your preferences not just based on what you've already seen. The focus is on creating a journey of exploring content that resonates with you because ultimately every movie night should be about discovering something that hits the spot. SHERPA is here to shake things up ensuring that finding your next favorite movie is a click eliminating the need, for endless scrolling.

CHAPTER TWO

LITERATURE SURVEY

Traditional Machine Learning Approaches

The traditional machine learning (ML) approaches in recommendation systems primarily concentrate on collaborative filtering and content-based filtering strategies [13]. Collaborative filtering anticipates user preferences by analyzing interactions and drawing insights from users' behavior. While this technique is commonly used for its simplicity and effectiveness it often encounters challenges with new users and sparsity in user-item interactions [2]. On the content-based filtering suggests items based on their features and user preferences emphasizing the items metadata [1]. Nonetheless this method may result in diversity in recommendations as it tends to recommend items, to those already interacted with by the user.

Modern Machine Learning Approaches

Recent modern advancements recommendation systems have made progress in overcoming limitations. These advancements involve using machine learning techniques like Singular Value Decomposition (SVD) to analyze user item interactions and predict ratings revealing factors [6]. Furthermore, new algorithms such as Alternating Least Squares (ALS) with Weighted

Regularization have enhanced filtering by giving importance to known interactions and incorporating regularization to prevent overfitting [3].

By combining these approaches models that blend elements of both content based and collaborative methods have been developed. These hybrid systems provide recommendations by considering both user behavior and content characteristics. For instance, (SHERPA) Smart Hybrid Enhanced Recommendation and Personalization Algorithm integrates TF-IDF for content analysis with ALS featuring Weighted Regularization to enhance insights [15]. This integration not only improve recommendation accuracy also offers a deeper understanding of user preferences and content relevance paving the way for a new era in recommendation systems.

CHAPTER THREE

DATA PREPARATION

Data Collection

In our project we split the data into two parts: the Movie Titles Dataset and the Movie Ratings Dataset. The movie ratings dataset consists of than 50 million ratings from 480,000 Netflix users that were carefully chosen. These ratings cover 17,000 movie titles. Were gathered between October 1998 and December 2005 encompassing all ratings given during that timeframe [8], [10]. Each rating ranges from 1 to 5 stars to represent customer opinions. To ensure customer privacy unique customer IDs have been anonymized. Additionally, the dataset includes details such as the rating date, movie title, release year and corresponding movie ID, for each record.

1. Movie Titles Dataset File Description

Information on movies is contained in the 'movie_titles.csv' file [5], formatted as follows:

- Movie ID range sequentially from 1 to 17770.
- Released Year spans from 1890 to 2005 (in YYYY format) and may correspond to the DVD release date rather than the theatrical release.
- Movie Names are the Netflix movie titles and may not align with titles used on other platforms. Titles are in English.

- Director: Guides the film's artistic direction.
- Cast: Actors performing in the film.
- Genre: Defines the film's style and theme, e.g., Drama, Action, etc.
- Overview: Provides a brief summary of the film's plot.

Example:

- 1, 2003, Dinosaur Planet, Christian Slater, Scott Sampson, Animation, A four-episode animated series charting the adventures of four dinosaurs each on a different continent in the prehistoric world.
- 17, 2005, 7 Seconds, Simon Fellows, Wesley Snipes, Crime, Action, when an experienced thief accidentally makes off with a Van Gogh, his partner is kidnapped by gangsters in pursuit of the painting, forcing the criminal to hatch a rescue plan.
- 45, 1999, The Love Letter, Peter Ho Sun Chan, Kate Capshaw, Romance, the life of a provincial town becomes stormy after the appearance of an anonymous love letter.

2. Movie Ratings Dataset File Description

The movie ratings dataset comprises six files [9]:

- 'training_set_c1.txt' containing 16,837,634 ratings.
- ‘training_set_c2.txt' containing 18,884,313 ratings.
- 'test_set_c1.txt' containing 3,608,065 ratings.

- 'test_set_c2.txt' containing 4,046,639 ratings.
- 'validation_set_c1.txt' containing 3.608.065 ratings.
- 'validation_set_c2.txt' containing 4.046.639 ratings.

Table 1. Movie Ratings Dataset Distribution

Dataset	#Training	#Test	#Validation
Movie Ratings Dataset C1	*16,837,634*	*3,608,065*	*3,608,065*
Movie Ratings Dataset C2	*18,884,313*	*4,046,639*	*4,046,639*
Total Ratings	35,721,947	7,654,704	7,654,704

Table 1 describes training dataset contains total of 35,721,947 ratings, test dataset contains total of 7,654,704 ratings, validation dataset contains total of 7,654,704 ratings.

Each file follows a specific format:

The first line presents the MovieID followed by a colon. Subsequent lines correspond to a customer's rating and the date it was given.

- MovieID are sequentially numbered from 1 to 17770.
- CustomerID range from 1 to 2,649,429, with some numbers missing, representing a total of 480,189 users.
- Ratings are on a five-star scale, ranging from 1 to 5, where 5 represents the highest rating.

- The dates are consistently formatted as YYYY-MM-DD across all files.

Example:

- 1:
  401047,4,2005-06-03
- 2:
  2059652,4,2005-09-05
- 3:
  1025579,4,2003-03-29

Data Cleaning

In the data cleaning phase, we carefully refined our dataset to get it ready for the analytical phases, here is a detailed account of our process:

- Removed duplicate entries across datasets.
- Filled in missing values or removed incomplete records.
- Standardized inconsistent data, such as movie titles.
- Formatted all dates to a consistent YYYY-MM-DD format.
- Confirmed that ratings fell within the 1-5 scale.
- Anonymized Customer IDs to maintain privacy.
- Checked and corrected data types for each column.
- Implemented ongoing data quality checks.

Data Visualizations

Data visualizations is an important stage in exploratory data analysis. it helps us to understand complex patterns and trends in the dataset. Here we mainly focus on to generate two data visualizations how movie genres are distributed and how film production has changed over the years. The data visualization phase also assists us in gaining an understanding of the movie data and refining the SHERPA recommendation algorithm.

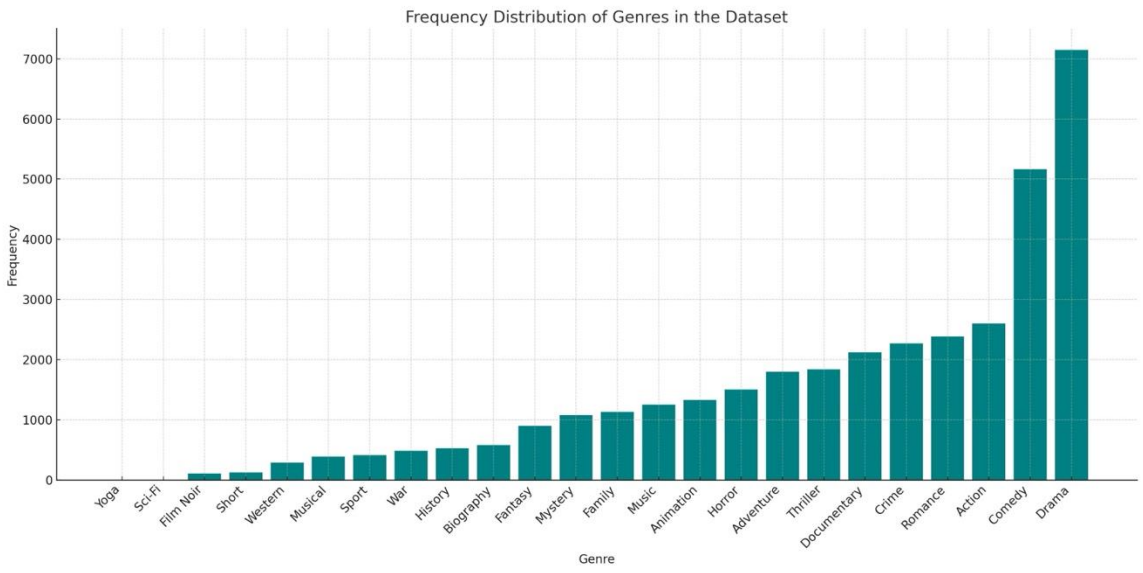

Figure 1. Frequency Distribution of Genres in the Dataset

Figure 1 displays movie genres on the axis (X axis) and their frequencies on the vertical axis (Y axis). The genres are arranged by frequency starting with "Yoga" having lowest no films on the left and highest no films with "Drama"

having on the right. Notably "Comedy" and "Action" are genres following "Drama " each having a number of films. The above figure visually displays the number of films in each genre, in the dataset making it easy to compare genre popularity.

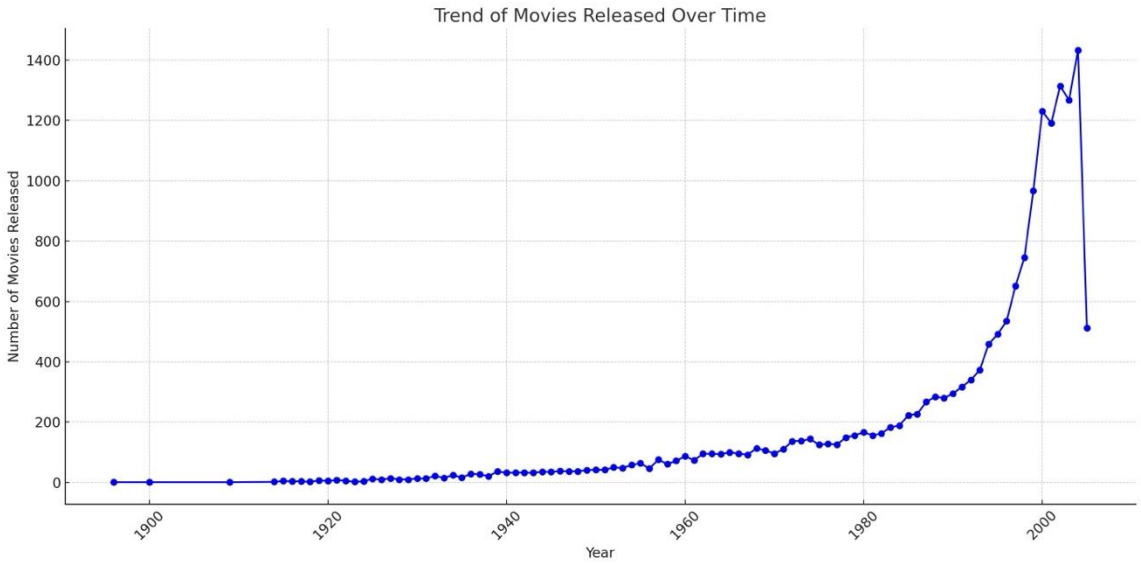

Figure 2. Trend of Movies Released Over Time

Figure 2 shows from the 1900s to the 2000s on the axis (X axis). The number of movies released on the vertical axis (Y axis). It illustrates an increase in movie production over time with a surge as it approaches the 2000s. The initial period saw a shift from the 1900s to mid-century followed by a rise starting in the 1960s and steepening in the 1980s. This upward trend accelerates significantly during the 1990s. Continues into the 2000s. Each data point is connected by lines to show year on year fluctuations in movie releases commonly used for

monitoring trends over time. This visual representation underscores how film industry productivity has expanded due to a growing number of movie releases.

Data Pre-Processing

During the data preparation stage, we will structure the unprocessed data to ensure it aligns with the format required for our machine learning model to learn effectively, Here's a comprehensive overview of our approach:

Data Parsing

- Implemented a parse_data function to read and process data from a file.
- Extracted the current movie ID when a line with a colon is encountered.
- Parsed customer ID and rating from lines without a colon.

Data Structuring

- Assembled the parsed data into a list with the structure [MovieID, CustomerID, Rating].
- Converted the list into a panda Data Frame for easier data manipulation.

Format Handling Issues

- Included error handling to skip lines that don't match the expected "MovieID: CustomerID, Rating, Date" format.

Data Cleaning

- Applied fillna method to replace any NaN values in the Data Frame with empty strings, preparing the data for further analysis.

CHAPTER FOUR

METHODOLOGY

This Chapter will outline the methodologies of different recommendation systems and their corresponding techniques.

Term Frequency-Inverse Document Frequency (TF-IDF)

Term Frequency Inverse Document Frequency is used to assess the importance of a word in a document within a collection of texts known as a corpus [13]. It improves upon term frequency, which counts how frequently a word appears in a document by considering the words frequency across all documents. Words that are frequent in one document but less common across others receive a TF-IDF value suggesting they could be crucial, for comprehending the content of that document [7].

Here's how we figure out the TF-IDF value through two components TF-IDF:

- TF (Term Frequency) *TF* is the number of times a term appears in a document relative to the total word count of that document.

TF = Term Frequency which is calculated as:

$$tf(t,d) = \frac{N_{t,d}}{N_d} \quad (1)$$

where:

$N_{t,d}$ = Number of times term *t* appears in document *d*

N_d = Number of terms in the document *d*

- IDF (Inverse Document Frequency) This measures the rarity of a term across all documents.

IDF= Inverse Document Frequency which is calculated as:

$$idf(t, D) = log \frac{N}{|d \in D : t \in d|} \quad (2)$$

where:

N = total number of documents in the collection.

$|d \in D : t \in d|$ counts the number of documents that contains term *t*.

By combining (1) & (2), The TF-IDF score for a term in a document is given by:

$$tf - idf(t, d, D) = tf(t, d) \times idf(t, D) \quad (3)$$

Words with high TF-IDF scores in a document are used more in that document and less in others, making them key indicators of what the document is about.

Example:

Picture this scenario; You have a collection of movie storylines. You want to discover films that resemble a user's movie "Galactic Quest," famous, for its distinctive mix of space exploration and humor.

Term Frequency (TF): In the plot summary of "Galactic Quest," the word 'spaceship' appears 5 times out of 500 words. So, the TF for 'spaceship' is 5/500 = 0.01.

Inverse Document Frequency (IDF): If your database contains 10,000 movie plots and 'spaceship' appears in 50 of these, the IDF is calculated as log (10000 / 50).

Now, you multiply these two figures to get the TF-IDF score for 'spaceship' in "Galactic Quest." This process is repeated for each relevant term in the plot summary of "Galactic Quest." With each movie in your database represented as a vector of TF-IDF scores for a shared set of terms, you can now compare them.

To recommend movies, you look for other movies with high TF-IDF scores for terms like 'spaceship,' 'alien,' or 'satire.' These scores help identify plots that share thematic elements with "Galactic Quest." The system then uses cosine similarity, comparing the angle between the TF-IDF vectors, to rank other movies. The smaller the angle (or the higher the cosine similarity), the more similar the movies.

Singular Value Decomposition (SVD)

Singular Value Decomposition (SVD) is a technique that breaks down a matrix into three matrices [6]. This process allows us to uncover connections in the data. For example, when we have information about how users rated items such, as movies, but not every user rated every item SVD comes in to complete the missing information [14].

Mathematical Formulation of SVD

$$R = U \sum V^T \quad (4)$$

Where:

- R is the original user-item rating matrix.
- U: A matrix where each row represents a user in terms of latent factors.
- Σ: A diagonal matrix with singular values that indicate the importance of each latent factor.
- V^T: The transpose of a matrix where each column represents an item in terms of latent factors.

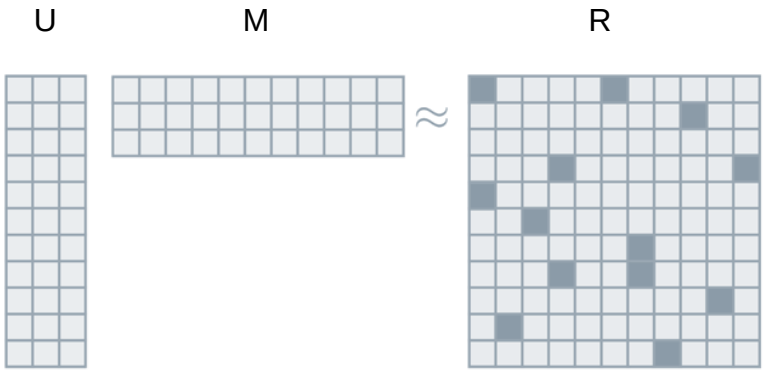

Figure 3. User-item Rating Matrix [14].

Figure 3 illustrates the user-item rating matrix 'R' as the product of three matrices in SVD. 'U' representing the user feature matrix, 'Σ' diagonal matrix with

singular values, 'V ^T' is the item feature matrix. This factorization enables the prediction of user preferences for various items.

The Mechanics of SVD

Imagine you've got a spreadsheet, with rows representing users, columns representing movies and ratings from users for those movies, in the cells. Here's the catch. Not all users have rated all movies leaving some cells empty. SVD comes into play by helping us predict what those missing ratings could be based on the patterns found in the existing ratings.

To do this, SVD takes our original spreadsheet (the matrix R) and transforms it into three new matrices ($\mathrm{U}, \Sigma \text{ and } \mathrm{V}^{\mathrm{T}}$):

- R: This is our starting spreadsheet with users, movies, and their ratings.
- U: This matrix represents each user with certain preferences or tastes.
- Σ: Think of this as a list that shows which preferences are most to least important.
- V^T: This matrix represents each movie according to those same preferences.

SVD to Make Predictions

To predict the missing ratings, we basically put these three matrices back together. It's like making educated guesses about the empty cells in our spreadsheet based on the patterns we've seen in the ratings.

Example:

Let's say we only have three users and three movies. After we've done the math with SVD, we can fill in a rating for a movie that User 1 hasn't seen yet, based on how similar users rated that movie.

For instance, we start with this:

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

After using SVD and choosing to focus on the two most important preferences, we calculate:

$$U = \begin{bmatrix} 0.6 & 0.8 \\ 0.5 & -0.6 \\ 0.6 & -0.2 \end{bmatrix}, \qquad \Sigma = \begin{bmatrix} 10 & 0 \\ 0 & 1.5 \end{bmatrix}, \qquad V^T = \begin{bmatrix} 0.7 & 0.7 & 0.1 \\ -0.2 & 0.1 & 0.98 \end{bmatrix}$$

We then multiply these together to guess the ratings for the movies that haven't been rated by each user, allowing us to make personalized movie recommendations.

Alternating Least Squares (ALS)

Alternating Least Squares (ALS) is a recommendation algorithm that handles sparse data by alternating between solving for user and item factors in a matrix, optimizing only on observed values. Unlike Singular Value Decomposition (SVD), which considers all entries in the user-item interaction matrix (including unknown or missing values), ALS focuses only on the known ratings, and it

scales well for large datasets and integrates regularization directly to prevent overfitting, making it ideal for collaborative filtering [3].

By treating each update of U and M as a least squares problem, we can update one matrix by fixing the other and using the known entries of R.

Update Procedure

1. Initialize the item matrix M with average values or a random start.
2. Fix M and solve for U by minimizing the loss function with respect to U.
3. Fix U and solve for M by minimizing the loss function with respect to M.
4. Repeat steps 2 and 3 until the model converges (i.e., the decrease in the loss function is below a threshold) or a specified number of iterations is reached.

ALS with Weighted-λ-Regularization

ALS (Alternating Least Squares) with Weighted-λ-Regularization is an enhancement to the standard Alternating Least Squares approach [15]. It introduces a regularization term to the optimization process, which helps to avoid overfitting a common problem where a model performs well on the training data but poorly on unseen data. The regularization particularly becomes essential when dealing with a lot of parameters and a sparse dataset (many unknown ratings).

The goal of ALS with Weighted-λ-Regularization is to find user and item feature matrices that predict how users would rate items, even new or previously unrated ones.

The Loss Function

The effectiveness of this method is measured by a loss function that captures two things [15]:

1.How well the model predicts the known ratings.

2.How complex the model is (the size of the user and item feature matrices).

The loss function is represented mathematically as:

$$f(U,M) = \sum_{(i,j)\in I} (r_{ij} - u_i^T m_j)^2 + \lambda \left(\sum_i n_{u_i} \|u_i\|^2 + \sum_j n_{m_j} \|m_j\|^2\right) \quad (5)$$

Where:

- r_{ij} is the actual rating of item j by user i.
- u_i is the feature vector representing user i.
- m_j is the feature vector representing item j.
- I is the set of all (user, item) pairs for which the rating is known.
- λ is the regularization weight that controls the trade-off between fitting the training data well and keeping the model simple to avoid overfitting.
- n_{u_i} is the number of items rated by user i,which weighs the user's feature vector.

- n_{m_j} is the number of users who have rated item j, which weighs the item's feature vector.

Example Application of ALS

Suppose you have a user-item rating matrix R where only some items are rated by each user. Using ALS, you would start by guessing the item features, fix those, and solve for the user features that best predict the known ratings. Then, with the new user features fixed, you would solve for the item features, and so on, iteratively improving your estimates.

In essence, ALS is particularly well-suited for sparse datasets common in collaborative filtering problems and offers a more targeted approach by focusing on known interactions rather than attempting to account for all possible user-item pairs as in SVD.

Content Based Filtering

Content based filtering is a way for recommendation systems to suggest items by looking at the content of the items and comparing it to what a user likes. The idea, behind it is that if a user enjoys one item, they would probably like items that're similar in content. This system creates profiles for both the items and users using details, like genre, description, and tags to figure out similarities and provide recommendations [1], [13], [15].

TF-IDF is chosen over traditional techniques because it provides a more sophisticated way to evaluate the importance of words (or terms) in the content. Unlike simple frequency counts, TF-IDF accounts for the rarity of terms across all

documents, thus giving higher weight to terms that are unique to a particular item. This is crucial in differentiating items with similar but not identical content, as common terms do not overly influence the similarity score.

Implementing TF-IDF in content-based filtering involves three steps:

- Data Cleaning: Start by cleaning and refining the text data by removing any elements, like punctuation and common words that don't add value.
- Vectorization Process: Next convert the text of each item into a vector using the TF-IDF method. Each dimension in the vector represents a term with its value indicating how significant that term is within the document compared to the dataset.
- Similarity Calculation: Determine the similarity between the TF-IDF vectors of a user’s preferred items and other items in the dataset. This comparison can be achieved through methods, like cosine similarity or other distance metrics, which help identify items closely aligned with the users’ preferences.

By implementing TF-IDF, the content-based filtering system can effectively gauge the content relevance of items to a user's interests, enabling more precise and meaningful recommendations.

Collaborative Based Filtering

Collaborative filtering functions, as a recommendation system algorithm that forecasts a user's preferences by considering the preferences of users. It operates on the premise that if users A and B share viewpoints on an item it is probable that A will align with Bs perspective on another item that A has not yet encountered. By analyzing user item interactions like ratings or viewing history the algorithm detects patterns and resemblances, among users or items. This approach enables tailored recommendations by tapping into the preferences of the user community making it widely adopted in suggesting movies, music and various products [2].

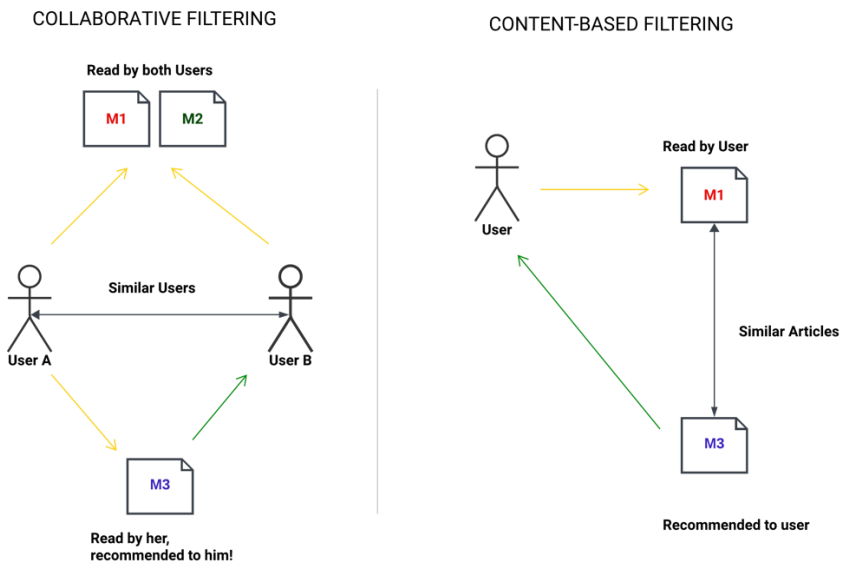

Figure 4: Content based filtering vs Collaborative filtering [12].

Figure 4 illustrates the mechanisms of collaborative and content-based filtering techniques. collaborative filtering recommends items by identifying patterns among similar users, while content-based filtering suggests items based on their similarity to content previously liked by the user.

Hybrid Filtering

A Hybrid filtering algorithm enhances recommendation systems by merging Collaborative and Content-based filtering strategies leveraging the strengths of each to compensate for their shortcomings [6]. This strategy integrates the SVD (Singular Value Decomposition) technique, which forecasts user preferences based on patterns, in user item interactions with TF-IDF. which examines item content to gauge its significance. By merging the personalized forecasts of SVD and the content specificity of TF-IDF the hybrid model provides varied and thorough recommendations effectively tackling issues, like the cold start dilemma and enhancing recommendation accuracy.

SHERPA

The SHERPA algorithm is a recommendation system that cleverly combines the strengths of two different methods: ALS with Weighted Regularization [4], [3], for collaborative filtering and TF-IDF for content-based filtering [15]. By utilizing ALS SHERPA effectively manages datasets. Enhances recommendation accuracy by considering user item interactions along with a

regularization parameter to prevent overfitting. Simultaneously incorporating TF-IDF enables SHERPA to assess and leverage the content of items ensuring recommendations are not solely based on user behavior patterns but, on the semantic relevance of the items themselves. This dual strategy empowers SHERPA to provide contextually relevant recommendations overcoming drawbacks of traditional approaches and enriching user satisfaction.

CHAPTER FIVE

EXPERIMENTAL RESULTS

Evaluation Metrics

This chapter will describe the metrics utilized to evaluate the performance of the movie recommendation algorithms employed in this project.

Root Mean Square Error (RMSE)

RMSE is a standard way to measure the error of a model in predicting quantitative data. It's particularly useful in recommender systems to evaluate the difference between predicted and actual ratings. RMSE provides a way to quantify the magnitude of prediction errors, taking the square root of the average squared differences between the prediction and the actual observation.

The formula for RMSE is: $RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(P_i - A_i)^2}$ (6)

Where:

- P_i represents the predicted value for the i^{th} instance.
- A_i is the actual value for the i^{th} instance.
- N is the total number of instances.

A lower RMSE value indicates a better fit of the model to the data. It's especially effective in highlighting the impact of large errors, given that it squares the differences before averaging. However, it should be noted that RMSE can be sensitive to outliers and might not be well-suited if the error distribution is not uniform.

In the context of our project, RMSE will serve as a key indicator of the accuracy of our recommendation system's predictions, allowing us to fine-tune the algorithm for optimal performance.

Evaluation Scenarios

We have designed two distinct scenarios to assess the performance of the SHERPA System. One is for the existing users and another for new users. These scenarios are constructed to evaluate the system's responsiveness to each user's unique needs whether they're browsing casually or conducting specific searches based on their past interactions.

For Existing Users

I. Existing User Log in & Without Search: When user 401047 logs in, without searching for anything the system uses their interactions to recommend movies. Since the user is simply browsing collaborative filtering is used. This involves the algorithm analyzing the activities of users, with interests and suggesting movies that those users have enjoyed.

II. Existing User Log in & Search with Keyword: When user 401047 logs in and searches, for "The Company " the system transitions to a recommendation method. It merges the user’s data with the search query to suggest options that cater not to popular or like-minded users but also those directly related to the search term.

For New Users

I. New User Log in & Search with Keyword: When a new user looks up "The Company" without any viewing history the platform uses content-based filtering. This approach involves analyzing details like genre, storyline, and actors of the film to suggest movies, with content to "The Company." The aim is to offer tailored recommendations solely based on the search query.

Results

In the Experimental Results and Analysis phase, we are comparing the SHERPA System's performance against traditional Hybrid systems using RMSE metric across Training, Test, and Validation datasets as detailed below:

Table 2. Evaluation of Recommendation Models by RMSE

Models	Training RMSE	Test RMSE	Validation RMSE
HYBRID	2.828920	2.948704	2.949163
SHERPA	0.860550	0.903856	0.904121
Improvement	69.6%	69.4%	69.3%

Table 2 presents a comparison between the HYBRID and SHERPA recommendation systems showing their Root Mean Square Error (RMSE) values on training, testing and validation sets.

Training Dataset Comparison

In the training dataset, we're really looking at how well each model learns from the data it's been given. The HYBRID model's RMSE is a bit on the high side at 2.8289, suggesting some difficulties in capturing the subtleties of user preferences. SHERPA, though, is a game changer, trimming that RMSE down to 0.8606. This impressive 69.6% improvement isn't just about numbers – it reflects a model that's getting a much better read on what users are likely to enjoy.

Test Dataset Comparison

Moving to the test dataset, we're in the real proving ground – how well can the models predict what users will like when they encounter fresh, unseen movies? HYBRID's showing an RMSE of 2.9487, hinting that it might miss the mark now and then. SHERPA, on the other hand, hits a sweet spot with an RMSE of 0.9039. That's about a 69.4% leap towards more spot-on recommendations, making a solid case for SHERPA's ability to understand and predict user tastes.

Validation Dataset Comparison

The validation dataset is all about fine-tuning and getting the model just right. HYBRID is at an RMSE of 2.9492, which means there's room for improvement. Enter SHERPA, with its RMSE of 0.9041 – it's not only 69.3% better at minimizing errors, but it's also showing us that it can stay consistent and reliable, no matter the dataset.

Key Areas Where SHERPA Outperforms Traditional Hybrid

- Advanced Matrix Factorization Technique : SHERPA's Alternating Least Squares (ALS) approach efficiently tackles the problem of sparsity in user-item, which is a common challenge in collaborative filtering. Unlike SVD which attempts to factorize the entire matrix including unobserved entries, ALS iteratively optimizes for known ratings leading to more accurate predictions.
- Regularization : SHERPA employs Weighted-λ-Regularization to balance fitting the model to complex datasets while preserving simplicity. it also gives better performance on unseen data compared to traditional hybrid methods.
- Parallelization: SHERPA enhances scalability and reduces computation time through parallel processing the distributes the computational load of updating user and item matrices (U and M) across multiple computers outpacing traditional SVD's computational demands.
- Computational Efficiency: Traditional Hybrid algorithms relies on SVD often requires higher computational resources due to matrix densification but whereas SHERPA handles large datasets efficiently using parallelization and increase Computational Efficiency performance optimization.
- Hybrid Filtering Approach: SHERPA refines recommendation accuracy by merging content-based filtering (using TF-IDF) and collaborative filtering

(using ALS). This combination of methods facilitates a more personalized and precise recommendation process rather traditional hybrid approaches eliminating diversity, data dependency and over specialization problems.

- Continuous Learning and Model Updating: SHERPA dynamically updates based on new user interactions continually refining its model to enhance recommendation accuracy, surpassing traditional models that may not update as frequently or effectively.

Overview SHERPA Performance Over Traditional Hybrid

SHERPA not only just slightly outperform and also it exceeds by a margin of 70% than the regular traditional HYBRID Algorithm. which is a quite significant improvement. It effectively tackles the shortcomings of content based and collaborative filtering techniques by using TF-IDF to capture content nuances and ALS with regularization to focus on filtering without overfitting. This balanced approach ensures that SHERPA isn't just technically superior and also provides more personalized recommendation experience to users.

CHAPTER SIX

CONCLUSION

This project has successfully introduced and evaluated the Smart Hybrid Enhanced Recommendation and Personalization Algorithm (SHERPA) an advanced machine learning algorithm created to enhance and personalize the movie recommendation process. By combining content-based filtering using TF-IDF and collaborative filtering through ALS with Weighted Regularization SHERPA has shown an improvement, in recommendation accuracy and user satisfaction.

Through analysis using metrics like RMSE, SHERPAs performance compared to traditional hybrid models was highlighted. Notably SHERPA achieved a decrease in prediction errors with enhancements of around 70% across training, testing and validation datasets when compared to its predecessor. This emphasizes the algorithms improved capability to comprehend and forecast user preferences providing relevant content suggestions.

Moreover, SHERPA’s innovative methodology tackles issues seen in existing recommendation systems such as overfitting and addressing the cold start problem. This ensures a scalable solution that caters to user interactions. Its proficiency in managing datasets and customizing content based on user behaviors as well as item traits sets a new standard in recommendation system technology.

To summarize the SHERPA algorithm signifies a progression, in recommendation systems. The users content discovery experience is enhanced by SHERPA, which also paves the way, for advancements in machine learning and artificial intelligence research and development. In the changing world personalized recommendation systems like SHERPA play a crucial role, in driving future innovations.

APPENDIX A

MODEL CODE

```
# Imports
import pandas as pd
import numpy as np
from scipy.sparse import csc_matrix
from scipy.sparse.linalg import svds
from numpy.linalg import lstsq
from sklearn.metrics import mean_squared_error
from math import sqrt
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import sigmoid_kernel
from sklearn.metrics import ndcg_score

# Parsing the data
def parse_data(file_path):
    data = []
    current_movie_id = None
    with open(file_path, 'r', encoding='utf-8-sig') as file:
        for line in file:
            try:
                if ':' in line:
                    current_movie_id = int(line.split(':')[0])
                else:
```

```
                customer_id, rating, _ = line.strip().split(',')
                data.append([current_movie_id, int(customer_id), int(rating)])
            except ValueError:
                continue
    return pd.DataFrame(data, columns=['MovieID', 'CustomerID', 'Rating'])

# Function to load data for training, test, and validation
def load_dataset(file_paths):
    data_frames = [parse_data(file_path) for file_path in file_paths]
    combined_data = pd.concat(data_frames)
    return combined_data

# File paths setup
train_files = ['training_set_c1.txt','training_set_c2.txt']
test_file = ['test_set_c1.txt', 'test_set_c2.txt']
validation_file = ['validation_set_c1.txt','validation_set_c2.txt']

# Load datasets
# a) Movie Titles Dataset
movies_df = pd.read_csv("movie_titles.csv", on_bad_lines='skip')

# b) Movie Ratings Dataset
```

```
train_movie_data = load_dataset(train_files)
train_movie_data.head()

#1. Content-Based Filtering
movies_df = movies_df.fillna('')

# Function to create weighted text
def create_weighted_text(row):
    return (row['Overview'] + ' ') * 45 + (row['Genre'] + ' ') * 25 + \
          (row['Director'] + ' ') * 15 + (row['Cast'] + ' ') * 15
movies_df['weighted_text'] = movies_df.apply(create_weighted_text, axis=1)

# Initialize TF-IDF Vectorizer
tfv = TfidfVectorizer(min_df=3, max_features=None, strip_accents='unicode',
                      analyzer='word', token_pattern=r'\w{1,}',
                      ngram_range=(1,3), stop_words='english')
tfv_matrix = tfv.fit_transform(movies_df['weighted_text'])
sig = sigmoid_kernel(tfv_matrix, tfv_matrix)
indices = pd.Series(movies_df.index,
index=movies_df['Movie_Name']).drop_duplicates()

#2. Colloborative based Filtering
```

```
# Create a user-item matrix
train_ratings_df = train_movie_data.pivot(index='CustomerID',
columns='MovieID', values='Rating').fillna(0)
train_ratings_matrix = csr_matrix(train_ratings_df.values)

# 2.1  SVD - Singular Value Decomposition
k=11
U, sigma, Vt = svds(train_ratings_matrix, k=k)
sigma = np.diag(sigma)

# To make Predictions
def predict(matrix, U, sigma, Vt):
    mean_user_rating = matrix.mean(axis=1).reshape(-1, 1)
    preds = np.dot(np.dot(U, sigma), Vt) + mean_user_rating
    return preds

    # RMSE calculation
def calculate_rmse(actual, predicted):
    mask = actual.nonzero()
    actual_filtered = actual[mask].flatten()
    predicted_filtered = predicted[mask].flatten()
    return sqrt(mean_squared_error(actual_filtered, predicted_filtered))
```

```
# train_preds is your predictions
train_preds = predict(train_ratings_df.values, U, sigma, Vt)

# Calculate RMSE
print('Training RMSE:', calculate_rmse(train_ratings_matrix.toarray(),
train_preds))

# 2.2 ALS - Alternating Least Squares
def update_U(M, U, lambda_reg, ratings): // Proprietary software code
def update_M(M, U, lambda_reg, ratings): // Proprietary software code
def ALS(ratings, num_factors=50, lambda_reg=0.1, iterations=10): // Proprietary
software code
num_factors = 11
lambda_reg = 0.1
iterations = 5

# Running ALS on the ratings matrix
U, M = ALS(train_ratings_matrix, num_factors=num_factors,
lambda_reg=lambda_reg, iterations=iterations)

# Generate predictions
```

```
train_predictions = U.dot(M.T)
train_preds_df = pd.DataFrame(train_predictions,
columns=train_ratings_df.columns, index=train_ratings_df.index)

#SHERPA Hybrid Recommendation System
movies_df.rename(columns={'Sl_No': 'MovieID'}, inplace=True)
def hybrid_recommendations(user_id=None, movie_name=None,
preds_df=None, movies_df=movies_df, sig=sig, indices=indices, top_n=10):
    if preds_df is None:
        raise ValueError("The predictions dataframe (preds_df) is required.")
    final_recs = []

    # Fetch Content-Based Recommendations
    content_based_recs = []
    if movie_name in indices:
        idx = indices[movie_name]
        sig_scores = list(enumerate(sig[idx]))
        sig_scores = sorted(sig_scores, key=lambda x: x[1], reverse=True)
        movie_indices = [i[0] for i in sig_scores[1:top_n+1]]
        content_based_recs = movies_df.iloc[movie_indices]['Movie_Name'].tolist()
```

```
    # For existing users with a search query, combine collaborative and content-based recommendations
    if user_id and movie_name:
        # Fetch collaborative filtering recommendations based on historical ratings
        collaborative_recs_ids = preds_df.loc[user_id].sort_values(ascending=False).head(top_n * 2).index.tolist()
        collaborative_recs_names = movies_df[movies_df['MovieID'].isin(collaborative_recs_ids)]['Movie_Name'].tolist()
        # Combine lists with simple deduplication, prioritizing content-based recommendations
        seen = set(content_based_recs)
        combined_recs = content_based_recs + [rec for rec in collaborative_recs_names if rec not in seen]
        # Limit to top_n recommendations after combining
        final_recs = combined_recs[:top_n]
    elif user_id:
        # Only collaborative recommendations for existing users without search query
        collaborative_recs_ids = preds_df.loc[user_id].sort_values(ascending=False).head(top_n).index.tolist()
```

```
        final_recs =
movies_df[movies_df['MovieID'].isin(collaborative_recs_ids)]['Movie_Name'].tolist
()
    else:
        # Only content-based recommendations for new users with a search query
        final_recs = content_based_recs
    return final_recs

# Testing the function with your scenarios
user_id = 401047  # Example user ID
movie_name = "The Company"  # Example movie name

print("Collaborative Recommendations for Existing User (No Search):")
collab_recs = hybrid_recommendations(user_id=user_id,
preds_df=train_preds_df, top_n=10)
for movie in collab_recs:
    print(movie)

print("\nHybrid Recommendations for Existing User (With Search):")
hybrid_recs = hybrid_recommendations(user_id=user_id,
movie_name=movie_name, preds_df=train_preds_df, top_n=10)
for movie in hybrid_recs:
```

```
    print(movie)


print ("\nContent-Based Recommendations for New User (With Search):")
content_recs = hybrid_recommendations(movie_name=movie_name,
preds_df=train_preds_df, top_n=10)
for movie in content_recs:
    print(movie)
def calculate_rmse(actual, predictions):
    mask = actual.nonzero()
    actual = actual[mask]
    predictions = predictions[mask]
    return sqrt(mean_squared_error(actual, predictions))


#Calculate RMSE for training set
rmse = calculate_rmse(train_ratings_matrix.toarray(), train_predictions)
print('Training RMSE:', rmse)


###### TEST PHASE ######


# Load datasets
test_movie_data = load_dataset(test_file)
```

```
# Create a Train user-item matrix
test_ratings_df= test_movie_data.pivot(index='CustomerID', columns='MovieID',
values='Rating').reindex(index=train_ratings_df.index,
columns=train_ratings_df.columns).fillna(0)

# Convert to CSR format
test_ratings_matrix = csr_matrix(test_ratings_df.values)

# Mapping test user and movie indices to training set indices
test_user_indices = [np.where(train_ratings_df.index == uid)[0][0] for uid in
test_ratings_df.index if uid in train_ratings_df.index]
test_movie_indices = [np.where(train_ratings_df.columns == mid)[0][0] for mid in
test_ratings_df.columns if mid in train_ratings_df.columns]

# Generate predictions for test set
test_predictions = U[test_user_indices, :] @ M.T[:, test_movie_indices]

# Generate predictions
test_preds_df = pd.DataFrame(test_predictions,
columns=test_ratings_df.columns, index=test_ratings_df.index)

# Calculate RMSE for test set
```

```
test_rmse = calculate_rmse(test_ratings_matrix.toarray(), test_predictions)
print('Test RMSE:', test_rmse)

###### VALIDATION PHASE ######

# Load datasets
validation_movie_data = load_dataset(validation_file)

# Create a user-item matrix
validation_ratings_df= validation_movie_data.pivot(index='CustomerID',
columns='MovieID', values='Rating').reindex(index=train_ratings_df.index,
columns=train_ratings_df.columns).fillna(0)

# Convert to CSR format
validation_ratings_matrix = csr_matrix(validation_ratings_df.values)

# Mapping validation user and movie indices to training set indices
validation_user_indices = [np.where(train_ratings_df.index == uid)[0][0] for uid in
validation_ratings_df.index if uid in train_ratings_df.index]
validation_movie_indices = [np.where(train_ratings_df.columns == mid)[0][0] for
mid in validation_ratings_df.columns if mid in train_ratings_df.columns]
```

```
# Generate predictions for validation set
validation_predictions = U[validation_user_indices, :] @ M.T[:,
validation_movie_indices]

validation_preds_df = pd.DataFrame(validation_predictions,
columns=validation_ratings_df.columns, index=validation_ratings_df.index)

# Calculate RMSE for validation set
validation_rmse = calculate_rmse(validation_ratings_matrix.toarray(),
validation_predictions)
print ('validation RMSE:', validation_rmse)
```

REFERENCES

[1] Armadhani Hiro Juni Permana Juni, Agung Toto Wibowo, “Movie Recommendation System Based on Synopsis Using Content-Based Filtering With TF-IDF and Cosine Similarity,” International Journal on Information and Communication Technology (IJoICT), Jawa Barat, Indonesia, 2023, pp. 1-14, doi:10.21108/ijoict.v9i2.747.

[2] Ching-Seh Mike Wu, Deepti Garg, and Unnathi Bhandary, "Movie Recommendation System Using Collaborative Filtering," 2018 IEEE 9th International Conference on Software Engineering and Service Science (ICSESS), Beijing, China, 2018, pp. 11-15, doi:10.1109/ICSESS.2018.8663822.

[3] Indah SurvyanaWahyudi, Achmad Affandi, Mochamad Hariadi. M., "Recommender engine using cosine similarity based on alternating least square-weight regularization," 15th International Conference on Quality in Research (QiR), Nusa Dua, Bali, Indonesia, 2017, pp. 256-261, doi: 10.1109/QIR.2017.8168492.

[4] István Pilászy, Dávid Zibriczky, Domonkos Tikk, "Fast als-based matrix factorization for explicit and implicit feedback datasets," 4th ACM conference on Recommender systems, New York, USA, 2010, pp. 71–78, doi: 10.1145/1864708.1864726.

[5] Jianjun Ni, et al., "Collaborative filtering recommendation algorithm based on TF-IDF and user characteristics," Applied Sciences, Nanjing, China, 2021, doi: 10.3390/app11209554.

[6] Mayur Rahul, Vinod Kumar, Vikash Yadav, "Movie Recommender System Using Single Value Decomposition and K-means Clustering," 1st International Conference on Computational Research and Data Analytics (ICCRDA), Rajpura, India, 2020, doi:10.1088/1757-899X/1022/1/012100.

[7] Mohamed Chiny, et al., "Netflix recommendation system based on TF-IDF and cosine similarity algorithms," International Conference on Big Data, Modelling and Machine Learning (BML), Kenitra, Morocco, 2021, pp. 15–20,doi: 10.5220/0010727500003101.

[8] Netflix, "Netflix Prize," Wikipedia, Available: https://en.wikipedia.org/wiki/Netflix_Prize [Accessed: August 2023].

[9] Netflix, "Netflix Prize data," Kaggle, Available: https://www.kaggle.com/datasets/netflix-inc/netflix-prize-data. [Accessed: May 2023].

[10] Netflix, "Netflix Prize Leaderboard," Wayback Machine, Available: https://web.archive.org/web/20150813135722/http://www.netflixprize.com/leaderboard. [Accessed: September 2023].

[11] Robin Burke, "Hybrid recommender systems: Survey and experiments," User modeling and user-adapted interaction, 2002, pp. 331–370, doi: 10.1023/A:1021240730564.

[12] Sanket Doshi, "Brief on Recommender Systems," Towards data science, Available: https://towardsdatascience.com/brief-on-recommender-systems-b86a1068a4dd [Accessed: September 2023].

[13] Simon Philip, Peter Shola, A. Ovye, "Application of content-based approach in research paper recommendation system for a digital library," International Journal of Advanced Computer Science and Applications (IJACSA), 2014, pp. 37-40, doi:10.14569/IJACSA.2014.051006.

[14] Triyanna Widiyaningtyas, Muhammad Iqbal Ardiansyah, Teguh Bharata Adji, "Recommendation Algorithm Using SVD and Weight Point Rank (SVD-WPR)," Big Data and Cognitive Computing 6, Indonesia, 2022, doi: 10.3390/bdcc6040121.

[15] Yunhong Zhou, et al., "Large-scale parallel collaborative filtering for the netflix prize," 4th International Conference of Algorithmic Aspects in Information and Management (AAIM), Shanghai, China, 2008, doi: 10.1007/978-3-540-68880-8_32.