JUMPSTART: TACKLING COLD START ISSUE IN RECOMMENDATIONS

Report submitted in partial fulfilment of requirements for the B.Tech degree in Computer Science and Engineering

By

NAME OF THE STUDENT	ROLL NUMBER
FARHAN ALAM	2019UCO1510
VARUN JAIN	2019UCO1520
RUDRAKSH TANEJA	2019UCO1521
MANAS BOHAT	2019UCO1536

Under The Supervision

Of

Dr. Geetanjali

Department of Computer Science and Engineering

Netaji Subhas University of Technology (NSUT)

New Delhi, India – 110078

MAY, 2023

CANDIDATES DECLARATION

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

We Farhan Alam (2019UCO1510), Varun Jain (2019UCO1520), Rudraksh Taneja (2019UCO1521), Manas Bohat (2019UCO1536) of B.Tech Department of Computer science & engineering, hereby declare that the project titled **“JUMPSTART: Tackling Cold Start Issue in Recommendations”** which is submitted by me to the Department of Computer science & engineering, Netaji subhas university of technology (NSUT) Dwarka, New Delhi in partial fulfilment of the requirement for the award of the degree of bachelor of technology is original and not copied from the source without proper citation. The manuscript has been subjected to plagiarism check by Turnitin software. This work has not previously formed the basis for the award of any degree.

Place: New Delhi, India

Date:

Farhan Alam	Varun Jain	Rudraksh Taneja	Manas Bohat
2019UCO1510	2019UCO1520	2019UCO1521	2019UCO1536

CERTIFICATE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

This is to certify that the work embodied in project thesis titled, "**JUMPSTART: Tackling Cold Start Issue in Recommendations**" by *Farhan Alam (2019UCO1510), Varun Jain (2019UCO1520), Rudraksh Taneja (2019UCO1521), Manas Bohat (2019UCO1536)* is the bonafide work of the group submitted to **Netaji Subhas University Of Technology** for consideration in 8th Semester B.Tech Project Evaluation.

The original research work was carried out by the team under my guidance and supervision in the academic year 2022-2023. This work has not been submitted for any other diploma or degree of any university. On the basis of declaration made by the group, we recommend the project report for evaluation.

Dr Geetanjali

(Associate Professor)

Department of Computer Science and Engineering

Netaji Subhas university of technology

ACKNOWLEDGEMENT

We would like to express my gratitude and appreciation to all those who make it possible to complete this project. Special thanks to our project supervisor Dr Geetanjali whose help, stimulating suggestions and encouragement helped us in writing this report. We also sincerely thank our colleagues for the time spent proofreading and correcting our mistakes.

We would also like to acknowledge with much appreciation the crucial role of the staff in computer science and engineering, who gave us permission to use the lab and the systems and giving permission to use all necessary things related to the project.

Farhan Alam	Varun Jain	Rudraksh Taneja	Manas Bohat
2019UCO1510	2019UCO1520	2019UCO1521	2019UCO1536

ABSTRACT

The project aims to investigate, analyse, and develop a method that tackles the cold start issue in recommender systems. The cold start issue refers to the difficulty of making recommendations for new users or items with little to no historical data. The cold start issue can be particularly challenging for platforms that have a large and constantly changing inventory of items. The project tries to find a suitable alternative for already existing methods that either work on hybrid methods with overhead of switching between various algorithms, or are computationally expensive with separate training to handle cold start users and items. Performing a study over the nature of item features and ratings, we identified the existence of human bias towards certain features in the items in ratings. Another thing we noted was the better efficiency in case of recommendations which are based on user item interactions (collaborative filtering) rather than item features (content-based filtering). We used the above two facts to use feature weighting to modify our feature-based algorithm to approximate and incorporate the results of the user interaction-based algorithm, in the same model. We made a machine learning model using sophisticated methods and rigorous training to train the feature weights for each of the features in items. We utilized the concept of early stopping and performed hyperparameter tuning to optimize our training time and the results. These learned feature weights are finally used to provide the recommended list of items. We created a web application where these recommended movies are displayed, and the user feedback is incorporated in the model whenever the recommended feed is refreshed.

Keywords: Recommendations systems, Cold Start Problem, Machine Learning, Feature Weighting, Feature based training

INDEX

CANDIDATES DECLARATION i

CERTIFICATE ii

ACKNOWLEDGEMENT iii

ABSTRACT iv

INDEX v

LIST OF FIGURES vii

LIST OF TABLES viii

LIST OF ALGORITHMS ix

CHAPTER 1

INTRODUCTION 1-5

1.1 Motivation 1

1.2 Key Challenges 3

1.3 Problems addressed in thesis 4

1.4 Approach to the problem and organisation of thesis 5

CHAPTER 2

LITERATURE REVIEW 6-16

2.1 Previous Work 6

2.2 Summary of Research Papers 11

2.2 Proposed solution 13

CHAPTER 3

MODEL FORMULATION AND DEVELOPMENT 17-48

3.1 Dataset 17

3.2 Model 30

3.3 Application 44

3.4 Technologies used 47

CHAPTER 4

RESULTS 49-58

4.1 Selecting Metrics 49

4.2 Hyperparameter Tuning 52

4.3 Comparison of Model Results 55

4.4 The User Interface 57

CHAPTER 5

CONCLUSION AND FUTURE WORK 59-60

5.1 Conclusion 59

5.2 Future Work 60

REFERENCES 61-63

PLAGIARISM REPORT 64-70

LIST OF FIGURES

CHAPTER 1

2.1 System Model of proposed Framework 16

CHAPTER 3

3.1 Movie file in dataset 18

3.2 Wikipedia Format 19

3.3 Screenshot of dataset before scraping 20

3.4 Screenshot of dataset after scraping 20

3.5 Screenshot of generated ICM, Dense format 28

3.6 Screenshot of generated ICM, Sparse format 28

3.7 Screenshot of generated URM, Dense format 29

3.8 Screenshot of generated URM, Sparse format 29

CHAPTER 4

4.1 Screenshot of the login page 57

4.2 Screenshot of the interface as user logins 57

4.3 Screenshot of user giving feedback on the movies 58

4.4 Screenshot of the interface showing movies recommended to the user. 58

LIST OF TABLES

CHAPTER 2

TABLE 2.1 Literature Summary 11

CHAPTER 4

TABLE 4.1 Comparison of RMSE values 55

TABLE 4.2 Comparison of precision, recall and F-measure 56

TABLE 4.3 Comparison of NDCG values 56

LIST OF ALGORITHMS

CHAPTER 3

3.1. Algorithm for web scraping from Wikipedia 20

3.2 Algorithm for k fold data splitting 24

3.3 Algorithm for warm k fold data splitting 24

3.4 Algorithm for creating for creating Incremental Sparse Matrix 26

3.5 Algorithm for building Item-Content Matrix 26

3.6 Algorithm for building User-Rating Matrix 29

3.7 Algorithm for handling Popularity Bias 33

3.8 Algorithm for creating Collaborative based Item Similarity Matrix 34

3.9 Algorithm for creating Content based Item Similarity Matrix 35

3.10 Algorithm for identifying Similar items 36

3.11 Algorithm for calculating loss value 39

3.12 Algorithm for updating the feature weights through gradient descent 39

3.13 Algorithm for Early Stopping 41

3.14 Algorithm for retrieving top k items for a particular user 43

CHAPTER-1: INTRODUCTION

1.1) MOTIVATION

The internet has resulted in a massive growth in content of all kinds, and users are craving more and more personalized content, be it video, audio or any other type. Recommendation systems have hence become a crucial part of many online platforms, from social media to e-commerce to streaming services. These systems are designed to help users discover content that they are likely to enjoy based on their previous interactions or profile information.

The effectiveness of a recommendation system depends heavily on the quality of its predictions, which in turn relies on the availability of data about user behavior and content features. However, a significant challenge that many recommendation systems face is the **cold start problem**. This problem arises when a system must make recommendations for new users or items that have little to no historical data. In the case of new users, the system has no information about their preferences, making it difficult to make personalized recommendations. Similarly, for new items, there is no data on how users have interacted with them in the past, making it challenging to predict how a user might respond to them.

The cold start issue can be particularly challenging for platforms that have a large and constantly changing inventory of items. New films, TV shows, reels, and articles etc are constantly being added to streaming services like Netflix, and video content sites like YouTube and Instagram, and in such situations, the lack of interactions in the newly added videos prevents recommendation models from making predictions.

As new content is added to the platforms, the cold start problem becomes more acute because there may be a higher proportion of new items with no interactions or features available. Thus, it is essential that recommendation engines strive for fresh and 'state-of-the-art' ways to suggest the most recent additions to its customers.

Addressing the cold start issue is important because it can impact the user experience and the effectiveness of the recommendation system. If users are not presented with relevant and interesting content, they may lose interest in the platform and not return. Additionally, if the recommender is not able to accurately predict user preferences, it may not be able to optimize engagement and revenue for the platform. Consider a new video posted on YouTube. Since the video has few interactions, it might never be able to make it to the recommended feed of users even though it might be something they like. In turn, some other video, which might not be the best pick based on the user's interests, might be recommended to them just because some field like genre was matching their general interests.

There are two basic approaches to develop a recommender system: Collaborative Filtering (CF) and Content-Based Filtering (CBF) models. The core idea behind the CF method is that if two people have the same view on one subject, they are more likely to do so on a different subject than two people who were picked at random. Hence, Collaborative filtering typically relies on user-item interaction data to make recommendations based on user similarity. While on the other hand, a content-based model uses data about item features, such as genre or cast, to make recommendations based on item similarity. However, when new items are added to the system, they may contain none or very few interactions, and hence there may not be enough data available to make accurate recommendations. The CF model may not be able to make accurate predictions because to recommend items it relies on their interactions, and in case of CBF, if a new item has not yet been analyzed for its features, the model may not be able to include it in its recommendations. This leads to the cold start problem.

Hence, we aim to find a solution to tackle the cold start problem in recommender systems in an efficient way, which allows new users to find the best content personalized especially for them. At the same time, this will enable businesses to profit due to high consumer traffic on their platforms.

1.2) KEY CHALLENGES

There are several key challenges in developing recommendation systems due to the cold start issue, and handling the issue.

One of the most significant challenges in handling the cold start problem is data sparsity. In most cases, recommendation systems rely on the availability of user-item interactions data to train their models and make recommendations. However, when a new user or item is added to the system, there is little or no data available to make personalized recommendations. For instance, a newly launched e-commerce platform might have a small inventory of items and few interactions between users and items. Hence, this would hinder a great user experience because of less personalized recommendations. This would lead to a churn rate. As a result, the system cannot rely on user-item interactions to make recommendations, making it difficult to provide a personalized experience to users.

The content inventory of platforms is constantly changing, resulting in a higher proportion of new items with no interactions or features available. Hence, the recommendation systems should be expected to accept the huge amount of content that is being generated daily, and include them in the recommendation process. Otherwise, a situation where only popular items are taken into consideration and may arise, which is called as the ‘Popularity Bias’.

Different types of content, such as videos, music, and books, may have different characteristics and require different approaches to address the cold start problem. Recommendation systems are expected to handle multimodal data, where each of the type of data may have its own different characteristics and processes to work with.

Interpretability is also a significant challenge in recommendation systems. As recommendation systems become more complex, it becomes difficult to understand how they make recommendations. This can lead to a lack of transparency and accountability, making it difficult for users to trust the recommendations. For instance, a music streaming platform might recommend a song to a user, but it may not be apparent why that song was actually recommended to that particular user.

1.3) PROBLEM ADDRESSED IN THESIS

The project aims to address the cold start problem which is a significant challenge in recommender systems. It refers to the difficulty of making recommendations for new users or items with little to no historical data. For new users, the system has no information about their preferences, making it difficult to make personalized recommendations. Similarly, for new items, there is no data on how users have interacted with them in the past, making it hard to predict how a user might respond to them.

The cold start issue is a huge challenge for platforms that have a large and constantly changing inventory of items. As new content is added to the platforms, the cold start problem becomes more acute because there may be a higher proportion of new items with no interactions or features available.

We addressed the problem by developing a hybrid approach that incorporates the collaborative similarities in the content-based model. We observed the existence of human bias towards certain features in items, and used this fact to weigh item features to approximate them to a collaborative model. We apply machine learning to learn these feature weights. We name our approach as ‘Jumpstart’.

1.4) APPROACH AND ORGANIZATION OF THESIS

The main focus of our project was to build the recommender system model to tackle the cold start issue. Moreover, we aimed to develop a web application through which the model behaviour will be represented. We began with the model and then implemented the application. We worked in several stages starting from planning, design, implementation (coding) and evaluation.

We started with research and literature review, and studied various research papers and journals to examine the existing methods for addressing the new item cold start problem in recommendations, and identified the challenges and limitations of these methods. Then we moved on to data collection and pre-processing. We collected, acquired and expanded a suitable dataset for the project, including entities such as user-item interactions, item metadata, and other relevant information. We pre-processed the data to make it ready for analysis and modeling.

We then proceeded to develop a suitable recommendation model to address the cold start problem. This was the main step of the whole project, where we designed the main approach to recommend new users suitable items. We further implemented and tested the model on the pre-processed data. There were many iterations within these steps leading to the refinement of the algorithm. After that, we evaluated the performance of the model using suitable evaluation metrics such as accuracy, precision, recall, and F1 score on the cold start items. This also involved hyperparameter tuning; fine-tuning the model by adjusting the hyperparameters to improve its performance. This step involved training the model with different combinations of hyperparameters to find the best set of parameters. Then analysis of results and comparison with the existing methods was done. We conclude the report by summarizing the findings and contributions of the project and identify areas for future work and improvement.

CHAPTER-2: LITERATURE SURVEY

2.1) PREVIOUS WORKS

Currently, there are several approaches to tackling this cold start issue in recommendation systems. A **Collaborative Filtering (CF)** model cannot provide suggestions for a new item due to a lack of interactions, and in the event of few interactions, the quality will be quite low. This can also lead to a problem known as ‘Popularity Bias’ [13], in which products with few interactions are suggested poorly, receive less interactions than others, and hence lose awareness and end up in this vicious cycle. Even though the items had been in the catalog for months, they might receive very few interactions as a result of this.

The **Content Based Filtering (CBF)** approach, which doesn't rely on interactions but rather item properties, is, in principle, less vulnerable to the cold start issue. This is only true, though, when the features of the newly added item are already available, such as the title, director, and cast of a recommended movie. Some attributes, however, cannot be known at the beginning of the cold start because they are obtained from user-generated content such as tags and reviews. CBF is believed to be less accurate than CF since it excludes helpful collaboration information and relies on expensive to gather and frequently inaccurate human-generated textual metadata.

To quickly gather some preference information is one of the choices to reduce the recommendation domain roughly and achieve more interactions in less time. Depending on how much information is needed, there are many approaches to do it. This can be done either directly (by asking questions or surveys to users) or implicitly (by keeping an eye on the user's behavior). In both scenarios, the cold start issue would suggest that before the system can start making intelligent recommendations, the user must invest some time utilizing it in its "dumb" form and building their user profile, which can make the process time consuming and users may enter inaccurate information to complete the process quickly.

The above process can be automated by applying active learning (a subset of machine learning). The basic objective of active learning is to direct the user through this process of preference elicitation so that he only rates the things that, from the viewpoint of the recommender, will be the most instructive. This is accomplished by analyzing the information supplied and determining the data values (e.g., interactions, ratings). This method however runs the danger of overloading the algorithm with irrelevant samples. Recent years have seen certain recommendation techniques that have significantly improved the quality of recommendations and considered as ‘state-of-the-art’ methods. Neural Collaborative Filtering is one which is a deep learning-based recommendation technique that combines traditional collaborative filtering with neural networks to learn a more accurate representation of user-item interactions, outperforming the traditional CF. However, it bears the same limitations as a CF model in case of cold start items.

To specifically tackle the cold start issue in recommender systems, Tahmasebi et al. [7] suggest a hybrid recommendation strategy, as a pure collaborative filtering method is not resistant to the cold start problem. By combining user demographic information with rating information, the approach broadens users' rating profiles, which in turn improves the user base in their immediate vicinity. The suggested mechanism produces a denser user-item rating matrix, which boosts recommender system performance, particularly when making predictions about yet-to-be-discovered items. In comparison to existing recommendation approaches, the extended rating profiles provide improved accuracy and rate coverage metrics by using similarity values between users to forecast unseen things.

One very effective solution was produced by J. Zhang et al. [5], that divides users into groups based on their profile traits and designates a virtual opinion leader to represent the cluster in order to overcome these problems. After significantly reducing the dimensions of the original user-item matrix, the recommendation results are generated using a Weighted Slope One-VU approach applied to the virtual opinion leader-item matrix. The suggested method achieves equivalent recommendation performance with noticeably less time complexity than conventional clustering-based CF recommendation schemes.

Further, in order to make **deep learning-based recommendation systems** (such as collaborative filtering with several layers) resistant to cold objects, a recent idea called DropoutNet [3] proposes randomly removing ratings of items and users during training.

The crucial distinction in this case is that, in contrast to typical dropouts in neural network training, they drop the features instead of the nodes. They could do this to lessen the neural network's reliance on particular ratings and increase its generalizability to objects or individuals with lower ratings. The benefit of their method is that it can be applied to any recommender system powered by neural networks and may be used from scratch.

Reinforcement Learning is also being employed for cold start which uses trial-and-error to learn optimal recommendation policies. It has been shown to be effective in personalized recommendation and has been applied to a variety of recommendation tasks, including sequential recommendation and multi-objective recommendation. However, the major drawback with these deep learning methods has been that they either require a huge amount of data or they need significant exploration, which is not suitable for a cold start scenario. Considering all the above limitations, the most promising strategy specific to effectively tackling the cold start problem when we aim to design a computationally inexpensive system proves to rely on hybrid recommenders, which combine two categories or models to counteract the drawbacks of one.

There are many ways different models can be hybridized [4]. Each one is specific to some aspects and has its own limitations. Weighted hybridization uses a linear formula to combine the results from each of the used recommendation models, integrating their scores to provide a forecast or list of suggestions. The recommendations can be more accurate if both user preferences and item attributes are considered. However, the technique may not be suitable for highly diverse recommendation domains, where different recommendation sources may not have enough overlap or agreement.

Switching hybrid method switches between uses multiple recommendation techniques based on the context of the recommendation. It combines the strengths of different recommendation methods and adapts to changing user preferences or item availability. This technique requires careful consideration of the context and appropriate switching criteria to ensure that the best recommendation method is used at any given time.

In case of switching hybridization when we use CBF for cold items and CF for warm items, it may sometimes suffer with a poor suggestion quality displayed by content-based recommenders in situations where it is challenging to provide an in-depth description of the item attributes. If the quality is too low for new users, a frequent tactic is to provide them with non-personalized suggestions. This implies that individuals could only receive recommendations for the products that are the most well-liked worldwide, locally, or in their native tongue, also called as ‘Popularity Based Recommenders’, but they are the last alternative that should be considered.

Cascade hybrid technique combines the benefits of both content-based and collaborative filtering techniques by dividing the recommendation process into two stages. In the first stage, CBF is applied to create an initial list of recommended items (irrespective of context unlike switching method). In the second stage, CF is used to refine the list of recommendations. The assumption behind this technique is that CBF will remove items that are unlikely to be of interest to the user, leaving only items that are potentially relevant. Then, CF is applied to this reduced set of items to generate the final list of recommendations. It may require a large number of computational resources and can be slow to execute.

A recent approach is to create a hybrid CBF recommender in which features are weighted in accordance with the user's assessment of relevance. These features can be of user or of items. Several characteristics (such as the actors, director, nation, and title) will be given varying weights in order to help the user choose a movie they would enjoy. The features of one model can be modified or enhanced using the output of another model for better Recommendations.

Deldjoo et al. [8] present a movie recommender system that addresses the cold start problem by utilizing audio and visual descriptors of video content, referred to as the "movie genome." The system incorporates canonical correlation analysis (CCA) for data fusion and trains a collaborative filtering model on warm items. This learned model is then applied to recommend cold items based on the movie genome. Experimental results demonstrate the effectiveness of the proposed approach in addressing the new item cold start problem in movie recommendation systems.

Now we describe some state-of-the-art recommendation systems employed by giants like YouTube and LinkedIn.

Paul Covington et al. [20] shed light on the overall architecture of the YouTube recommender system. The system utilizes two neural networks, one of them is used for candidate generation and the other one for ranking. The candidate generation component applies collaborative filtering techniques to offer personalized recommendations based on a user's activity history on YouTube. The ranking network further refines the recommendations and presents a few top choices to the user, guided by a specific task or objective function.

The model architecture involves generating embeddings of high dimensions for each item in a predefined vocabulary. These embeddings are then input into a feedforward neural network. The embeddings are trained in a multimodal manner, such as through summation or component-wise max, alongside model parameters using gradient descent and backpropagation.

In a study conducted by Sahin Cem Geyik et al. [11], the authors discuss LinkedIn's information retrieval, search, and recommendation systems, which specifically focus on talent search. LinkedIn aims to recommend candidates who not only match the recruiter's query but also demonstrate interest in the job opportunity. The system architecture of LinkedIn's talent search involves collecting the recruiter's query and ranking it against the candidates' qualifications and search parameters. The ranking models are periodically trained using recruiter usage logs in an offline modeling pipeline. These logs are generated from interactions between recruiters and candidates, including responses to recruiter messages displayed in search results. This approach enhances the relevance of search results and increases the likelihood of successful recruiter-candidate interactions.

2.2) SUMMARY OF RESEARCH PAPERS

Here we provide a description of few of the above-described mentioned papers, summarized with their description and limitations.

TABLE 2.1: Summary of Research Papers

Name	**Description**	**Limitations**
Wang et al. (2018) [15]	RippleNet uses a knowledge graph to represent users, items, and their relationships and employs a graph convolutional network (GCN) to propagate user preferences throughout the graph and generate recommendations. Useful when there is limited user data or where there are complex relationships between users and items.	Harder to understand than feature weighting models, and building and maintaining a knowledge network can be resource-intensive
Deldjoo et al. (2019) [8]	The paper aims to develop a hybridized technique that can be with the CBF model acting as the core model without any switching to CF. Instead, the authors suggested an idea of feature weighting that can adjust the features of CBF based on the CF model. The paper focused on extracting the audio and video features and performing data fusion.	The idea of feature weighting but specific to the audio-visual domain, where a lot of fixes and changes would be required to adapt to a new domain for the same idea developed here.
Zhang et al. (2020) [5]	The objective of the paper is to present a recommendation algorithm with high efficiency to address the scalability issue and practical usage feedback in movie recommendation systems. The algorithm being proposed uses the profile attributes of	The time complexity problem is mitigated by the algorithm proposed, it may still face scalability issues when applied to very large datasets. The paper does not

	users to divide them into multiple clusters and create a virtual opinion leader for each of these clusters. By reducing the dimensionality of the initial user-item matrix, the issue of time complexity is improved.	provide an analysis of the algorithm's performance on very large datasets, which limits its practical applications.
Rahman et al. (2022) [1]	The objective of the paper is to suggest an inference system based on rules that can enhance the precision of the recommendation process. The system under consideration employs a predetermined set of rules, which are derived from the user's preferences, to enhance the suggestions produced by both the CBF and CF method. These rules are designed to capture the user's implicit preferences and provide more personalized recommendations, especially in situations where there is limited or no data available on the user's past behavior.	The paper does not provide any empirical evidence of the system's effectiveness in addressing the cold start problem. The assessment of the proposed system relies entirely on the subjective judgment of a panel of specialists, which may not reflect the genuine behavior of users.
Weimann et al. (2022) [2]	The authors propose a model which is designed to resemble commonly used sequential recommenders, which typically include an embedding layer. However, instead of using a traditional embedding layer, this model uses a dynamically sized cache that does not require side information to represent new items.	To learn the recommendation task successfully, a sizable amount of datasets are needed.Other approaches might work well if side information is available.

2.3) PROPOSED SOLUTION

After doing thorough research of recommender systems, we went ahead with a hybrid approach of content based filtering and collaborative filtering. Our aim was to get the benefits of both.

i. The accuracy and efficiency of collaborative filtering
ii. The ability to give good results during cold start from content-based filtering

An important point to note is that we are not following the cascade hybrid approach, but a hybrid approach that is much different and gives much better results. To reveal the dichotomy between users and features, we, on the other hand, develop a feature-weighted user profile.

Our main goal is to combine collaborative filtering's effectiveness with content-based filtering's cold-start resistance. We will use Content Based technique as our base model, and we will adjust the input needed by CBF to produce recommendations based on the results of the CF Model. In this manner, the CF approach will still have an impact on the final recommendations while still adhering to the CBF algorithm.

We employ a technique called feature weighting to improve the accuracy and efficiency of our solution, because some features are more informative or significant than others in predicting the outcome variable. Every item is represented by a feature vector or an attribute profile in content-based recommendations. The features reflect specific characteristics of the item, such as colour, price, etc., with numeric or nominal values. The similarity of two objects can be calculated using a number of different distance measurements between the feature vectors. The ranked list of suggested items is then created using the similarity values. If Euclidean or cosine similarity is taken into account, all traits are implied to be of equal importance. However, when comparing two items, human judgement frequently gives varying weights to certain features. For example, while buying a mobile phone, the battery backup may be more important than the phone’s colour attribute.

Feature weighting methods can handle the cold start problem better than collaborative filtering methods, as they can recommend items based on the user's profile attributes, even if the user has not interacted with any items yet. Collaborative filtering methods require user-item interactions to make recommendations, so they cannot make recommendations for new users.

Another advantage of feature weighting methods is their interpretability. Feature weighting methods assign a weight to each feature or attribute of an item, which can be interpreted as an indicator of the importance of that feature for the recommendation. This makes it easier for users to understand why a particular item was recommended to them and can help build trust in the recommendation system.

Feature weighting methods are also computationally efficient. These methods typically involve simple mathematical operations, such as dot products and summations, and do not require complex neural network architectures or training procedures. This makes them faster to implement and can be more suitable for small-scale recommendation systems or resource-constrained environments.

When it comes to sparse data too, feature weighting methods work very well. This is because they explicitly consider the sparsity of the data in their calculations, and adjust the weights of the features accordingly. Neural network methods, on the other hand, can struggle with sparse data because they are designed to work with dense, high-dimensional feature vectors. While there are techniques to handle sparsity in neural networks, such as using embeddings or applying dropout regularization, feature weighting methods are typically more effective for sparse data.

Now having discussed why our approach is favourable, let us describe our system. Our system comprises multiple components majorly including data preprocessing, collaborative filtering, feature weighting and content based filtering, which are integrated together to provide accurate recommendations efficiently.

There are two sources of data in our system which are the item profile and user profile, and the eventual output of our system is the list of recommended items for each user. The User profile contains the previous interactions of the user with various items, and the Item profile is a matrix of items vs their features and each cell contains the relevance of each feature for the item. Because of the amount of data being large, the model does the processing incrementally wherever required. We use Machine Learning to train our model on these data sources and optimize the hyperparameters by validating the model.

We describe here the different components of our system:

- **Data Preprocessing**: This is the first component of our model. The user profile and the item profiles are fed into this unit to be processed and they are converted into suitable matrix formats for further processing. Data Preprocessing involves several steps described as follows:
    - **Adding Features:** through Web scraping. This is a necessary step as every domain may not contain the required or enough features. There can be different methods to collect features but we include Web Scraping because it is the most convenient and widely used method.
    - **Data Cleaning:** To improve the accuracy of our system, not all user-item ratings are significant, setting up correct thresholds and cleaning data is the part of this step
    - **Converting dataset into incremental sparse matrix:** This is required because the data is huge and matrix operations cannot be done directly on these huge datasets
    - **Building the required matrices:** We build two matrices from our dataset, one is the **URM (User Rating Matrix)** and **ICM (Item Content Matrix).**
    - **Data Splitting:** The URM Matrix built is split into training, validation and test sets using k-fold cross validation method.
- **Collaborative Filtering:** The collaborative model learns over the training URM matrix and generates top K similar items for each item. This item-item similarity matrix (IIM) is then fed for feature weighting as one of the inputs. CF also involves hyperparameters that penalize the popular items so that they don’t dominate less popular items in the similarity metric.
- **Feature Weighting:** It takes as input the training URM set, the ICM set, and the Item similarity matrix which is obtained from the CF component. The item features in the ICM set are modified and adjusted as per the IIM. This modified ICM matrix along with URM is then passed to the core recommendation model of our system, i.e. Content Based Model.

- **Content Based Model:** The CBF takes the enhanced ICM and URM as shown in Figure 1, and finds the recommended items for each user based on the relevance of these features. These recommendations are provided to the users, and users provide feedback in the form of ratings to those recommendations. These feedbacks update the URM (and also add onto the user profile dataset) and new recommendations are hence generated as per the URM. The ICM is also updated when new movies or items are added or uploaded on the platform.

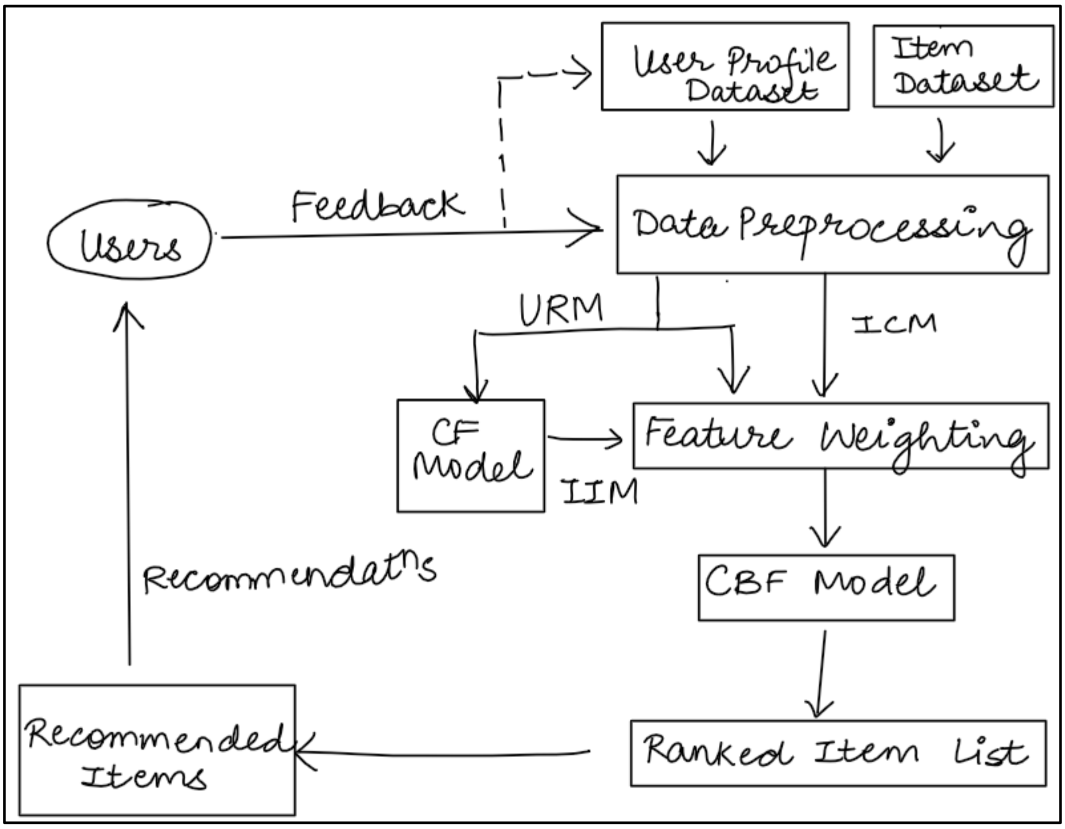


Figure 2.1: The System Model

CHAPTER 3: MODEL FORMULATION AND DEVELOPMENT

3.1) DATASET

Movie recommendation systems have gained significant popularity in recent years, with several online platforms such as Netflix, Amazon Prime, and Hulu using them to provide personalized recommendations to their users. In order to build and test such systems, researchers and data scientists require access to large and diverse datasets that contain information about movies, users, and their preferences.

3.1.1) SELECTING THE DATASET

Our primary requirements for building a movie recommendation system is to have access to movie features, such as the title, genre, director, cast, and release date. In addition, it is also required to have the information of user ratings of movies, as this information is used to train and evaluate recommendation algorithms. There are several popular datasets available that meet these requirements.

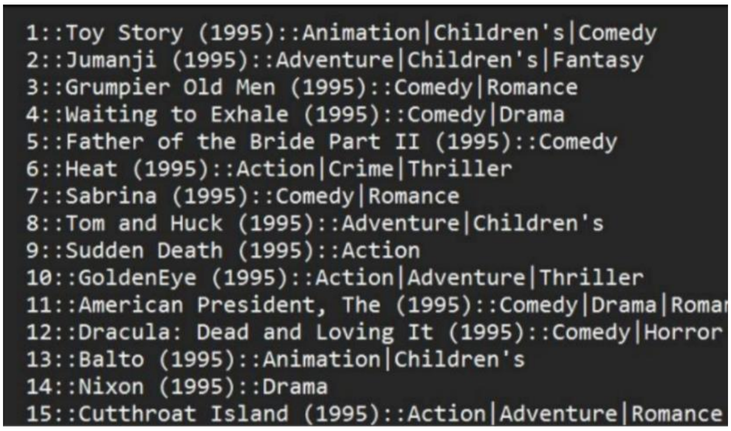
The **MovieLens 1M dataset** is a widely used and popular movie recommendation systems' dataset. It contains 1,000,000 (1M) ratings from over 6,000 people of 4,000 movies, along with geographical information about the users and genre information about movies. The **MovieLens 20M dataset** is a larger version of the MovieLens, containing 20,000,000 (20M) ratings from over 138,000 people of around 27,000 movies. It also includes tags applied by users to the movies, providing additional information about movie content. This dataset is useful for evaluating recommendation algorithms that can handle larger datasets and more complex features.

The **IMDb dataset** is another popular dataset for movie recommendation systems. It contains information about movies, TV shows, and their cast and crew, as well as ratings and reviews from users. The dataset is available for download from IMDb and can be used for research purposes.

The **Amazon Movies and TV Reviews dataset** contains over 8 million reviews from Amazon customers on over 400,000 movies and TV shows. In addition to star ratings, the dataset also includes text reviews, which can be used to train recommendation algorithms that take into account textual features and sentiment analysis. This dataset is particularly useful for evaluating recommendation algorithms that can handle large amounts of textual data.

The above-mentioned datasets have their own advantages and limitations. However, the MovieLens 1M dataset is a popular choice for building and evaluating movie recommendation systems due to its several advantages. Despite its relatively small size compared to other datasets, the dataset's 1 million ratings are sufficient for evaluating recommendation algorithms, especially for researchers with limited computing resources. The dataset is also diverse in terms of quantity of movies and users included, as well as their demographics. This diversity makes it a good dataset for evaluating recommendation algorithms that need to work across a range of users and movies, which makes it more representative of actual user behavior. Additionally, the ratings in the MovieLens 1M dataset are known for their high quality and accuracy, having been collected over a long period of time from real users. Finally, the dataset is freely available for research purposes, allowing for comparisons between different recommendation algorithms that have been trained and evaluated on the same dataset.

Hence, we choose Movielens 1m as our dataset. The dataset consists of three .dat files: ratings, users, and movies. The ratings.dat file contains the user ratings, with each row representing a single rating from a user for a movie. The file includes the user ID, the movie ID, the rating, and a timestamp of when the rating was made.



```
1::Toy Story (1995)::Animation|Children's|Comedy
2::Jumanji (1995)::Adventure|Children's|Fantasy
3::Grumpier Old Men (1995)::Comedy|Romance
4::Waiting to Exhale (1995)::Comedy|Drama
5::Father of the Bride Part II (1995)::Comedy
6::Heat (1995)::Action|Crime|Thriller
7::Sabrina (1995)::Comedy|Romance
8::Tom and Huck (1995)::Adventure|Children's
9::Sudden Death (1995)::Action
10::GoldenEye (1995)::Action|Adventure|Thriller
11::American President, The (1995)::Comedy|Drama|Roma
12::Dracula: Dead and Loving It (1995)::Comedy|Horror
13::Balto (1995)::Animation|Children's
14::Nixon (1995)::Drama
15::Cutthroat Island (1995)::Action|Adventure|Romance
```

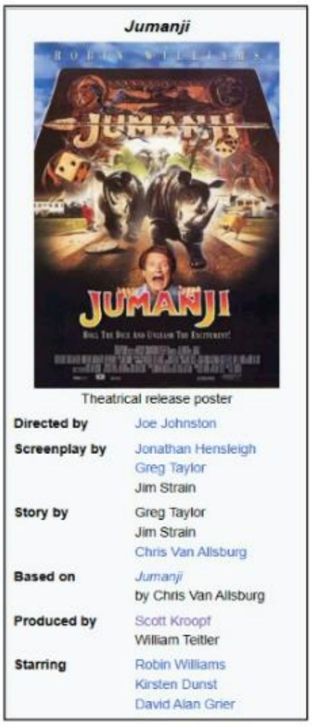
Figure 3.1: The list of movies in .dat file

The users.dat file contains demographic information about the users. Finally, the movies.dat file contains information about each movie, including its title, release year, and genres.

Although, MovieLens 1M dataset is the most suitable dataset in our scenario, it only provides basic information such as movie name, release year and genres. In order to create a more comprehensive recommendation system, we needed to gather additional information such as the movie's cast and director. To accomplish this, we used the help of data scraping.

3.1.2) DATA SCRAPING

Selenium is an automation tool which offers several types of selectors to identify the HTML tags and attributes we needed to scrape. We evaluated different selectors based on their simplicity and time complexity, and ultimately selected the xpath selector. The position of the content we needed varied slightly depending on the type of movie search, but the xpath selector allowed us to accurately extract the required information. Selenium automation tool to scrape the required attributes from a website that consistently provides the information we needed: **Wikipedia.**



Jumanji

Theatrical release poster

Directed by	Joe Johnston
Screenplay by	Jonathan Hensleigh Greg Taylor Jim Strain
Story by	Greg Taylor Jim Strain Chris Van Allsburg
Based on	*Jumanji* by Chris Van Allsburg
Produced by	Scott Kroopf William Teitler
Starring	Robin Williams Kirsten Dunst David Alan Grier

Fig 3.2: The Wikipedia Format

One significant issue we encountered was the high number of null values in the scraped data. We found that simply searching for the movie name could lead to a webpage that did not contain the desired information, as many movie names can also refer to generic words. To address this issue, we modified our scraping method to include the movie's release year as an additional parameter in the search, ensuring that we selected the most appropriate search result to obtain the desired information.

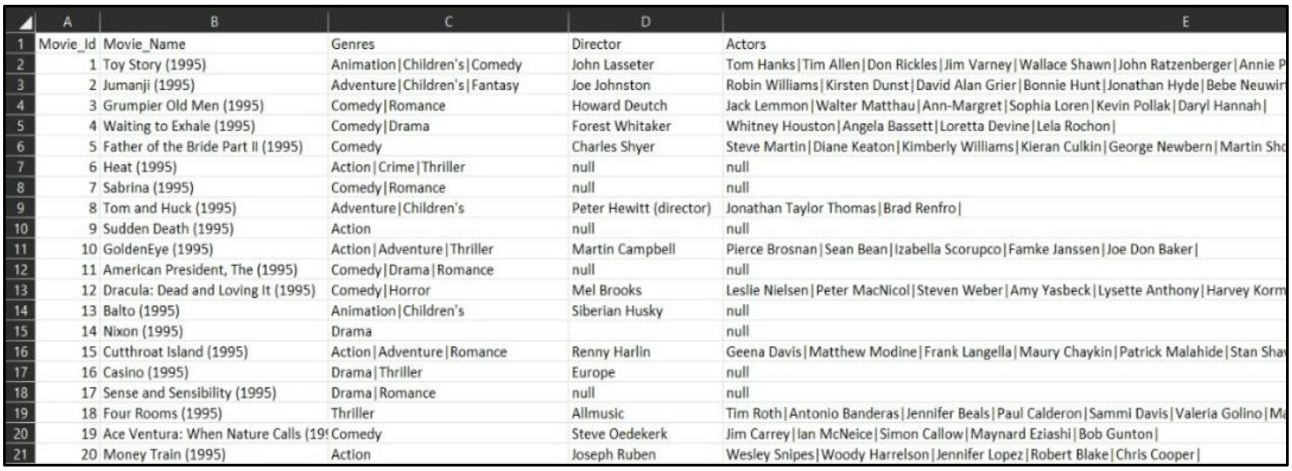
After scraping all the necessary attributes for each movie, we stored the data along with the initial attributes in the movies.csv file. With this additional information about each movie's cast and director, we were able to build a more comprehensive recommendation system that takes into account important factors in movie selection. The pseudocode for scraping is shown below.

```
# Define the Wikipedia URL for the movie you want to scrape
url = "https://en.wikipedia.org/wiki/Movie_Title"
driver.get(url)
actors = driver.find_elements(By.XPATH, 'xpath_for_actors')  # Replace 'xpath_for_actors' with the XPath expression to locate the actors' details
for actor in actors:
        # Extract relevant information from the actor element
```

```
            a.  actor_name = actor.find_element(By.XPATH, 'xpath_for_actor_name').text
            b.  # Process or store the actor information as desired
    5.   # Scrape the directors' details
        directors = driver.find_elements(By.XPATH, 'xpath_for_directors')  # Replace
        'xpath_for_directors' with the XPath expression to locate the directors' details
    6.  for director in directors:
            a.  director_name = director.find_element(By.XPATH,
                'xpath_for_director_name').text
            b.  # Process or store the director information as desired
```

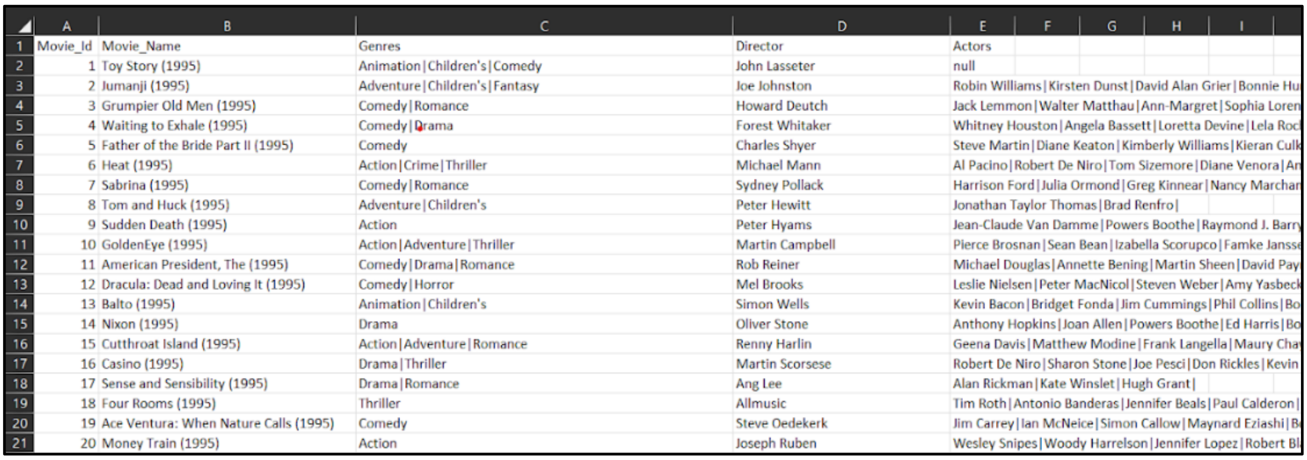
Algorithm 3.1: Scraping

The code initializes the URL of a Wikipedia page and navigates to that page using Selenium. Then, it finds and extracts the details of actors and directors from the web page using XPath expressions, storing the information for further processing or storage.



	A	B	C	D	E
1	Movie_Id	Movie_Name	Genres	Director	Actors
2	1	Toy Story (1995)	Animation|Children's|Comedy	John Lasseter	Tom Hanks|Tim Allen|Don Rickles|Jim Varney|Wallace Shawn|John Ratzenberger|Annie P
3	2	Jumanji (1995)	Adventure|Children's|Fantasy	Joe Johnston	Robin Williams|Kirsten Dunst|David Alan Grier|Bonnie Hunt|Jonathan Hyde|Bebe Neuwir
4	3	Grumpier Old Men (1995)	Comedy|Romance	Howard Deutch	Jack Lemmon|Walter Matthau|Ann-Margret|Sophia Loren|Kevin Pollak|Daryl Hannah|
5	4	Waiting to Exhale (1995)	Comedy|Drama	Forest Whitaker	Whitney Houston|Angela Bassett|Loretta Devine|Lela Rochon|
6	5	Father of the Bride Part II (1995)	Comedy	Charles Shyer	Steve Martin|Diane Keaton|Kimberly Williams|Kieran Culkin|George Newbern|Martin Sho
7	6	Heat (1995)	Action|Crime|Thriller	null	null
8	7	Sabrina (1995)	Comedy|Romance	null	null
9	8	Tom and Huck (1995)	Adventure|Children's	Peter Hewitt (director)	Jonathan Taylor Thomas|Brad Renfro|
10	9	Sudden Death (1995)	Action	null	null
11	10	GoldenEye (1995)	Action|Adventure|Thriller	Martin Campbell	Pierce Brosnan|Sean Bean|Izabella Scorupco|Famke Janssen|Joe Don Baker|
12	11	American President, The (1995)	Comedy|Drama|Romance	null	null
13	12	Dracula: Dead and Loving It (1995)	Comedy|Horror	Mel Brooks	Leslie Nielsen|Peter MacNicol|Steven Weber|Amy Yasbeck|Lysette Anthony|Harvey Korm
14	13	Balto (1995)	Animation|Children's	Siberian Husky	null
15	14	Nixon (1995)	Drama		null
16	15	Cutthroat Island (1995)	Action|Adventure|Romance	Renny Harlin	Geena Davis|Matthew Modine|Frank Langella|Maury Chaykin|Patrick Malahide|Stan Sha
17	16	Casino (1995)	Drama|Thriller	Europe	null
18	17	Sense and Sensibility (1995)	Drama|Romance	null	null
19	18	Four Rooms (1995)	Thriller	Allmusic	Tim Roth|Antonio Banderas|Jennifer Beals|Paul Calderon|Sammi Davis|Valeria Golino|Ma
20	19	Ace Ventura: When Nature Calls (19!	Comedy	Steve Oedekerk	Jim Carrey|Ian McNeice|Simon Callow|Maynard Eziashi|Bob Gunton|
21	20	Money Train (1995)	Action	Joseph Ruben	Wesley Snipes|Woody Harrelson|Jennifer Lopez|Robert Blake|Chris Cooper|

Figure 3.3: Dataset after scraping before modifying the scraping method



	A	B	C	D	E	F	G	H	I
1	Movie_Id	Movie_Name	Genres	Director	Actors				
2	1	Toy Story (1995)	Animation|Children's|Comedy	John Lasseter	null				
3	2	Jumanji (1995)	Adventure|Children's|Fantasy	Joe Johnston	Robin Williams|Kirsten Dunst|David Alan Grier|Bonnie Hu				
4	3	Grumpier Old Men (1995)	Comedy|Romance	Howard Deutch	Jack Lemmon|Walter Matthau|Ann-Margret|Sophia Loren				
5	4	Waiting to Exhale (1995)	Comedy|Drama	Forest Whitaker	Whitney Houston|Angela Bassett|Loretta Devine|Lela Roc				
6	5	Father of the Bride Part II (1995)	Comedy	Charles Shyer	Steve Martin|Diane Keaton|Kimberly Williams|Kieran Culk				
7	6	Heat (1995)	Action|Crime|Thriller	Michael Mann	Al Pacino|Robert De Niro|Tom Sizemore|Diane Venora|An				
8	7	Sabrina (1995)	Comedy|Romance	Sydney Pollack	Harrison Ford|Julia Ormond|Greg Kinnear|Nancy Marchan				
9	8	Tom and Huck (1995)	Adventure|Children's	Peter Hewitt	Jonathan Taylor Thomas|Brad Renfro|				
10	9	Sudden Death (1995)	Action	Peter Hyams	Jean-Claude Van Damme|Powers Boothe|Raymond J. Barry				
11	10	GoldenEye (1995)	Action|Adventure|Thriller	Martin Campbell	Pierce Brosnan|Sean Bean|Izabella Scorupco|Famke Janss				
12	11	American President, The (1995)	Comedy|Drama|Romance	Rob Reiner	Michael Douglas|Annette Bening|Martin Sheen|David Pay				
13	12	Dracula: Dead and Loving It (1995)	Comedy|Horror	Mel Brooks	Leslie Nielsen|Peter MacNicol|Steven Weber|Amy Yasbeck				
14	13	Balto (1995)	Animation|Children's	Simon Wells	Kevin Bacon|Bridget Fonda|Jim Cummings|Phil Collins|Bo				
15	14	Nixon (1995)	Drama	Oliver Stone	Anthony Hopkins|Joan Allen|Powers Boothe|Ed Harris|Bo				
16	15	Cutthroat Island (1995)	Action|Adventure|Romance	Renny Harlin	Geena Davis|Matthew Modine|Frank Langella|Maury Cha				
17	16	Casino (1995)	Drama|Thriller	Martin Scorsese	Robert De Niro|Sharon Stone|Joe Pesci|Don Rickles|Kevin				
18	17	Sense and Sensibility (1995)	Drama|Romance	Ang Lee	Alan Rickman|Kate Winslet|Hugh Grant|				
19	18	Four Rooms (1995)	Thriller	Allmusic	Tim Roth|Antonio Banderas|Jennifer Beals|Paul Calderon|				
20	19	Ace Ventura: When Nature Calls (1995)	Comedy	Steve Oedekerk	Jim Carrey|Ian McNeice|Simon Callow|Maynard Eziashi|B				
21	20	Money Train (1995)	Action	Joseph Ruben	Wesley Snipes|Woody Harrelson|Jennifer Lopez|Robert Bl				

Figure 3.4: Dataset after adding the required features through web scraping

3.1.3) DATA PREPROCESSING

After reading the raw scraped data from the MovieLens dataset, the pre-processing steps the dataset object is passed through are chained together in a FIFO order and are as follows -

1. Keeping only users and items with at least 25 interactions (i.e., "k_cores_value=25") in the ‘training set’, which is a way to remove less active users and items that may not be as informative.
2. Sampling a subset of users from the dataset, with a user quota of 0.3, which means that only 30% of the users will be kept in the dataset after this step. Sampling datasets has advantages such as reduced computational cost, better resource utilisation, and easier data exploration. By processing a subset of the data, computational cost can be significantly reduced, making processing and analysis faster. Exploring data and identifying patterns can be easier when working with smaller subsets of the data.

3.1.4) DATA CLEANING

Data cleaning is an essential step when utilizing the MovieLens 1M dataset or any other dataset for movie recommendation systems. The purpose of data cleaning is to improve accuracy, consistency and usability of the data. By performing data cleaning tasks such as handling missing values, eliminating duplicates, standardizing data formats, and addressing inconsistencies, you can ensure the dataset is reliable and of high quality. This, in turn, contributes to more precise and meaningful recommendations from your movie recommendation system. Data cleaning is crucial for creating a solid foundation and maximizing the effectiveness of your movie recommendation algorithms.

We focused on these steps:

1. Handling Missing Values: I carefully dealt with any missing values by either removing rows with missing values or filling them in using appropriate techniques such as mean or median imputation.
2. Eliminating Duplicates: I thoroughly checked for and eliminated any duplicate entries in the dataset, ensuring that each movie and rating appeared only once.

3. Handling Outliers: I identified and addressed outliers in the rating values, making informed decisions on whether to keep or remove them based on thorough analysis and domain knowledge.
4. Addressing Inconsistencies: I paid close attention to inconsistencies in the data, such as mismatched movie titles or genres, and took necessary steps to correct them, ensuring data consistency and reliability.

3.1.5) DATA SPLITTING

In our system, data splitting serves the purpose of evaluating and validating the performance of the recommendation model. The main goal is to simulate real-world scenarios where the model is trained on a portion of the data and tested on unseen data to assess its generalization capabilities, and it ensures that the model is not overfitting the training data.

Data splitting also supports hyperparameter optimization, where different hyperparameter settings are tested and compared on the test set. This process helps in finding the optimal hyperparameter values that lead to improved performance of the recommendation model.

Our movie recommendation system utilizes k-fold data splitting as a means to assess the effectiveness and generalization capabilities of the recommendation model. The objective behind employing k-fold cross-validation is to derive a more trustworthy evaluation of the model's performance through the utilization of multiple data subsets for both training and testing purposes.

K-fold data splitting involves the division of the dataset into k folds or subsets. The model then undergoes k iterations, each time using a different fold as the testing set while employing the remaining folds for training. This methodology enables a comprehensive assessment of the model's performance by testing it on various portions of the dataset. By averaging the outcomes from these k iterations, we obtain a more reliable estimation of the model's performance, allowing for an evaluation of its ability to generalize.

The purpose of the warm data splitting is to provide a specialized implementation of data splitting for cross-validation that ensures every user has at least one interaction in all splits. This class is designed for scenarios where it is important to have a representative distribution of user interactions across different folds while maintaining the integrity of the warm users.

The main functionality of doing warm k-fold splitting is to address the issue of "**cold**" users, who have very few interactions in the dataset. In a typical k-fold cross-validation, there is a possibility that some users may end up with no interactions in one or more of the folds, resulting in incomplete evaluations for those users. This class tackles this problem by only allowing warm users in all splits, where warm users are defined as those with interactions greater than or equal to the number of folds.

The warm k fold data split performs the warm-item splitting by distributing the interactions of each user across different folds while ensuring that warm users are present in all splits. It utilizes techniques such as shuffling and dividing the interactions of each user into equal portions for each fold. This process guarantees that every user contributes to the training and testing sets in a balanced manner.

```
 # Shuffle the data randomly
shuffled_data = randomize(data)
foldSize = len(data) // k
# Perform k-fold cross-validation
for x in range(K):
        # Find the start and end indices for the current fold
        first_index = x * foldSize
        last_index = (x + 1) * foldSize
      # Split the data for the current fold
        validation_data = shuffledData[first_index:last_index]
        training_data = shuffledData[:first_index] + shuffled_data[last_index:]
       # Add the training and validation sets to the respective lists
        training_sets.append(training_data)
        validation_sets.append(validation_data)
return training_sets, validation_sets
```

Algorithm 3.2: K fold data splitting

The code performs k-fold cross-validation by shuffling the data randomly, dividing it into folds, and creating training and validation sets for each fold. It returns a list of training sets and a list of validation sets.

```
# Shuffle the data randomly
shuffle_data(data)
warm_users = {}
# Iterate over each interaction in the data
for interaction in data:
    user = interaction['user']
    item = interaction['item']
    # Check if the user is already in the warm users' dictionary
    if user in warm_users:
    # Add the interaction to the train data
    train_data.append(interaction)
    else:
    # Add the interaction to the test data
    test_data.append(interaction)
    # Add the user to the warm users' dictionary
    warm_users[user] = True
    # Check if we have enough warm users for k-fold
    if len(warm_users) >= k:
        break
return train_data, test_data
```

Algorithm 3.3: Warm K fold data splitting

The code shuffles a given dataset randomly and splits it into training and test data. It ensures that each user's interactions are either included in training data or the test data, but not both. The splitting is based on a warm user criterion, where warm users are added to the training data until a specified limit (k) is reached.

3.1.6) INCREMENTAL SPARSE MATRIX

The Incremental Sparse Matrix is used in our recommendation system to efficiently handle and update the user-item rating matrix. The user-item rating matrix used is extremely large and sparse (i.e., most entries are empty), and storing the entire matrix in memory would be inefficient. Hence, Incremental Sparse Matrix is preferred becaue it stores only the non zero cells of the matrix, resulting in significant memory savings.

This is not the only reason why these matrices have been chosen. They allow for incremental updates to the matrix. In a recommendation system, users may rate new movies or provide additional ratings for existing movies over time. The incremental update capability enables efficient addition of new data points without needing to reconstruct the entire matrix.

To build the sparse matrix, we first initialize an empty sparse matrix with the required dimensions (number of rows and columns). This matrix is typically stored in one of the formats such as Compressed Sparse Row (CSR) or Compressed Sparse Column (CSC), which store only the non-zero entries of the matrix to save memory. Next, we read the dataset in batches to avoid loading the entire dataset into memory. For each batch of data, we convert it to sparse format and add it to the sparse matrix using matrix addition operations. Finally, as new data is received, we can add new rows or columns to the sparse matrix as needed by appending the new row or column to the existing matrix using the appropriate matrix addition operation. By building the matrix in this incremental way, we avoid the need to reconstruct the entire matrix each time new data is received.

There are several classes that collectively provide functionality for managing and manipulating sparse matrices. The main functionality achieved by each class:

1. `IncrementalSparseMatrix_ListBased`:
    a. Represents a sparse matrix using a list-based approach.
    b. Provides methods to add data points to the matrix.
    c. Supports mapping of original row and column IDs to matrix indices.
    d. Can generate a sparse matrix representation using SciPy.
2. `IncrementalSparseMatrix`
    a. Uses fixed-size arrays and dynamically increases their size as new data points are added.

   b. Overrides the `get_SparseMatrix` method to create a sparse matrix representation using the updated array-based data structure.

3. `IncrementalSparseMatrix_FilterIDs` class (inherits from `IncrementalSparseMatrix`):
   a. Further extends `IncrementalSparseMatrix` with the ability to filter and control the IDs of rows and columns.
   b. Allows for defining rules on whether to add or ignore new row and column IDs.
   c. Maintains dictionaries to map original IDs to matrix indices and handles the addition of new IDs based on the defined rules.

```
# Initialize empty sparse matrix
sparse_matrix = initialize_sparse_matrix(num_rows, num_cols)
# Read dataset in batches
for batch in dataset_batches:
    # Convert batch to sparse format
    sparse_batch = convert_to_sparse(batch)
    # Add batch to sparse matrix
    sparse_matrix = add_sparse_batch(sparse_matrix, sparse_batch)
# Add new rows or columns to sparse matrix
if new_data_received:
    # Append new row or column to existing matrix
    sparse_matrix = append_new_data(sparse_matrix, new_data)
```

Algorithm 3.4: Creating Incremental Sparse Matrix

The code initializes an empty sparse matrix and then reads a dataset in batches. Each batch is converted to a sparse format and added to the sparse matrix. If new data is received, the code appends a new row or column to the existing matrix.

3.1.7) BUILDING MATRICES

In a recommender system, the URM and the ICM are two key components used to recommend items to users. These two matrices are commonly referred to as URM and ICM, respectively.

ITEM CONTENT MATRIX (ICM)

The ICM is a matrix where the rows describe the items and their features such as the genre, director, actors, language, release year etc are represented by the columns, that describe those items.

ICM is used to develop a profile of the user's preferences based on the features of the items they have interacted with or shown interest in. By analyzing the presence or absence of certain features in the ICM for the items that the user has positively rated or engaged with, we can identify patterns and preferences specific to that user. It also facilitates the calculation of similarity between items. By comparing the feature vectors of different items in the matrix, we can quantify the similarity or dissimilarity between them. This information is crucial for generating recommendations based on item similarity, where items with similar features are suggested to users who have shown interest in related items.

We have decided to use genre, director, and actors as the features in our project. Each element in the matrix corresponds to the presence of a feature for an item. The ICM is typically very sparse because not all items have all features. The pseudo code for the algorithm to generate ICM is given below.

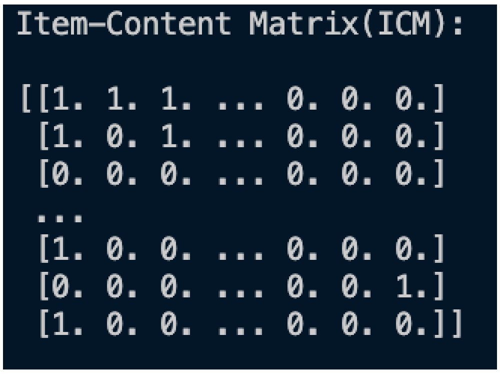
```
# Assume we have a list of items and their corresponding features
items = ['item1', 'item2', 'item3', ...]
features = ['genre', 'director', 'actors', ...]

# Iterate over each item
for item in items:
        # Initialize an empty feature vector for the item
        feature_vector = {}
        # Iterate over each feature
```

```
for feature in features:
# Check if the item has the feature
        if item_has_feature(item, feature):
                feature_vector[feature] = 1
        else:
                feature_vector[feature] = 0
                # Add the feature vector to the ICM
                ICM[item] = feature_vector
```

Algorithm 3.5: Building ICM

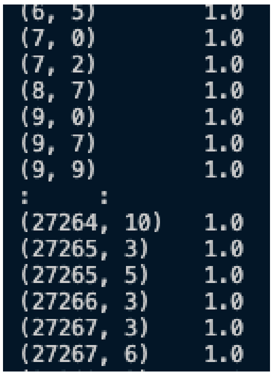
The code generates an ICM where each item is represented by a feature vector indicating the presence (1) or absence (0) of various features. It iterates over each item, checks if it has each feature, and updates the corresponding feature vector in the ICM with the presence/absence information.



```
Item-Content Matrix(ICM):

[[1. 1. 1. ... 0. 0. 0.]
 [1. 0. 1. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [1. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 1.]
 [1. 0. 0. ... 0. 0. 0.]]
```

Figure 3.5: Generated ICM (Dense)



```
(6, 5)        1.0
(7, 0)        1.0
(7, 2)        1.0
(8, 7)        1.0
(9, 0)        1.0
(9, 7)        1.0
(9, 9)        1.0
:     :
(27264, 10)   1.0
(27265, 3)    1.0
(27265, 5)    1.0
(27266, 3)    1.0
(27267, 3)    1.0
(27267, 6)    1.0
```

Figure 3.6: Generated ICM (Sparse)

USER RATING MATRIX (URM)

The User-Item Rating Matrix (URM) is a structured representation that records the engagements or ratings between users and items. In the matrix, users are represented as rows, and items are represented as columns. Each cell within the matrix denotes the specific interaction or rating between an individual user and a particular item.

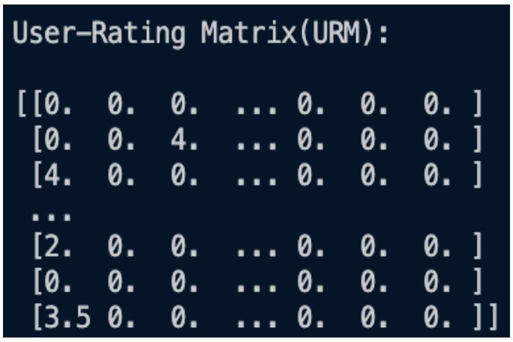
The entries in the URM can have different interpretations. They can be binary, indicating whether there is an interaction between the user and the item or not. Alternatively, they can represent ratings given by users to items. Our project uses like and dislike buttons, and uses a binary format to represent the ratings.

By collecting and organizing this data, the URM reflects the preferences of individual users. It reflects which items a particular user has disliked or liked, and to what extent, based on their ratings. This information is valuable for generating personalized recommendations. The pseudo code to generate URM is given below.

```
# Step 1: Extract unique user and item IDs
user_ids = set()
item_ids = set()
for interaction in interactions:
    user_ids.add(interaction['user_id'])
    item_ids.add(interaction['item_id'])
# Step 2: Create an empty URM matrix
num_users = len(user_ids)
num_items = len(item_ids)
URM = [[0] * num_items for _ in range(num_users)]
# Step 3: Populate the URM matrix with ratings
for interaction in interactions:
        user_id = interaction['user_id']
        item_id = interaction['item_id']
        rating = interaction['rating']
        # Map user and item IDs to matrix indices
        user_index = user_ids.index(user_id)
        item_index = item_ids.index(item_id)
        # Store the rating in the URM matrix
        URM[user_index][item_index] = rating
```

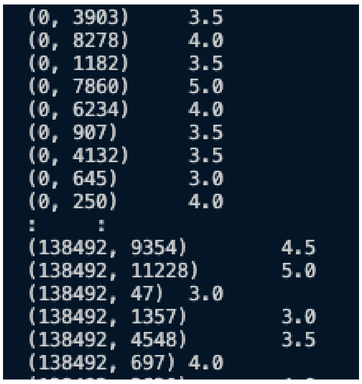
Algorithm 3.6: Building URM

The code populates the URM matrix with ratings by mapping the user and item IDs to their corresponding matrix indices and storing the ratings in the matrix. Figure 3.7 and 3.8 show the generated URM, in both dense and sparse formats.



```
User-Rating Matrix(URM):

[[0.  0.  0.  ... 0.  0.  0. ]
 [0.  0.  4.  ... 0.  0.  0. ]
 [4.  0.  0.  ... 0.  0.  0. ]
 ...
 [2.  0.  0.  ... 0.  0.  0. ]
 [0.  0.  0.  ... 0.  0.  0. ]
 [3.5 0.  0.  ... 0.  0.  0. ]]
```



```
  (0, 3903)     3.5
  (0, 8278)     4.0
  (0, 1182)     3.5
  (0, 7860)     5.0
  (0, 6234)     4.0
  (0, 907)      3.5
  (0, 4132)     3.5
  (0, 645)      3.0
  (0, 250)      4.0
  :     :
  (138492, 9354)        4.5
  (138492, 11228)       5.0
  (138492, 47)  3.0
  (138492, 1357)        3.0
  (138492, 4548)        3.5
  (138492, 697) 4.0
```

Figure 3.7: Generated URM (Dense)

Figure. 3.8: Generated URM (Sparse)

3.2) THE MODEL

Our implementation of the collaborative filtering algorithm draws inspiration from the RP3beta model introduced by Paudel et al. This model demonstrated competitive recommendation quality compared to state-of-the-art models, while keeping the computational cost low by not relying on machine learning. The RP3beta model operates on a graph-based approach, which involves simulating a random walk between two groups of nodes: users and items. Each user and item involved in an interaction are connected, allowing the model to generate an item-item similarity matrix from the graph adjacency matrix. This similarity matrix reflects the transition probability between two items and can be easily obtained from the User-Item Interaction Matrix (URM).

The first step in our algorithm is **Candidate Generation.** The vast corpus of entities (movies in our case) to be recommended is whittled down to hundreds of movies that may be of interest to the user during candidate generation. The main focus of recommendation algorithms is based on this step.

In a pure CBF candidate generation technique, the user profile is scanned for the items that the user has positively interacted with and items similar to those items form the candidate set for the recommendations. These similar items are discovered from the item-item similarity matrix that is developed in the CBF technique. Because our algorithm also uses CBF as the core model, generation of candidates follows the same procedure, however the similarity values in that item-item similarity matrix involve an element of collaborative similarity as well. Hence, the candidate set formed is not exactly the same as it would have been in the case of pure CBF method.

A typical item based CF candidate generation algorithm attempts to identify items that are similar based on user interactions and recommend items that are similar to the ones the user has already interacted with. This also involves an item-item similarity matrix as well where the similarity values between each pair of items are present.

As per our proposed solution, we identified that human comparison of two objects frequently assigns various features varying weights. We take advantage of this fact to solve our purpose. We assign a weight to each of the features. These weights will modify the item content matrix and hence the content based item similarity matrix as well. Our aim is to have such values of features weights such that the resultant content based item similarity matrix approximates to the collaborative based item similarity matrix. This modification will let our candidate set also include the items that are similar based on user interactions but still having an element of feature based similarity.

To obtain the desired values for our feature weights, we apply Machine Learning. For every epoch, we calculate the similarity for certain pairs of items based on the content based item similarity matrix values as well as the feature weights, while our target similarity value is obtained from the collaborative based item similarity matrix. The difference between the two becomes our error function and try to minimize it by changing the feature weights, in the process of machine learning.

We can summarise our candidate generation algorithm in the following steps:

1. **Building item similarity Matrices:** Building item-item similarity matrices for both collaborative and content based filtering
2. **Identifying Similar Items:** Collecting similar feature item pairs also having some collaborative similarity
3. **Machine Learning**: Initialize weights for each feature and loop over each item pair identified in step 2, use common features between them and their feature weights to generate a similarity value for the item pair. This is the calculated similarity while the target similarity comes from the collaborative item-item similarity matrix. The difference between the two is the loss. Apply gradient descent to minimise this loss and learn the feature weights on the way.
4. **Modifying the Item Content Matrix:** Using the learned feature weights, modify the item content matrix and generate the new hybrid item-item similarity matrix
5. **Select the Suitable Candidates:** All the items similar to those items with which the target user has positively interacted, are added to the candidate set

3.2.1) BUILDING ITEM SIMILARITY MATRICES

The first step of our candidate generation algorithm is building item similarity matrices for both our recommendation methods. We describe each of them in the following subsections. Note that in the subsequent subsections we denote collaborative based item similarity matrix as $\text{IIM}_{\text{collab}}$ and content based item similarity matrix as $\text{IIM}_{\text{content.}}$

COLLABORATIVE FILTERING

To implement the collaborative filtering algorithm, we were inspired by the RP3beta (Paudel et al.) model, which achieved a recommendation quality comparable to the start of the art models without involving Machine Learning and hence at a very low computational cost. It essentially is a graph-based technique which simulates a random walk between two groups of nodes, such as user and items. Each user and the item involved in an interaction are connected to each other. The model generates an item-item similarity matrix generated from the graph adjacency matrix, which is easily obtained from the URM, and which indicates the transition probability between the two items.

We also include a hyperparameter to handle the issue of Popularity Bias in the model in order to achieve better results. **Popularity Bias** is a situation in which products with few interactions are suggested poorly, receive less interactions than others, and hence lose awareness and end up in this same vicious cycle. Even though the items had been in the catalogue for months, they might receive very few interactions as a result of this. So The similarity scores are increased by an ‘alpha’ coefficient which determines the weight of user item interactions and divided by the popularity of each item, which is adjusted through ‘beta’ coefficient. The latter serves as a reranking step that accounts for the popularity bias. These hyperparameters will be discussed in detail later in chapter 4.

```
1. Normalize the user-item matrix (URM_train) row-wise to get Mui.
   Mui = normalize(URM_train, norm='l1', axis=1)
2. Transpose the URM_train and convert it to a boolean matrix, where the non-zero
   values are replaced with 1.
   Miu_bool = URM_train.transpose(copy=True)
   Miu_bool.data = np.ones(Miu_bool.data.size, np.float32)
3. Calculate the sum of each row of the boolean matrix Miu_bool to find the degree of each item.
   Miu_bool_sum = np.array(Miu_bool.sum(axis=1)).ravel()
4. Create an array 'degree' with the same shape as the number of items in the URM_train matrix, with all elements initially set to 0.
5. Create a boolean array 'nonZeroMask' with the same shape as the number of items in the URM_train matrix, where each element is True if the sum of the corresponding row in Miu_bool_sum is non-zero, else False.
6. For each item i where nonZeroMask[i] is True, calculate the degree of that item using the formula: degree[i] = (Miu_bool_sum[i])^(-beta), where beta is a hyperparameter.
7. If alpha is not equal to 1, raise both the Mui and Miu matrices to the power of alpha.
```

Algorithm 3.7: Handling popularity bias

The above code calculates the degree of each item, which is used to penalize popular items in the recommendation process. The degree is inversely proportional to the sum of interactions of an item, with a power of beta applied as a parameter.

The main focus of the collaborative filtering algorithm is to generate the item-item similarity matrix. Because the URM from which the matrix is to be generated is very huge in size, and matrix multiplication operations are supposed to be performed to obtain the desired result, the whole matrix cannot be operated upon at once. We take a block size of 200, and perform a block-wise matrix multiplication to reduce the memory usage of the algorithm.

The pseudo-code for the algorithm is:

```
1. Initialize the Collaborative Filtering Class with URM_train
2. Define the fit method that takes in several hyperparameters such as alpha, beta, min_rating, and normalize_similarity (bool variable for whether to normalize the values or not)
```

```
3. Compute Mui by normalizing URM_train along the rows.
4. Compute Miu by normalizing the transposed URM_train along the rows.
5. Unpack the multiplication into blocks for memory reasons. Set the block dimension to 200 and data block to 10000000.
6. Use a loop to process each block. For each data block:
    a. Compute the similarity block as the dot product of Miu and Mui, converting it to an array.
    b. For each row in the similarity block, compute the row data by element-wise multiplication with the degree array, setting the diagonal value to 0.
    c. Find the top K values and corresponding indices from the row data.
    d. Add the values and indices to the respective arrays.
    e. If the number of cells is equal to the length of the rows array, increase the size of the rows, cols, and values arrays.
7. Create a CSR matrix from the rows, cols, and values arrays and set it as the $IIM_{collab}$ attribute of the class instance.
```

Algorithm 3.8: Creating Collaborative based Item Similarity Matrix

The code unpacks the multiplication into blocks to avoid memory issues and loops through each block to compute the similarity block, element-wise multiply it with the degree array, and find the top K values and indices for each row. It adds the values and indices to their respective arrays and creates a CSR matrix from them, setting it as the W_sparse attribute of the class instance.

CONTENT BASED FILTERING

In this part of the candidate generation algorithm, we compute item-item similarities and select the top-k similar items based on their features and store these values in a sparse matrix. This sparse matrix is used to find the topK items to be recommended to a particular user. It starts by iterating through each item and computing their similarities by comparing their features with other items that share the same feature. The similarity values are incremented based on the feature values and a weighting factor. The computed similarities are then normalized and adjusted using shrinkage and specific coefficients if required. It returns a sparse matrix containing the computed similarities for each item. The pseudo-code for the step is described below:

```
1. Iterate through each itemIndex from 0 to end_col_local i.e the number of columns in ICM.
2. Compute item similarities using the computeItemSimilarities method and store them in self.this_item_weights.
3. In the computeItemSimilarities method , for a particular item, retrieve all the features of that item and iterate through all the items which have that particular feature and then increment the similarity value by multiplying their values with the row_weight.
4. Normalize and apply shrinkage to the computed similarities, ensuring the denominator is not zero.
5. If TopK is 0 (indicating all similarities should be stored), assign the computed similarities to self.W_dense matrix.
6. If TopK is non-zero, perform the following steps:
    a. Create an array this_item_weights_np to store the computed similarities in the desired ordering.
    b. Iterate through the this_item_weights_id data structure and add the weights to this_item_weights_np.
    c. Determine the number of items to consider for the TopK (minimum of TopK and this_item_weights_counter).
    d. Find the unordered set of topK items by partitioning this_item_weights_np.
    e. Sort the elements in the partition to get the topK items in the correct order.
    f. Store the original indices of the topK items in top_k_idx.
7. Add the computed similarities in the sparse matrix and return this matrix to find topK recommended items for a particular user.
```

Algorithm 3.9: Creating Content based Item Similarity Matrix

This same component is reutilized again to generate the hybrid item similarity matrix when the ICM is modified with the feature weights and IIM is again re-computed from this modified ICM.

3.2.2) IDENTIFYING SIMILAR ITEMS

After creating IIM_{collab} and $IIM_{content}$ we want to find those pairs of items which have similarity based on features as well as user interactions. These item pairs will be used in the step of Machine Learning because only they contain a non-zero similarity component in both cases and hence can contribute towards effective and fast convergence in learning.

We identify these pairs of items and pass on to the next step. Though the logic for this step is fairly simple, the process involves some important hyperparameters that can improve the efficiency of our system. One of them worth noting is ‘add_zeros_perc’,

The add_zeros_perc parameter is used to control the sparsity of the generated training data. When add_zeros_perc is set to a non-zero value, the function randomly sets some of the cell values to zero with a probability equal to the add_zeros_perc value. This is done to ensure that the model is exposed to some negative examples during training, i.e. pairs of items that are not similar to each other. This can help improve the model's ability to distinguish between similar and dissimilar items, and can prevent overfitting to the positive examples only.

The pseudo-code for this step is shown below:

```
1. Create empty numpy arrays to store row indices, column indices, and data values of $IIM_{content}$ matrix for sparse matrix representation. These arrays will store the similar item pairs. Note that in $IIM_{content}$, the rows as well as the columns represent items.
2. Iterate over all items in the $IIM_{content}$ matrix.
    a. For each item, retrieve its content-based similarity coordinates and collaborative similarity coordinates.
    b. Determine the number of common coordinates between the content-based and collaborative similarity matrices for the current item.
    c. For each coordinate in the content-based similarity matrix, randomly determine whether to include it in the training data based on the add_zeros_perc parameter.
    d. If the coordinate is common, add its value to the data list.
```

```
        e.  If the coordinate is not common and the random value is less than or equal to the add_zeros_perc add a zero to the data list.
        f.  If the number of samples exceeds the estimated number of samples, increase the size of the numpy arrays.
```

Algorithm 3.10: Identifying Similar items

The add_zeros_perc parameter is usually set to a small value, such as 0.1 or 0.2, to ensure that most of the cells still have non-zero values. The above code loops over each item and for each item, it identifies the set of similar items from the content-based similarity matrix and the set of similar items from the item-based similarity matrix. Note that the array for data value contains the similarity values from IIM_{collab}, because these values will be required during machine learning to compute the loss.

3.2.3) MACHINE LEARNING

Our main consideration behind the algorithm is the fact that a CF model will outperform a CBF model in terms of suggestion quality and will be better equipped to represent the user's perspective. And as discussed, using this fact we aim to learn feature weights to make our modified hybridized $IIM_{content}$ approximate to IIM_{collab}. Hence, our main algorithm can be summarized as the following optimization problem:

$$\underset{\mathbf{D}}{\operatorname{argmin}} \quad \left|\left|\mathbf{S}^{(\mathbf{CF})} - \mathbf{S}^{(\mathbf{D})}\right|\right|_F^2 + \alpha\,||\mathbf{D}||_F^2 + \beta\,||\mathbf{D}|| \qquad (1)$$

where $S^{(CF)}$ represents the IIM_{collab} from which we want to learn, $S^{(D)}$ is the item-item hybrid similarity matrix, D is the feature weight matrix, α and β are the regularization parameters.

Eq. (1) represents the loss function of the machine learning we are supposed to perform. Because we want to approximate our feature based similarity to user interaction based similarity, we will calculate the similarities between item pairs using the common features from the ICM and the learned feature weights, whereas our IIM_{collab} similarities will be target values. The difference between the two will determine the error which we want to minimize. The formula for the similarity calculated using common features is described in Eq.(2)

The following equation is used to calculate the similarity between the items:

$$sim(i,j) = \frac{\mathbf{f}_i \, D \, \mathbf{f}_j}{||\mathbf{f}_i| |_F^2 \, ||\mathbf{f}_j| |_F^2} \qquad (2)$$

where f_i is the feature vectors for movie i, $|| ||^2$ is the magnitude/absolute value of the feature vector. The feature weight matrix is D, which we are supposed to learn represents the importance of each feature.

So the feature weight array is hence initialized. Next, we iterate over the item pairs identified in the last step, and first extract the feature vectors of both of these items from the ICM and then find the common features between them because they are to be used in finding the similarity value. These common features are stored in an array along with their values. Here the values are the products of the ICM values corresponding to the respective item and features. A cell value in ICM denotes the relevance of a particular feature in a particular item. A nested loop must begin next iterating over all the identified common features where the similarity value is calculated as the product of the corresponding feature weight and the common feature data. This value is considered our predicted similarity score for the two items and our target similarity score is present in the data value array (which contains the values of IIM_{collab}), and the loss function is hence computed as the difference between the two.

The snippet of code involving the iteration over common features and calculating the loss value is shown in algorithm below

```
similarity_value_weighted = 0.0


for feature_index in range(num_common_features):
    feature_id = self.common_features_id[feature_index]
    similarity_value_weighted += self.D[feature_id] *
self.common_features_data[feature_index] * self.dropout_mask[feature_id]

# The gradient is the prediction error
error = similarity_value_weighted - similarity_value_target
cumul_loss += error**2
```

Algorithm 3.11: Calculating loss value

The above code uses a regularization hyperparameter called the dropout mask. Here the dropout technique involves randomly dropping out (i.e., setting to zero) a fraction of the units in the common features during training. This can help prevent the learning from becoming too reliant on any one set of weights, which can lead to overfitting. The dropout mask has been randomly generated for each training example and kept fixed for all iterations of the training process. The cum_loss used in the above code refers to the cumulative sum of the squared errors across all training samples processed so far.

Once the loss is generated, it is used to update the feature weights. The gradient is calculated and hence the feature weight for each feature is updated according to the step size which is calculated using the gradient and learning rate. Learning rate is another hyperparameter here. The code as shown below:

```
# For every common feature update the corresponding weight
for feature_index in range(num_common_features):
    feature_id = self.common_features_id[feature_index]

    if self.dropout_mask[feature_id]:
        gradient = error * self.common_features_data[feature_index]
        adaptive_gradient = self.compute_adaptive_gradient(feature_id, gradient)
        self.D[feature_id] -= self.learning_rate * (adaptive_gradient + self.l1_reg +
2*self.l2_reg * self.D[feature_id])
```

Algorithm 3.12: Updating the feature weights through gradient descent

The reason why we are updating the weights in every iteration is because we are using stochastic gradient descent. In each iteration of the algorithm, a single data point (in this case, a pair of items with a similarity value) is used to compute the error and update the weights. This is in contrast to batch gradient descent, where all data points are used to compute the error and update the weights in each iteration.

Updating the weights in the same iteration allows the model to learn from the current data point and adjust the weights accordingly. The adaptive gradient method used in this code also allows the weights to be updated with different step sizes depending on their past gradients, which can improve the convergence of the algorithm.

The l1_reg and l2_reg used in the above code are regularization parameters. They add penalty terms to the gradient. The idea behind these penalties is to prevent overfitting and encourage the model to learn more generalizable patterns in the data.

This training is performed for a specific number of epochs. However, because we focus on the computationally inexpensive aspect of the algorithm, we implement our training in an Early Stopping Manner. Early stopping is used to prevent overfitting and improve the generalization performance of a model during the training process. The idea behind early stopping is to monitor the performance of the model on a validation set during the training process, and stop the training when the performance on the validation set stops improving or starts to degrade, instead of waiting for the model to converge on the training set. By doing so, early stopping can help to prevent the model from overfitting to the training data and capturing noise instead of the underlying patterns in the data.

The pseudo-code for early stopping is shown below in algorithm below:

```
1. Set initial values:
   - best_validation_metric = None
   - lower_validations_count = 0
   - convergence = False
   - epochs_best = 0
   - epochs_current = 0

2. While epochs_current < epochs_max and not convergence:
       Run the current epoch
       If no validation required, set epochs_best to epochs_current
       Else if validation step is required:
            Prepare the model for validation
            Evaluate the recommender using the evaluator_object
            Update the optimal model:
              - Compute the current metric value
              - If it is the best metric value so far:
                  - Update best_validation_metric
                  - Update the best model
                  - Set epochs_best to epochs_current + 1
                  - Reset lower_validations_count
              - Else, increment lower_validations_count
3.  If stop_on_validation is True, and lower_validations_count >=
    lower_validations_allowed, and epochs_current >= epochs_min:
              convergence = True
4.  Increment epochs_current
```

Algorithm 3.13: Early stopping

As shown in the above Figure, at regular intervals (specified by a validation_every_n parameter), the algorithm's performance is evaluated on a validation dataset using a specified metric. If the algorithm's performance on the validation dataset improves, the current model is saved as the best model. If the algorithm's performance on the validation dataset does not improve for a specified number of consecutive validations (specified by lower_validations_allowed), training is terminated early and the best model is returned.

Hence, now we have obtained our learned feature weight array.

3.2.4) MODIFYING ITEM CONTENT MATRIX

Now that we have obtained the learned feature weight array, we are supposed to incorporate it in our matrices. First, the direct effect of different weights to the features is reflected in the ICM. All the columns corresponding to each feature are multiplied by their corresponding feature weight.

We convert our learned feature array to a weighted diagonal matrix; a weighted diagonal matrix places all the elements of the array as its diagonal elements and all the other elements remain 0. Multiplying ICM with this weighted diagonal matrix will produce our required modified ICM. The process is described below

$$\begin{bmatrix} d_0 & 0 & 0 \\ 0 & d_1 & 0 \\ 0 & 0 & d_2 \end{bmatrix} * \begin{bmatrix} v_{00} & v_{01} & v_{02} \\ v_{10} & v_{11} & v_{12} \\ v_{20} & v_{21} & v_{22} \end{bmatrix} = \begin{bmatrix} d_0v_{00} & d_0v_{01} & d_0v_{02} \\ d_1v_{10} & d_1v_{11} & d_1v_{12} \\ d_2v_{20} & d_2v_{21} & d_2v_{22} \end{bmatrix}$$

Figure 3.7: The weighted D matrix multiplied with ICM gives modified ICM with feature weighted values

Because selecting the candidates for a target user requires the use of an item similarity matrix, it is generated using the modified ICM matrix. The IIM is generated from the same piece of code described to initially create the $\text{IIM}_{\text{content}}$ from ICM. We call the IIM now formed as $\text{IIM}_{\text{hybrid}}$. We'll now use this IIM to find similar items for user's preferred items as the recommendations for users.

3.2.5) SELECT THE SUITABLE CANDIDATES

The final step in our model is to suggest a list of recommended items to a user. This correlates with the ranking step of recommendation algorithms, in general. It looks at the user's past interactions with items and uses that information to find other items that are similar. This is done by scanning the user profile from the URM. It calculates a similarity score between each item the user has interacted with and all other items. Then, it selects the top-k items with the highest similarity scores. To avoid suggesting similar items multiple times, it combines the selected items and removes any duplicates. Finally, it sorts the selected items based on how strongly they are related to the user's past interactions and returns the top-k items as recommendations.

The pseudo code is given below:

```
1. Retrieve the user's interaction history from the URM and convert it to a dense array format.
2. Compute the average similarity row (user_row) using the item-item similarity matrix for the items that the user has interacted with.
3. Initialize an empty list (top_k_idx) to store the indices of the top-k items.
4. Iterate over each item in the user's interaction history.
```

```
        ● Sort the item-item similarity values for each item in the history and
          select the top-k indices.
        ● Append the top-k indices to the top_k_idx list.
5. Combine the indices in the top_k_idx list and remove any duplicates.
6. Compute the similarity scores between the selected top-k items and the
   user_row.
7. Sort the indices based on the decreasing order of similarity scores and return the
   top-k items.
```

Algorithm 3.14: Algorithm for retrieving top k items for a particular user

The resultant top-k items are then passed to the application, and the application displays these results to the user. This recommendation algorithm is triggered whenever the user session is refreshed or reconnected. However, we have optimized the algorithm such that only those rows and columns are recomputed in the IIM matrices corresponding to the users and items for which there has been a feedback (i.e a like or dislike).

3.2.6) MAKING OUR MODEL INCREMENTAL

We have optimized our algorithm such that the training over the whole ICM and URM matrices is done only once and all the feedbacks provided by users in the form of likes and dislikes are updated in the matrices and only corresponding changes are made. The feedbacks are updated in the matrix and the optimized algorithm is triggered at every refresh or whenever the user session reconnects or the user logins.

We considered all the spots in the algorithm where the feedback would make changes. First, in the construction of the $\text{IIM}_{\text{collab}}$, changes will be there because feedback updates the URM matrix from which $\text{IIM}_{\text{collab}}$ is made. We observed that only the particular rows and columns corresponding to the updated cells in the URM are modified in $\text{IIM}_{\text{collab}}$.

Hence, we optimize that step to recompute only those cells and values. The code change corresponding to the optimization is reflected in the loop, where instead of now working with data blocks and iterating over the whole matrix, we only iterate over the rows of the modified cells. Iterating the columns is not required even, because IIM is a symmetric matrix and column values can just be copied with row values.

The other modifications have been done while finding similar items in our algorithm. Instead of iterating over the whole $\text{IIM}_{\text{content}}$ to find the items with both collaborative and feature based similarity, we would now only need to traverse through those rows and columns which have been recomputed in the earlier step. This causes the number of item pairs to be passed on the machine learning algorithm far less than during the initial training. Hence, because there are very few item pairs to iterate upon, our machine learning trains the feature weights much quickly.

3.3) APPLICATION

Our model behaviour is made visible through the application. In this subsection, we describe the flow of our application.

Scenario 1: Existing Users

In our system, existing users are individuals who have previously provided feedback or interacted with the system. These users have already expressed their preferences, either by rating movies, providing reviews, or indicating their interests in certain genres or actors. Whenever the user interface (UI) refreshes, a new list of personalized recommendations is presented to these warm users based on their past feedback and interactions.

The UI refresh process for warm users takes into account the latest available data, ensuring that the recommendations are up-to-date and reflect the user's evolving interests. This may involve periodically retraining the recommendation model with new user feedback or incorporating real-time data to capture the user's changing preferences. By providing tailored recommendations, the system aims to enhance user satisfaction and engagement, ultimately fostering a more personalized movie-watching experience.

Scenario 2: New Users

Cold users in the movie recommendations system are individuals who have not yet provided any feedback or interacted with the system. These users are newcomers to the platform or have not engaged with the recommendation features in the past. To cater to this user segment, the system presents them with a list of top K items, where K represents the number of most popular movies to showcase.

For cold users, the UI providse additional features to help them discover movies of interest. This includes a genre-based filters. By encouraging user engagement and enabling them to explore various movie options, the system aims to convert cold users into warm users over time. As these users start interacting with the system and providing feedback, their movie recommendations will gradually become more personalized, transitioning them into the warm user category.

3.3.1) BACKEND LOGIC

The backend code of our recommendation system is implemented using Flask, MongoDB, and external APIs. The code follows the following logic:

1. The home route is the main endpoint for generating movie recommendations. When a GET request is received, the user's ID is retrieved from the session, and the `retrain_model()` function is called to generate movie recommendations based on the user's voting history. Movie data is fetched from the MongoDB database, and a list of movies with relevant information is constructed.
2. The signup and login routes handle user registration and login. If the request is a POST request, the form data is processed, the password is hashed using bcrypt, and the new user is inserted into the MongoDB user’s collection.
3. The vote route handles movie voting. The movie ID and vote value are extracted from the JSON payload, and the votes dictionary stored in a cookie is updated.
4. The `get_movie_poster_url()` function utilizes the OMDB API to retrieve the poster URL for a given movie name.
5. The `retrain_model()` function retrieves the user's ID and vote data from the session cookie. It then calls the `retrainModel()` function, presumably defined in the "retrain_model" module, to retrain the movie recommendation model based on the user's votes. The generated movie recommendations are returned.

3.3.2) DATABASE LOGIC

Our database contains the following schemas or collections:

a. "users" collection: It stores user information for login and signup, and includes fields such as "userId", "name," "email," and "password”

b. "csvs" collection: It stores the movie attributes from the movies.csv file, and includes fields representing various attributes of each movie, such as "movieId," "title," and other relevant information.

The "users" collection allows the system to manage user authentication and registration, while the "csvs" collection provides a storage mechanism for movie data used for UI representation and ICM creation. By utilizing these collections, the movie recommendation system can effectively store and retrieve data from MongoDB enabling seamless user experiences and personalized recommendations based on user preferences.

3.4) TECHNOLOGIES USED

Selecting the suitable technologies for each system or application is a vital step in determining how smooth the system will be able to perform, how much complexity will be present in the integration of different components and how maintainable the application will be.

Learning from our previous projects, we analyzed and identified the appropriate technologies which will help our application to be computationally efficient, less complex while working with specific data structures like datasets and matrices, and easy to integrate between the frontend and backend of our application. Following is the set of technologies we have used in our project:

3.4.1) The Model

In our project, we needed to work with large datasets and CSV files, perform extensive operations on matrices, and deal with sparse matrices as well. Python provides excellent support for handling data in different formats, including CSV files, and has many powerful libraries for data manipulation, such as NumPy and Pandas. Hence, Python was our choice for the implementation of the model.

For implementing our machine learning model, we needed a programming language that has strong support for numerical computing and scientific computing libraries. Python has an excellent ecosystem for machine learning with many powerful libraries such as Scikit-learn. Furthermore, to make the execution of our model faster, we have also used Cython. Cython is a superset of Python and provides an easy way to write C extensions for Python. It allows us to write Python code that can be compiled to C code, which runs much faster than pure Python code. This is especially useful when dealing with large datasets or computationally intensive operations.

3.4.2) The Application

In the development of our movie recommendation system, we have utilized a range of technologies to provide an effective and user-friendly experience for our users. These technologies span the front-end, back-end, and database layers ensuring a seamless flow of information and recommendations.

3.4.3) User Interface

For the user interface (UI), we have employed HTML, CSS, JavaScript (JS), and Ajax. HTML forms the backbone of our web pages, providing the structure and layout, while CSS enhances the visual presentation, including styling, fonts, and colors. JS adds interactivity to the UI, enabling dynamic behavior such as user interactions, form validations, and AJAX requests. Ajax plays a crucial role in asynchronous data retrieval from the server without requiring a full page reload, enhancing the responsiveness of our application.

3.4.4) Backend

On the back-end side, we have chosen Flask as our web framework. Flask is a lightweight and flexible framework written in Python, offering simplicity and ease of use for building web applications. It provides a robust foundation for handling HTTP requests, routing, and managing application logic. With Flask, we can efficiently process user requests, fetch data from the database, and generate personalized movie recommendations based on the collaborative filtering algorithm.

3.4.5) Database

We have opted for MongoDB as our choice of the database management system. MongoDB is a NoSQL database that stores data in a flexible, document-oriented format, making it well-suited for handling large amounts of unstructured data, such as movie information and user preferences. It allows us to store and retrieve data easily, perform complex queries, and handle scalability as our user base grows.

CHAPTER 4: RESULTS

4.1) METRICS USED

There are several metrics to choose from to evaluate our performance. We consider the following metrics, and we describe the reasons for choosing them.

RMSE (Root Mean Square Error) is a widely used metric for assessing the performance of recommendation systems. Its purpose is to determine the average magnitude of the differences between predicted and actual values.
RMSE is determined using following formula shown in Eq. (3):

$$RMSE = \sqrt{\frac{1}{n}\sum_{(u_i,v_j)\in TestSet}(r_{i,j} - estimated\_rating)^2}. \tag{3}$$

where $r_{i,j}$ denotes the actual rating of the item and n is the number of ratings in the test set.
This metric offers insight into how well the model's predictions align with the true values, where lower RMSE values indicate higher accuracy.

RMSE proves particularly valuable in the context of recommendation systems as it enables quantification of the errors made by the model when predicting user preferences or ratings for items. By squaring the differences, larger errors are penalized more heavily, resulting in a comprehensive evaluation of the overall prediction quality.

An advantage of RMSE is its interpretability in the same unit as the predicted and actual values. For example, if a recommendation system is predicting movie ratings on a scale of 1 to 5, the RMSE value will also fall within the range of 1 to 5.

NDCG stands for **Normalized Discounted Cumulative Gain** and is commonly used to evaluate the effectiveness of recommendation systems in ranking items. It measures the quality of the ranked list of recommended items based on their relevance to the user.

NDCG takes into account both the relevance of the recommended items and their positions in the ranking. The positions of the items in the ranking are weighted such that higher-ranked items have a greater impact on the overall NDCG score.
The NDCG score ranges from 0 to 1, with 1 indicating the perfect ranking where all the relevant items are at the top positions.

NDCG is calculated with the help of DCG and IDCG. DCG is evaluated as shown in Eq. (4) :

$$DCG_u = \sum_{i=1}^{N} \frac{r_{u,i}}{log_2(i+1)} \tag{4}$$

where $r_{u,i}$ denotes the actual rating for the item appearing at position i for a particular user u, and N denotes the number of items.

Since user behavior determines how ratings are distributed, there is no direct comparison between the DCG scores for various people. As a result, the total gain for each user needs to be normalized. To do this, the optimal DCG for user u, designated as $IDCG_u$, is calculated.
The overall NDCG is computed by averaging $NDCG_u$ across all users as shown in Eq. (5).

$$NDCG_u = \frac{DCG_u}{IDCG_u} \tag{5}$$

Precision is a metric commonly used to evaluate the accuracy of recommendation systems. It measures the proportion of recommended items that are relevant to the user's interests out of all the recommended items. Precision provides insights into the quality of the recommendations in terms of relevance.
To calculate precision, we need to determine what is considered a "relevant" item based on user feedback or other relevance criteria. For example, in our case, if the user provides ratings or feedback on items, we can consider items with high ratings or

positive feedback as relevant. Once we have defined relevance, we can calculate precision using the Eq. (6)

$$\text{Precision}=\frac{\sum \text{Relevant items}}{\sum \text{All items}} \tag{6}$$

Precision value of 0.6 means that out of all the items recommended by the system, 60% of them are relevant to the user's interests.

Recall quantifies the ratio of relevant items that have been accurately retrieved or recommended out of the total number of relevant items available in the system.Recall does not consider the number of irrelevant items that might have also been recommended. It solely focuses on the ability of the system to retrieve relevant items, without penalizing false positives (irrelevant items that are recommended). Recall is calculated as shown in Eq. (7).

$$r = \frac{\#\text{ of our recommendations that are relevant}}{\#\text{ of all the possible relevant items}} \tag{7}$$

High recall indicates that a significant portion of movies that a user would enjoy have been suggested.

F measure is a metric that combines precision and recall into a single measure to evaluate the performance of recommendation systems. It provides a balanced assessment of both precision and recall by considering their harmonic mean.
The F-measure is calculated using the formula mentioned in Eq. (8).

$$F1 = 2.\,\frac{Precision\ \times Recall}{Precision + Recall} \tag{8}$$

The F-measure is particularly useful when there is an uneven distribution between relevant and irrelevant items, as it takes into account both false positives (irrelevant items recommended) and false negatives (relevant items not recommended). It provides a comprehensive evaluation of the system's ability to recommend relevant items while minimising irrelevant ones.

4.2) HYPER-PARAMETER TUNING

Hyper-parameter tuning aims to optimize the performance of the system by fine-tuning the values of specific parameters. The proposed method involves two kinds of parameter tuning. Initially the collaborative filtering (CF) model is trained and tuned. To find the best hyper-parameter set, the collaborative filtering recommender is trained on all the train folds individually. The hyperparameters corresponding to the best average recommendation quality across all folds, measured using Mean Average Precision (MAP) are selected. Secondly, the hyper-parameters of the feature weighting component are tuned in a similar manner. This enables the selection of hyper-parameter values that result in the best recommendation quality for the feature weighting model.

4.2.1) COLLABORATIVE FILTERING

It includes various hyper-parameters like alpha, beta, topK and normalize_similarity which drastically affects the performance of our model.

- **Influence Power (Alpha)**: The 'alpha' parameter determines the weight assigned to the user-item interactions when calculating the item-item similarity matrix. It essentially controls the influence of explicit user-item ratings or interactions in the recommendation process. A higher value of alpha (close to 1) places greater importance on the user-item interactions, indicating that the recommender system heavily relies on the ratings given by users to different movies. This can be useful when user ratings accurately reflect their preferences and when the dataset contains a significant number of explicit ratings. On the other hand, a lower value of alpha means that the recommender system focuses more on identifying similarities between movies based on the underlying patterns in the data, such as genre, director, or actor information.

- **Popularity Penalty (Beta)**: The 'beta' parameter is utilized to calculate the degree of each item in the item-user interaction matrix (degree refers to a measure of the strength or importance of each item). By taking the sum of co-occurrences for each item and applying the required formula, the impact of popular items is penalized.The 'beta' parameter controls the decay rate of the degree values. A higher value of 'beta' (closer to 1.0) leads to a slower decay, implying that even items with fewer co-occurrences can maintain a relatively higher degree. On the other hand, a lower value of 'beta' accelerates the decay, meaning that items with fewer co-occurrences will have a lower degree.By adjusting the 'beta' parameter, we can control the influence of popular items in the recommendation process. Penalizing top popular items helps diversify the recommendations and reduces the tendency to excessively recommend frequently chosen items.

4.2.2) FEATURE WEIGHTING

The key hyperparameters used in feature weighting include learning rate, dropout_perc, l1 and l2 regularization and topK.

- **Dropout Percentage (dropout_perc)**: The 'dropout_perc' parameter is used to control the dropout rate during training.It specifies the fraction or percentage of connections that will be randomly set to zero during each training iteration. Tuning the 'dropout_perc' parameter allows us to control the amount of dropout regularization applied during training. By setting the value of dropout_perc to 0.7, we were able to achieve the best results. Higher values, such as 0.7 , indicate a higher dropout rate, where 70% of the connections will be randomly dropped during each training iteration. This is useful when dealing with complex models or datasets with high dimensionality, as it helps prevent overfitting and improves the robustness of the learned feature weights.

- **Gradient Descent Variant (sgd_mode):** The 'sgd_mode' parameter is used to specify the mode of the stochastic gradient descent (SGD) optimization algorithm used during the training process. The choice of the SGD mode can impact the training process and convergence speed of the recommender system. Some of the SGD modes that are available include 'sgd' (standard stochastic gradient descent), 'adagrad' (AdaGrad optimization), 'rmsprop' (RMSprop optimization). Different variants of SGD have distinct characteristics in terms of learning rate adaptation, handling of gradients, and convergence behavior. Our model gives best performance when sgd mode is set to ‘Adam’.

- **Learning rate , l1_reg and l2_reg:** The 'learning_rate' parameter determines the step size or rate at which the model's parameters are updated during optimization. We choose the 'learning_rate' as 0.0001 to guarantee a stable and controlled convergence during the training process. The 'l1_reg' and 'l2_reg' parameter represents the strength or weight of L1 regularization and L2 regularization applied during the training respectively.The former encourages sparsity in the learned feature weights by driving some weights to exactly zero while the latter encourages the learned feature weights to be small but non-zero. They help us prevent overfitting and improve the model's generalization capability.

- **TopK:** The 'topK' parameter determines the number of most similar items to consider during the recommendation process. By setting 'topK' to a specific value, the recommender system focuses on the most similar items, effectively limiting the number of items to be considered for generating recommendations. The choice of 'topK' can influence the diversity and coverage of the recommendations. Higher values of 'topK' leads to more diverse recommendations, while lower values prioritize accuracy by focusing on the most similar items.

4.3) COMPARING RESULTS

We compare the RMSE values of our proposed feature weighting method to those of many other current recommendation approaches in order to assess its performance. The comparison includes two model-based methods as well as two memory-based methods and conventional user-based collaborative filtering (CF). Table 4.1 lists the conclusions of this comparison.

TABLE 4.1: Comparison of RMSE values

Algorithm	**RMSE**
User-based [26]	0.993
NMF [23]	1.127
PLSA [25]	1.093
RSM [24]	1.016
ACF [6]	1.008
Feature Weighting (our approach)	0.998

In order to evaluate the performance of different recommendation algorithms on the MovieLens-1M dataset, we compare their precision, recall, and F-measure values. We gather the top 10 items that have been recommended for each user to calculate these metrics.We compare our approach with different algorithms as presented in Table 4.2.

TABLE 4.2: Comparison of the F-measure, precision, and recall for the DR, TDM, and other recommendation algorithms.

Algorithm	**Precision**	**Recall**	**F-measure**
Item-CF [26]	8.25%	5.66%	5.29%
TDM [12]	14.06%	10.55%	9.49%
Youtube DNN [20]	11.87%	8.71%	7.96%
DR [3]	20.58%	10.89%	12.32%
Feature Weighting (our approach)	15.35%	10.73%	9.57%

For comparing NDCG values, we assess the values across the top 3 recommended items to each user ie. NDCG@3. We evaluated it over two scenarios : existing users for warm items and cold users for warm items as presented in table 4.3.

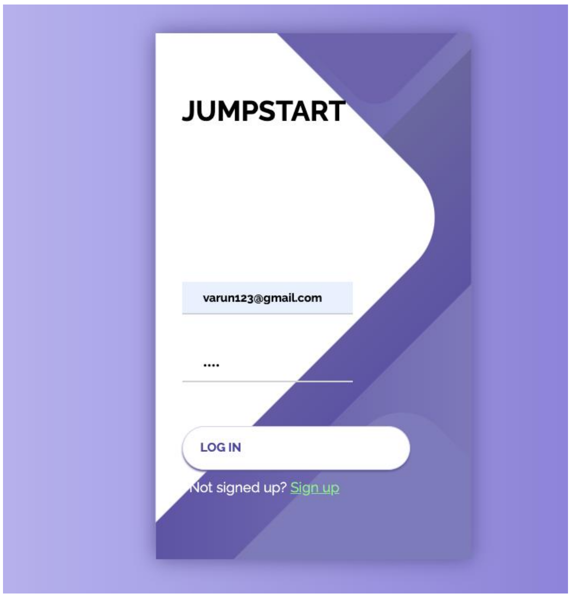
TABLE 4.3: Comparison of NDCG values

Algorithm	**Cold users**	**Warm users**
MeLU [10]	0.8810	0.8904
MetaCS-DNN [9]	0.8559	0.9090
Item-level RUM [26]	0.7420	0.9102
Feature-level RUM [16]	0.7642	0.9120
MAMO [4]	0.8967	0.8866
Feature Weighting (our approach)	0.8832	0.9093

4.4) USER INTERFACE

The system provides an intuitive and user-friendly interface that allows users to easily navigate and interact with the system.

Figure 4.1 shows the login page, where the credentials needed are username and password. On a successful login, the user is landed to the home page, as shown in figure 4.2. It shows the list of recommended movies to the users. The users can also provide feedback to these movies in the form of like and dislike.

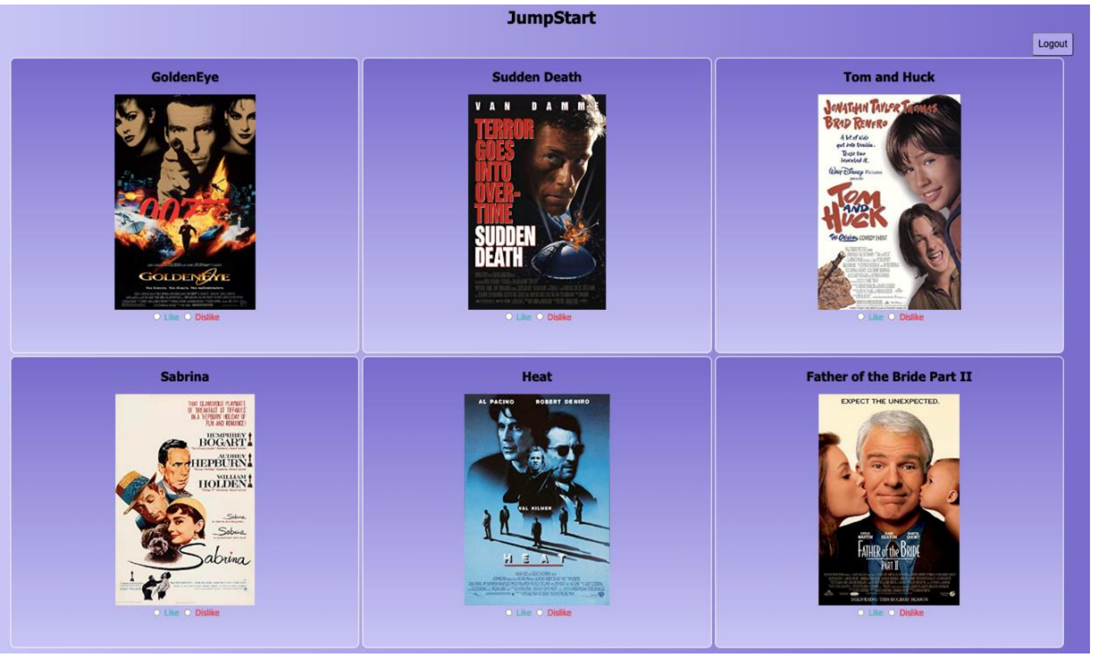


JUMPSTART

varun123@gmail.com

....

Figure 4.1: Screenshot of login Page



JumpStart

GoldenEye ○ Like ○ Dislike

Sudden Death ○ Like ○ Dislike

Tom and Huck ○ Like ○ Dislike

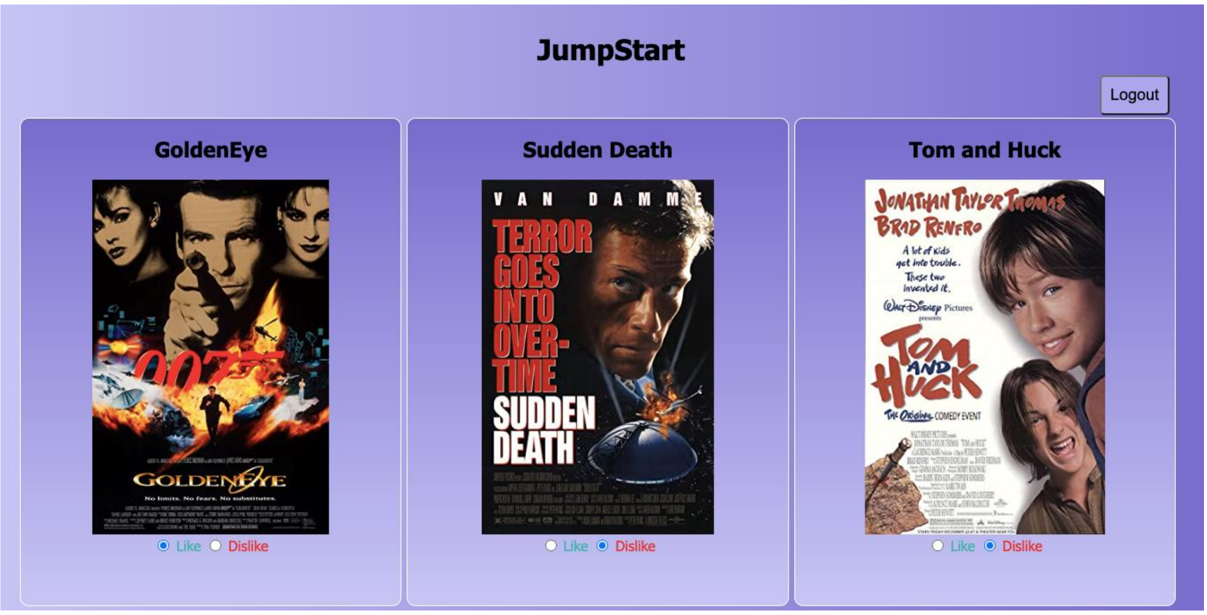
Sabrina ○ Like ○ Dislike

Heat ○ Like ○ Dislike

Father of the Bride Part II ○ Like ○ Dislike

Figure 4.2: Screenshot of the interface as user logins for the first time

We provide some feedback to particular movies, as shown in figure 4.3. The feedback is sent to the backend using AJAX and Flask and the recommended movie list is updated dynamically based on the feedback provided by the user. The final list of recommendation systems is hence displayed to the user, as shown in Figure 4.4.

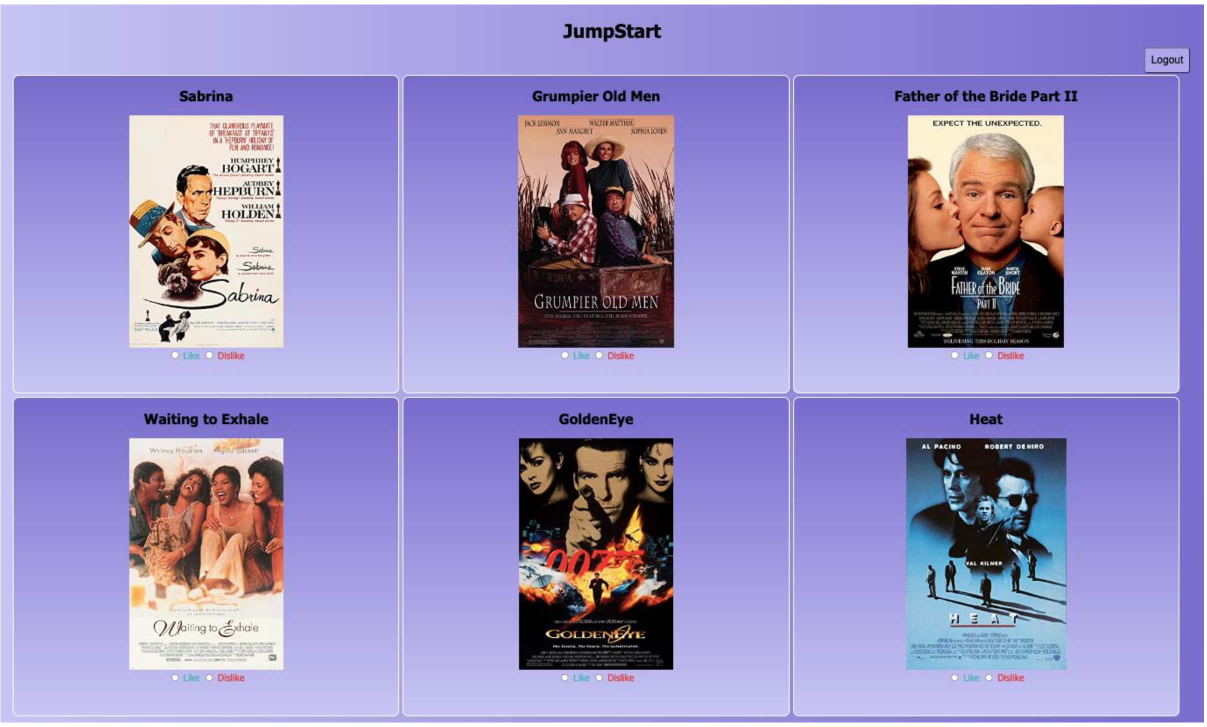


GoldenEye ● Like ○ Dislike

Sudden Death ○ Like ● Dislike

Tom and Huck ○ Like ● Dislike

Figure 4.3: Screenshot of user giving feedback on the movies.



Sabrina ○ Like ○ Dislike

Grumpier Old Men ○ Like ○ Dislike

Father of the Bride Part II ○ Like ○ Dislike

Waiting to Exhale ○ Like ○ Dislike

GoldenEye ○ Like ○ Dislike

Heat ○ Like ○ Dislike

Figure 4.4: Screenshot of the interface showing movies recommended to the user.

CHAPTER 5: CONCLUSION AND FUTURE WORK

5.1) CONCLUSION

The cold-start problem, which occurs when a recommender lacks enough data to provide a useful recommendation to the user, must be handled by recommender systems in order for them to function seamlessly in dynamic situations.

In this report, we proposed a hybrid approach of collaborative and content-based filtering with emphasis on feature weighting. It had many advantages over other hybrid approaches like cascade hybrid techniques as well as the pure collaborative filtering methods that most state-of-the-art algorithms use. It also had benefits over neural snetwork-based techniques like computational efficiency, performance with sparse data, and interpretability.

We discovered the existence of human bias towards some specific qualities in the items in ratings after conducting a study on the nature of item attributes and ratings. Another thing we saw was that recommendations that are based on user-item interactions (collaborative filtering) as opposed to item features (content-based filtering) are more effective. With the help of the two aforementioned facts, we modified our feature-based algorithm using feature weighting in order to estimate and incorporate the output of the user interaction-based algorithm into the same model. In order to train the feature weights for each of the features in items, we created a machine learning model employing complex techniques and extensive training. To optimize training time and the outcomes, we applied the concept of early stopping and engaged in hyperparameter tuning. The final step is to offer the suggested list of items using these learned feature weights.

We also created a web application where these recommended movies are displayed, and the user feedback is incorporated in the model whenever the recommended feed is refreshed.

5.2) FUTURE SCOPE

We integrated our model with the application, where the movies are displayed as per the recommended list of items provided by the users. We can improve our applications by incorporating some personalization features described below:

1. Context-Aware Recommendations: Recommender systems have the system to become more context-aware, taking into account factors such as time of day and weather to make more personalized recommendations.
2. Multi-Modal Recommendations: Our system can be extended to take into account multiple sources of data, such as text, images, and videos.
3. Social Integration: Integrating social features into the recommendation system can enhance the discovery process for new users. This can involve leveraging social media platforms, allowing users to connect with their friends or influencers, and incorporating social recommendations or reviews. By leveraging social connections and trusted sources, the system can help new users discover movies recommended by people they trust.
4. User Engagement Features: The user interface for new users can include engaging features to encourage exploration and discovery. This can include curated movie lists, thematic collections, or personalized movie quizzes to help users discover movies aligned with their interests. By fostering user engagement and providing an enjoyable experience, the system can increase the likelihood of converting cold users into warm users.
5. Real-time Updates: The system can be designed to provide real-time updates for existing users. By continuously monitoring and analyzing user behavior, the system can adapt to their evolving preferences and interests more effectively. This can involve incorporating real-time data streams, such as user interactions with the platform, social media data, or trending movie discussions, to provide the most up-to-date recommendations.

REFERENCES

[1] Rahman, Md. Mijanur & Shama, Ismat & Rahman, Siamur & Nabil, Rahmatullah. (2022). HYBRID RECOMMENDATION SYSTEM TO SOLVE COLD START PROBLEM. Journal of Theoretical and Applied Information Technology. 100.

[2] Weimann, Kuba & Conrad, Tim. (2022). Forget Embedding Layers: Representation Learning for Cold-start in Recommender Systems.

[3] Weihao Gao† , Xiangjun Fan† , Chong Wang† , Jiankai Sun, Kai Jia, Wenzhi Xiao, Ruofan Ding, Xingyan Bin, Hui Yang, Xiaobing Liu. 2021. Deep Retrieval: Learning A Retrievable Structure for Large-Scale Recommendations.

[4] Dong, M., Yuan, F., Yao, L., Xu, X., & Zhu, L. (2020). MAMO: Memory-Augmented Meta-Optimization for Cold-start Recommendation. Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining.

[5] J. Zhang, Y. Wang, Z. Yuan and Q. Jin, "Personalized real-time movie recommendation system: Practical prototype and evaluation," in Tsinghua Science and Technology, vol. 25, no. 2, pp. 180-191, April 2020, doi: 10.26599/TST.2018.9010118.

[6] Liao, X., Wu, H., Wang, Y.: Ant collaborative filtering addressing sparsity and temporal effects. IEEE Access 8, 32783–32791 (2020)

[7] Tahmasebi, F., Meghdadi, M., Ahmadian, S., & Valiallahi, K. (2020). A hybrid recommendation system based on profile expansion technique to alleviate cold start problem. Multimedia Tools and Applications, 80, 2339-2354.

[8] Deldjoo, Y., Dacrema, M.F., Constantin, M.G. et al. Movie genome: alleviating new item cold start in movie recommendation. User Model User-Adap Inter 29, 291–343 (2019).

[9] Homanga Bharadhwaj. 2019. Meta-Learning for User Cold-Start Recommendation. In 2019 International Joint Conference on Neural Networks (IJCNN). IEEE, 1–8.

[10] Hoyeop Lee, Jinbae Im, Seongwon Jang, Hyunsouk Cho, and Sehee Chung. 2019. MeLU: Meta-Learned User Preference Estimator for Cold-Start Recommendation. In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. ACM, 1073–1082

[11] Geyik, Sahin & Guo, Qi & Hu, Bo & Ozcaglar, Cagri & Thakkar, Ketan & Wu, Xianren & Kenthapadi, Krishnaram. (2018). Talent Search and Recommendation

Systems at LinkedIn: Practical Challenges and Lessons Learned. 1353-1354. 10.1145/3209978.3210205.

[12] Han Zhu, Xiang Li, Pengye Zhang, Guozheng Li, Jie He, Han Li, and Kun Gai. 2018. Learning tree-based deep model for recommender systems. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 1079–1088.

[13] Hou, L., Pan, X. & Liu, K. Balancing the popularity bias of object similarities for personalised recommendation. Eur. Phys. J. B 91, 47 (2018).

[14] Kang, W.-C. and McAuley, J. Self-attentive sequential recommendation. In Proceedings of the IEEE International Conference on Data Mining (ICDM '18), pp. 197–206. IEEE, 2018.

[15] Wang Hongwei, Zhang Fuzheng, Wang Jialin, Zhao Miao, Li Wenjie, Xie Xing, and Guo Minyi. 2018. RippleNet: Propagating user preferences on the knowledge graph for recommender systems. In Proceedings of the 27th ACM International Conference on Information and Knowledge Management.

[16] Xu Chen, Hongteng Xu, Yongfeng Zhang, Jiaxi Tang, Yixin Cao, Zheng Qin, and Hongyuan Zha. 2018. Sequential recommendation with user memory networks. In Proceedings of the eleventh ACM international conference on web search and data mining. ACM, 108–116.

[17] Y. Kumar, et al., "IceBreaker: Solving Cold Start Problem for Video Recommendation Engines," in 2018 IEEE International Symposium on Multimedia (ISM), Taichung, Taiwan, 2018 pp. 217-222.

[18] Maksims Volkovs, Guang Wei Yu, and Tomi Poutanen. 2017. DropoutNet: Addressing Cold Start in Recommender Systems. In Advances in Neural Information Processing Systems, Vol. 30. Long Beach, CA, USA, 4957--4966.

[19] Frolov, E., & Oseledets, I. (2016). Fifty Shades of ratings: How to Benefit from a Negative Feedback in top-n Recommendations Tasks. In Proc. of the 10th ACM conf. on recommender systems (pp. 91–98). USA: ACM, New York, NY.

[20] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In Proceedings of the 10th ACM conference on recommender systems. 191–198.

[21] Maciej Kula. 2015. Metadata Embeddings for User and Item Cold-start Recommendations. In CBRecSys@RecSys 2015.

[22] Wang, Y., Liao, X., Wu, H., & Wu, J. (2012). Incremental Collaborative Filtering Considering Temporal Effects. ArXiv, abs/1203.5415.

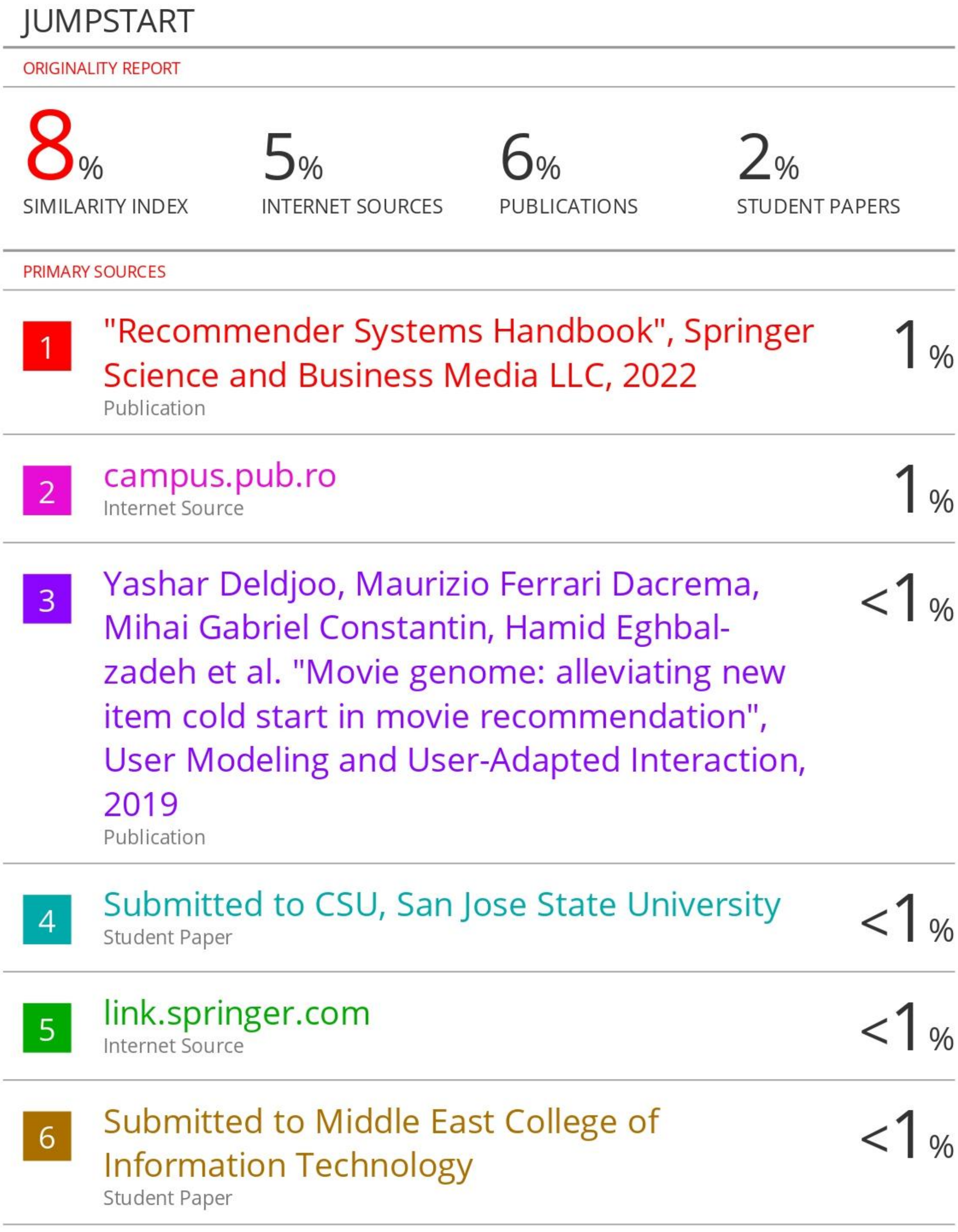
[23] S. Zhang, W.H. Wang, J. Ford, F. Makedon. Learning from Incomplete Ratings Using Nonnegative Matrix Factorization. In 6th SIAM Conference on Data Mining : 548-552, 2006.

[24] P. Han, B. Xie, F. Yang, and R.M. Shen. An Adaptive Spreading Activation Scheme for Performing More Effective Collaborative Recommendation. In Proceeding of International conference on database and expert systems applications: 95-104, 2005

[25] T. Hofmann. Latent Semantic Models for Collaborative Filtering. In ACM Transaction Information System 22(1): 89-115, 2004.

[26] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In Proceedings of the 10th WWW Conference: 285-295, 2001.

PLAGIARISM REPORT



JUMPSTART

ORIGINALITY REPORT

8%	5%	6%	2%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	"Recommender Systems Handbook", Springer Science and Business Media LLC, 2022 Publication	1%
2	campus.pub.ro Internet Source	1%
3	Yashar Deldjoo, Maurizio Ferrari Dacrema, Mihai Gabriel Constantin, Hamid Eghbal-zadeh et al. "Movie genome: alleviating new item cold start in movie recommendation", User Modeling and User-Adapted Interaction, 2019 Publication	<1%
4	Submitted to CSU, San Jose State University Student Paper	<1%
5	link.springer.com Internet Source	<1%
6	Submitted to Middle East College of Information Technology Student Paper	<1%

7	"Advances in Knowledge Discovery and Data Mining", Springer Science and Business Media LLC, 2017 Publication	<1 %
8	www.hindawi.com Internet Source	<1 %
9	www.researchgate.net Internet Source	<1 %
10	Submitted to Clarkston Community Schools Student Paper	<1 %
11	Submitted to Liverpool John Moores University Student Paper	<1 %
12	Submitted to Middle East Technical University Student Paper	<1 %
13	deepai.org Internet Source	<1 %
14	vdoc.pub Internet Source	<1 %
15	ebin.pub Internet Source	<1 %
16	www.arxiv-vanity.com Internet Source	<1 %
17	Submitted to Rutgers University, New Brunswick	<1 %

	Student Paper	
18	"Knowledge Science, Engineering and Management", Springer Science and Business Media LLC, 2019 Publication	<1 %
19	"ECAI 2020", IOS Press, 2020 Publication	<1 %
20	docs.recombee.com Internet Source	<1 %
21	www.slideshare.net Internet Source	<1 %
22	Amine Boulemtafes, Abdelouahid Derhab, Yacine Challal. "A review of privacy-preserving techniques for deep learning", Neurocomputing, 2020 Publication	<1 %
23	www.mdpi.com Internet Source	<1 %
24	"Recommender System with Machine Learning and Artificial Intelligence", Wiley, 2020 Publication	<1 %
25	Rim Fakhfakh, Anis Ben, Chokri Ben. "Deep Learning-Based Recommendation: Current Issues and Challenges", International Journal	<1 %

	of Advanced Computer Science and Applications, 2017 Publication	
26	github.com Internet Source	<1 %
27	"Foundations of Intelligent Systems", Springer Science and Business Media LLC, 2014 Publication	<1 %
28	Submitted to Griffith College Dublin Student Paper	<1 %
29	arxiv.org Internet Source	<1 %
30	export.arxiv.org Internet Source	<1 %
31	Submitted to Morgan Park High School Student Paper	<1 %
32	Mukund Deshpande. "Item-based top-*N* recommendation algorithms", ACM Transactions on Information Systems, 1/1/2004 Publication	<1 %
33	Submitted to Brisbane Catholic Education Student Paper	<1 %
34	Submitted to Institute of Technology, Nirma University Student Paper	<1 %

35	Rahmat Ullah, Yinhuan Dong, Tughrul Arslan, Siddharthan Chandran. "A Machine Learning-Based Classification Method for Monitoring Alzheimer’s Disease Using Electromagnetic Radar Data", IEEE Transactions on Microwave Theory and Techniques, 2023 Publication	<1 %
36	pure.rug.nl Internet Source	<1 %
37	Submitted to University of Durham Student Paper	<1 %
38	Submitted to Unviersidad de Granada Student Paper	<1 %
39	eprints.gouni.edu.ng Internet Source	<1 %
40	www.theseus.fi Internet Source	<1 %
41	Emre Yalcin, Alper Bilge. "Evaluating unfairness of popularity bias in recommender systems: A comprehensive user-centric analysis", Information Processing & Management, 2022 Publication	<1 %
42	R. Lavanya, Utkarsh Singh, Vibhor Tyagi. "A Comprehensive Survey on Movie Recommendation Systems", 2021	<1 %

	International Conference on Artificial Intelligence and Smart Systems (ICAIS), 2021 Publication	
43	kth.diva-portal.org Internet Source	<1 %
44	towardsdatascience.com Internet Source	<1 %
45	"Hybrid Artificial Intelligent Systems", Springer Science and Business Media LLC, 2019 Publication	<1 %
46	Jenq-Haur Wang, Yi-Hao Chen. "A Distributed Hybrid Recommendation Framework to Address the New-User Cold-Start Problem", 2015 IEEE 12th Intl Conf on Ubiquitous Intelligence and Computing and 2015 IEEE 12th Intl Conf on Autonomic and Trusted Computing and 2015 IEEE 15th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom), 2015 Publication	<1 %
47	Submitted to University of Birmingham Student Paper	<1 %
48	Vito Bellini, Vito Walter Anelli, Tommaso Di Noia, Eugenio Di Sciascio. "Auto-Encoding User Ratings via Knowledge Graphs in Recommendation Scenarios", Proceedings of	<1 %

	the 2nd Workshop on Deep Learning for Recommender Systems - DLRS 2017, 2017 Publication	
49	dr.ntu.edu.sg Internet Source	<1 %
50	file.allitebooks.com Internet Source	<1 %
51	s3-ap-southeast-1.amazonaws.com Internet Source	<1 %
52	soar.wichita.edu Internet Source	<1 %
53	Azhar Yousf Mir, Majid Zaman, Syed Mohammad Khurshid Quadri, Sheikh Amir Fayaz. "An Adaptive Classification Framework for Handling the Cold Start Problem in Case of News Items", Revue d'Intelligence Artificielle, 2022 Publication	<1 %
54	Han, Shuli, Yujiu Yang, and Wenhuang Liu. "Incremental Learning for Dynamic Collaborative Filtering", Journal of Software, 2011. Publication	<1 %
55	Sang-Min Choi, Dongwoo Lee, Chihyun Park. "On the Smaller Number of Inputs for Determining User Preferences in Recommender Systems", Mathematics, 2020	<1 %

Exclude quotes	On	Exclude matches	Off
Exclude bibliography	On		