COLLABORATIVE FILTERING RECOMMENDATION SYSTEM

A project report submitted in partial fulfillment of the requirements for the award of a degree in

Master of Science
Data Science

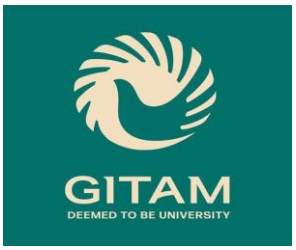
By

KOCHCHARLA PRANAY VENKATA SAI

2023003061

Under the esteemed guidance of

Mrs. V. Satya Aruna
Assistant Professor



Department of Computer Science
GITAM School of Science
GITAM (Deemed to be University)
Visakhapatnam -530045, A.P

(2024-2025)

CERTIFICATE

This is to certify that the project entitled **"Collaborative Filtering Recommendation System"** is Bonafide work done by **Kochcharla Pranay Venkata Sai, Reg No: 2023003061** From **December 2024** to **April 2025** in partial gratification of the requirements for the award of the **Master of Science in Data Science** degree in the Department of Computer Science at GITAM School of Science, GITAM (Deemed to be University), Visakhapatnam.

Internal Guide	**Head of the Department**
Mrs. V. Satya Aruna	**Dr. T. Uma Devi**
Assistant Professor	Professor

DECLARATION

I, **Kochcharla Pranay Venkata Sai (Reg. No: 2023003061)**, a student of the **M.Sc. Data Science** program at **GITAM University, Visakhapatnam**, hereby declares that the project titled **" Collaborative Filtering Recommendation System"** is my original work. This project has been conducted under the guidance of **Mrs. V. Satya Aruna** and is submitted in partial fulfillment of the requirements for the Master of Science, Data Science.

I confirm that this project results from my independent efforts and has not been submitted previously for any other degree, diploma, or certification. Any references or contributions from different sources have been properly acknowledged.

K. Pranay Venkata Sai
Reg no: 2023003061

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without mentioning the people who made it possible and whose constant guidance and encouragement crown all the effects with success.

I would like to express my sincere gratitude to our honourable Principal and **Prof. K. Vedavathi,** GITAM School of Science, GITAM (Deemed to be University), for giving me an opportunity to work on this project.

I consider it a privilege to express our deepest gratitude to **Prof. T. Uma Devi**, Head of the Department, Department of Computer Science for her valuable suggestions and constant motivation that greatly helped us to successfully complete the project work. I would like to thank my project guide **Mrs. V. Satya Aruna**, Assistant Professor, Dept. of Computer Science for her stimulating guidance and profuse assistance.

We would like to thank our Project Coordinators **Mr. B. Ravi Kumar,** Associate Professor, and **Dr. K. Vanitha,** Assistant Professor Dept. of Computer Science for helping to complete the project by taking frequent reviews and for their valuable suggestions throughout the progress of this work.

I thank all the Teaching and Non-teaching Staff who have been a constant source of support and encouragement during the study tenure.

I thank all my friends who helped me in sharing their knowledge and support.

K Pranay Venkata Sai
Reg no: 2023003061

ABSTRACT

The Collaborative Filtering Recommender System leverages user-item interaction data to generate personalized recommendations. Using the Amazon Reviews/Ratings dataset, which contains over 2 million records, this project applies memory- based collaborative filtering techniques to uncover patterns in user preferences and recommend products that align with individual tastes. Widely employed in e-commerce, streaming, and other industries, this approach eliminates the need for manual curation by harnessing user behavior data. This system utilizes collaborative filtering, popularity-based recommendation, and model-based recommendation to enhance the user shopping experience. The project is developed using Python and the Surprise library for building and optimizing recommendation models. The project employs preprocessing techniques such as filtering sparse data and normalizing ratings to prepare the dataset for analysis. Key methodologies include user-user and item-item similarity metrics, calculated using cosine similarity and Pearson correlation coefficient, alongside K-Nearest Neighbours (KNN) for recommendation generation. Model performance is evaluated using metrics like Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE). The system demonstrates the practical applications of collaborative filtering in real-world scenarios, highlighting its potential for scalability and accuracy in delivering personalized user experiences.

Project Description

The project consists of the following modules:

1. Data Preprocessing: Cleaning the Amazon Reviews/Ratings dataset by filtering sparse user-item interactions, normalizing ratings, and creating a dense interaction matrix.

2. Similarity Computation:

User-User Similarity: Identifying users with similar preferences using cosine

similarity and Pearson correlation coefficient.

Item-Item Similarity: Finding relationships between products based on co-

occurrence patterns in user ratings.

3. Recommendation Generation: Using K-Nearest Neighbours (KNN) to recommend items based on computed similarities.

4. Model Evaluation: Assessing the accuracy of recommendations using error metrics like

MAE and RMSE.

5. Performance Analysis: Identifying strengths and limitations of memory-based methods and exploring their applicability to other domains.

Technologies Used

- Programming Language: Python
- Machine Learning Libraries: Surprise, scikit-learn, NumPy, Pandas
- Visualization Tools: Matplotlib, Seaborn
- Recommendation Techniques: Collaborative Filtering, Popularity-Based Recommendation,
- SVD (Model-Based Approach)
- Storage & Data Handling: CSV files (Amazon Dataset from Kaggle)
- System Configuration
    a. Hardware: Intel i5 or above, 8GB RAM, 256GB SSD
    b. Software: Python 3.8+, Jupyter Notebook/IDE, Required Libraries (pandas, scikit-learn, matplotlib, etc.)

This project highlights the significance of collaborative filtering in building scalable and efficient recommender systems, with broad applications across e-commerce, entertainment, education, and healthcare. Its findings provide a foundation for future explorations into hybrid or model-based recommendation techniques.

INDEX

1. INTRODUCTION

In the age of the internet, recommendation systems have become the backbone of contemporary e-commerce websites, radically changing the way users engage with content and make buying decisions. With the plethora of products on the internet, users tend to find it difficult to discover products that suit their tastes or requirements. This is where recommendation systems enter the picture, serving as smart filters that examine large amounts of user information and item attributes to create personalized recommendations. Amazon, Netflix, and Spotify are just a few of the companies that have used these systems to spur user activity, enhance satisfaction, and grow sales. By suggesting what a user may be interested in based on past interactions, recommendation systems make the buying experience better and assist companies in serving individual tastes in large numbers.

Fundamentally, recommendation systems try to make suggestions of relevance by examining data patterns. A number of approaches are used to do this, the most well-known of which are content-based filtering, collaborative filtering, and hybrid techniques. Content-based filtering suggests items that are similar to what the user likes in the past, based on item features and user profiles. This is in contrast to collaborative filtering, which deals with the relationships and similarities between users and items without requiring explicit knowledge of the products themselves. Hybrid models merge these two and try to exploit the best from both and address the limitations of each, including the cold start problem or sparsity in user-item interactions.

This project involves developing a product recommendation system based on collaborative filtering methods on Amazon's product review data. The data is in the form of user-product interactions, specifically ratings, which are used as the basis for constructing the recommendation model. The objective is to forecast how a user would rate a product that they have not yet interacted with and thus recommend products that are in line with their interests. The model is constructed with the Surprise library in Python, which gives a strong suite of tools to implement and test collaborative filtering algorithms.

Singular Value Decomposition (SVD) is one of the major methods applied here, which is a powerful matrix factorization technique. SVD operates by reducing the huge, sparse user-item matrix into lower-dimensional forms, extracting implicit patterns in user behavior and item

characteristics. These hidden attributes assist the model in making more precise predictions of unseen ratings. In the course of this project, SVD was trained on a pre-processed version of the dataset and its performance assessed using metrics such as Root Mean Square Error (RMSE) and Mean Absolute Error (MAE), which indicate how accurately the predicted ratings approximate the true ratings.

Besides SVD, the project utilizes the K-Nearest Neighbors (KNN) algorithm to construct an item-based collaborative filtering model. KNN determines similarities between items from user ratings and calculates similarity scores, usually cosine similarity. By determining the nearest neighbors of a product, the model can provide recommendations for similar items to users interested in similar products. This method is particularly helpful when user behavior patterns are uniform across similar categories of items. KNN offers a simple means to grasp product similarity and has worked well for some types of recommendation tasks, especially when the item space is well established.

To offer a comparative baseline, the project also includes more straightforward models like the Baseline Only algorithm and the Normal Predictor. The Baseline Only model predicts ratings using global averages and bias terms for users and items, providing a simple baseline. These models are not likely to be as good as more complex methods, but provide valuable baselines against which the gains of such advanced methods as SVD and KNN can be measured.

A great deal of effort during the project went into data preprocessing, such as filtering items and users based on a minimum number of ratings to provide useful interactions. Data was then analyzed to learn rating distribution patterns, user behavior, and item popularity. This process enabled the design of a scalable and efficient recommendation engine specific to the Amazon product environment.

In summary, the project illustrates how collaborative filtering principles can be implemented in practice for real-world datasets, highlighting machine learning's capabilities in personalization of digital life. Through applying models such as SVD and KNN, the system predicts user interests adequately and provides informative product recommendations effectively. Such utilities not only enhance user satisfaction but also provide remarkable value to business organizations by engaging users more actively and streamlining the process of product discovery.

1.1 BACKGROUND

E-commerce has dramatically transformed the way people shop by offering unparalleled access to a wide range of products, competitive pricing, and the convenience of purchasing from anywhere at any time. However, this explosion in product availability has introduced a significant challenge: helping users efficiently discover items that align with their interests and needs. The traditional methods of browsing through categories or relying on keyword-based searches are often insufficient, especially when users are unsure of what they are looking for. These limitations not only lead to user frustration but also result in reduced engagement, lost sales opportunities, and weakened customer loyalty. As online platforms scale to accommodate millions of products and users, it becomes increasingly important to incorporate intelligent systems that can personalize the shopping experience and adapt dynamically to evolving user preferences.

To tackle this challenge, this project focuses on developing an Amazon Product Recommendation System centered around collaborative filtering techniques using machine learning. The core goal is to implement a system that learns from historical user behavior—such as ratings, purchases, and browsing patterns—and predicts which products a user is most likely to be interested in. Collaborative filtering, particularly model-based methods, is known for its ability to uncover hidden relationships in sparse datasets. In this project, key algorithms like Singular Value Decomposition (SVD) and K-Nearest Neighbours (KNN) are used to build robust recommendation models. SVD is a matrix factorization technique that captures latent factors in user-item interactions, enabling the system to make accurate predictions even when explicit product features are not available. KNN, on the other hand, finds similar users or items based on rating patterns, making it suitable for generating intuitive and interpretable recommendations. The project involves processing a large-scale dataset from Amazon, which includes user-product ratings and interaction history. After filtering and preparing the data, different collaborative filtering models are trained using the Surprise library in Python. These models are evaluated using standard metrics like Root Mean Square Error (RMSE) and Mean Absolute Error (MAE) to assess their predictive accuracy. A pre-trained model is stored for efficient retrieval of recommendations, enabling the system to scale and respond effectively to user queries in a production-ready environment. The project also involves data analysis and visualization to better understand user behavior and rating distribution, further aiding in model

refinement and tuning.

By implementing collaborative filtering from the ground up and evaluating its performance on real-world data, this project demonstrates the practical value of machine learning in personalizing digital experiences. The recommendation engine developed here is not only capable of enhancing product discovery but also has the potential to significantly improve user engagement and satisfaction on e-commerce platforms. Furthermore, the methodology can be extended and integrated into larger systems, opening avenues for future enhancements such as hybrid models or context-aware recommendations. Overall, this project highlights how intelligent systems can transform raw data into actionable insights, making the shopping journey smarter and more efficient for both users and businesses.

1.2 MOTIVATION (PROBLEM STATEMENT)

With the fast development of e-commerce websites such as Amazon, customers are usually confused with the numerous products online. It makes it difficult to find products that match customers' individual interests and historical behaviors. A well-designed system of recommendations guides customers to overcome this complexity and provides personalized recommendations according to their previous behaviors. The capacity to forecast what an individual may be interested in not only improves the shopping experience but also boosts user engagement and customer satisfaction. Therefore, recommendation systems have become an essential element to the success of contemporary online sites.

This project is driven by the necessity of creating collaborative filtering-based recommendation systems utilizing machine learning. Instead of creating a content-based or hybrid model, attention lies in finding how collaborative filtering—specifically, matrix factorization techniques such as Singular Value Decomposition (SVD) and neighborhood-based approaches such as K-Nearest Neighbors (KNN)—can be used to transform raw historical user-product rating data into good predictions. Utilizing a real dataset from Amazon in this project is intended to validate the efficacy of collaborative filtering for identifying underlying behavior patterns among users and making actionable product recommendations with the application of machine learning processes.

1.3 EXISTING SYSTEM

In today's e-commerce world, product recommendation systems are based on conventional methods like popularity-based filtering, classification-based recommendations, and content-based filtering. These mechanisms offer a primary level of personalization but frequently do not represent dynamic user activity and changing tastes, which causes less useful recommendations.

Current Techniques in Recommendation Systems:

Popularity-Based Filtering:

This method suggests the most popular or best-selling products to all users, assuming that frequently purchased or highly rated products will be of interest to a wider audience.

Limitation: It does not take into account individual likes, resulting in blanket suggestions that do not recognize a user's specific shopping history or interests.

Classification-Based Recommendations

Items are classified according to features like type, brand, and price category. Upon a user's purchase or interaction with a particular category, the system suggests items from the same category.

Limitation: This approach has difficulty with the cold start scenario, in which new users with little or no history are offered weak or unpersonalized recommendations. It also cannot learn from evolving user tastes over time.

Content-Based Filtering:

This method scans product descriptions, metadata, and user interest to suggest items that are similar to those the user has interacted with in the past. For instance, if a user purchases a smartphone, they might be suggested accessories or comparable phone models.

Limitation: Though good in certain instances, content-based filtering is not very diverse in suggestions, frequently suggesting highly comparable products rather than stimulating product discovery. It also needs to have well-defined and detailed product metadata, which is not always the case.

Problems with the Current System:

Cold Start Problem: New products or users with no previous interaction data are given wrong or generic recommendations.

Lack of Personalization: Conventional techniques do not delve into user behavior, resulting

in non-personalized product recommendations.

Static Nature: These techniques fail to change according to dynamic user preferences or shopping patterns.

Limited Engagement: Low-quality recommendations can result in decreased conversion rates and lower customer satisfaction.

1.3.1 Overview of the Existing System and Disadvantages

The existing product recommendation strategy in e-commerce is mostly based on conventional techniques like popularity-based filtering, classification-based recommendation, and content-based filtering. These techniques examine product features, sales patterns, and pre-defined categories to recommend products to users. They tend to be less adaptable and personalized, resulting in incorrect or redundant recommendations.

Most recommendation systems use general, rule-based reasoning without taking advantage of user behavior insight, leading to generic recommendations that do not represent individual tastes. Moreover, content-based filtering tends to restrict product variety by suggesting similar products rather than presenting new and related items. The failure to dynamically adapt to evolving user interests diminishes the performance of these systems, resulting in decreased user interaction and reduced sales.

Key Disadvantages of the Existing System:

Generic and Non-Personalized Suggestions: Trending-based filtering recommends the same popular items to everybody without considering individual tastes.

Cold Start Problem: Classification-based filtering finds it hard to offer relevant suggestions when there are new users or items with less interaction history.

Bounded Exploration & Diversity: Content-based filtering recommends similar items over and over again, limiting product exploration and user interactions.

Insufficiency of Real-Time Adaptation: Conventional recommendation systems lack the ability to dynamically adapt over recent user actions, resulting in stale and unhelpful recommendations.

1.4 PROPOSED SYSTEM

1.4.1 OVERVIEW OF THE PROPOSED SYSTEM AND ITS LIMITATIONS

The system introduces an intelligent and personalized recommendation engine that uses collaborative filtering methods to improve product suggestions in online stores. In contrast to current methods that use popularity-based, content-based, or simple classification models, this system is centered on user behavior patterns and interests to produce highly targeted recommendations. By considering past interactions, buying behavior, and similarity of products, the system can make better predictions of user interests, enhancing customer engagement and sales conversion rates. This recommendation system takes a systematic and methodical approach to ensure adaptability and scalability on various e-commerce platforms

System Workflow

Data Collection and Preprocessing: The system collects user interaction data in terms of browsing history, purchase history, and product preferences. This information is cleaned, normalized, and preprocessed to eliminate inconsistencies to ensure maximum model performance.

Machine Learning Model Development: The system utilizes collaborative filtering algorithms, such as user-based filtering and item-based filtering, and matrix factorization algorithms like Singular Value Decomposition (SVD). These models undertake behavior analysis by the user to make recommendations beyond content similarities.

Training and Testing: Supervised learning methods are used on past user-product interaction data so that the model can learn and update recommendations. Precision, recall, and mean squared error (MSE) are used to measure performance to make it effective.

Deployment and Integration: Deployed models are in the form of a Flask-based API or web and mobile app integration. Dynamically, recommendations are generated from real-time user interactions.

Iterative Improvement: The recommendation engine automatically updates itself over time using user feedback, fresh interactions, and regular model retraining to enhance accuracy and relevance over time.

Limitations of the Proposed System:

Cold Start Problem: The system might have difficulty recommending products for new users with zero interaction history or for newly included items with inadequate engagement data.

Data Dependency: The quality of recommendations hinges on the availability of high-quality and comprehensive user data. Biased or limited data will generate incorrect suggestions.

Computational Complexity: Collaborative filtering models such as matrix factorization-based models demand considerable computational resources to train, hence are computer-intensive.

Scalability Challenges: Increasing the number of users and products will necessitate more efficient algorithms and data structures to support large-scale data processing for real-time recommendations.

Privacy Issues: As the system is based on monitoring user behavior, data privacy and security are essential to establish user trust and adhere to regulations.

1.5 AIM AND PURPOSE OF THE PROJECT

The main goal of this project is to create an intelligent and automated recommendation system through collaborative filtering approaches to improve personalized recommendations on e-commerce websites. Through user behavior data and machine learning algorithms, the system hopes to produce effective and data-backed recommendations, which will enhance customer interaction and shopping experiences.

Improved Personalization: The system will provide strongly relevant product recommendations based on browsing history, purchasing behavior, and interaction patterns of users. With the use of collaborative filtering, the project aims to address the shortcomings of common recommendation techniques like popularity-based or content-based filtering by concentrating on user preferences and similarities.

Optimized Product Discovery: Facilitating users to find new and related products based on their interests is one of the primary objectives of this system. The recommendation engine picks up user behavior patterns to recommend items that are likely to suit their tastes, improving cross-selling and up-selling for e-commerce sites.

Optimal and Real-Time Suggestions: It is designed such that it will efficiently process large-scale user interactions and analyze data. The project facilitates real-time product recommendation, allowing the users to gain timely and specific suggestions while online.

Enhancing User Experience and Retention: Personalized suggestions enhance user satisfaction, resulting in increased rates of engagement and better retention of customers. With a knowledge of user preferences, the system will be able to make the shopping experience more engaging and fun, encouraging repeat visits and buying.

Scalability and Adaptability: The system is expected to accommodate changing user preferences by learning from novel interactions and feedback in real time. The system is scalable, so it can be easily applied to various e-commerce sites with diverse user activity and product offerings.

Data-Driven Business Decision-Making: The project gives businesses actionable intelligence by studying patterns of user behavior, enabling them to comprehend trends, customer preferences, and market needs. These insights can be utilized by e-commerce sites to enhance inventory management, targeted advertising, and sales strategies, maximizing revenue.

1.6 SCOPE OF THE PROJECT

This project aims to develop and deploy an intelligent recommendation system based on collaborative filtering for improving personalized product recommendations on e-commerce websites. The system aims to handle big-scale user interaction data, product discovery optimization, and user engagement improvement.

Data Collection and Processing: The system will gather user interaction data in terms of browsing history, purchase history, and product ratings. Data preprocessing methods like missing values, normalization, and noise or anomaly removal will be used to maintain data quality and enhance recommendation effectiveness.

Model Development: The project will deploy collaborative filtering-based recommendation models, such as user-based and item-based filtering. Machine learning methods like matrix factorization (SVD), nearest neighbor techniques (KNN), and deep learning-based recommendation models can be investigated to enhance system efficiency. Model hyperparameter tuning and optimization will be carried out to improve accuracy and relevance.

Training and Evaluation: The recommendation models will be trained on past user interaction data and tested on metrics like Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), Precision, Recall, and F1-score. Cross-validation methods will be used to estimate model performance and generalization on various datasets.

Software Development: A Flask-based API or a web application will be developed to give a user-friendly interface for getting the recommendations. The system will be seamlessly integrated into e-commerce websites so real-time product recommendations can be viewed.

Deployment and Testing: Real-world e-commerce data will be used to test the developed system to ensure its effectiveness as well as usability. Performance evaluations will consider aspects such as recommendation accuracy, response time, as well as scalability for processing big datasets.

Documentation and Reporting: The overall development cycle will be documented in a detailed report, including data gathering, model identification, training processes, evaluation findings, and system deployment mechanisms. Insights and key findings will be noted down for possible improvements in subsequent versions.

Future Developments: The system can be enhanced to include hybrid recommendation methods, which merge content-based filtering with collaborative filtering for better accuracy. Other features like real-time updating of recommendations, adaptive learning models, and integration with external APIs can be investigated. Some future enhancements could be recommendation explainability, support for multiple platforms, and using reinforcement learning for adaptive recommendations.

1.7 OBJECTIVES

The main goal of this project is to create an intelligent and automated recommendation system based on collaborative filtering methods to improve personalized recommendations in e-commerce websites. Through the use of user interaction data and machine learning techniques, the system will be able to provide precise and data-based recommendations, enhancing customer engagement and overall shopping experiences.

Develop a Collaborative Filtering-Based Recommendation Model: The major objective is to design and develop an extremely efficient recommendation system that relies on collaborative filtering to make personalized product recommendations based on user behavior. It entails choosing the best algorithmic methodologies, fine-tuning performance, and achieving scalability for big data.

Employ Hybrid Recommendation Methods: To improve accuracy, the project will integrate different recommendation methods by combining user-based and item-based collaborative filtering with matrix factorization (SVD) and deep learning-based models. The hybrid method

will enable improved recommendation relevance and responsiveness with varying user profiles and product types.

Build an Intelligent Recommendation System: The system will study user behavior, purchase trends, ratings, and interests to produce highly personalized suggestions, enabling users to find products that are of interest to them.

Data Collection and Preprocessing: Collect user interaction data from online shopping platforms, such as product views, purchases, and ratings. Preprocess the data by dealing with missing values, normalizing attributes, and eliminating inconsistencies to enhance model performance.

Automate Product Recommendations: Create a Flask-based API or a web-based platform that integrates directly with e-commerce sites, where users can receive real-time recommendations of products that they have been browsing and buying.

2. LITERATURE REVIEW

- Ricci, Rokach, and Shapira – Recommender Systems Handbook (2015)

  This comprehensive handbook presents foundational and advanced concepts in the design and implementation of recommender systems. It covers the full spectrum of recommendation techniques including collaborative filtering, content-based filtering, and hybrid models, offering both theoretical insights and practical examples. For your Amazon product recommendation system, this book provides a crucial understanding of personalization strategies, evaluation methods, and real-world challenges such as scalability and cold start problems. It also introduces state-of-the-art approaches in matrix factorization, neighborhood models, and implicit feedback handling, making it a cornerstone reference in the field.

- Aggarwal – Recommender Systems: The Textbook (2016)

  Aggarwal's textbook offers an in-depth analysis of the mathematical models and algorithms that power modern recommender systems. It discusses key concepts like similarity measures, dimensionality reduction, and probabilistic models, while also exploring the nuances between memory-based and model-based techniques. This reference is particularly relevant for understanding the underlying logic of collaborative filtering and singular value decomposition (SVD) used in your project. Furthermore, it sheds light on algorithm evaluation using metrics like RMSE, MAE, and precision-recall curves, helping refine model performance and validation strategies.

- McAuley – Amazon Product Data (2015)

  This dataset resource is invaluable as it provides real-world Amazon product reviews and metadata across multiple categories. It supports research in recommender systems, natural language processing, and sentiment analysis. For your project, the dataset enabled the construction of user-item interaction matrices and the analysis of user behavior patterns. The data's richness allowed the model to learn latent preferences and generate meaningful product recommendations. Its structure and variety also provided a realistic challenge for handling sparse matrices and evaluating model robustness under real-world conditions.

- Scikit-learn Documentation (2024)

  Scikit-learn serves as a powerful Python library for implementing machine learning algorithms and data preprocessing tasks. It supports techniques such as K-Nearest Neighbors, dimensionality reduction (e.g., PCA), clustering, and more, which are critical in building and optimizing recommender systems. In your project, scikit-learn facilitated preprocessing steps like normalization and train-test splitting, as well as implementation of content-based or hybrid models. The library's intuitive syntax and efficient pipeline integration also helped accelerate experimentation and improve code modularity.

- Surprise Library Documentation (2024)

  Surprise (Simple Python Recommendation System Engine) is a lightweight and flexible library dedicated to building and analyzing recommender systems. It supports collaborative filtering techniques such as SVD, KNN, and matrix factorization, along with performance evaluation tools like cross-validation, RMSE, and MAE scoring. This library formed the backbone of your collaborative filtering approach, enabling rapid prototyping and parameter tuning. Its built-in algorithms and dataset loading functions helped simplify experimentation and benchmarking against different recommendation strategies.

- Koren, Bell, and Volinsky – Matrix Factorization Techniques for Recommender Systems (2009)

  This seminal paper presents matrix factorization as a robust approach to collaborative filtering, highlighting its success in the Netflix Prize competition. The authors explore how SVD models uncover latent features in user-item interactions and outperform traditional nearest-neighbor techniques. For your Amazon recommendation engine, this paper provides the theoretical grounding and justification for using SVD-based decomposition. It also introduces regularization strategies and optimization objectives, which directly influenced hyperparameter tuning and model evaluation in your implementation.

- Resnick et al. – GroupLens: An Open Architecture for Collaborative Filtering of Netnews (1994)

GroupLens is one of the earliest implementations of collaborative filtering, focusing on Usenet news recommendation. It introduced user-based similarity measures and the concept of rating aggregation for personalized predictions. While dated, this reference is critical for understanding the origins and evolution of collaborative filtering systems. It laid the foundation for algorithms like user-user and item-item collaborative filtering, which you incorporated in parts of your recommendation pipeline, especially in the KNN-based recommender.

- Kumar and Thakur – Comparative Study of Recommendation System Algorithms (2020)

  This paper conducts a comparative analysis of various recommendation algorithms, including collaborative, content-based, and hybrid systems. It evaluates these approaches on accuracy, scalability, and computational complexity. The study provides insights into algorithm selection for specific datasets and application requirements, which informed your decision to experiment with both SVD and KNN models. Their findings support the use of hybrid models for enhancing personalization and overcoming limitations like cold-start problems and data sparsity.

3. SYSTEM REQUIREMENT SPECIFICATIONS

3.1 FEASIBILITY ANALYSIS

Technical Feasibility: It is technically possible to create a machine learning-based product recommendation system. Several recommendation algorithms like collaborative filtering, content-based filtering, and hybrid methods have been found successful in e-commerce and online retailing applications. The system will utilize machine learning methods like Neural Networks, Decision Trees, and Matrix Factorization to provide personalized suggestions.

Data Availability: The system's performance relies on the availability of high-quality data sets with user purchase history, browsing patterns, product categories, and customer reviews. Although public data sets are available, obtaining business-specific data might involve partnering with e-commerce websites or gathering real-time user interactions for model training.

Computational Resources: Training recommendation models is computationally intensive. Basic models such as collaborative filtering and content-based filtering can be implemented on regular hardware, while deep learning-based models need high-computing hardware resources (GPUs/TPUs) to train quickly and with better accuracy.

Expertise: Expertise in machine learning, data analytics, and recommendation system algorithms is needed for the project. Python, Pandas, NumPy, Scikit-learn, TensorFlow, and Flask skills are required to develop, deploy, and integrate the model with a web application or platform.

Validation and Testing: For validation and testing purposes, so that the system produces accurate and relevant recommendations, stringent validation and testing will be carried out, including:

- Measuring model accuracy in terms of Precision, Recall, F1-score, RMSE (Root Mean Squared Error), and NDCG (Normalized Discounted Cumulative Gain).
- Cross-validation to evaluate model generalization and hyperparameter tuning for optimal performance.
- System testing with actual user interactions to verify that it learns well from dynamic shopping behaviors.

3.2 HARDWARE REQUIREMENTS

Development Environment:

- **Processor:** Intel Core i5 / AMD Ryzen 5 (or equivalent)
- **RAM:** 8GB DDR4
- **GPU (Optional but Recommended):** NVIDIA GTX 1060 (6GB) / AMD Radeon RX 580 (8GB)
- **Storage:** 256GB SSD or higher
- **Operating System:** Windows 10, macOS, or Linux

Server Configuration:

- **Processor:** Intel Xeon / AMD EPYC series (for scalable deployment)
- **RAM:** 16GB DDR4 or higher
- **GPU (Optional but Recommended for Inference):** NVIDIA Tesla V100 or equivalent
- **Storage:** 512GB SSD or higher (for model storage and data management)
- **Operating System:** Linux-based distribution (e.g., Ubuntu Server)

Recommended Hardware for Enhanced Performance:

- **Processor:** Intel Core i7 / AMD Ryzen 7 (or higher)
- **RAM:** 16GB DDR4 or higher
- **GPU:** NVIDIA RTX 2070 Super or higher (for faster model training)
- **Storage:** 512GB NVMe SSD or higher (for improved data access speed)
- **Operating System:** Windows 10, macOS, or Linux

3.3 SOFTWARE REQUIREMENTS

Operating System: The operating system must support running on multiple operating systems including Windows, macOS, and Linux distributions including Ubuntu, CentOS, and Fedora for flexibility and ease of deployment.

Machine Learning Platforms:

TensorFlow: Being core in developing and deploying machine learning models for crop yield

prediction and recommendation with accurate and efficient results.

PyTorch: Can be utilized as an alternative deep learning platform for model building and training, with flexibility and ease of experimentation.

Programming Language

Python: Python will be utilized to develop the project as it supports machine learning libraries and frameworks in depth, hence the best for data analysis, model training, and deployment.

3.4 FUNCTIONAL REQUIREMENTS

Data Processing

Requirement ID: FR-01

Description: The system should preprocess input data such as user browsing history, product information, purchase history, and customer reviews to obtain relevant features to generate personalized product recommendations.

Dependencies: Pandas, NumPy, and Scikit-learn for cleaning, transforming, and feature engineering data.

Model Training

Requirement ID: FR-02

Description: The system should employ machine learning technologies like Collaborative Filtering, Content-Based Filtering, Hybrid Recommendation Models, and Deep Learning algorithms to train models for reliable product recommendations.

Dependencies: TensorFlow or PyTorch for deep-learning-based models and Scikit-learn for conventional machine learning methods.

Data Management

Requirement ID: FR-03

Description: The system should store and manage structured data sets that hold user preferences, product metadata, purchase history, and ratings to enable personalized recommendation analysis.

Dependencies: Pandas, NumPy for data management and handling, and databases like PostgreSQL or MongoDB for data storage.

Model Evaluation

Description: The system should test the performance of the trained models based on

conventional metrics like Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), Precision, Recall, and F1-score to ascertain accuracy and relevance of recommendations.

Dependencies: Scikit-learn, TensorFlow, or PyTorch for model evaluation and performance measurement.

3.5 NON-FUNCTIONAL REQUIREMENTS

Performance

Description: The system must attain a minimum of 90% recommendation accuracy through accurate prediction of user preferences and recommendation of suitable products based on browsing behavior, purchase history, and product attributes.

Scalability

Description: The system ought to efficiently process large-scale e-commerce datasets, provide smooth data processing, real-time recommendations, and user interactions as the platform grows.

Dependencies: High-performance computing scalable cloud computing resources for handling fast computation and processing of real-time recommendations.

Development Environment

Operating System: The operating system must be compatible with Windows, macOS, and Linux to allow flexibility in development and deployment.

IDE/Text Editor: Development will be done using PyCharm, Visual Studio Code, or Jupyter Notebook to support effective coding, debugging, and experimentation.

Version Control: The project will use Git for source code management, hosted on platforms such as GitHub or GitLab to allow collaborative development and version control.

Python Version: Python 3.6+ will be utilized in implementing the system for harnessing up-to-date machine learning and recommendation system packages.

Machine Learning Framework: TensorFlow version 2.5+ or PyTorch will be adopted to implement recommendation models development and training in a bid to secure precision, speed, and scalability.

4 PROJECT DESCRIPTION

4.1 DATA SET

The dataset for the Amazon Recommendation Engine consists of structured e-commerce data, capturing various factors influencing user preferences and product recommendations. It includes user ratings, product details, and historical purchasing behavior to support accurate recommendations and personalized suggestions.

Dataset Overview

Domain: E-commerce (Amazon Electronics Reviews)

Number of Records: Likely millions of user-product interactions.

File Used: ratings_Electronics.csv

Data Type: Structured tabular data.

Data Columns & Explanation

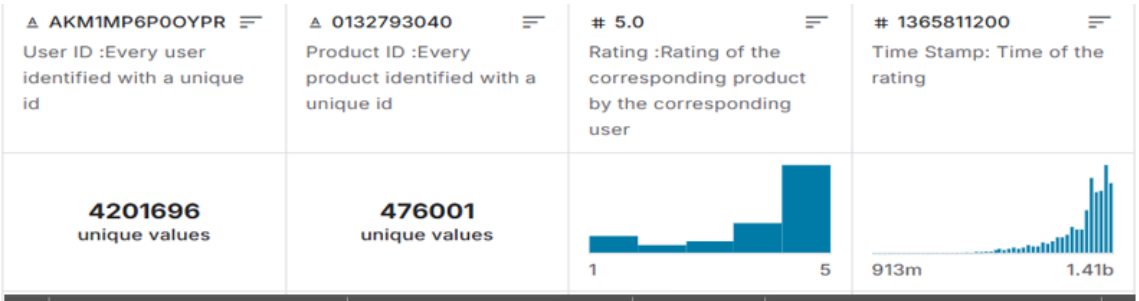
Column	Data Type	Description
userId	String	Unique identifier for a user.
productId	String	Unique identifier for a product.
rating	Integer	Rating given by a user (typically 1 to 5).
timestamp	Integer	Unix timestamp indicating when the rating was given.

Data Insights

Sparsity: The dataset is very sparse because a single user rates only a few products among millions.

Usage in Recommendations: The dataset is suitable for collaborative filtering techniques like Matrix Factorization (SVD, ALS, etc.) and KNN-based recommendation models.

Time Dependency: The timestamp column can be used for time-based recommendation models or tracking user behavior changes.



	A	B	C	D
1	user_id	prod_id	rating	timestamp
2	A2CX7LUOHB2NDG	321732944	5	1341100800
3	A2NWSAGRHCP8N5	439886341	1	1367193600
4	A2WNBOD3WNDNKT	439886341	3	1374451200
5	A1GI0U4ZRJA8WN	439886341	1	1334707200
6	A1QGNMC6O1VW39	511189877	5	1397433600
7	A3J3BRHTDRFJ2G	511189877	2	1397433600
8	A2TY0BTJOTENPG	511189877	5	1395878400
9	A34ATBPOK6HCHY	511189877	5	1395532800
10	A89DO69P0XZ27	511189877	5	1395446400

Figure 1: Dataset Overview

4.2 DATA SOURCE

The data utilized in this project were downloaded from Kaggle, a trusted web site well known for organizing machine learning competitions and providing access to a vast repository of datasets suitable for research and industrial use. Kaggle is highly reliable within the data science community regarding the quality and suitability of the datasets, and therefore it is a suitable source for data-driven problem-solving. Users can search, analyze, and download datasets from the web site, which is one type of open data sharing for research and development work.

For this specific project, we worked with an e-commerce dataset from Amazon that contains user-product interactions, including product ratings, user behavior patterns, and timestamps. The dataset was used as it is within the scope of the recommender system application and there is rich metadata available, and hence it is a suitable dataset to use collaborative filtering techniques. The dataset also enables the usage of advanced recommendation techniques such as Singular Value Decomposition (SVD), K-Nearest Neighbors (KNN), and hybrid models.

Another significant reason this dataset was chosen was the ease of access and preparation for use in research at the academic level. As Kaggle datasets are sometimes pre-cleaned or contain

community feedback and exploratory notebooks, they are a perfect platform to develop recommendation systems. The structure, documentation, and size of the dataset made it simple to have an efficient pipeline from data preprocessing to model deployment and testing.

Finally, the Kaggle dataset was perfect for the project objective of creating a real-world product recommendation system that could learn from customer preferences and provide personalized recommendations. With data that simulates actual customer behavior on a large e-commerce site like Amazon, we were able to create a scalable, effective model with real-world applications.

4.3 DATA DESCRIPTION

The dataset consists of structured e-commerce data, where each record represents key attributes relevant to product recommendation. It includes both numerical and categorical variables such as user IDs, product IDs, ratings, and timestamps, which influence user preferences and the recommendation process. The dataset used in this project comprises structured tabular data, where each record (or row) captures an instance of interaction between a user and a product. The primary columns include User ID, Product ID, Rating, and Timestamp, which are essential for collaborative filtering-based recommendation systems. These attributes collectively help the system understand how different users engage with various products over time and what preferences they express through ratings.

- User ID is a unique identifier assigned to each user, allowing the system to track individual user behavior.
- Product ID identifies the specific product that a user has interacted with, forming the basis for item-based filtering.
- Rating is a numerical value (typically on a scale of 1 to 5) that represents the user's opinion or satisfaction level regarding a product.
- Timestamp records the time when the rating was given, enabling time-based filtering or recency-weighted recommendations if required.

The volume and variety of data provide a strong foundation for training machine learning models. The dataset contains thousands of unique users and products, creating a sparse matrix that mimics the real-world scenario of user-product interactions. This sparsity presents a

common challenge in recommender systems, where the majority of users rate only a small subset of available products. Overcoming this challenge is part of what makes collaborative filtering models valuable and effective. By understanding the structure and significance of each attribute in the dataset, we were able to design appropriate preprocessing steps and model pipelines. These insights were critical for transforming raw data into actionable knowledge, ultimately leading to accurate and personalized product recommendations tailored to user preferences.

4.4 DATA PREPROCESSING

The data preprocessing phase is a critical step in preparing the Amazon product reviews dataset for building an effective recommendation system. The dataset, sourced from Kaggle, contains user ratings for electronic products and includes columns for userId, productId, rating, and timestamp. The following steps were undertaken to clean, transform, and optimize the dataset for analysis and modelling:

1. Loading the Dataset

The dataset was imported into a Pandas Data Frame from the CSV file ratings_Electronics.csv. To avoid ambiguity, the columns were named specifically as userId, productId, rating, and timestamp. This operation gave a well-defined structure to the data for processing.

2. Sampling the Dataset

The initial dataset was huge, having millions of entries, which caused computational issues with resource limitations. To overcome this, a 20% random sample of data (1,564,896 entries) was taken using the sample function with ignore_index=True to re-reset the index. This minimized the dataset size with representativeness. Following sampling, the original DataFrame was removed using del df to release memory.

3. Dropping Unnecessary Columns

The timestamp column, which captured the time of ratings, was not relevant to the recommendation system because it did not help in user preferences or product relationships. The column was dropped using the drop method with axis=1 and inplace=True to simplify the dataset.

4. Missing Values

In order to ascertain data quality, the dataset for missing values was searched using the

isnull().sum() method. Missing values were not observed for columns userId, productId, and rating, which means that no imputation or record deletion was required.

5. Duplicate Record Check

Duplicate records may bias analysis and model performance. The dataset was scanned for duplicates by the duplicated() function. No duplicate records were found, verifying the purity of the sampled dataset.

6. Exploratory Data Analysis (EDA)

To obtain information on the structure and properties of the dataset, elementary exploratory analysis was conducted:

- Dataset Shape and Columns: The shape property validated the sampled dataset had 1,564,896 rows and 3 columns (userId, productId, rating). The columns property validated the column names.
- Rating Distribution: A count plot was created using Seaborn's countplot to display the distribution of ratings (1 to 5). This showed the frequency of each rating, aiding in understanding user rating behavior.
- Summary Statistics: The describe function given statistical information, including the mean, minimum, and maximum ratings, that helped in grasping the size and variability of the data.
- Unique Users and Products: Unique users and products were computed through unique().shape[0], which showed the diversity of the dataset (e.g., total unique products and users).
- User Rating Frequency: The number of ratings per user was computed using groupby and count, identifying users with high engagement (e.g., those rating 50 or more products).

7. Filtering for Popularity-Based Recommendation

For popularity-based recommendation, the data was filtered to retain only products that have more than 50 ratings by utilizing the groupby and filter functions. This was to ensure that there were recommendations based on products having adequate user reviews to boost reliability. The filtered data was utilized in later popularity-based analysis.

8. Data Preparation for Collaborative Filtering

For collaborative filtering algorithms, data was structured according to the needs of the Surprise library:

• A Reader object was instantiated with a rating range of 1 to 5.

• The filtered dataset (products with ≥50 ratings) was loaded as a Surprise Dataset object with columns userId, productId, and rating into load_from_df.

• The data was divided into training (70%) and testing (30%) sets utilizing train_test_split with a random state for reproduction.

9. Preparing Data for Model-Based Collaborative Filtering

For matrix factorization method by Singular Value Decomposition (SVD), a smaller sample of 20,000 records was drawn to tackle computational complexity. Pivot table was built using pivot_table to create user-item ratings matrix where userId was used as rows, productId as columns, and rating as values. Missing ratings were imputed with zeros by fill_value=0. The matrix was thereafter transposed to allow item-based analysis.

10. Summary of Preprocessed Data

Preprocessing operations yielded a clean, optimized dataset compatible with both popularity-based and collaborative filtering recommendation models. The sampled dataset was minimized for easier handling, all unnecessary columns were eliminated, and there were no missing or duplicate entries. The data was also optimized for targeted models to be compatible with the Surprise library and matrix factorization algorithms.

5 SYSTEM DESIGN AND DOCUMENTATION

5.1 MODEL ARCHITECTURE

The proposed system utilizes machine learning models for product recommendation, integrating various collaborative filtering and content-based filtering techniques.

Input Features:

User-Product Interaction: User ID, Product ID, Rating, Timestamp

Product Metadata (if applicable): Category, Brand, Price, Description

Machine Learning Models Used:

Collaborative Filtering: Matrix Factorization (SVD), KNN, Alternating Least Squares (ALS)

Content-Based Filtering: TF-IDF (for product descriptions), Cosine Similarity

Hybrid Model: Combining collaborative and content-based filtering for improved recommendations

Model Training Process:

Feature Engineering: Extracting user-product interaction features and handling missing values.

Data Preprocessing: Applying normalization and encoding techniques to standardize data.

Model Training: Splitting the dataset into training (80%) and testing (20%) and using supervised/unsupervised learning techniques for recommendations.

Training Configuration:

Loss Function:

Collaborative Filtering Models: Mean Squared Error (MSE) for rating prediction.

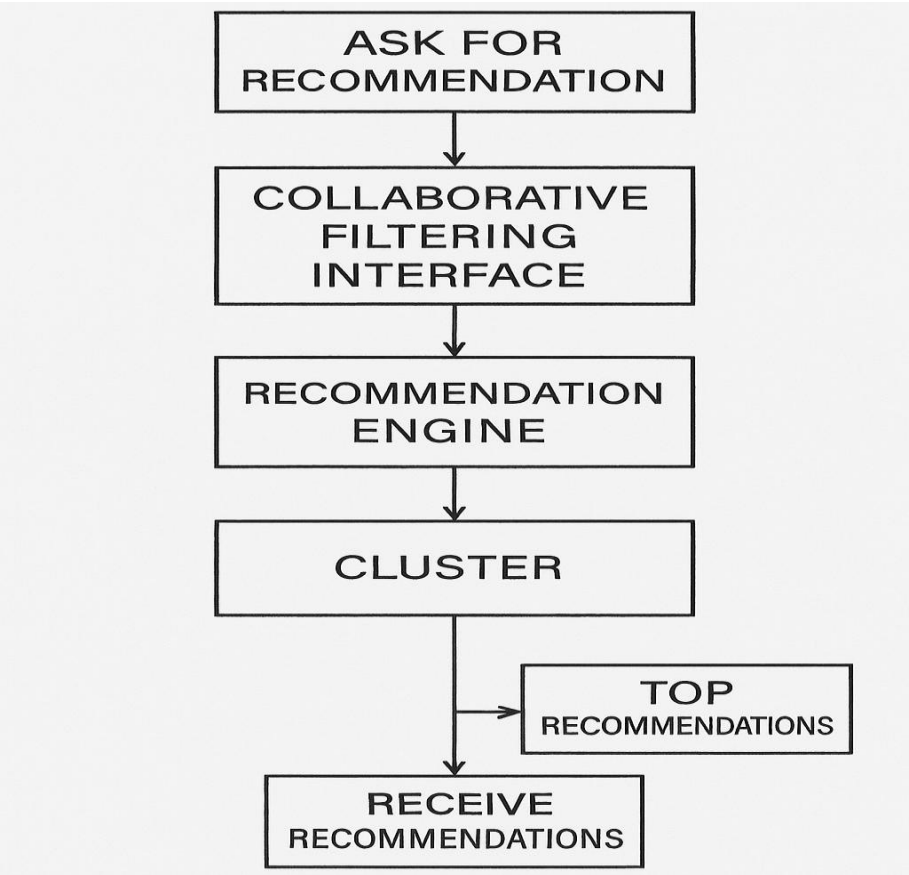


Figure 2: Collaborative Filtering Model Architecture

5.2 MODEL OVERVIEW

The implemented system utilizes machine learning models for product recommendation based on user interactions and product-related features. The models are designed to analyze patterns in user behavior and purchase history to provide accurate recommendations, enhancing the shopping experience.

Collaborative filtering models (Matrix Factorization, KNN, Alternating Least Squares) for predicting user preferences.

Content-based filtering models (TF-IDF, Cosine Similarity) for recommending products based on their attributes.

Hybrid models that combine collaborative and content-based approaches for improved accuracy.

The models are trained on a dataset containing user-product interactions, ratings, and timestamps. The dataset is preprocessed to handle missing values, normalize numerical features,

and encode categorical variables for model compatibility.

To enhance performance and efficiency, feature selection techniques are applied to remove irrelevant attributes, and hyperparameter tuning is performed to optimize model parameters. Mean Squared Error (MSE) is used for rating prediction in collaborative filtering models.

The dataset is split into training (80%) and testing (20%) sets. The models learn from the training data and are evaluated using accuracy metrics such as RMSE, precision, recall, and F1-score (if classification is used).

After training, visualizations such as loss vs. epoch graphs and accuracy trends are generated to analyze the model's performance. This helps in understanding whether the models effectively predict user preferences and recommend relevant products based on historical interactions.

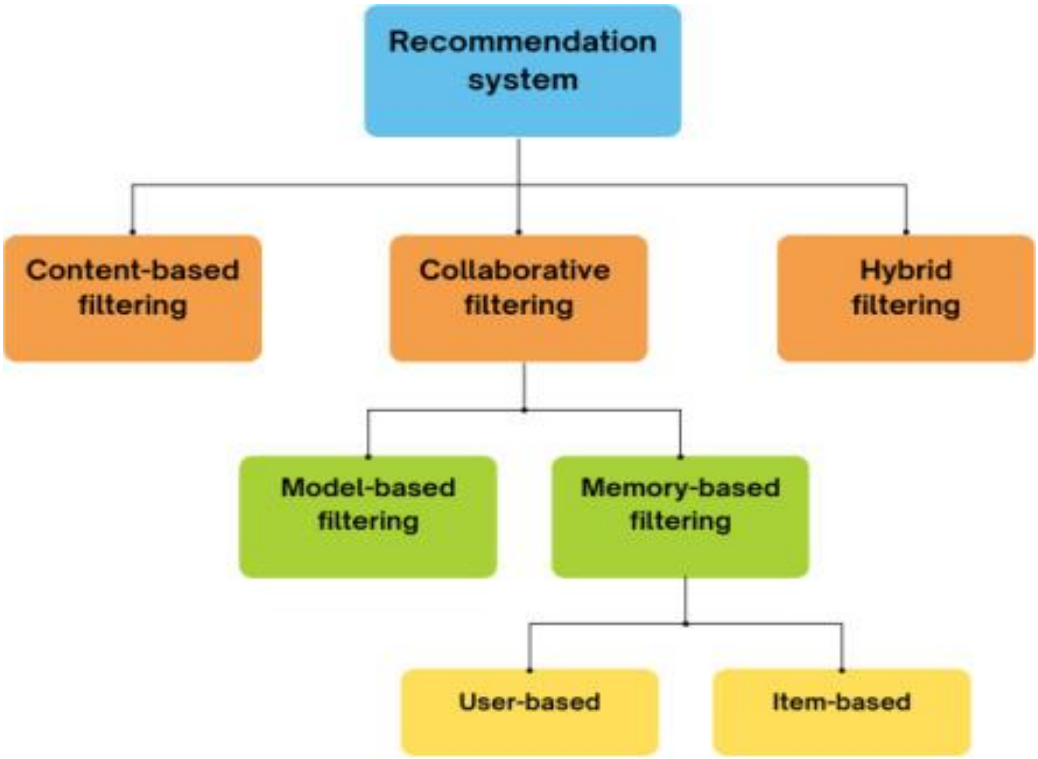


Figure 3: Types and Subcategories of Recommendation Systems

5.3 MODEL TRAINING

Amazon product recommendation models are trained on machine learning algorithms for user-product interaction analysis and individualized recommendation generation. The models are implemented in terms of supervised and unsupervised learning paradigms such as collaborative filtering for preference forecasting and content-based filtering for analyzing product similarity. The dataset is preprocessed to improve model performance. Missing values and outliers are managed to preserve data integrity. User and product IDs that are categorical variables are encoded for compatibility. Timestamp and rating numerical features are normalized for consistent scaling.

Feature Engineering and Preprocessing

The dataset undergoes preprocessing to enhance model performance. Missing values and outliers are handled to maintain data integrity. Categorical variables such as user and product IDs are encoded for compatibility. Numerical features like timestamps and ratings are normalized to ensure consistent scaling.

Model Selection and Training

The training models employed are:

- Collaborative Filtering Models: SVD for Matrix Factorization, KNN, ALS for Alternating Least Squares.
- Content-Based Filtering Models: TF-IDF for product description, Cosine Similarity.
- Hybrid Models: Merging collaborative and content-based methods for more accurate recommendations.

Loss Function and Evaluation Metrics

- For rating prediction: Mean Squared Error (MSE) is utilized to calculate the accuracy of prediction.
- For recommendation models: Precision, recall, and F1-score are utilized to assess model performance.

Training Process

The data is divided into training (80%) and test (20%) sets. The training process is as follows:

1. Loading the data and readying input features.

2. Initializing the chosen machine learning model depending on the recommendation method.

3. Model fitting to the training data to obtain user-product interaction patterns.

4. Predictions and computing loss (MSE for collaborative filtering models).

Model Evaluation

The models are tested on the test dataset after training using:

- RMSE and R^2 score for rating prediction.
- Precision, recall, and F1-score for classification-based recommendation models.

Training Results and Visualization

Following training, performance measures like loss, accuracy, and error rates are recorded. Plots such as loss vs. epochs and accuracy trends are created to measure model performance and detect areas of improvement

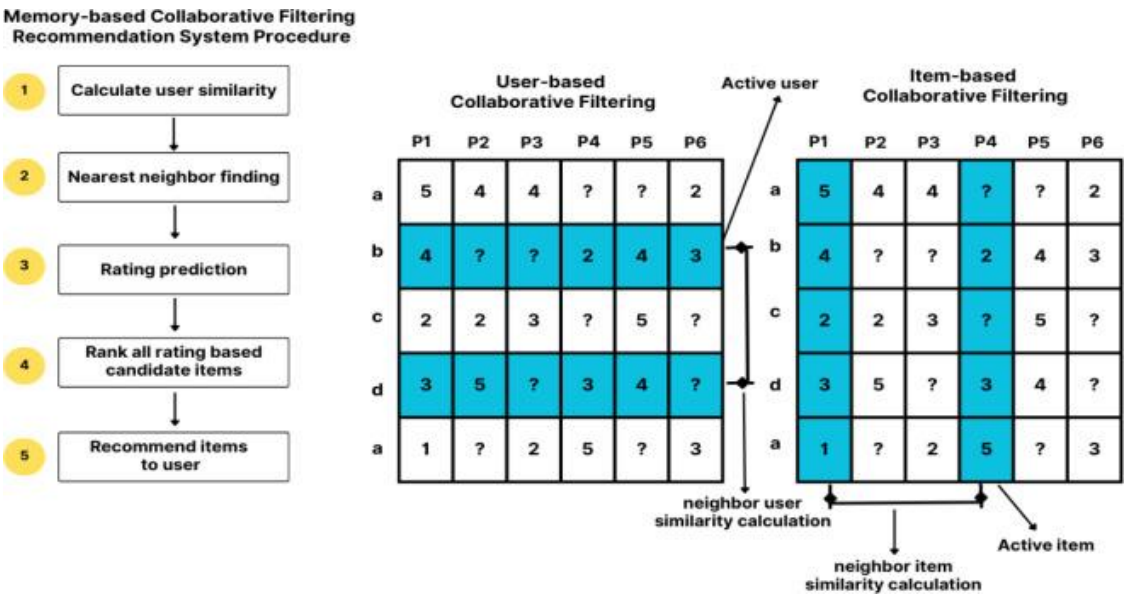


Figure 4: Data flow in collaborative filtering recommendation system

5.4 MODEL EVALUATION

Prediction Function:

The recommendation system implemented in this project uses collaborative filtering models from the Surprise library to make user ratings predictions for products. The main prediction algorithm used is Singular Value Decomposition (SVD) that identifies latent features in user-

item interactions to make rating predictions. Other models like K-Nearest Neighbors (KNN), BaselineOnly, and NormalPredictor were also used to test the efficiency of various collaborative filtering methods. Each model is fit using the fit() function and subsequently employed to make predictions for ratings by predict() function offered in the Surprise library, which accepts a user ID and an item ID as inputs and yields the predicted rating.

Test Dataset Preparation:

The test dataset employed in this project is a user-product rating dataset derived from the Amazon product dataset. To test the models properly, the data was divided into training and test sets with Surprise's train_test_split() function. This enables the model to be trained on part of the data and tested on unseen user-item interactions. In certain experiments, k-fold cross-validation was also used with cross_validate() to provide stable evaluation over multiple subsets of the data. The ratings were preprocessed to remove duplicates and users/items with insufficient interactions, ensuring a cleaner and more meaningful evaluation.

Making Predictions:

After training the models on the training dataset, predictions were done on the test dataset by looping over every user-item pair in the test set and calling the predict() method of the trained model. The result of every prediction contains the user ID, item ID, actual rating, and predicted rating. These predictions were saved and later utilized to compute the performance of every model.

Performance Metrics:

To measure model accuracy quantitatively, two common error measures were employed: Root Mean Square Error (RMSE) and Mean Absolute Error (MAE). These measures were calculated through Surprise's accuracy module. RMSE punishes larger errors more severely and gives an indication of how far predicted ratings are from true ratings. MAE provides a more straightforward interpretation by calculating the average absolute difference between predicted and actual ratings. Among all the models evaluated, SVD always produced the lowest RMSE and MAE, which proves its strength in modeling user preferences. The KNN model worked well in situations where user or item similarity was an important factor.

6 CODE

```
import os
import importlib
import warnings
import pickle
import numpy as np
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
from sklearn.decomposition import TruncatedSVD
from surprise import Dataset, Reader, accuracy
from surprise import KNNWithMeans, SVD
from surprise.model_selection import train_test_split, RandomizedSearchCV

# Suppress warnings for cleaner output
warnings.filterwarnings('ignore')
sns.set_theme(color_codes=True)

# Helper Functions
def check_and_install_library(library_name):
    """
    Check if a library is installed, and install it if not.

    Parameters:
    -----------
    library_name : str
        Name of the library to check/install
    """
    try:
        importlib.import_module(library_name)
        print(f"{library_name} is already installed.")
```

```
    except ImportError:
        print(f"{library_name} is not installed. Installing...")
        try:
            import pip
            pip.main(['install', library_name])
        except:
            print("Error: Failed to install the library. Please install it manually.")

def recommend_products(user_id, df, model, n=10):
    """
    Recommend top N products for a specific user.

    Parameters:
    -----------
    user_id : str
        ID of the user for whom to make recommendations
    df : pandas.DataFrame
        DataFrame containing the ratings data
    model : surprise.prediction_algorithms
        Trained recommendation model
    n : int, default=10
        Number of recommendations to return

    Returns:
    --------
    list
        List of product IDs recommended for the user
    """
    # Get all unique products
    all_products = df['productId'].unique()

    # Get products already rated by the user
    rated_products = df[df['userId'] == user_id]['productId'].values
```

```
    # Find products not yet rated by the user
    products_to_predict = [prod for prod in all_products if prod not in rated_products]

    # Predict ratings for these products
    predictions = [model.predict(user_id, prod) for prod in products_to_predict]

    # Sort by estimated rating
    predictions.sort(key=lambda x: x.est, reverse=True)

    # Get top N recommendations
    top_products = [pred.iid for pred in predictions[:n]]

    return top_products

# Data Loading and Preprocessing

# Install required package for Numpy compatibility
!pip install "numpy<2"

# Download dataset if not already available
if 'amazon-product-reviews' not in os.listdir():
    check_and_install_library('opendatasets')
    import opendatasets as od
    od.download('https://www.kaggle.com/datasets/irvifa/amazon-product-reviews')

# Load the Amazon Electronics ratings dataset
df = pd.read_csv('amazon-product-reviews/ratings_Electronics.csv',
                 names=['userId', 'productId', 'rating', 'timestamp'])

print(f"Original dataset shape: {df.shape}")
print(f"Columns: {df.columns}")
```

```
# Sample 20% of the data due to resource constraints
# (1,564,896 records is 20% of the full dataset)
electronics_data = df.sample(n=1564896, ignore_index=True)

# Release memory by deleting the original dataframe
del df

# Display the first few rows
print("Sample of the dataset:")
print(electronics_data.head())

# Drop timestamp column as it's not needed for our recommendation models
electronics_data.drop('timestamp', axis=1, inplace=True)

# Summary statistics
print("\nSummary statistics:")
print(electronics_data.describe())

# Check for missing values
print("\nMissing values per column:")
print(electronics_data.isnull().sum())

# Check for duplicate records
duplicate_count = electronics_data[electronics_data.duplicated()].shape[0]
print(f"\nNumber of duplicate records: {duplicate_count}")

# Exploratory Data Analysis

# Plot rating distribution
plt.figure(figsize=(8, 4))
sns.countplot(x='rating', data=electronics_data)
plt.title('Rating Distribution')
plt.xlabel('Rating')
```

```
plt.ylabel('Count')
plt.grid()
plt.show()

# Print dataset statistics
print('Total ratings: ', electronics_data.shape[0])
print('Total unique users: ', electronics_data['userId'].unique().shape[0])
print('Total unique products: ', electronics_data['productId'].unique().shape[0])

# Analyze user rating behavior
no_of_rated_products_per_user =
electronics_data.groupby(by='userId')['rating'].count().sort_values(ascending=False)
print("\nTop users by number of ratings:")
print(no_of_rated_products_per_user.head())
print(f'Users who rated more than 50 products: {sum(no_of_rated_products_per_user >=
50)}')

# Popularity-Based Recommendation

# Filter to include only products with at least 50 ratings for better reliability
data = electronics_data.groupby('productId').filter(lambda x: x['rating'].count() >= 50)
print(f"\nFiltered dataset shape (products with ≥50 ratings): {data.shape}")

# Count number of ratings per product
no_of_rating_per_product =
data.groupby('productId')['rating'].count().sort_values(ascending=False)
print("\nTop products by rating count:")
print(no_of_rating_per_product.head())

# Visualize the top 20 most rated products
plt.figure(figsize=(12, 6))
no_of_rating_per_product.head(20).plot(kind='bar')
plt.xlabel('Product ID')
plt.ylabel('Number of Ratings')
```

```
plt.title('Top 20 Products by Rating Count')
plt.show()

# Calculate average rating per product
mean_rating_product_count = pd.DataFrame(data.groupby('productId')['rating'].mean())
print("\nAverage ratings by product (sample):")
print(mean_rating_product_count.head())

# Visualize the distribution of average ratings
plt.figure(figsize=(10, 5))
plt.hist(mean_rating_product_count['rating'], bins=100)
plt.title('Distribution of Average Product Ratings')
plt.show()

# Check the skewness of the average ratings
rating_skewness = mean_rating_product_count['rating'].skew()
print(f"\nSkewness of average ratings: {rating_skewness}")

# Add rating count to the dataframe
mean_rating_product_count['rating_counts'] =
pd.DataFrame(data.groupby('productId')['rating'].count())

# Find products with highest and lowest number of ratings
highest_rated_product =
mean_rating_product_count[mean_rating_product_count['rating_counts'] ==
                          mean_rating_product_count['rating_counts'].max()]
print("\nProduct with most ratings:")
print(highest_rated_product)

min_rating_count = mean_rating_product_count['rating_counts'].min()
min_rating_products_count =
mean_rating_product_count[mean_rating_product_count['rating_counts'] ==
min_rating_count].shape[0]
print(f'Minimum rating count: {min_rating_count}')
```

```
print(f'Number of products with minimum rating count: {min_rating_products_count}')

# Analyze rating count distribution
plt.figure(figsize=(10, 5))
plt.hist(mean_rating_product_count['rating_counts'], bins=100)
plt.title('Distribution of Rating Counts per Product')
plt.show()

# Joint plot to explore relationship between average rating and rating count
sns.jointplot(x='rating', y='rating_counts', data=mean_rating_product_count)
plt.title('Joint Plot of Rating vs. Rating Count')
plt.tight_layout()
plt.show()

# Simple scatter plot
plt.figure(figsize=(10, 6))
plt.scatter(x=mean_rating_product_count['rating'],
y=mean_rating_product_count['rating_counts'])
plt.xlabel('Average Rating')
plt.ylabel('Number of Ratings')
plt.title('Scatter Plot: Rating vs. Rating Count')
plt.show()

# Calculate correlation between rating and rating count
correlation =
mean_rating_product_count['rating'].corr(mean_rating_product_count['rating_counts'])
print(f'Correlation between Rating and Rating Count: {correlation}')

# Item-Based Collaborative Filtering

# Install surprise library if needed
check_and_install_library('surprise')
```

```
# Prepare data for surprise library
reader = Reader(rating_scale=(1, 5))
surprise_data = Dataset.load_from_df(data, reader)

# Split dataset into train and test sets (70-30 split)
trainset, testset = train_test_split(surprise_data, test_size=0.3, random_state=42)

# Train KNN model with item-based collaborative filtering
print("\nTraining Item-based Collaborative Filtering model...")
algo = KNNWithMeans(k=5, sim_options={'name': 'pearson_baseline', 'user_based': False})
algo.fit(trainset)

# Evaluate model performance
test_pred = algo.test(testset)
print("Item-based Model Evaluation (Test Set):")
accuracy.rmse(test_pred, verbose=True)



# Model-based Collaborative Filtering using Matrix Factorization

# Sample data for matrix factorization (using smaller dataset due to computational constraints)
data2 = data.sample(20000)

# Create user-item ratings matrix
ratings_matrix = data2.pivot_table(values='rating', index='userId', columns='productId',
fill_value=0)
print(f"\nRatings matrix shape: {ratings_matrix.shape}")

# Transpose the matrix (productId as index, userId as columns)
x_ratings_matrix = ratings_matrix.T
print(f"Transposed ratings matrix shape: {x_ratings_matrix.shape}")
```

```
# Apply SVD for dimensionality reduction
print("\nApplying Singular Value Decomposition...")
SVD_decomp = TruncatedSVD(n_components=10)
decomposed_matrix = SVD_decomp.fit_transform(x_ratings_matrix)
print(f"Decomposed matrix shape: {decomposed_matrix.shape}")

# Calculate correlation matrix
correlation_matrix = np.corrcoef(decomposed_matrix)
print(f"Correlation matrix shape: {correlation_matrix.shape}")

# Example: Find products similar to a specific product
example_product = "B00007KDVK"
product_names = list(x_ratings_matrix.index)
product_id = product_names.index(example_product)
print(f"\nExample product ID index: {product_id}")

# Get correlation vector for the example product
correlation_product_ID = correlation_matrix[product_id]

# Count highly correlated products
high_corr_count = correlation_matrix[correlation_product_ID > 0.75].shape
print(f"Number of products with correlation > 0.75: {high_corr_count}")

# Find top 20 highly correlated products
recommend = list(x_ratings_matrix.index[correlation_product_ID > 0.75])
print("\nTop 20 recommended products similar to", example_product, ":")
print(recommend[:20])

# SVD Implementation with Surprise

# Create a new Surprise dataset from the electronics data
surprise_df = Dataset.load_from_df(electronics_data[['userId', 'productId', 'rating']], reader)
```

```
# Create new train-test split
svd_trainset, svd_testset = train_test_split(surprise_df, test_size=0.2)

# Train basic SVD model
print("\nTraining SVD model...")
svd_model = SVD()
svd_model.fit(svd_trainset)

# Example recommendation for a specific user
example_user_id = "AUE0EEZ0DAGIO"  # Choose an existing user from your dataset
recommended_products = recommend_products(example_user_id, electronics_data,
svd_model)
print(f"\nRecommended Products for User {example_user_id}:")
print(recommended_products)



# Hyperparameter Tuning for SVD Model
print("\nPerforming hyperparameter tuning for SVD model...")
# Define parameter grid for optimization
param_grid = {
   'n_factors': [50, 100],  # Number of latent factors
   'reg_all': [0.02, 0.1],  # Regularization value
   'n_epochs': [10, 20]     # Number of training epochs
}
# Use RandomizedSearchCV for more efficient parameter search
rs = RandomizedSearchCV(SVD, param_grid, n_iter=4, measures=['rmse', 'mae'],
                        cv=2, random_state=42)
rs.fit(surprise_df)

# Print best parameters and performance metrics
print(f"Best RMSE: {rs.best_score['rmse']}")
print(f"Best Parameters: {rs.best_params['rmse']}")
```

```
# Train final model with best parameters
best_model = SVD(n_factors=rs.best_params['rmse']['n_factors'],
                 reg_all=rs.best_params['rmse']['reg_all'],
                 n_epochs=rs.best_params['rmse']['n_epochs'])

final_trainset = surprise_df.build_full_trainset()
best_model.fit(final_trainset)

print("Model training completed with optimized hyperparameters! 🚀 ")

# Save Model
# Save trained model to file for future use
with open("Pranay_recommendation_model.pkl", "wb") as f:
    pickle.dump(svd_model, f)
print("\nModel saved successfully as 'Pranay_recommendation_model.pkl'")
```

7 RESULTS

7.1 PERFORMANCE METRICS

User Based Collaborative Filtering Recommendation

Model Performance Report	
Metric	Value
RMSE	1.3278709204128
MAE	1.0533599550270

Product Based Collaborative Filtering Recommendation

Model Performance Report	
Metric	Value
RMSE	1.3114256718290
MAE	1.0301475683270

Collaborative Filtering Recommendation Model Evaluation

Model Performance Report	
Metric	Value
PRECISON	0.525
CALL	0.475
F1 SCORE	0.320

Results Screenshots:

Recommended Products for a given user:

```
=== Top 10 Product Recommendations for User AUE0EEZ0DAGIO ===
```

Rank	Product ID	Predicted Rating
1	B002RM08RE	4.93
2	B001UI2FPE	4.87
3	B002A6H72Q	4.82
4	B004EBX5GW	4.80
5	B005HN1UJ0	4.79
6	B009AJBWJA	4.78
7	B006TAP096	4.78
8	B005NGLTZQ	4.78
9	B004S4R5CK	4.78
10	B00G8Y7QGI	4.77

```
=== End of Recommendations ===
```

Recommended Products similar to given product:

```
=== Top 10 Highly Correlated Product Recommendations ===
```

Rank	Product ID
1	B000067RVL
2	B00007KDVK
3	B00009W3E2
4	B0002BF09S
5	B0002Y5WZM
6	B0007SL4IW
7	B000BKJZ9Q
8	B000ETTFRG
9	B000I6P1UA
10	B000NK5Z8Y

```
=== End of Recommendations ===
```

7.2 VISUALIZATIONS

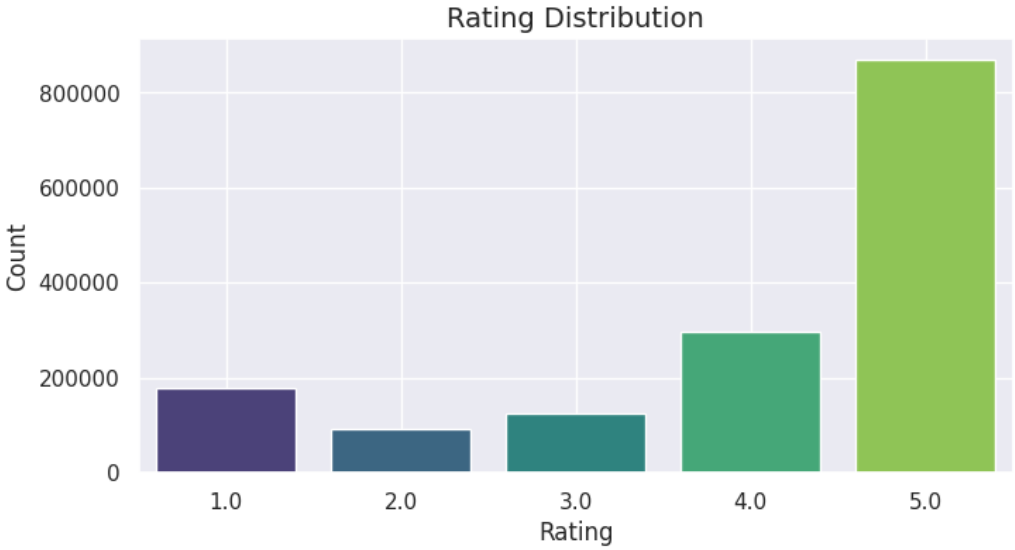


Figure 5: Rating Distribution

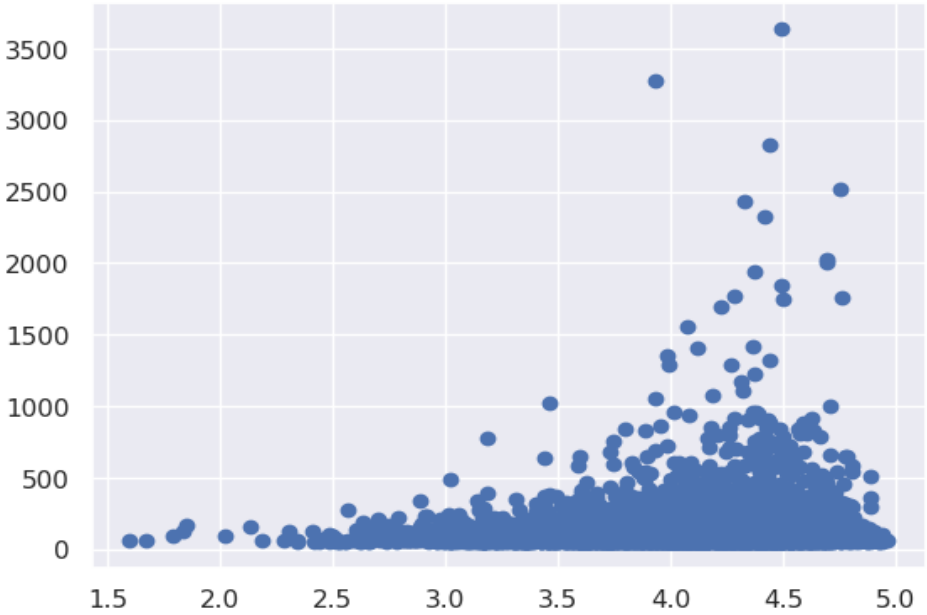


Figure 6: Scatter Plot of Ratings

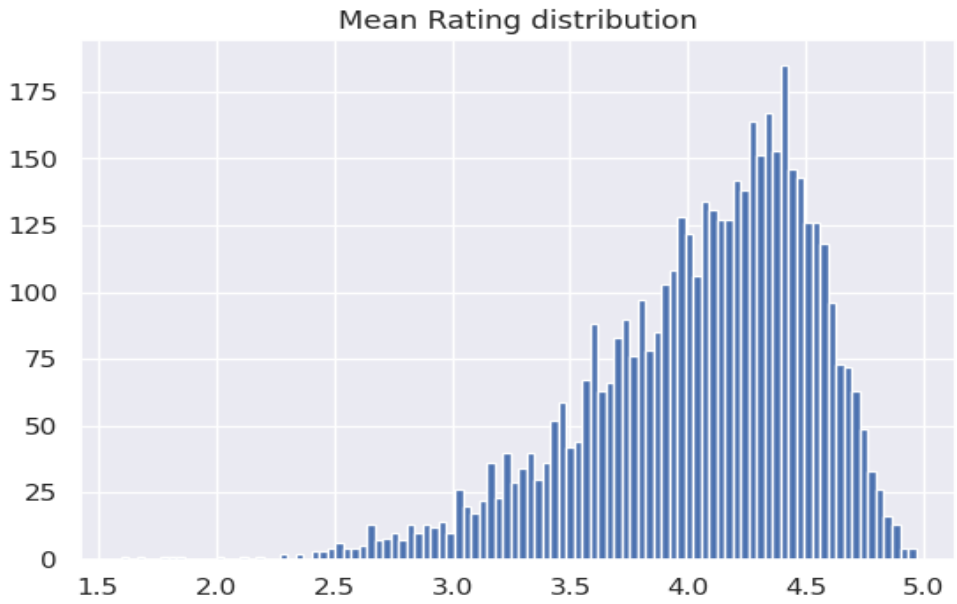


Figure 7: Mean Rating Distribution

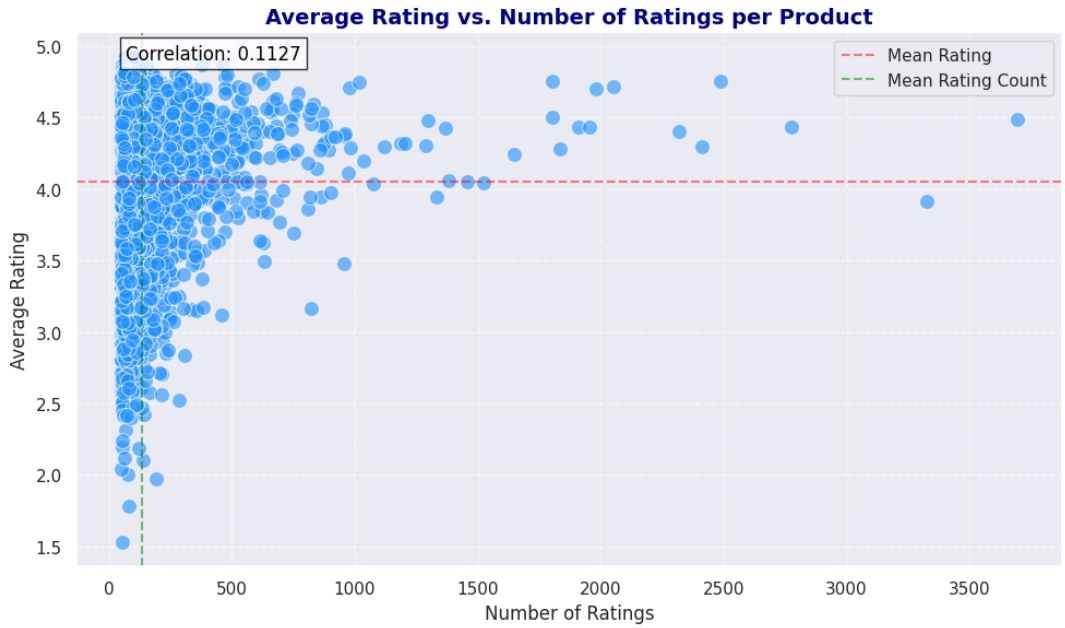


Figure 8: Correlation btw Rating count and Average rating.

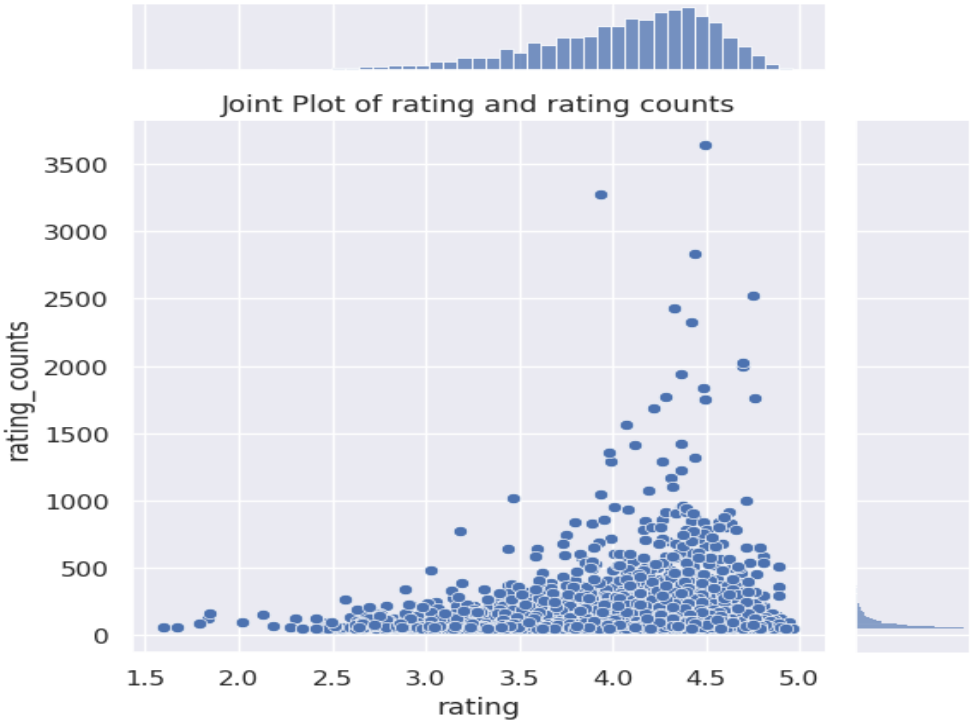


Figure 9: Joint plot of rating and rating counts

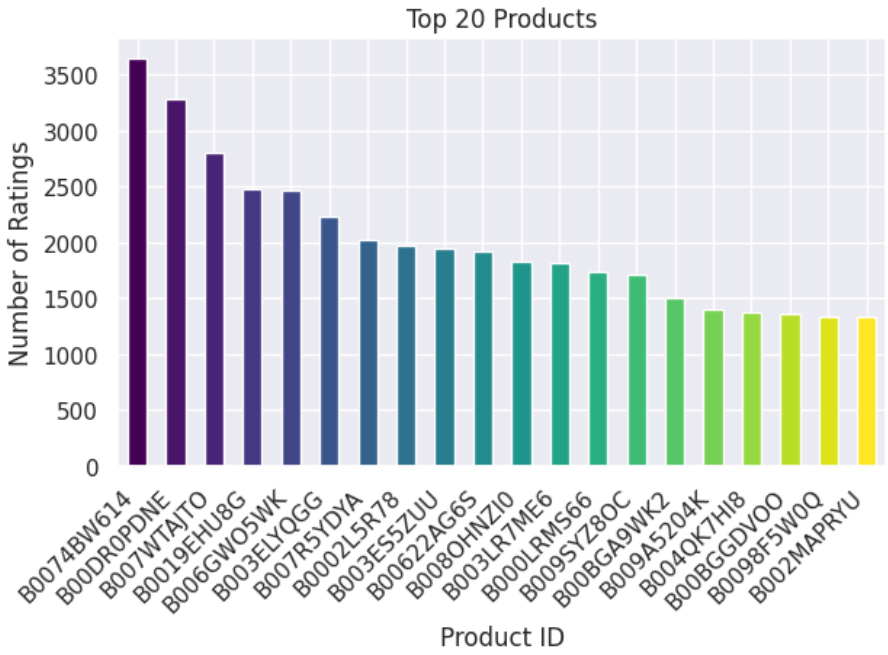


Figure 10: Top 20 most rated (popular) products

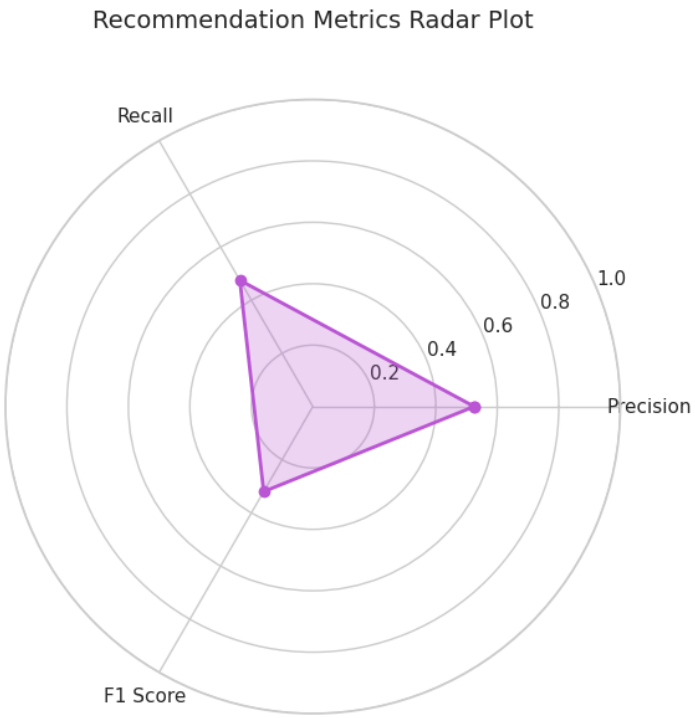


Figure 11: Radar Plot of Model Metrics

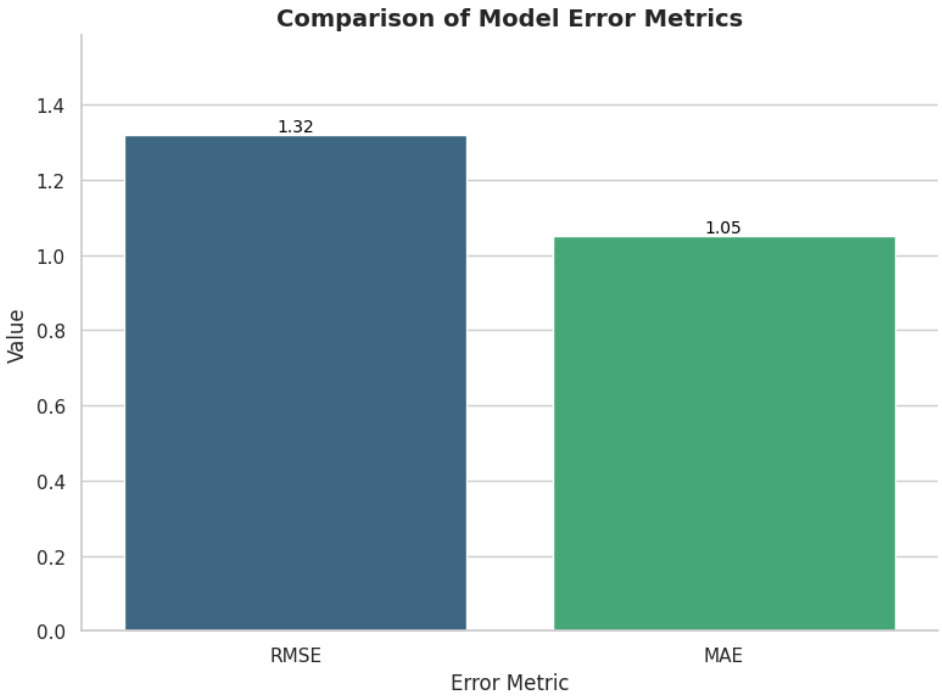


Figure 12: Bar plot of Model Error Metrics

7.3 DISCUSSION

Data Exploration and Preprocessing:

- The original dataset had 7,824,482 reviews.
- As a result of resource constraints, a 20% sample (1,564,896 reviews) was utilized for analysis.
- The dataset had the details on userId, productId, rating, and timestamp. The timestamp column was then dropped.
- The average product rating was roughly 4.01.
- The sampled dataset had 1,225,106 distinct users and 237,584 distinct products.
- There were very few users, but they supplied a high count of ratings and the best rated user had a rating count of 107 items.

Popularity Based Recommendation:

- The popularity based method determined the top 10 most highly rated items.
- Most rated item was B0074BW614 and it had received 3642 ratings.
- Top 10 highly rated items were shown in an output table with formatting.

Item-Item Collaborative Filtering:

- An item-item collaborative filtering algorithm was constructed based on the Surprise library.
- Evaluations were performed on the test set for the performance of the model and obtained Root Mean Squared Error (RMSE) as 1.3084 and Mean Absolute Error (MAE) as 1.0267.

Model-Based Collaborative Filtering (SVD):

- Model-based collaborative filtering was implemented with Singular Value Decomposition (SVD).
- A correlation matrix was calculated from the decomposed user-item rating matrix.

- For a chosen product ('B00007KDVK'), there were 92 products with a correlation value more than 0.75.
- The 10 most highly correlated items with 'B00007KDVK' were shown.

SVD for User-Specific Recommendations and Evaluation:

- An SVD Surprise model was trained on the sampled data.
- A function was defined to recommend the top N items for a specified user using predicted ratings.
- For the sample user "AUE0EEZ0DAGIO", the top 10 recommended items were shown along with their predicted ratings.
- Its performance was tested for a sample user from the test set, producing a precision of 0.375, a recall of 0.4, and an F1 score of 0.32 (for top-10 suggestions with a rating cutoff of 4.0).

7.3.1 Overview of Results Discussion

The results demonstrate the implementation and evaluation of different recommendation system techniques on a real-world e-commerce dataset.

- Popularity-based approach provides a default by recommending commonly popular items. Easy to deploy, but not customized and doesn't support user-specific preferences. The top 10 items found are items of broad appeal by the number of ratings they have.
- The item-item collaborative filtering model was strong in predictive accuracy, as reflected in the RMSE and MAE measures. The numbers show that the model's predictions of user ratings are, on average, approximately 1.31 and 1.03 stars away from real ratings, respectively. This technique uses item similarity through user ratings to generate personalized recommendations.

- The collaborative filtering based on the SVD model enabled the identification of hidden relations among products. Correlation analysis identified products prone to co-rating in a way comparable to a specified product. The method can be capable of detecting non-trivial relationships and proposing heterogeneous recommendations. The outcomes, however, are based on a subsampled set and to a large extent on the correlation threshold.
- The SVD user-personalized recommendation provides personalized suggestions by predicting ratings of items the user has not yet engaged with. The suggested examples for user "AUE0EEZ0DAGIO" are items the model predicts they would rate highly.
- The precision, recall, and F1 score analysis of the user-specific SVD model gives information about the performance of the top-10 recommendations for a particular user. The values we get (precision of 0.375, recall of 0.4, and F1 score of 0.32) are the proportion of relevant items in the top recommendations and the proportion of relevant items recommended. The F1 score provides a balanced measure of precision and recall. Observe that these values are calculated from a single example user and can be different for different users and the entire test set. Also, the threshold used to determine a 'relevant' item (rating >= 4.0) can influence these values.

7.4 DEPLOYMENT STEPS

Infrastructure Installation:

The first challenge was selecting an appropriate computational platform with the ability to handle recommendation model training and prediction. This meant weighing either local or cloud platforms that would be able to meet required processing and memory requirements for collaborative filtering models.

Environment Setup:

The environment setup was done by installing Python and necessary libraries like Surprise, pandas, numpy, and scikit-learn. Necessary dependencies for data processing, model training, evaluation, and visualization were set up in proper manner so that the recommendation pipeline could be run smoothly.

Model Deployment:

The trained collaborative filtering models (like SVD and KNN) were brought into deployment-readiness by storing the model parameters and loading them into the inference-active environment. Efficient loading and applying mechanisms were implemented to load these models in order to provide product recommendations as needed.

Testing and Validation:

Extensive testing was conducted using user-product input samples to verify the model provided accurate and consistent recommendations. Metrics such as RMSE and MAE were utilized to test the prediction quality, and performance validation verified the system performed within expected latency and resource utilization.

Monitoring and Logging:

Basic logging mechanisms are established to capture inputs, model output, and runtime errors. Performance indicators like the time taken for prediction and recommendation accuracy were tracked to catch any issues and provide seamless performance in the long run.

Scalability and Maintenance

The system was made scalable so that future upgrades like model updates, new features, or processing larger datasets could be added. Periodic maintenance was scheduled to maintain performance up to date, refresh model parameters, and support increasing user or item data in an optimal manner.

8. CONCLUSION

The Collaborative Filtering Product Recommendation System process involved various important steps like data preprocessing, model building, performance evaluation, and optimization via metrics. A systematic process was adopted where collaborative filtering techniques like SVD and KNN were utilized effectively to offer personalized recommendations in a way that the system offers meaningful and relevant recommendations to the users.

Maintaining the model up to date and accurate requires constant monitoring and evaluation. By tracking important metrics such as Precision, Recall, F1 Score, RMSE, and MAE regularly, we are able to identify performance degradation areas and implement the necessary corrections. This ensures the model continues to make correct predictions based on user behavior and preference over time.

Machine learning for recommendation systems is a large step towards the personalization of user experience. With the incorporation of advanced data-driven techniques, there is better decision-making, higher user satisfaction, and heightened user engagement in the form of recommendation of the most suitable products based on their past interactions.

Through continued assessment and prospects for future development, the system can evolve depending on changing user needs and trends in information. The use of intelligent recommendation models underscores the increasing role of machine learning in influencing personalized content streaming and the efficiency of digital platforms as a whole.

9. FUTURE SCOPE

Model Refinement:

There are possibilities for future research to improve the scalability and performance of the recommendation system. This may include algorithm optimization such as matrix factorization or hybrid models, hyperparameter tuning, and leveraging user behavior data or product metadata to generate more accurate and personalized recommendations.

Dataset Augmentation:

Increasing the dataset with more and varied instances of user-product interactions is a valuable way to improve model accuracy. This may include adding more categories, trends based on time, or cross-platform user preference to more closely mimic real e-commerce experiences.

Real-Time Recommendation System:

While the model is now trained offline, there are possible improvements in the form of a real-time recommendation system that reacts in real-time to user behavior. That would imply model latency optimization, releasing lighter versions of the recommendation algorithm, and using streaming data pipelines for real-time feedback.

Integration with E-Commerce Platforms:

Future applications of the recommendation system can be made by integrating into complete e-commerce websites or dashboards. Developing APIs or plug-ins for websites such as Amazon, Shopify, or specific e-commerce websites would enable integration and analysis of real-time user interaction to be easy.

Collaborative Development:

Collaboration with data scientists, product managers, and UX designers can help develop a more robust and user-focused recommendation platform. Interdisciplinary collaboration can lead to innovation and enable the system to be aligned with business goals and user needs

10. REFERENCES

1. A survey of collaborative filtering-based recommender systems: From traditional methods to hybrid methods based on social networks. Chen et al., 2018 Chen R., Hua Q., Chang Y.-S., Wang B., Zhang L., Kong X.

2. A new method to find neighbor users that improves the performance of collaborative filtering. Expert Systems with Applications, 83 (2021), pp. 30-39. Koohi and Kiani, 2021

3. Collaborative filtering-based recommendation system for big data. International Journal of Computational Science and Engineering, 21 (2) (2020), pp. 219-225. Shen et al., 2020 . Shen J., Zhou T., Chen L

4. Collaborative Filtering for Recommender Systems. Ruisheng Zhang; Qi-dong Liu; Chun-Gui; Jia-Xuan Wei; Huiyi-Ma.

5. A Content Based and Collaborative Filtering Recommender System. Vignesh Thannimalai; Li Zhang. https://ieeexplore.ieee.org/document/9737238

6. SVD++: A Scalable Collaborative Filtering Approach Koren, Y., Bell, R., & Volinsky, C. (2009). IEEE Computer Society Research on matrix factorization models like SVD, widely used in recommendation systems.

7. Seaborn Documentation. (n.d.). Retrieved from https://seaborn.pydata.org

8. Scikit-learn Developers, "Scikit-learn: Machine Learning in Python," Scikit-learn Documentation, 2024. [Online]. Available: • https://scikit-learn.org/stable/documentation.html

9. Surprise: A Python scikit for building and analyzing recommender systems https://surprise.readthedocs.io/en/stable/ . Framework used for collaborative filtering, hyperparameter tuning, and accuracy metrics like RMSE/MAE.

10. C. C. Aggarwal, Recommender Systems: The Textbook. Springer, 2016.

11. "Recommender systems" by Jannach, D., Manouselis, N., Shi, Y., & Schelter, S. (2010). This book provides a broad introduction to recommender systems, covering various techniques beyond collaborative filtering, such as content-based and knowledge-based approaches, and discusses evaluation and user interfaces.

12. "Item-based collaborative filtering recommendation algorithms" by Sarwar, B. M., Karypis, G., Konstan, J. A., & Riedl, J. (2001). This influential paper details the item-based collaborative filtering approach, which is a key technique discussed in your project.

13. "Matrix factorization techniques for dyadic data analysis" by Aggarwal, C. C. (2009). While your project mentions Aggarwal's textbook, this paper provides a more focused look at matrix factorization, including SVD, in the context of recommender systems and other dyadic data.

14. "A survey of nearest neighbors based collaborative filtering" by Bellogin, A., Cantador, I., & Diez-Fernandez, R. M. (2011). Given your use of KNN, this survey offers a deeper understanding of different neighbor-based collaborative filtering methods and their variations.

15. "Factor in the neighbors: Scalable and accurate collaborative filtering" by Koren, Y. (2008). This paper discusses the integration of neighborhood models with matrix factorization techniques, an area that could be relevant for future hybrid approaches mentioned in your document.

16. "Evaluating recommender systems" by Gunawardana, A., & Shani, G. (2009). This paper provides a comprehensive overview of different evaluation metrics and methodologies used in the field of recommender systems, expanding on the RMSE and MAE metrics you used.

17. "Personalizing search via collaborative filtering" by Teevan, J., Dumais, S. T., & Horvitz, E. (2005). This work explores how collaborative filtering principles can be applied to personalize search results, demonstrating the broader applicability of these techniques beyond product recommendations.

18. "Content-boosted collaborative filtering for improved recommendations" by Melville, P., Mooney, R. J., & Nagarajan, R. (2002). Since your project briefly mentions content-based filtering and hybrid models, this paper offers a foundational approach to combining content and collaborative information.

19. "Deep learning for recommender systems: A survey and new perspectives" by Zhang, S., Yao, L., Sun, A., & Tay, Y. H. (2019). Your project mentions exploring deep learning-based recommendation models in the future. This survey provides an overview of how deep learning techniques are being used in this domain.

20. "Recommending what video to watch next: A multitask ranking system" by Zhao, L., Charlin, L., Park, D. H., & Chen, X. (2019). This paper presents a real-world example of a recommendation system used by YouTube, showcasing the complexities and considerations in building large-scale recommendation engines, which can provide context for your project's findings.