Scalable Recommendation System using Hybrid Approach: Content and Collaborative Filtering

Project report submitted to
Visvesvaraya National Institute of Technology, Nagpur
In partial fulfillment of the requirements for the award of the degree

Bachelor of Technology
In
Computer Science and Engineering

Chaitya Chheda	**BT16CSE016**
Ayush Singh	**BT16CSE098**
Deval Mudia	**BT16CSE060**
Sadneya Pusalkar	**BT16CSE076**

under the guidance of
Dr. S.R. Sathe

Department of Computer Science and Engineering
Visvesvaraya National Institute of Technology
Nagpur 440 010 (India)

2020

Department of Computer Science and Engineering
Visvesvaraya National Institute of Technology,
Nagpur

DECLARATION

We hereby declare that this project work titled **Scalable Recommendation System using Hybrid Approach: Content and Collaborative Filtering** is carried out by us in the Department of Computer Science and Engineering of Visvesvaraya National Institute of Technology, Nagpur under the guidance of **Dr. S.R. Sathe**. The work is original and has not been submitted earlier whole or in part for the award of any degree/diploma at this or any other Institution/University.

Sr.No.	**Enrollment No**	**Names**	**Signature**
1	BT16CSE016	Chaitya Chheda	[illegible]
2	BT16CSE098	Ayush Singh	[illegible]
3	BT16CSE060	Deval Mudia	[illegible]
4	BT16CSE076	Sadneya Pusalkar	[illegible]

Date: 5th July 2020

Department of Computer Science and Engineering
Visvesvaraya National Institute of Technology,
Nagpur

CERTIFICATE

This is to certify that the project work entitled- "**Scalable Recommendation System using Hybrid Approach: Content and Collaborative Filtering**", is a bonafide work done by Mr. Chaitya Chheda, Ms. Sadneya Pusalkar, Mr. Deval Mudia and Mr. Ayush Singh under the guidance of Dr. S.R. Sathe in the Department of Computer Science & Engineering, Visvesvaraya National Institute of Technology, Nagpur, for the partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in "**Computer Science & Engineering**". The work is comprehensive, complete, and fit for final evaluation.

[illegible]

(Dr. U.A. Deshpande)
Head of Department
Department of Computer Science
and Engineering, VNIT, Nagpur

SRsathe

(Dr. S.R. Sathe)
Project Guide
Department of Computer Science
and Engineering, VNIT, Nagpur

Date: 5th July 2020

ACKNOWLEDGMENTS

We would like to express our sincere thanks and gratitude to our guide, Dr. S.R. Sathe for giving us the opportunity of working on a project that is widely applied in most online businesses, guiding us thoughtfully and allowing us to work at our own pace.

We would also like to thank all the teaching and non-teaching faculty of the Computer Science and Engineering Department for supporting us and providing necessary facilities at all times.

Last but not least we would like to acknowledge the contribution of those who have constantly supported and encouraged us which helped in the successful completion of our project.

ABSTRACT

A recommendation system assists users in finding desired content from enormous corpora. In a very general way, recommender systems are algorithms aimed at suggesting relevant items to users, to ensure he/she consumes the content of specific interest and relevance.

There exist two significant models of recommendation systems: **collaborative-based** and **content-based** methods. Collaborative filtering systems recommend items to a user that are preferred by other users with similar tastes. Content-based systems examine the properties of the items to recommend other similar items.

However, these methods suffer from inherent drawbacks individually and suffer from common problems like the *cold start problem*: insufficient information about user or item to draw any inferences, *sparsity*: feedback data is sparse and insufficient for identifying neighbors and *popular bias*: the popular and frequently rated items get a lot of exposure while less popular ones are under-represented

In this thesis, we discuss our implementation of a hybrid recommender system that leverages both content and collaborative data. We have chosen the base model for our **scalable collaborative filter** to be Singular Value Decomposition (SVD) since recommender systems based on SVD have proved to be versatile and efficient. However, building the SVD model suffers from being an extremely slow process. Therefore we used an incremental SVD algorithm based on a "fold-in" technique that could append new users so they receive their recommendations on the spot instead of waiting for the model to be rebuilt. For **content filtering**, we represented each item as a set of descriptors or terms from the metadata available and used cosine similarity to calculate a numeric quantity that denotes the similarity between two items. To overcome the shortcomings of both the approaches we have implemented **hybrid filtering** by taking the weighted average of the content recommendations and collaborative recommendation

In the evaluation of this system, we chose predictive accuracy metrics. We evaluated the three approaches using two metrics: Mean Absolute Error (MAE) and Root Mean Square Error (RMSE). The MAE and RMSE values for the hybrid recommender system turned out to be 0.733 and 0.973 respectively. We found that hybrid filtering gave better results than content-based filtering and collaborative filtering.

CONTENTS

List of Figures

List of Tables

Chapter 1

Introduction

Imagine the following scenario, you are currently in the mood for a new movie or TV series to begin watching. However, there isn’t anything particular in mind. So what are the available options? One option is to visit those "Top 10 movies/TV series" blogs or websites and choose one at random to watch. But the problem is that if those movies or series were really popular, you might have already seen them by now since these lists tend to be generic.

Another option could be asking your friends whom you share common interests with. They have always given you good recommendations in the past due to shared interest, so maybe they might recommend something good to you again! That is exactly where the main idea for recommender systems originated from. Basically seeking others with similar interests and formulating personal recommendations tailored to the user based on other like-minded users.

Not only are recommender systems more beneficial to helping the end-user find what he or she requires by giving a pleasurable browsing experience but also they have an extremely high potential for increasing sales by exposing users to items more tailored to their needs. Recommender systems are such a big concern to businesses that even Netflix once held a 1 Million dollar prize for whoever can decrease their error score metric by just 10% [8].

1.1 Motivation

In the age of unlimited digital content and due to the rise of E-commerce businesses, we are no longer restrained to the physical capacity of the shelves but to the receptive capacity of the end-user. For example, while physical book stores may only be able to offer a limited number of books that only their inventory can accommodate, there are online stores such as Amazon that can offer millions of books to their customers. In recent times, the growth of available digital information and the number of internet users has created a challenge of information overload which hinders timely access to items of interest available online. Search engines such as Google have partially solved this problem but prioritization and personalization still remain a pressing issue.

Fortunately, Recommender systems were designed to assist the user in consuming content of specific interest and relevance to the user; hence, recommenders solve the problem of information overload. Recommendation systems help users find and select items from the huge number available on the web or in other electronic information sources. Given a large set of items and a description of the user's needs, they present to the user a small set of items that are well suited to the description. The recommender should suggest the items the user might be interested in. Since, if the recommender system isn’t on par with the user’s expectations then the user will most likely not pay much attention to its recommendations and all future recommendations will be neglected or overlooked. This is why the quality of the recommendation the recommender system produces is vital; however, the recommendation quality is only half of the problem, the other half is the scalability of the system. The recommender system should be able to handle large amounts of data and produce recommendations on the spot for new and existing users, even if the base model consists of millions of user items.

1.2 Project Aim

The focus of this project is to implement a hybrid recommender system that is based on the combination of content-based filtering and collaborative filtering methods. The scalable collaborative filter is based on Singular Value Decomposition (SVD) and supports incremental updates while retaining the recommendation quality. So when new users enter the system they can be "folded in" directly and receive their recommendations instantaneously with good accuracy that is comparable to the base SVD model. Instead of originally waiting until the rebuilding of the whole model from scratch at the specified server time. In content-based filtering each item is represented by a set of features or attributes in the vector space model which characterize that item and use cosine similarity to calculate the similarity scores between the items. The content filtering and collaborative filtering are combined by taking the weighted average of the content recommendation and collaborative recommendation.

The main advantage of the Hybrid Recommender systems is that they combine two or more recommendation techniques to realize performance with fewer of the drawbacks of any of them. The hybrid recommendation system not only provides improved recommendation accuracy but also handles the cold start problem and sparsity of data.

1.3 Organization of Thesis

This Thesis is divided into the following chapters:

Chapter 2, Literature Survey: This chapter outlines and explains the terminologies and concepts for understanding how recommenders function. Any background knowledge or concept needed to understand this thesis is provided in this chapter.

Chapter 3, Methodology and Implementation: This chapter explains the approach and implementation of the hybrid recommender system.

Chapter 4, Results and Evaluation: This chapter introduces the different error metrics and datasets we used for evaluation. It also contains a discussion of our achieved results.

Chapter 5, Conclusion and Future Work: This chapter concludes the thesis and final discussion of the project and includes what additional features and ideas to include and examine in the next iteration of the project.

Chapter 2

Literature Survey

In this chapter, we shall introduce the motivation, terminology, and concepts needed to understand generally how recommender systems work then we will explain how recommendations are made, and we will demonstrate how classic collaborative filtering and content filtering is utilized in order to outline the difference between the classic recommendation technique and our hybrid model.

2.1 Methods of Recommendation

There are multiple ways recommendations could be produced such as:

1. **Hand curated content:** Recommendation content could be hand-curated by human staff. However, this has the problem of putting too much load on the human workforce and might be lacking user feedback hence it won't be personalized.

2. **Aggregate lists:** A simple aggregate list like "top 10" or "most popular items" (like Youtube's trending/most popular videos for example). However, this suffers from being too generic and not relatable to most users, hence those with more refined tastes tend to get overlooked.

3. **Recommender systems:** Recommenders could generate personalized recommendations specifically tailored to the user, just like that friend from chapter 1, the system knows what content the user consumes and what content he or she prefers based on previous feedback and interactions.

2.2 Categorisation of Recommender Systems

Generally, Recommender systems are categorized into three categories based on how recommendations are made:

1. **Content-based filtering:** Makes recommendations based on the user's past preferred item features (meta-data) for example such as text and/or item attributes (no other user ratings are required) from past highly-rated items by the user, usually the advances of information retrieval are utilized here [1, 11].

2. **Collaborative-based filtering:** Makes recommendations based on the user's past ratings in addition to the other user's past ratings to find items of similar interest to the user [9].

    a. **Memory-based:** Utilizes statistical techniques to find neighbors then it computes a prediction derived from the weighted combination of the neighbor's rating [13, 11].

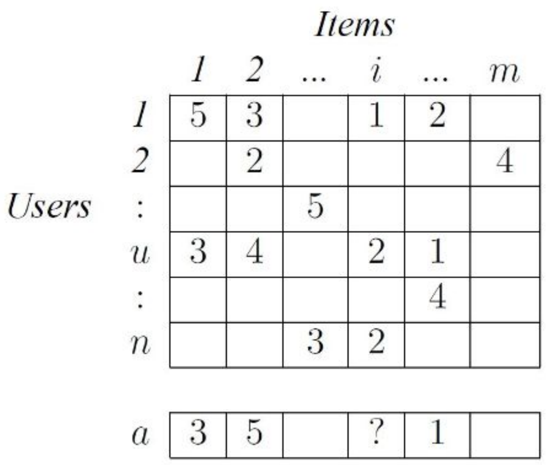
    b. **Model-based:** Utilizes Data mining or machine learning algorithms to create a model and recognize patterns based on pure data to make intelligent predictions based on the learned models [16].

3. **Hybrid:** Combines both of the content-based and collaborative-based approaches [1]. There are several strategies for hybrid recommenders; however, the most popular are ***weighted recommenders*** which combine the scores outputted by different recommender algorithms. Also ***switching recommenders*** which switch between recommender algorithms according to the best-expected results in a particular context [5]. A simple example of switching recommenders is using content-based filtering at the start when there are not enough users or ratings for the user but then switching to collaborative filtering when there are enough users or ratings for the user.

2.3 Utility Matrix

Recommender systems have been usually studied under their most common form of "rating prediction" [11, 1] where the recommendation problem has been simplified to just predicting ratings for items not seen/rated by the user, if the ratings are high enough, then the items get recommended to the user based on those predicted ratings [1].

The corresponding rating data is represented in the form of a "utility matrix" which is a user-item pair containing values from an ordered set (ex: integers from a certain range representing stars given) denoting the degree of the user's preference to that item [1, 9], it can be represented formally, Utility matrix *u: users* × *items* → *ratings* where users is the set of all users while items is the set of all possible items that can be recommended, and ratings are the values coming from an ordered set (values of a specific range) [1]. An example utility matrix is shown in Figure 2.1.



		Items					
		1	*2*	...	*i*	...	*m*
	1	5	3		1	2	
	2		2				4
Users	:			5			
	u	3	4		2	1	
	:					4	
	n			3	2		
	a	3	5		?	1	

Figure 2.1: The utility matrix which consists of utility matrix *u: user* × *item* → *ratings* where each cell r_{ui} corresponds to the rating of user u for item i, and it is required to find the rating of current user a which is r_{ai} [11]

As can be seen from Figure 2.1 not all items are rated by most users hence the matrix can be called "sparse" meaning most entries are "unknown" (there are blanks in the matrix) [9, 11], which formally means the utility u is not defined on the whole *users* × *items* space but only a subset of it [1]. These "not rated" or even "unseen" items form the pure essence of the recommender system problem, where we must "extrapolate" [1] or predict the missing ratings then use the predicted ratings or utility to formulate a list of items that if the user saw them would possibly give them a close high rating. Thus the recommender system problem is to predict the missing ratings; however, predicting every blank entry is not the main objective of a recommender, the main objective is finding the few highly valued items to the user and recommending them [9].

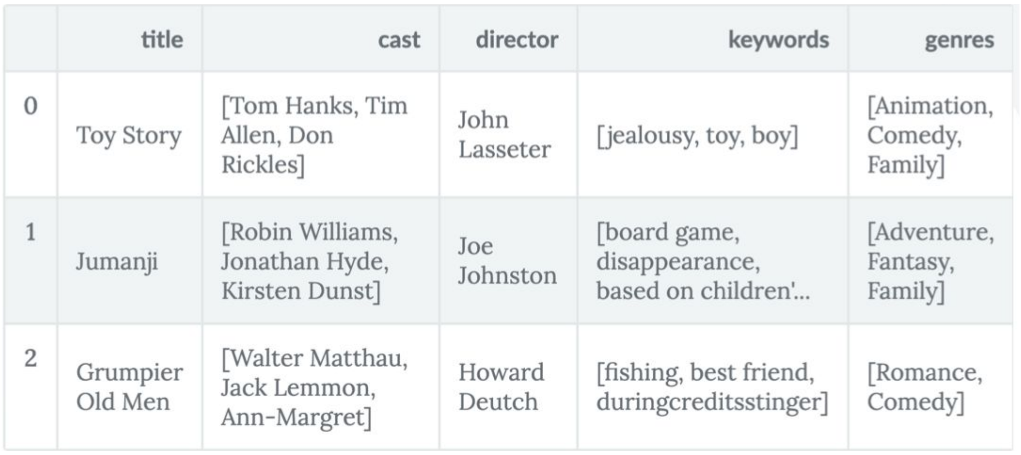
2.3.1 Collecting data for utility matrix

As can be deduced from the above section 2.3 for recommender systems to be able to recommend an item there must be a utility matrix with some initial user ratings; acquiring data for the utility is out of scope for this thesis; however, for the sake of completeness, it will briefly be mentioned here. Ratings are collected either explicitly or implicitly, for example, some websites explicitly ask users to give ratings, while others can create a certain formula that deduces ratings based on inferences from the user behavior, for example, buying a product or several products of the same type or even watching a movie for a specific duration on certain websites. [9].

2.4 Collecting data for feature representation

Using keywords to model items is an important step for many recommender systems. But extracting keywords of an item is also a difficult problem, especially in the media field, because it is very hard to extract text keywords from a video. Expert **tagging systems** like Pandora for music and Jinni are present for tagging the items. Let's take Jinni as an example, the researchers of Jinni defined more than 900 tags as movie gene, and they let movie experts make tags for them. These tags belong to different categories, including movie genre, plot, crew, cast, credits, and keywords.

Figure 2.2 shows the tags for the movies generated by the experts at MovieLens. As we can see from the figure, the tags from each movie are divided into many categories: Title, Cast, Director, Keywords and Genres. These tags contain all aspects of movie information, which can describe a movie very accurately.



	title	cast	director	keywords	genres
0	Toy Story	[Tom Hanks, Tim Allen, Don Rickles]	John Lasseter	[jealousy, toy, boy]	[Animation, Comedy, Family]
1	Jumanji	[Robin Williams, Jonathan Hyde, Kirsten Dunst]	Joe Johnston	[board game, disappearance, based on children'...	[Adventure, Fantasy, Family]
2	Grumpier Old Men	[Walter Matthau, Jack Lemmon, Ann-Margret]	Howard Deutch	[fishing, best friend, duringcreditsstinger]	[Romance, Comedy]

Figure 2.2: The metadata of the different movies which consists of tags of various categories including title, genres, crew, cast and keywords.

2.5 Content-based filtering

Content-based filtering is deeply based on well-established research in information retrieval and information filtering fields [1]. Since a popular method in content-based filtering is using the item features as the knowledge base and the user's profile or preference as an information retrieval query which then produces a recommendation to satisfy that query [1, 11]. These methods are best suited to situations where there are known data on an item (name, location, description, etc.), but not on the user.

In this system, keywords describe the items and a user profile is generated to suggest the type of item this particular user may like. In a more generic sense, these algorithms attempt to recommend items that are similar to what a user liked in the past, or is examining in the present.

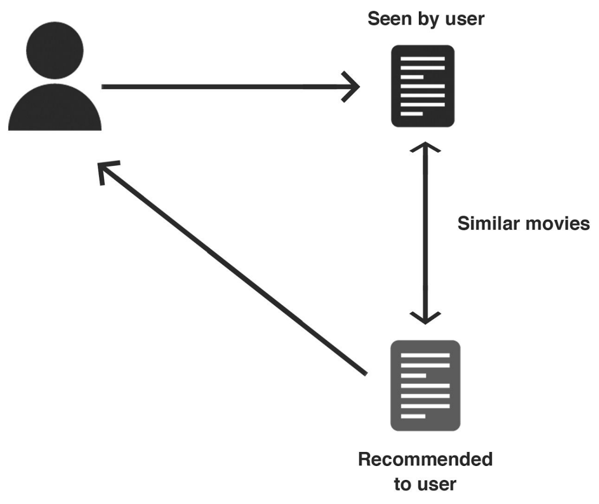


Figure 2.3 Content-based filtering system chooses items similar to those for which the user has already expressed a preference.

Another alternative to using information retrieval for collaborative filtering is treating the problem as a classification task and using one of the popular classification algorithms in machine learning such as naive Bayes classifier, k-nearest neighbor, decision trees [11].

2.5.1 Feature Representation

In content-based filtering, the first step is to identify keywords for representing the items. To avoid indexing useless words, a text retrieval system often associates a stop list with a set of items. The irrelevant words are called stop words (the, of, for, with, etc). There is a requirement that the information retrieval system has to identify groups of words, where groups are small syntactic variants of one another and collect only the common word stem per group. A group of various words may share an equivalent word stem. For example, apple and apples share a common word stem.

2.5.1.1 Vector Space Model

Before being deep into feature representation, it is very important to get the idea of document representation. Assume there are several documents and each document consists of one single sentence. Then we can represent the document as a model in Figure 2.4, which is called the Vector Space Model. Consider each feature of a movie as a term, then a feature can be represented by this model.

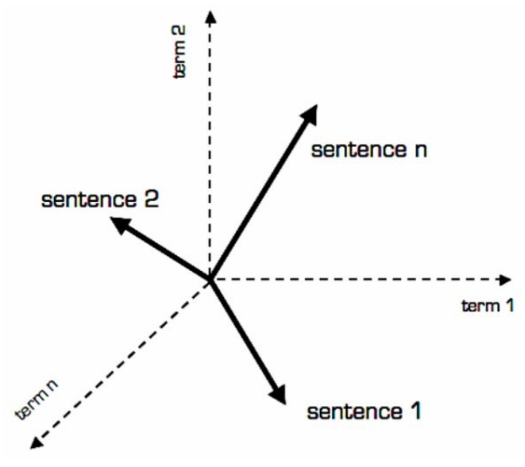


Fig 2.4 Vector Space Model of Documents

2.5.1.2 TF-IDF

TF-IDF, term frequency-inverse document frequency is a score/ weight given to a word that shows how important is a word in a document. The TF-IDF score for a word increases with the increase in the number of times the word appears in a document but decreases with the number of documents in the corpus in which it appears.

TF or term frequency is the number of times a word occurs in a document. IDF or inverse term frequency is the logarithm of the ratio of the total number of documents in the complete corpus to the number of documents in which the word occurs.

2.5.1.3 Term Frequency

Term frequency is about how many times a term t_i appears in document d_j, which can be represented by $TF(t_{ij})$. In the condition of removing stop-words, the more t_i appears in the document, the more important term t_i is for the document. It can be defined as:

$$TF(t_{ij}) = \frac{N(t_{ij},d_j)}{N(d_j)} \tag{2.1}$$

$N(t_i,d_j)$ is the number of times t_i appears in $d_{j,}$ and $N(d_j)$ is the total number of terms in document d_j.

2.5.1.4 Inverse Document Frequency

To understand inverse document frequency, let us see what document frequency is. Document frequency is how many times the term t_i appears in all documents C, which is represented by $N(t_i, C)$. The more term t_i appears in all documents C, the weaker term t_i can represent document d_j.

Inverse document frequency means that the representability of term ti for document d_j and its amount in all documents $N(t_i, C)$ is inverse proportion, which is represented by $IDF(t_i)$:

$$IDF(t_i) = log\frac{N(C)}{N(t_i,C)} \tag{2.2}$$

$N(C)$ is the total amount of documents, $IDF(t_i)$ decreases with the increase of $N(t_i, C)$. The less $N(t_i, C)$ is, the more representative t_i is for d_j.

2.5.2 Similarity

Calculating the similarity between items is the objective of content-based recommender systems, with the assumption that if a user likes a certain item like a movie, news article or a product, he/she may like similar items in the future.

2.5.2.1 Cosine Similarity

The usual similarity metric to compare vectors is the cosine similarity which measures the **cosine** of the angle between two vectors that are projected in a multi-dimensional space.

Using the Euclidean dot product formula the cosine for two non-zero vectors can be derived as:

$$A \cdot B = \|A\|\,\|B\|\, cos \tag{2.3}$$

Given two vectors of attributes, A and B, the cosine similarity, cos(θ), is represented using a dot product and magnitude as:

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{b} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \tag{2.4}$$

Where A_i and B_i are components of vectors A and B respectively.

The range of the similarity is between -1 and 1. -1 means that the direction of the two vectors is opposite and 1 means they are in the same direction. The cosine similarity is 0 if the two vectors have no relationship. For text matching, it is obvious that the weights are non-negative, so the range should be 0 to 1 in our case. Here is an example for illustrating cosine similarity [24].

Given two sentences:

• A: I like watching TV, but I don’t like watching films.

• B: I don’t like watching TV and films.

How can we calculate the similarity between the two sentences? The basic idea is: the more similar the words used by the two sentences are, the more similar the sentences are.

First step: Word segmentation.

• A: I / like / watching / TV, but / I / don't / like / watching / films.
• B: I / don't / like / watching / TV / and / films.

Second step: List all the words.

• I, like, watching, TV, but, don't, films, and

Third step: Word frequency calculation.

• A: I 2, like 2, watching 2, TV 1, but 1, don't 1, films 1, and 0.
• B: I 1, like 1, watching 1, TV 1, but 0, don't 1, films 1, and 1.

Fourth step: Get word frequency vector.

• A: [2, 2, 2, 1, 1, 1, 1, 0]
• B: [1, 1, 1, 1, 0, 1, 1, 1]

Then we can calculate the cosine of the two vectors by Equation 3.15.

$$cos() = \frac{2 \times 1 + 2 \times 1 + 2 \times 1 + 1 \times 1 + 1 \times 0 + 1 \times 1 + 1 \times 1 + 0 \times 1}{\sqrt{2^2 + 2^2 + 2^2 + 1^2 + 1^2 + 1^2 + 1^2 + 0^2} \times \sqrt{1^2 + 1^2 + 1^2 + 1^2 + 0^2 + 1^2 + 1^2 + 1^2}}$$

The value is 0.85 so that the two sentences are much similar.

2.6 Collaborative-based filtering

Collaborative filtering deduces recommendations based on the user's past ratings and other like-minded user's past ratings to find items of similar interest to the user [9, 13]. As stated before, Collaborative filtering is divided into ***Memory-based*** which utilizes statistical techniques for deducing neighborhoods that are used for producing recommendations [13, 11] and ***Model-based*** which utilize data mining and/or machine learning techniques to create the models which are used for producing recommendations [16].

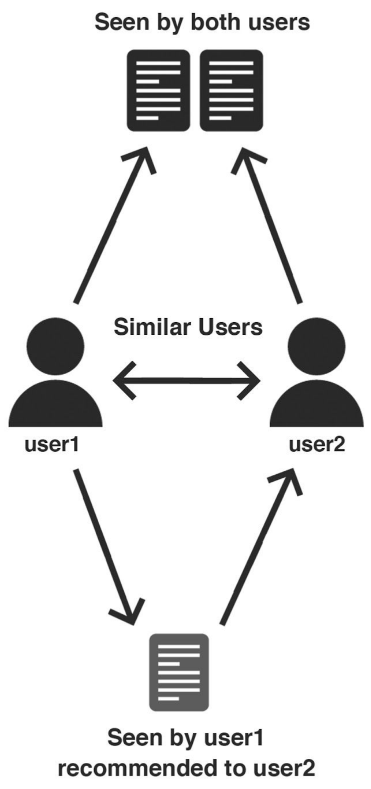


Figure 2.5 Content-based filtering system chooses items similar to those for which the user has already expressed a preference.

2.6.1 Memory-based: K-Nearest Neighbor Collaborative Filtering (k-NN)

The most traditional type of memory-based collaborative filtering is K-Nearest Neighbor; It was the first automated collaborative filtering method and forms the basis to current collaborative filtering recommender systems [5]. Additionally, it is also known as "user-user collaborative filtering" [5], at its base definition, it is simply finding a subset of other users whose past rating behavior is similar to the current user and then use those weighted (based on similarity to the current user) ratings to predict new items that the current user will value [5, 11].

2.6.1.1 Calculating Predictions

To predict ratings, we first need to compute the similarity using a certain similarity function (explained in the next subsection 2.6.1.2) to know how similar certain users are and to be able to gather a neighborhood $N \subseteq U$ where U is the space containing all of our users. Then the predicted rating simply becomes a "weighted average of the neighboring users' ratings which uses the similarity as weights" [5] as formulated in the following:

$$p_{u,i} = \overline{r_u} + \frac{\sum_{v \in N} simi(u,v)(r_{v,i} - \overline{r_v})}{\sum_{v \in N} |simi(u,v)|} \tag{2.5}$$

Where N is the neighbors of user u (a subset of the user base which is from the neighborhood of u) and v is the other user which is neighbor to our current user u, simi(u, v) is the "similarity function" which computes how similar current user u is with user v which adds the weights, i is the current required item's rating, $p_{u,i}$ is the current user's predicted rating for item i[5]. Notice that the average user rating $\overline{r_v}$ gets subtracted from $r_{v,i}$ to somewhat normalize the user's rating bias since a user could be a hard or easy rater, also the $\overline{r_u}$ is being added for the same reason, to compensate the user's rating bias so the predicted rating is close to the real rating [5, 9].

2.6.1.2 Computing Similarity

To be able to compute predictions we first need to determine how similar other users are to our current user, hence we need a "similarity function" to know how similar another user's "taste" is to our current user's, we can use one of several similarity functions which are either based on statistics or linear algebra.

- **Pearson correlation:** Computes the statistical correlation (Pearson's coefficient) the value is between [-1,1] inclusive where -1 is a negative correlation (opposing similarity therefore not similar), 0 no correlation (no similarity) and 1 is a positive correlation (strong similarity)

$$simi(u,v) = \frac{\sum_{i \in I_u \cap I_v} (r_{u,i} - \overline{r_u})(r_{v,i} - \overline{r_v})}{\sqrt{\sum_{i \in I_u \cap I_v} (r_{u,i} - \overline{r_u})^2} \sqrt{\sum_{i \in I_u \cap I_v} (r_{v,i} - \overline{r_v})^2}} \quad (2.6)$$

Where item i is the commonly rated item between user u and user v as denoted formally by $i \in I_u \cap I_v$ [5,11].

- **Cosine similarity:** Measures the cosine distance (The vector's closeness (angle)) [5, 9] since the cosine similarity is based on the cosine's angle, therefore, the cosine can only take values from [-1,1] inclusive where -1 means the vectors are exactly opposite in direction, 0 means vectors are orthogonal to each other, and 1 means the vectors are in the same direction.

$$simi(u,v) = cos(u,v) = cos(\theta_{u,v}) = \frac{r_u \cdot r_v}{\|r_u\| \|r_v\|} = \frac{\sum_i r_{u,i}\, r_{v,i}}{\sqrt{\sum_i r_{u,i}^2} \sqrt{\sum_i r_{v,i}^2}} \quad (2.7)$$

From the below figure 2.6 it can be shown that users can be represented as vectors in a two-dimensional space where each axis represents an item.

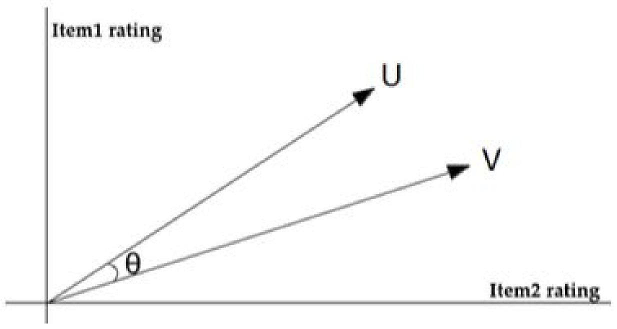


Figure 2.6: user u and user v represented as vectors in a 2d item space with angle θ in between which signifies the cosine distance between those two vectors since the angle translates to similarity (cosine similarity)

2.6.1.3 Computing a movie rating using k-NN

Given *user* × *movie* matrix in figure 2.7 we want to find User C's predicted rating for the movie "Equilibrium" ($p_{c,e}$). We will be using Pearson correlation as our similarity function and a neighborhood of size 2.

	Batman Begins	*Alice in Wonderland*	*Dumb and Dumber*	*Equilibrium*
User A	4	?	3	5
User B	?	5	4	?
User C	5	4	2	?
User D	2	4	?	3
User E	3	4	5	?

Fig 2.7 An Example *user x movie* matrix on a 5 point rating scale [5]

The mean rating of user C is $\overline{r_c}$= 3.667, only two users have rated "Equilibrium" therefore only two candidates for the neighborhood even though some might not be similar to our user but to meet the neighborhood quota we must consider them. simi(c, a)= 0.832 and simi(c, d)=-0.515 as can be seen, d isn't similar to user c but it will be taken into consideration to meet the neighborhood quota; however, it's weight will reflect it's dissimilarity to user c, to get $p_{c,e}$ we can just apply equation 2.1 [5]

$$p_{c,e} = \overline{r_c} + \frac{simi(c,a)(r_{a,e} - \overline{r_a}) + simi(c,d)(r_{d,e} - \overline{r_d})}{|simi(c,a)| + |simi(c,d)|} \tag{2.8}$$

$$= 3.667 + \frac{0.832 \cdot (5-4) + -0.515 \cdot (2-3)}{|0.832| + |0.515|} = 4.667$$

2.6.2 Model-based: Singular Value Decomposition (SVD)

Singular Value Decomposition is a dimensionality reduction technique with deep roots and origins in information retrieval where it is usually called latent semantic indexing (LSI), this information retrieval technique was patented by Deerwester et al. [4, 12, 16]. It was mainly used to solve the problems of synonymy which is having several words containing the same meaning and polysemy which is having words containing more than one meaning [12, 4].

2.6.2.1 SVD Equation Definition

SVD is a factorization technique that factors a $m \times n$ matrix into three matrices that retain the essence of the original matrix as shown in the following equation:

$$A_{mn} = U_{mr} \Sigma_{rr} V_{rn}^{T} \qquad (2.9)$$

where A is the original utility matrix, $U^T U = I$ and $V^T V = I$ which means both V and U have to be orthogonal matrices and Σ is a diagonal matrix containing the square roots of Eigenvalues from U or V in descending order [2] . So for clearer visualization of the U,V, Σ formalization:

$$U = Orthonormal\{EigenVectors(AA^T)\},$$

$$V = Orthonormal\{EigenVectors(A^T A)\}, \qquad (2.10)$$

$$\Sigma = Diagonal\{Descending < Sqrt[EigenValues(AA^T\ OR\ A^T A)] >\}$$

The main idea of SVD is that we can represent the principal factors (dimensions) of a matrix using only a couple of the dimensions while the rest is just a linear combination of those principal axes. This is why after getting $U\Sigma V^T$ we truncate the dimensions to at least the rank of the matrix; however, we can even go further and reduce it to the most significant k factors which would be less than the rank of the matrix as shown in figure 2.8 and table 2.1. So $A_k \approx U_{mk}\Sigma_{kk}V^T{}_{kn}$ is the best rank-K approximation to the original kn matrix A since A_k minimizes the "Frobenius norm" which is the measure of the error of how close the lower rank matrix A_k is to the real matrix A [5, 10], where the Frobenius norm would be mathematically represented as $|F| = \sqrt{\sum_{ij}(A_{ij} - A_{k_{ij}})^2}$ [10] where F is the Frobenius norm and A the original matrix and A_k the resulting approximation of A that occurs from multiplying the reduced $U_k \Sigma_k V_k$. This means $A \approx A_K$, where matrix A_k is the best rank-k approximation for matrix A [5]. If the k is chosen correctly, this new reduced dimensional space will show the true essence of the data without any noise [12, 10, 2, 3].

A_k	Best rank-k approximation to matrix A
U	User vectors
Σ	Singular values (latent factors)
V	Item vectors
m	Number of users
n	Number of items
k	Number of factors (reduced factors)
r	Rank of matrix A

Table 2.1: Interpretation of SVD components in recommender setting if the original matrix A contained the rows as users and columns as items.

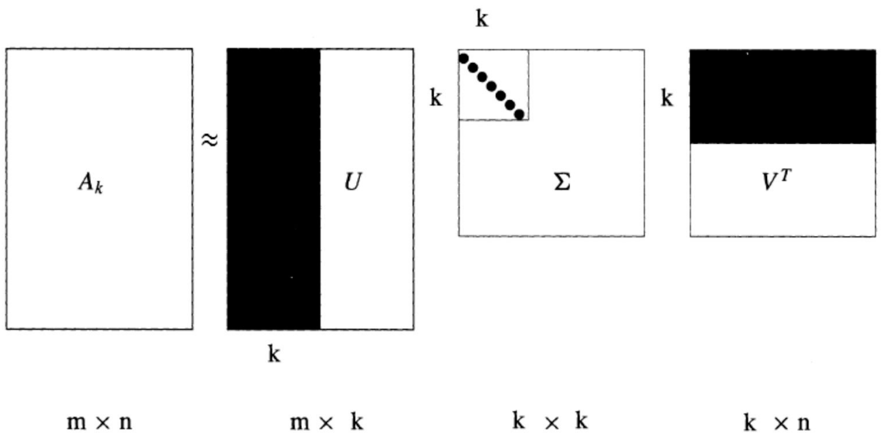


Figure 2.8: Shows matrix A_k being broken down into three matrices $U_{mk}\Sigma_{kk}V^T_{kn}$, notice that matrix U contains the users' vector as a row and matrix V contains item vector as columns while Σ contains the "concept strength" as shown in the above table 2.1 [3]

2.6.2.2 Calculating Predictions using SVD

A user's item prediction can be computed by applying the following algorithm formulated in Sarwar et al.'s research paper [12]:

1. For any empty entries, fill them in using the item average rating (Sarwar et al. stated that item average performed better than user average), since SVD is a mathematical operation so as with any mathematical operation, all values must be present, if there are missing values then SVD cannot be computed therefore in step1 we filled in with the product average rating this method is called "imputation" [8, 5].

2. Normalize by subtracting user Average from each rating (Sarwar et. al states normalizing by this method works better than using the Z-score) since some users rate higher/lower than others.

3. We can now factor the matrix into UΣV using SVD then reduce the matrices' dimensions to k.

4. Compute and let $Q = U_k\sqrt{\Sigma_k}$ and $Z = \sqrt{\Sigma_k}V^T_k$ (Note that in other references such as [5], doing $Q = U_k$ and $Z = \Sigma_k V_k^T$ worked as well.)

5. To compute prediction of the uth user for the ith item, just do the dot product to $Q_{u,\_}$ and $Z_{\_,i}$ which is the uth row in Q and the ith column in Z then add back the average user rating, more formally denoted by this equation:

$$p_{u,i} = \overline{r_u} + (Q_{u,-} \cdot Z_{-,i}) \tag{2.11}$$

Notice that what occurred in the above equation is the same as doing $P = U_k\Sigma_k V_k^T$ then the predicted rating $p_{u,i}$ would be just $P_{u,i} + \overline{r_u}$.

2.6.2.3 Intuition on why SVD works

Intuitively what SVD does is that instead of imposing the different categories or latent factors of the movies like action, comedy, drama, oriented towards males, oriented towards females, serious, escapist ...etc. It deduces these categories or latent/hidden factors using the rating patterns given by the users for the movies [8]. It can be considered as a data-driven categorization method for both the movies and users, the secret lies in the singular matrix Σ, which contains the weights of the latent factors [10].

Each movie and user is a linear combination of the basis unit vectors where these basis unit vectors are the categories. But these categories must be different or independent from each other thus they are orthonormal meaning they are vectors of unit length and their dot product equals zero which means they are perpendicular and independent (since they are used for being bases for linear combination). Figure 2.9 illustrates a simplified categorization that occurs when multiplying the reduced dimension user matrix U_k with Σ_k then graphing it and reduced dimension movie matrix V_k with Σ_k then graphing it [3]. Also, figure 2.10 shows the clustering that occurs on each latent factor vector which acts as the base axis for the movies and users (can only visually show a 2D plot with 2-factor vectors as the base axis).

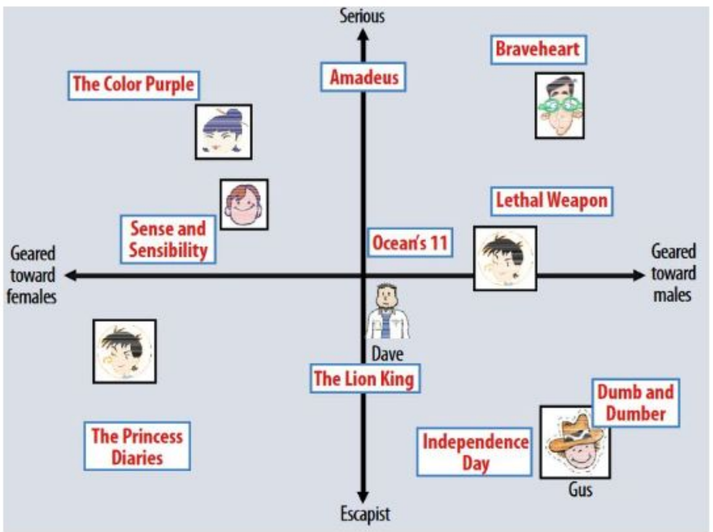


Figure 2.9: A simplified illustration of the latent factors, which characterizes both users and movies on some deduced latent factors [8].

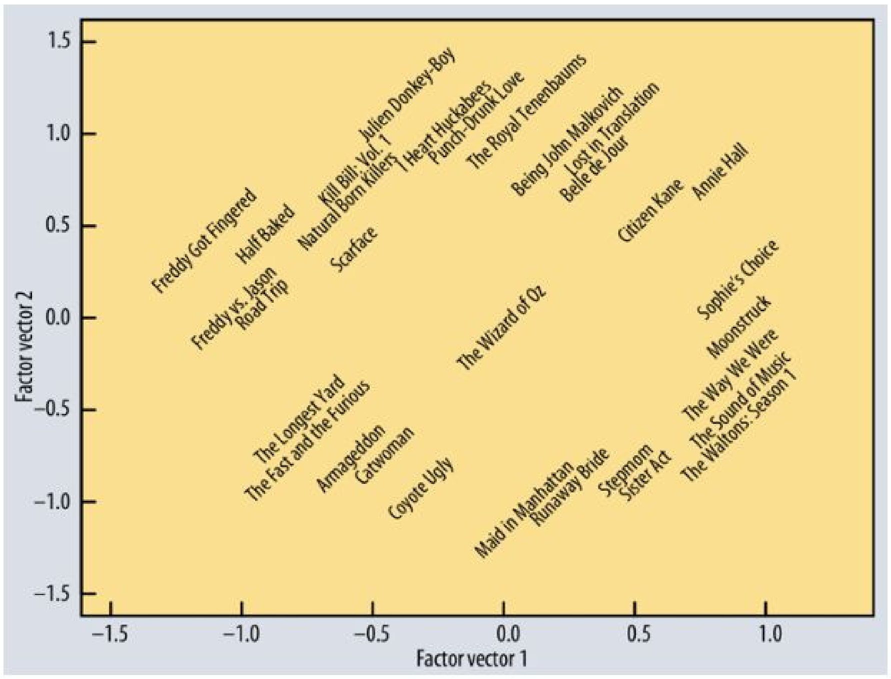


Figure 2.10: “The first two vectors from a matrix decomposition of the Netflix Prize data. Selected movies are placed at the appropriate spot based on their factor vectors in two dimensions. The plot reveals distinct genres, including clusters of movies with strong female leads, fraternity humor, and quirky independent films.” [8].

2.6.2.4 Interpretation example on SVD

Given the simple *user* × *movie* matrix in figure 2.11 from source [10], we want to use the information from subsection 2.6.2.3 to intuitively understand the components of SVD and the latent factors.

$$\begin{array}{r|ccccc|}
 & \text{Matrix} & \text{Alien} & \text{Star Wars} & \text{Casablanca} & \text{Titanic} \\
\hline
\text{Joe} & 1 & 1 & 1 & 0 & 0 \\
\text{Jim} & 3 & 3 & 3 & 0 & 0 \\
\text{John} & 4 & 4 & 4 & 0 & 0 \\
\text{Jack} & 5 & 5 & 5 & 0 & 0 \\
\text{Jill} & 0 & 0 & 0 & 4 & 4 \\
\text{Jenny} & 0 & 0 & 0 & 5 & 5 \\
\text{Jane} & 0 & 0 & 0 & 2 & 2 \\
\hline
\end{array}$$

Figure 2.11: *user* × *movie* → *rating* utility matrix M [10].

After doing SVD via NumPy we arrive at figure 2.12, in this contrived example the matrix is of rank=2 (number of linearly independent vectors) so the k=2 which means this is the full SVD formulation so $U\,\Sigma V^T$ effectively equals matrix M and doesn't approximate it in this instance. U contains the "user-to-concept" (user to latent factor) matrix, V contains the "movie-to-concept" (movie to latent factor) matrix (so V^T is "concept-to-movie"), and Σ is the concept strength (latent factor scale or weight) [10].

$$\underbrace{\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 3 & 3 & 3 & 0 & 0 \\ 4 & 4 & 4 & 0 & 0 \\ 5 & 5 & 5 & 0 & 0 \\ 0 & 0 & 0 & 4 & 4 \\ 0 & 0 & 0 & 5 & 5 \\ 0 & 0 & 0 & 2 & 2 \end{bmatrix}}_{M} = \underbrace{\begin{bmatrix} .14 & 0 \\ .42 & 0 \\ .56 & 0 \\ .70 & 0 \\ 0 & .60 \\ 0 & .75 \\ 0 & .30 \end{bmatrix}}_{U} \underbrace{\begin{bmatrix} 12.4 & 0 \\ 0 & 9.5 \end{bmatrix}}_{\Sigma} \underbrace{\begin{bmatrix} .58 & .58 & .58 & 0 & 0 \\ 0 & 0 & 0 & .71 & .71 \end{bmatrix}}_{V^{\mathrm{T}}}$$

Figure 2.12: Full SVD for matrix M [10].

From figure 2.12 and Σ, it's easy to see that there are two concepts or latent factors present, the "Sci-fi" factor and the "Romance" factor, this can be deduced by looking at the columns of V^T since we know that Matrix, Alien, Starwars are sci-fi and we can see that they are not zeroes in the 1st row (dimension) but are zeroes in the 2nd row so their sci-fi factor weight corresponds to 12.4. While Casablanca and Titanic are non zero on the other row (dimension) so their romance factor weight is 9.5.

For users, using the same thought process as above, it can be seen that column 1 in U is the sci-fi latent factor dimension and column 2 is the romance latent factor dimension, if we multiply U by Σ (user-to-concept multiplied by concept weight) we project the users on the latent factor axis and could plot them, it can be seen where each user falls into place and on which latent factor vector such as shown in figure 2.13. Notice that Joe and Jane rated their movies low so they had small factors in U in their respective rows, this is also reflected in the graph.

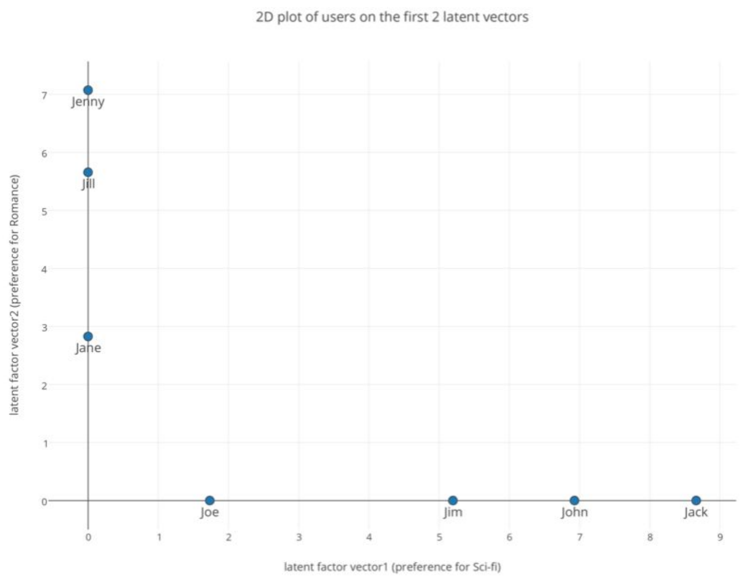
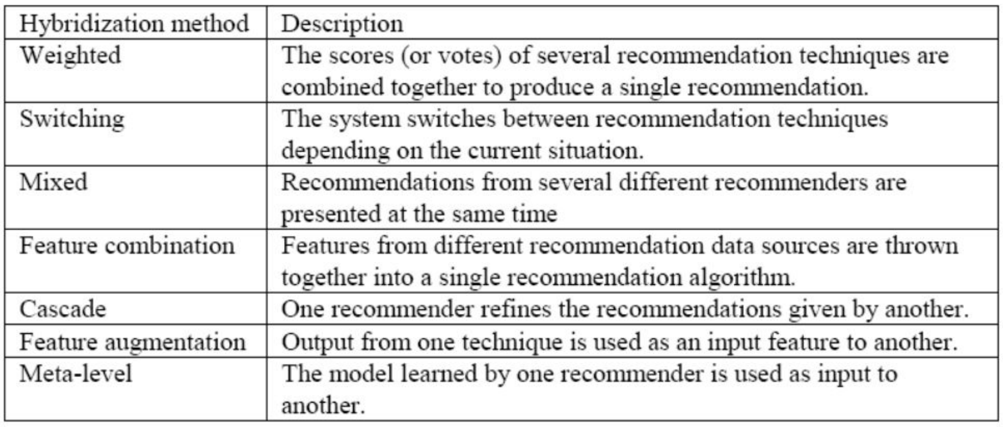


Figure 2.13: Graph of U multiplied by Σ which produced projection of users on the two latent factor vectors which act as axes.

2.7 Hybrid Filtering

To gain superior performance and curb the drawbacks of individual algorithms, two or more recommendation techniques can be combined. The collaborative filtering is often combined with other techniques to overcome common problems like recommending new items. Table 2.2 shows some of the common combination methods.



Hybridization method	Description
Weighted	The scores (or votes) of several recommendation techniques are combined together to produce a single recommendation.
Switching	The system switches between recommendation techniques depending on the current situation.
Mixed	Recommendations from several different recommenders are presented at the same time
Feature combination	Features from different recommendation data sources are thrown together into a single recommendation algorithm.
Cascade	One recommender refines the recommendations given by another.
Feature augmentation	Output from one technique is used as an input feature to another.
Meta-level	The model learned by one recommender is used as input to another.

Table 2.2: Hybridization Techniques

2.7.1 Weighted

The score of a recommended item is calculated from the individual recommendation methods' results in the system.

Suppose:

$R_{CF}(i)$ is the predicted rating for an item i computed by algorithm CF

$R_{CB}(i)$ is the predicted rating for an item i computed by algorithm CB

Then the hybrid rating can be computed:

$$R_H(i) = \alpha R_{CF}(p) + (1-\alpha)R_{CB}(i) \tag{2.12}$$

The P-Tango system uses a linear combination of recommendation scores. The content-based and collaborative recommenders are initially given equivalent weights. As the predictions about the user ratings are validated, the weights are gradually adjusted [28].

Pazzani's method uses a consensus scheme where the output of each recommender is treated as a set of votes [26].

2.7.2 Switching

In a switching recommendation system, some criterion is used to switch between available recommendation techniques.

$$\exists\, k : 1....n \qquad R_{switching}(u, i) = R_k(u, i) \tag{2.13}$$

The DailyLearner [26] system utilizes a hybrid system comprising both content and collaborative methods. Firstly, the content-based recommendation method is employed and then the collaborative recommendation is attempted if the former is not able to make recommendations with sufficient confidence.

The collaborative technique ensures recommendations that are not close in a semantic way to the highly-rated items are also made.

But because both collaborative and the content-based systems suffer from the cold start problem, switching hybrid does not completely avoid the ramp-up problem. Also, another level of parameterization is introduced to determine the switching criteria which makes the system much more complex.

2.7.3 Mixed

In a mixed recommendation system, recommendations from multiple methods are presented together.

$$R_{mixed} = \bigcup_{k=1}^{n} [R_k(u, i), k] \tag{2.14}$$

The PTV system [30] uses content-based techniques based on the textual description of TV shows and collaborative information about the preferences of users. Finally, the recommendations from both methods are mixed to recommend a program.

Mixed hybrid ensures the user is not stuck in local minima, that is, diverse content is always present on the recommendation list.

2.7.4 Cascade

In the cascade hybrid technique, different recommendation systems are involved in a staged process. A rough ranking of contenders is produced by one recommendation technique, then a second technique is used to refine the results from the above contenders set.

EntreeC is a restaurant recommendation system that employs a cascaded collaborative and knowledge-based recommender. Based on the user's stated interests, the system uses its knowledge of restaurants to make recommendations. The recommendations are placed in buckets of equal preference (equal utility) and the collaborative technique is employed to break ties.

The major advantage of cascading is that it prevents employing another round of recommendation algorithms on items that are already well -differentiated by any recommendation technique applied earlier or items that are so poorly-rated they need not be recommended.

2.7.5 Feature Combination

In the feature combination method of hybrid systems, the collaborative data is treated as additional feature data associated with each example and the content-based techniques are used over this augmented data set.

Example: The user's ratings as well as the content features are used in Basu, Hirsh & Cohen's (1998) report [27] where the inductive rule learner Ripper is used for recommending movies.

2.7.6 Feature Augmentation

A rating or classification of an item is achieved through one technique and that information is used in the processing of the next recommendation technique.

The Libra System [31] uses a naive Bayes text classifier to make content-based recommendations of books based on data in Amazon where the text data includes "*related authors*" and "*related titles*" which are generated by Amazon's internal collaborative systems.

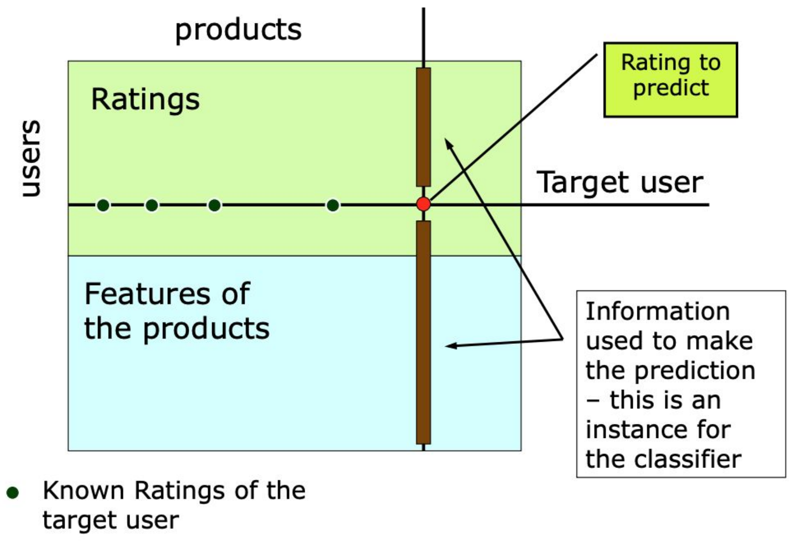

products
users
Ratings
Rating to predict
Target user
Features of the products
Information used to make the prediction – this is an instance for the classifier
● Known Ratings of the target user

Figure 2.14: Feature Augmentation

Chapter 3

Methodology and Implementation

In this chapter, we shall explain our approach and implementation of the hybrid recommender system based on the principle mentioned in Chapter 2 using Incremental SVD technique that updates the SVD model with new users instead of building from scratch and popularity based content filtering that models the movies into the vector space and calculates similarity using cosine similarity to recommend critically acclaimed movies of user's preference.

3.1 Dataset

All the movie data we used is from **The Movies Dataset by Kaggle**. The dataset consists of movies with metadata for all 45,000 movies listed in the Full MovieLens Dataset. Data points include cast, crew, plot keywords, budget, revenue, posters, release dates, languages, production companies, countries, TMDB vote counts, and vote averages.

For the perspective of the recommender system, a movie can be described by a collection of features, which can be genres, actors, directors, and so on.

- **Crew:** The directors, producers, editors, etc. For our purpose, we will be considering the directors only. Most of the movies only have one director, but some of them have two or more.

- **Actor:** A movie normally has a lot of actors, but most of them are useless for the recommender system and bring disadvantageous effects. So we only get three main actors for one movie.

- **Keywords:** The movie plot keywords provide the essence of our movies.

- **Genres:** Theme of the movie that describes movies from a different perspective than keywords. Eg. Sci-Fi, Romance, Thriller, Horror, etc.

- **Ratings:** The set of ratings from different users to the movies.

3.2 Collaborative Filtering

To overcome the limitations of content-based filtering, collaborative filtering relies on how the users interact with the items (implicitly like the number of clicks, or explicitly like ratings). This allows for serendipitous recommendations. The user is recommended items that similar users have given a high rating to. The similarity between the users is calculated based on their rating patterns

We have created a 2-dimensional matrix using the given data which contains the explicit users' feedback which specifies how much they liked a particular movie by providing a numerical rating (in our case a number between 0 and 5).

To establish recommendations, the collaborative filtering systems use latent factor models that transform both items and users to the same latent factor space. The latent space tries to explain ratings by characterizing both products and users on factors automatically inferred from user feedback.

For instance, if the latent factors (embedding vector) for the movies are fixed, then the model can learn an embedding vector for the users to best explain their preferences. As a result of this, the embeddings of users with similar preferences will be close together.

3.2.1 Matrix Factorisation

To create these embeddings we have used Matrix Factorisation. Matrix factorization is a simple embedding model which learns embeddings from dyadic/relational data (each matrix entry is a dyad, e.g., user-item rating, document-word count, user-user link, etc.).

Given a user-item rating matrix $A \in R^{m \times n}$, where m is the number of users and n is the number of items, the model learns:

- A user embedding matrix $P \in R^{m \times k}$, where row i is the embedding vector(latent factors) for a user i.
- An item embedding matrix $Q \in R^{n \times k}$, where row j is the embedding vector(latent factors) for an item j.

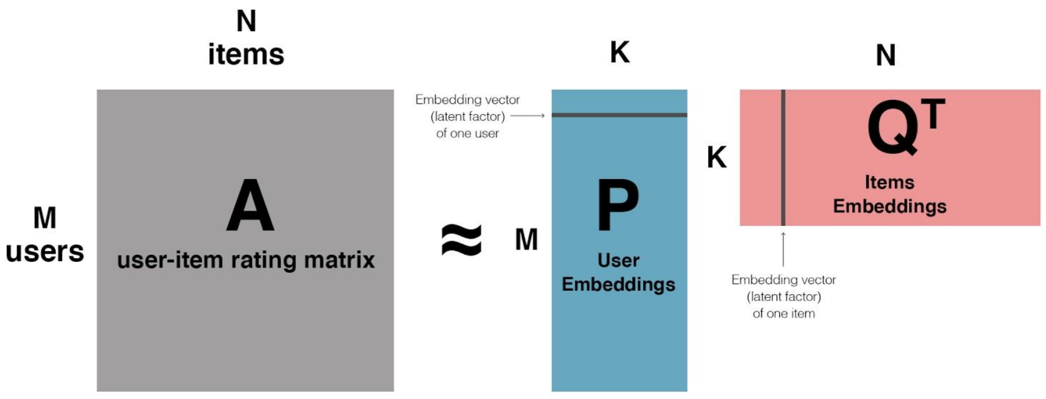


Fig 3.1 Illustrating the Matrix Factorization

The embeddings are learned such that the product PQ^T is a good approximation of the feedback matrix A. The (i,j) entry of $P.Q^T$ is simply the dot product $\langle U_i,\ V_j\rangle$ of the embeddings of user i and item j, which we want to be close to $A_{i,j}$.

$$\text{The prediction } \bar{r}_{ui} \text{ is set as } \hat{r}_{ij} = \mu + b_i + b_u + q^T{}_i p_u \qquad (3.1)$$

Where,
μ is the global mean,
b_i is the item bias,
b_u is the user bias,
q is the item matrix,
p is the user matrix.

To estimate all the unknown, we minimize the following regularized squared error:

$$\sum_{ui \in train} (r_{ui} - \hat{r}_{ui})^2 + \lambda (b_i^2 + b_u^2 + \| q_i \|^2 + \| p_u \|^2) \tag{3.2}$$

The minimization of this error function is performed by a very straightforward stochastic gradient descent.

3.2.2 Folding in a new user

When a new user arrives his user embedding has very few ratings. To recommend movies to the new user we cannot compute the matrix factorization all over again since it is an expensive computation.

We use the error function mentioned above and the gradient descent algorithm. But this time we keep the item matrix fixed and try predicting values only for the new user embedding instead of the complete user matrix, thereby reducing computation drastically.

$$bu \leftarrow bi + \gamma (eui - \lambda\, bu) \tag{3.3}$$
$$\mathrm{p}u \leftarrow \mathrm{p}i + \gamma (eui - \lambda\, \mathrm{p}u) \tag{3.4}$$

3.2.3 The Algorithm

Figure 3.2 shows the architectural flow for our implementation. Below are the steps followed by our algorithm.

1. Load the data set using Pandas into dataframes.

2. We convert the ratings dataframe into a utility matrix where the rows are the user ids and the columns are the movie ids.

3. Then we perform the matrix factorisation which converts our utility matrix to 2 matrices p and q where p contains the user embeddings and q contains the item embeddings.

4. The matrix factorisation is performed considering the error function:

$$\sum_{ui \in train} (r_{ui} - \hat{r}_{ui})^2 + \lambda (b_i^2 + b_u^2 + \| q_i \|^2 + \| p_u \|^2) \quad (3.2)$$

5. To get the estimated rating for a user u for movie i, we simply take the dot product of the embeddings.

The prediction $\bar{r}_{ui}$ is set as $\hat{r}_{ij} = \mu + b_i + b_u + q^T_i p_u$ (3.1)

Where,
μ is the global mean,
b_i is the item bias,
b_u is the user bias,
q is the item matrix,
p is the user matrix.

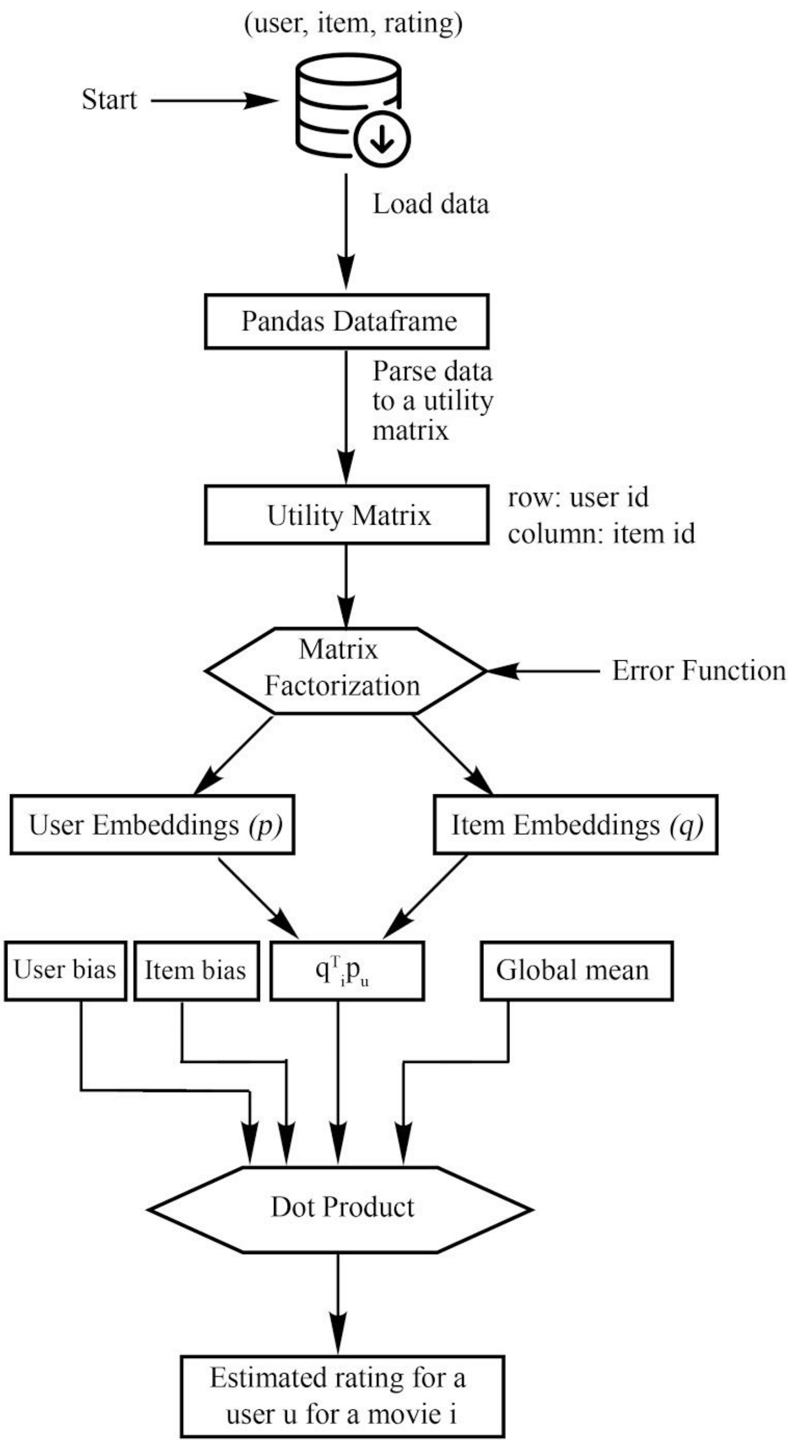
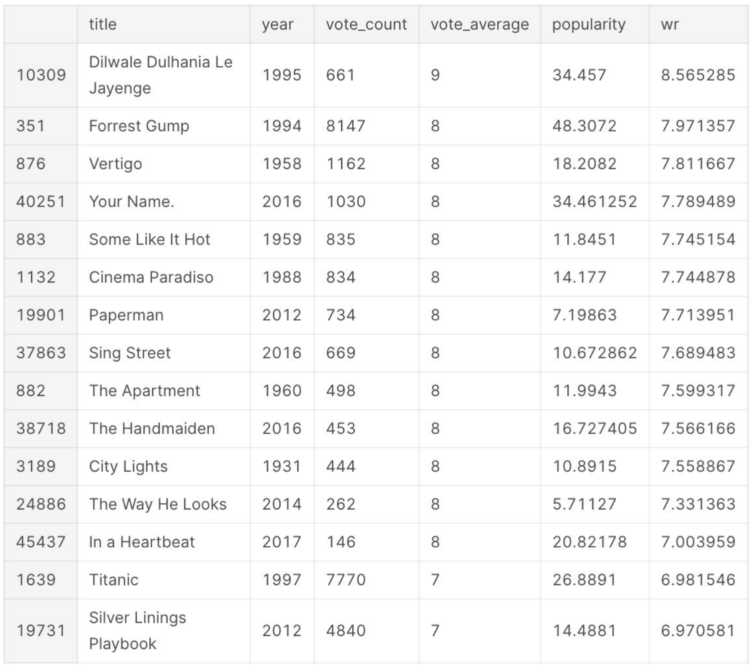


Figure 3.2: Architectural Flow diagram to the algorithm to find the estimated rating of a movie by a user.

3.3 Content Filtering

The standard recommendation systems built purely on movie popularity and genre end up recommending movies that are more popular and more critically acclaimed. This model does not give personalized recommendations based on the user.

For example, this is the list of Top 15 Romance movies according to the popularity rating (***wr***).



	title	year	vote_count	vote_average	popularity	wr
10309	Dilwale Dulhania Le Jayenge	1995	661	9	34.457	8.565285
351	Forrest Gump	1994	8147	8	48.3072	7.971357
876	Vertigo	1958	1162	8	18.2082	7.811667
40251	Your Name.	2016	1030	8	34.461252	7.789489
883	Some Like It Hot	1959	835	8	11.8451	7.745154
1132	Cinema Paradiso	1988	834	8	14.177	7.744878
19901	Paperman	2012	734	8	7.19863	7.713951
37863	Sing Street	2016	669	8	10.672862	7.689483
882	The Apartment	1960	498	8	11.9943	7.599317
38718	The Handmaiden	2016	453	8	16.727405	7.566166
3189	City Lights	1931	444	8	10.8915	7.558867
24886	The Way He Looks	2014	262	8	5.71127	7.331363
45437	In a Heartbeat	2017	146	8	20.82178	7.003959
1639	Titanic	1997	7770	7	26.8891	6.981546
19731	Silver Linings Playbook	2012	4840	7	14.4881	6.970581

Figure 3.3: Similar movie recommendations that don't take personalization into account

Consider a person who loves Dilwale Dulhania Le Jayenge, Kabhi Alvida Na Kehna, and Dil To Pagal Hai. One inference we can obtain is that the person loves the actor Shahrukh Khan and the director Karan Johar. Even if s/he were to access the generic romance chart like Fig 3.3, s/he wouldn't find these as the top recommendations.

3.3.1 Our Approach: Extending Content Filtering with Popularity

As discussed in Section 2.5.1, a document can be represented as a set of features in the vector space model. Each movie in our case is a document, represented by the Vector Space Model and each feature for the movie is a term in the document. Then we use cosine similarity discussed in Section 2.5.2 to calculate similarity for each movie.

To personalize the recommendations more, we suggest movies that are most similar to a particular movie that a user liked but still takes into consideration the popularity of the movies so that bad movies are not recommended to any user based on his/her taste.

3.3.2 The Algorithm

Figure 3.5 shows the architectural flow for our implementation. Below are the steps followed by our algorithm.

1. Load the data set using Pandas into dataframes.

2. To pre-process the keywords, calculate the frequency counts of every keyword that appears in the dataset. Keywords that occur only once are not used and can be safely removed.

3. Convert every word to its stem using SnowballStemmer so that words such as *Dogs* and *Dog* are considered the same.

4. Merge the movies metadata(*name, id*) dataframe with the crew, cast, and the keywords dataframe information thus creating a metadata dump or *soup* for every movie.

    - Only the top 3 actors from the credits list are chosen because supporting roles do not impact the opinion of a movie significantly.

- Only the director is selected as the feature from the crew list since the remaining production doesn’t have a major contribution to the *feel* of the movie.

  For example, consider the movie ‘**Toy Story**’. Table 3.1 shows the metadata information that will be merged to form a dump shown in Fig 3.4

Director(Crew)	John Lasseter
Cast	Tom Hanks, Tim Allen, Don Rickles
Keywords	jealousy, toy, boy, friendship, friend, rivalry, boynextdoor, newtoy, toycomestolife
Genres	Animation, Comedy, Family

Table 3.1 Information for Toy Story

[‘jealousy toy boy friendship friend rivalry boynextdoor newtoy toycomestolife tomhanks timallen donrickles johnlasseter Animation Comedy Family']

Figure 3.4 The representation of Toy Story

5. To convert a movie to the vector space model, use sklearn’s CountVectorizer to convert this combined metadata soup of all the movies into a matrix of token counts.

6. Compute the cosine similarity between a given movie and all the movies represented in the count matrix from Step 4 using sklearn’s **linear_kernel**. This returns similar movies regardless of ratings and popularity.

For example, it is true that **Batman and Robin** has a lot of similar characters as compared to **The Dark Knight** but it was a terrible movie that shouldn't be recommended to anyone.

7. To ensure popular movies with good critical response are returned, take the top 25 movies based on similarity scores from 6 and use IMDB's *weighted rating* formula:

$$\text{Weighted Rating (WR)} = \left(\frac{v}{v+m} \cdot R\right) + \left(\frac{m}{v+m} \cdot C\right) \qquad (3.5)$$

   where,

   - v is the number of votes for the movie
   - m is the minimum votes required to be listed in Top 250 chart
   - R is the mean rating of the movie as a number from 1 to 10
   - C is the mean vote across the whole report (currently 7.0)

   to construct a chart of the most popular movies from those 25 movies.

8. To determine an appropriate value for m, the minimum votes required to be listed in the chart, use the 75th percentile as our cutoff. In other words, for a movie to feature in the charts, it must have more votes than at least 75% of the movies on the list.

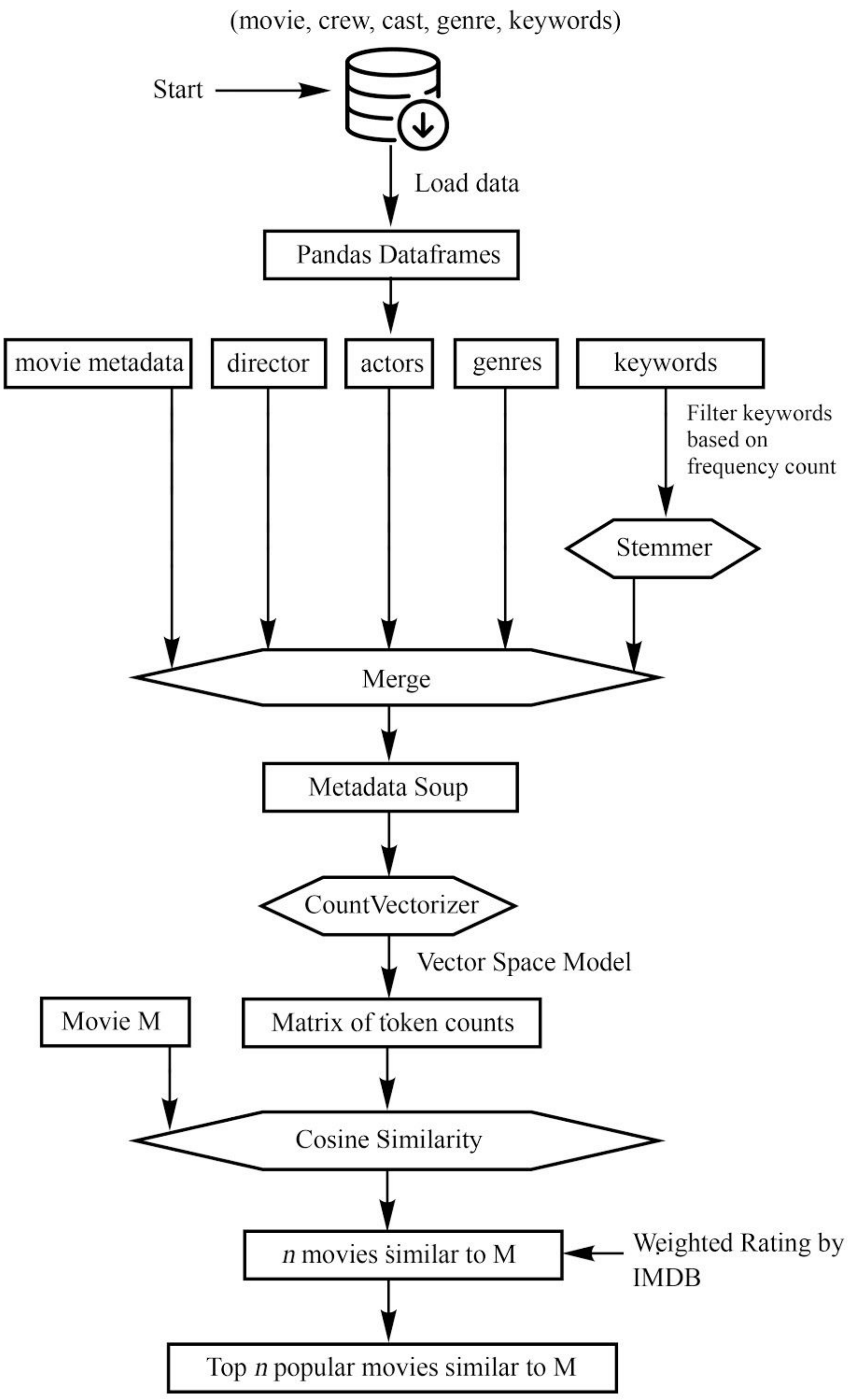


Figure 3.5: Architectural Flow diagram to the algorithm to find the top ‘n’ popular movies that are similar to the given movie.

3.4 Hybrid Filtering

As mentioned in chapter 2, **Collaborative recommenders** rely on how the users interact with the items (implicitly like the number of clicks, or explicitly like ratings). For our project, we consider the movie ratings. These collaborative filters figure out how similar users rate movies using the available rating data.

Collaborative filters face two common problems. The first one is the **Cold start problem** which arises because the new items do not have any ratings and the second one is the problem to **recommend novel or niche items** that do not have enough user ratings.

Content Filters do not rely on user ratings so they can tackle *cold start problem*, *popularity bias,* and can even recommend novel and niche movies.

They depend on the similarity between the items and hence recommend movies having similar features which can result in giving out movies that have lower ratings.

To overcome the limitations of content and collaborative filters we need to create a **hybrid filter** that takes advantage of both filtering techniques. It recommends movies taking into account both the benefits that is, the recommendations contain high rated movies according to the user's rating pattern but those will be curated according to the taste of the user.

3.4.1 Combining Recommendations

Given a number of recommendation scores obtained by using different methods, e.g. $R_{CF}(u, d_i)$, $R_{CB}(u, d_i)$, etc., an overall score can be obtained using

$$R(u, d_i) = f(R_{CF}(u, d_i), R_{CB}(u, d_i), \ldots) \tag{3.6}$$

with f an aggregation function. The P-Tango system [28] uses a linear combination of collaborative and content-based scores to make its recommendations and adjusts.

Aggregation of two or more methods can be performed using several functions with different properties and behavior. The use of non-linear or more complicated functions would enable some recommenders to fine-tune the ranking process, creating less irrelevant and more accurate predictions.

3.4.2 The Algorithm

1. We combine the outputs of collaborative and content filtering to produce recommendations.

2. Given a user u for whom we need to compute the recommendations first we get *'n'* recommendations from the collaborative filtering.

3. Using these *'n'* movies we get *'m'* movies from content filtering.

4. Now for these *'n+m'* movies, we calculate a hybrid score for all these movies using the formula :

$$\alpha \text{ x } \textit{Collaborative Score} + \beta \text{ x } \textit{content score (wr)}$$

Where,

$$\beta = 1 - \alpha \tag{3.7}$$

$$\text{Weighted Rating } (\boldsymbol{wr}) = \left(\frac{v}{v+m} \cdot R\right) + \left(\frac{m}{v+m} \cdot C\right) \tag{3.8}$$

where,
v is the number of votes for the movie
m is the minimum votes required to be listed in Top 250 chart
R is the mean rating of the movie as a number from 1 to 10
C is the mean vote across the whole report (currently 7.0)

The *weighted rating* (wr) is similar to a Bayesian posterior mean.

5. We sort the recommendations based on this hybrid score and recommend movies.

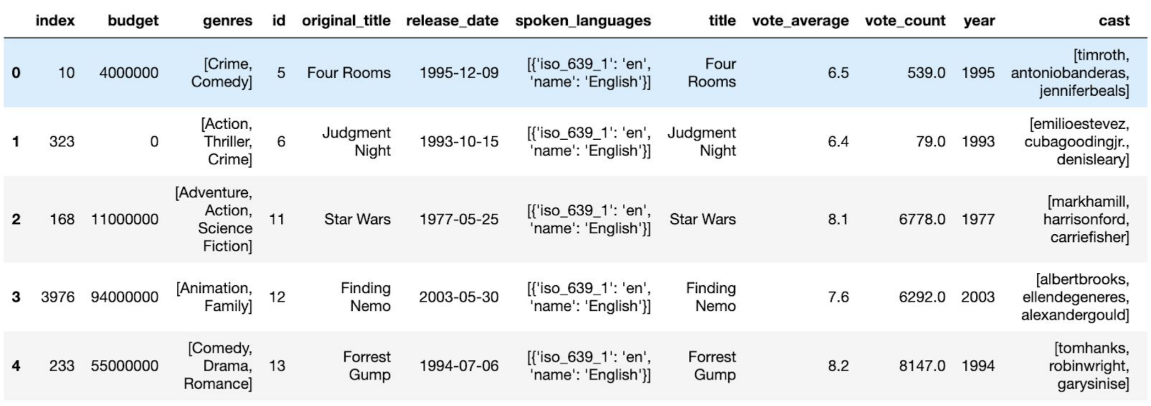
3.5 Demonstration and visual validation

We developed a working algorithm to showcase our hybrid recommendation filtering. Content-based and collaborative filtering were implemented separately and a weighted hybrid method was utilized to present the recommendations.

3.5.1 Content Filtering

As discussed in Section 3.3.1, using content-based filtering we suggest movies that are most similar to a particular movie that a user liked but still takes into consideration the popularity of the movies.

Fig 3.6 shows the metadata soup created for the movies that will be used to represent each movie in the vector space model.

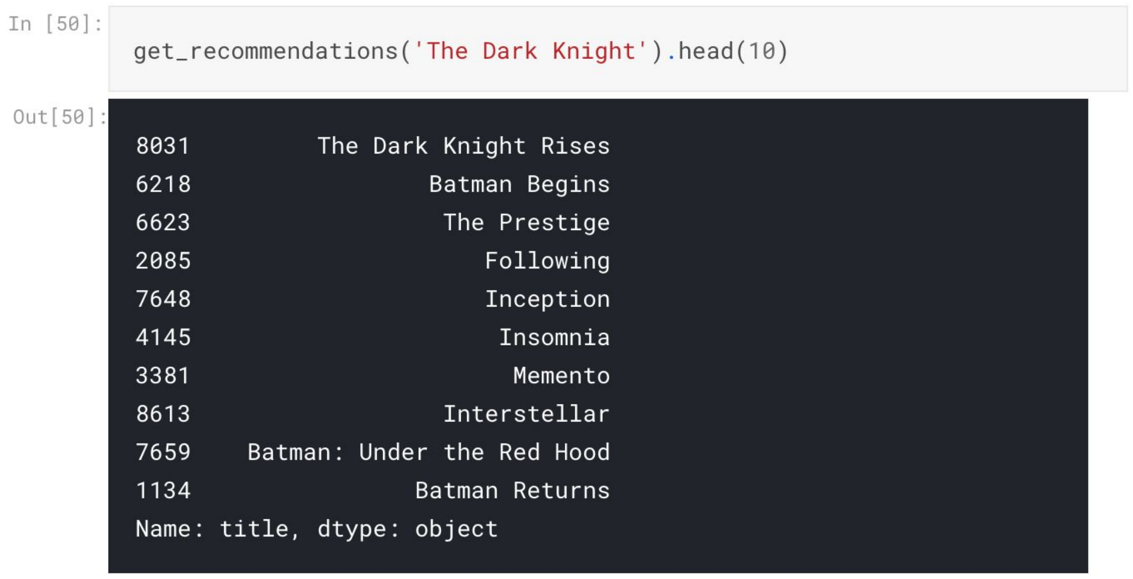


	index	budget	genres	id	original_title	release_date	spoken_languages	title	vote_average	vote_count	year	cast
0	10	4000000	[Crime, Comedy]	5	Four Rooms	1995-12-09	[{'iso_639_1': 'en', 'name': 'English'}]	Four Rooms	6.5	539.0	1995	[timroth, antoniobanderas, jenniferbeals]
1	323	0	[Action, Thriller, Crime]	6	Judgment Night	1993-10-15	[{'iso_639_1': 'en', 'name': 'English'}]	Judgment Night	6.4	79.0	1993	[emilioestevez, cubagoodingjr., denisleary]
2	168	11000000	[Adventure, Action, Science Fiction]	11	Star Wars	1977-05-25	[{'iso_639_1': 'en', 'name': 'English'}]	Star Wars	8.1	6778.0	1977	[markhamill, harrisonford, carriefisher]
3	3976	94000000	[Animation, Family]	12	Finding Nemo	2003-05-30	[{'iso_639_1': 'en', 'name': 'English'}]	Finding Nemo	7.6	6292.0	2003	[albertbrooks, ellendegeneres, alexandergould]
4	233	55000000	[Comedy, Drama, Romance]	13	Forrest Gump	1994-07-06	[{'iso_639_1': 'en', 'name': 'English'}]	Forrest Gump	8.2	8147.0	1994	[tomhanks, robinwright, garysinise]

Fig 3.6: Metadata soup of each movie that includes genres, crew, cast and keywords

Let us first see the output recommendations without using the popularity filter. Using cosine similarity, we compute the similarity between a given movie and all the movies represented in the count matrix.

As can be seen from Fig 3.7, when **The Dark Knight** was given as input to the content filtering algorithm, the recommendations seem to have recognized other Christopher Nolan movies and put them as top recommendations.



```
In [50]:
get_recommendations('The Dark Knight').head(10)
```

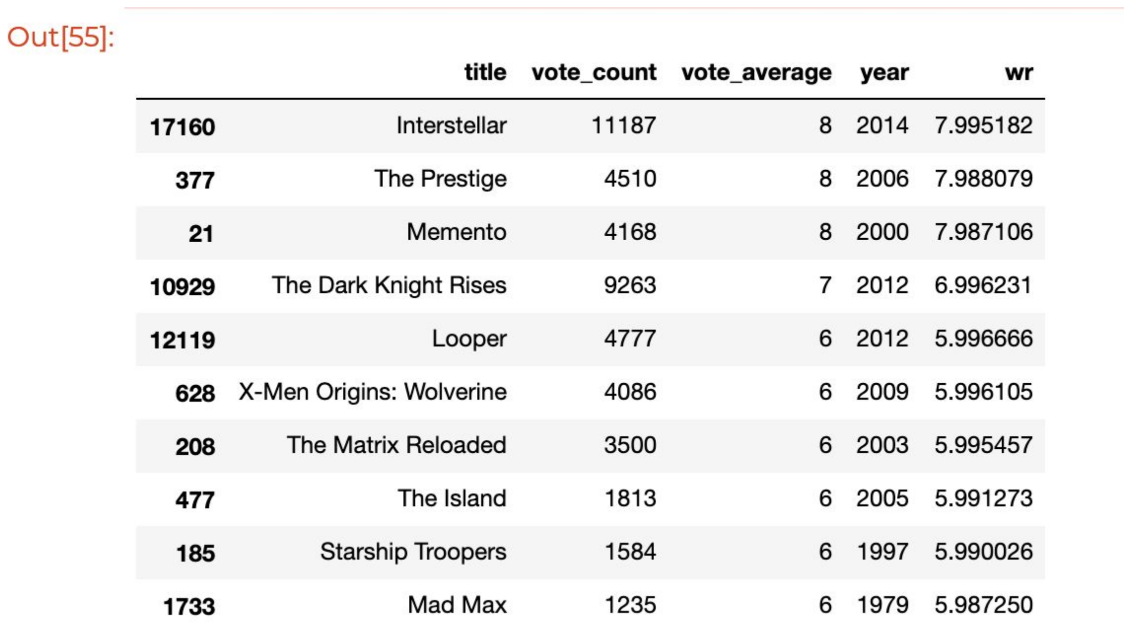
```
Out[50]:
8031         The Dark Knight Rises
6218                 Batman Begins
6623                  The Prestige
2085                     Following
7648                     Inception
4145                      Insomnia
3381                       Memento
8613                  Interstellar
7659    Batman: Under the Red Hood
1134                Batman Returns
Name: title, dtype: object
```

Fig 3.7 List of similar movies to **The Dark Knight**

But **Batman and Robin**, though has a lot of similar characters as compared to **The Dark Knight,** is a terrible movie that shouldn't be recommended to anyone.

Now, to ensure popular movies with good critical responses are returned, we take the top 25 movies based on similarity scores and use IMDB's *weighted rating* formula to present the final recommendations.

As seen from Fig 3.8, when movie **Inception** was given as input, less popular movies, however similar they may be, are placed lower whereas more popular movies are present at the top of the recommendation list.



Out[55]:

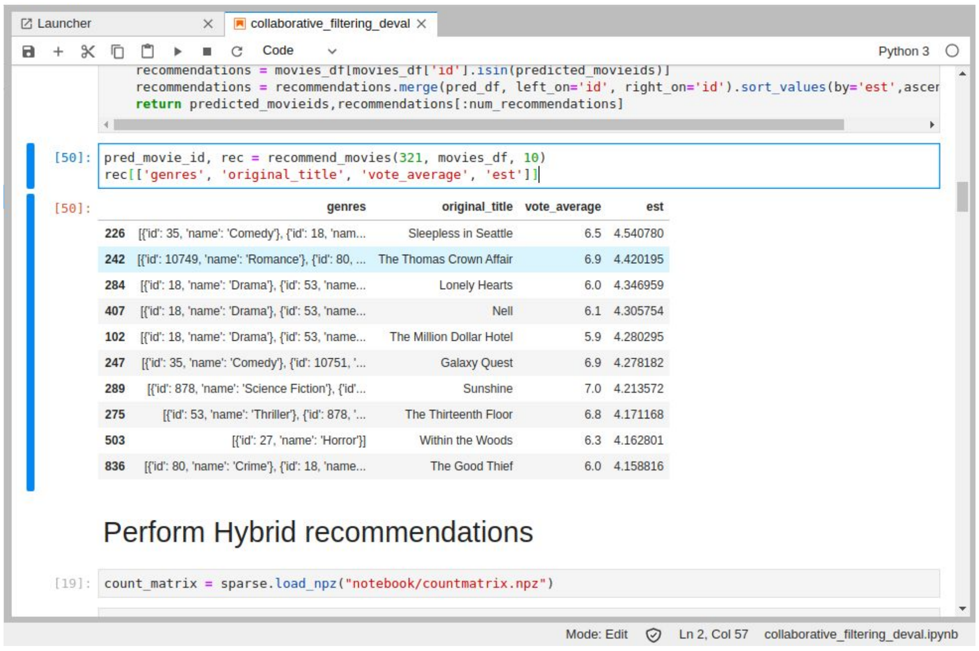
	title	vote_count	vote_average	year	wr
17160	Interstellar	11187	8	2014	7.995182
377	The Prestige	4510	8	2006	7.988079
21	Memento	4168	8	2000	7.987106
10929	The Dark Knight Rises	9263	7	2012	6.996231
12119	Looper	4777	6	2012	5.996666
628	X-Men Origins: Wolverine	4086	6	2009	5.996105
208	The Matrix Reloaded	3500	6	2003	5.995457
477	The Island	1813	6	2005	5.991273
185	Starship Troopers	1584	6	1997	5.990026
1733	Mad Max	1235	6	1979	5.987250

Fig 3.8: Improved Content-based filtering using popularity filter

3.5.2 Collaborative Filtering

As discussed in Section 3.2, our scalable collaborative filtering model uses the incremental SVD algorithm based on the “fold-in” technique that could append new users so they receive their recommendations on the spot instead of waiting for the model to be rebuilt.

Fig 3.9 shows for the user ‘*321*’ the top 10 recommendations according to his/her preferences found using Singular Value Decomposition on the utility matrix. The predicted ratings are found using Matrix Factorization and Fold-in Technique as discussed in Section 3.2.1 and Section 3.2.2.



```
    recommendations = movies_df[movies_df['id'].isin(predicted_movieids)]
    recommendations = recommendations.merge(pred_df, left_on='id', right_on='id').sort_values(by='est',ascer
    return predicted_movieids,recommendations[:num_recommendations]
```

```
[50]: pred_movie_id, rec = recommend_movies(321, movies_df, 10)
      rec[['genres', 'original_title', 'vote_average', 'est']]
```

[50]:

	genres	original_title	vote_average	est
226	[{'id': 35, 'name': 'Comedy'}, {'id': 18, 'nam...	Sleepless in Seattle	6.5	4.540780
242	[{'id': 10749, 'name': 'Romance'}, {'id': 80, ...	The Thomas Crown Affair	6.9	4.420195
284	[{'id': 18, 'name': 'Drama'}, {'id': 53, 'name...	Lonely Hearts	6.0	4.346959
407	[{'id': 18, 'name': 'Drama'}, {'id': 53, 'name...	Nell	6.1	4.305754
102	[{'id': 18, 'name': 'Drama'}, {'id': 53, 'name...	The Million Dollar Hotel	5.9	4.280295
247	[{'id': 35, 'name': 'Comedy'}, {'id': 10751, '...	Galaxy Quest	6.9	4.278182
289	[{'id': 878, 'name': 'Science Fiction'}, {'id'...	Sunshine	7.0	4.213572
275	[{'id': 53, 'name': 'Thriller'}, {'id': 878, '...	The Thirteenth Floor	6.8	4.171168
503	[{'id': 27, 'name': 'Horror'}]	Within the Woods	6.3	4.162801
836	[{'id': 80, 'name': 'Crime'}, {'id': 18, 'name...	The Good Thief	6.0	4.158816

Perform Hybrid recommendations

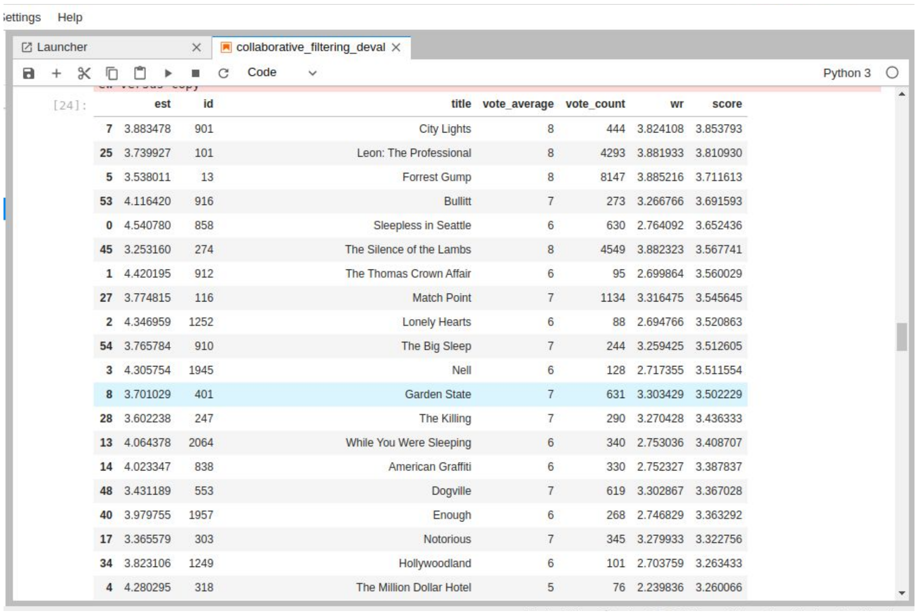
```
[19]: count_matrix = sparse.load_npz("notebook/countmatrix.npz")
```

Mode: Edit   Ln 2, Col 57   collaborative_filtering_deval.ipynb

Fig 3.9: Result collaborative filtering for user ***321***

3.5.3 Hybrid Filtering

The results from the content-based and collaborative filtering are combined using a weighted average. Fig 3.10 shows the result of hybrid filtering.



[24]:

	est	id	title	vote_average	vote_count	wr	score
7	3.883478	901	City Lights	8	444	3.824108	3.853793
25	3.739927	101	Leon: The Professional	8	4293	3.881933	3.810930
5	3.538011	13	Forrest Gump	8	8147	3.885216	3.711613
53	4.116420	916	Bullitt	7	273	3.266766	3.691593
0	4.540780	858	Sleepless in Seattle	6	630	2.764092	3.652436
45	3.253160	274	The Silence of the Lambs	8	4549	3.882323	3.567741
1	4.420195	912	The Thomas Crown Affair	6	95	2.699864	3.560029
27	3.774815	116	Match Point	7	1134	3.316475	3.545645
2	4.346959	1252	Lonely Hearts	6	88	2.694766	3.520863
54	3.765784	910	The Big Sleep	7	244	3.259425	3.512605
3	4.305754	1945	Nell	6	128	2.717355	3.511554
8	3.701029	401	Garden State	7	631	3.303429	3.502229
28	3.602238	247	The Killing	7	290	3.270428	3.436333
13	4.064378	2064	While You Were Sleeping	6	340	2.753036	3.408707
14	4.023347	838	American Graffiti	6	330	2.752327	3.387837
48	3.431189	553	Dogville	7	619	3.302867	3.367028
40	3.979755	1957	Enough	6	268	2.746829	3.363292
17	3.365579	303	Notorious	7	345	3.279933	3.322756
34	3.823106	1249	Hollywoodland	6	101	2.703759	3.263433
4	4.280295	318	The Million Dollar Hotel	5	76	2.239836	3.260066

Fig 3.10: Result from the combination of content-based and collaborative filtering.

Chapter 4

Results and Evaluation

In this chapter, we shall discuss our achieved results when running our algorithm on the MovieLens dataset. We will also analyze the distribution of the datasets and the optimal number of latent dimensions. Also, we will initially introduce the class of evaluation metrics we have chosen for evaluating our recommender system.

4.1 Evaluation Metrics

To claim that a recommender is better than another recommender system, one needs a standard evaluation metric which can be the same for all recommender systems. Since different datasets can give different accuracy readings for the same algorithm [5]. There are a lot of factors that can affect the error score of a recommender, possible examples can include:

1. **Sparsity:** How empty is the dataset? Some recommenders can work well with dense data while others perform poorly on sparse data and vice versa.

2. **Rating scale:** Is the dataset on a scale from 1-5 or 1-10? An error score on one scale could be good while on another scale it could be bad.

3. **Portion used for testing:** Which portion of the dataset was used for training and which for testing? The parts of the data used must be consistent for all when evaluating a recommender. This is usually true in a competition like the Netflix challenge.

4.1.1 Predictive Accuracy Metrics

The class of metrics we decided to use for evaluation are the predictive accuracy metrics. These metrics measure how accurate or close the predicted ratings are to the real user ratings [6, 7]. They were chosen due to their consistent use in academic papers and their inclusion in the Netflix competition, thus proving relevancy.

4.1.1.1 Mean Absolute Error (MAE)

The MAE measures the average deviation between the predicted and real user rating [7]

$$\text{MAE} = \frac{\sum_{u,i} \left| p_{u,i} - r_{u,i} \right|}{n} \tag{4.1}$$

Where n is the number of elements held out for testing and $p_{u,i}$ is the predicted rating for the real hidden user rating $r_{u,i}$. So if MAE=0.75 then that means the algorithm was off by 0.75 stars on average therefore less is better [5].

4.1.1.2 Root Mean Square Error (RMSE)

The RMSE is similar to MAE; however, it places more emphasis on large errors. For example, a system is penalized more if it's off by 2 stars in 1 single prediction than for being off 1/4 of a star in 8 predictions [5]. Therefore less is also better here. RMSE has been popularized by the Netflix prize since it was the chosen error metric to be decreased by just 10% for the 1 million dollar prize [5, 6].

$$\text{RMS} = \sqrt{\frac{\sum_{u,i} (p_{u,i} - r_{u,i})^2}{n}} \tag{4.2}$$

Where n is the number of elements held out for testing and $p_{u,i}$ is the predicted rating for the real hidden user rating $r_{u,i}$.

4.2 MovieLens Dataset

The dataset we have tested our recommender system on was The Movies Dataset on Kaggle[1]. The dataset contains “45,000 ratings from 700 users on all the 45,000 movies”. The rating scale for this dataset is from 1 to 5. The data points of the dataset include cast, crew, plot keywords, budget, revenue, posters, release dates, languages, production companies, countries, TMDB vote counts, and vote averages.

In the upcoming subsections, we will discuss the user distribution, finding optimal weights given to each method for calculating the hybrid score, and the achieved results.

4.2.1 User Distribution

We examined the user distribution to get a clearer image of the data. So we implemented a simple algorithm in python that counts the number of rated items that each user had rated. Then it counted the frequency of the number of occurrences of each rating per user count. Then it proceeded to plot a histogram for visualization. Figure 4.1 shows the distribution of the number of ratings per user count. This is important because it means that the dataset has different types of users, i.e, the users who rated only a few movies and the ones who rated a lot.

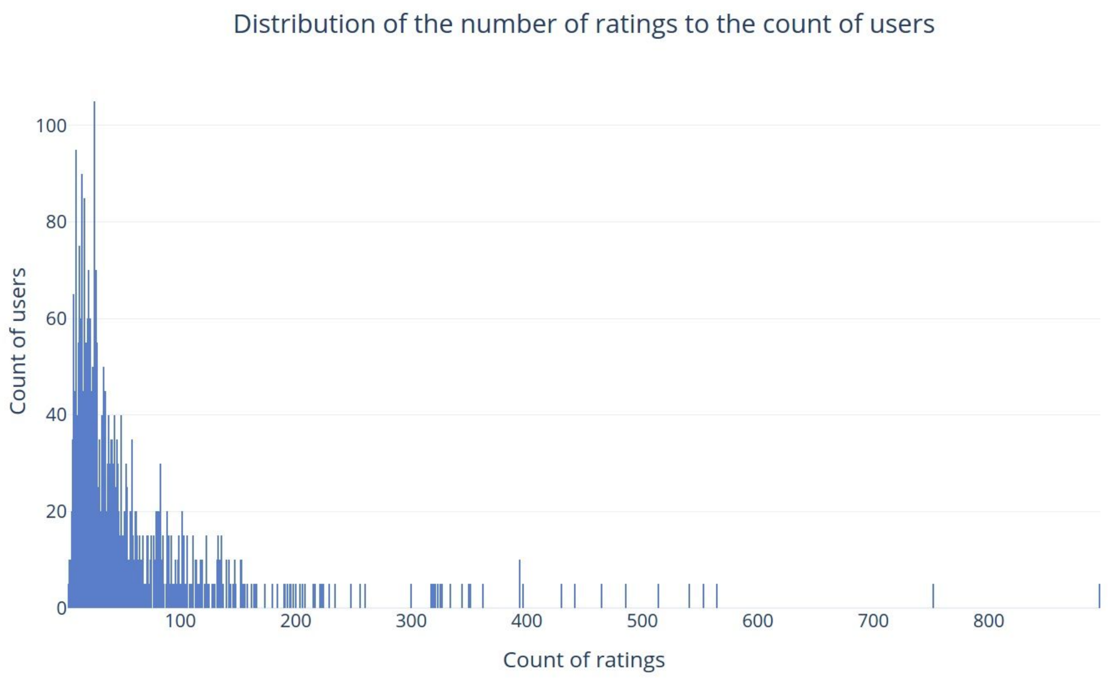


Figure 4.1: Distribution of the number of ratings to the count of users

4.2.2 Evaluating our Hybrid model versus the standard model - our model vs collaborative alone, our model vs content alone

The RMSE and MAE values calculated for our hybrid model and the standard collaborative and content model alone are summarized in table 4.1. It is observed that the error value is least for the hybrid model.

Algorithm	**RMSE**	**MAE**
Collaborative Filtering	1.012	0.745
Content Filtering	1.296	1.082
Hybrid Filtering	**0.973**	**0.733**

Table 4.1: Summary of Results

4.2.2.1 Division of data into Training and Testing datasets

We divided the dataset into training and testing. The training set is the data that the SVD will be built from, which is used for the collaborative filtering and the testing test is used to calculate the root mean square error.
The rating dataset is divided such that 75% of it is used as a training dataset and the remaining 25% is used as a testing dataset.

4.2.2.2 Calculating the RMSE value for the testing dataset

The training data of rating is used to train the collaborative model to predict the rating for the given movie and the user and content model use movie data to calculate the rating. The RMSE is calculated for the testing data, i.e, 25% of the total rating data. For each movie rated by a user in a testing dataset, a rating is predicted by our hybrid by taking a weighted average of values from collaborative and content models. RMSE value is calculated for all predicted ratings against the actual rating in the dataset.

4.2.2.3 Finding optimal value of *alpha*

The rating of a movie predicted by the hybrid model is equal to the weighted average of rating predicted by the collaborative model and the content model. The weight given to the rating predicted by the collaborative model is taken as *alpha* and the rating predicted by the content model is taken as (1 - *alpha)*. To calculate the optimal value of *alpha* to get the least RMSE value, its value is varied from 0 to 1 and corresponding RMSE value is calculated for the testing dataset as shown in Figure 4.2. We get the minimum RMSE value for *alpha* as 0.65.

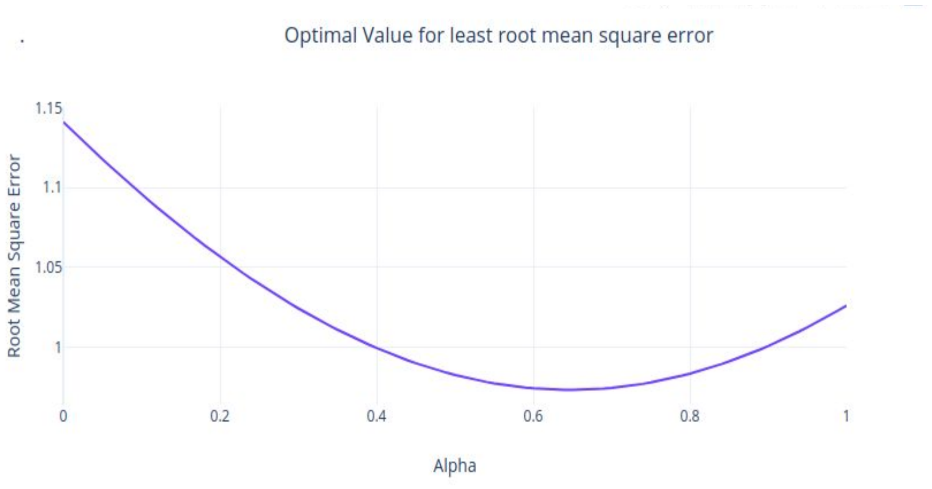


Figure 4.2: Optimal Value of alpha for least Root Mean Square Error

4.3 Discussion

In this section, we shall discuss our findings for the optimal *alpha* and Error scores in the dataset.

4.3.1 The Optimal *alpha*

As discussed in section 4.2.2.3, we calculated the RMSE value by varying *alpha*. It is found that for the value of *alpha* equal to 0.65, the RMSE value is the least. This implies that the score from the collaborative model has relatively more weightage than the content model in the hybrid model. This optimal *alpha* value may vary depending on the item for which the rating is calculated. Here the item considered is a movie that has attributes such as crew, cast, genre, etc which are used for calculating the similarity in movies. But for some other items (say food products) for which a similar recommendation model is to be developed, the attributes will vary and will not be the same as that for movies, hence the optimal value for *alpha* will be different.

4.3.2 Discussing and Validating RMSE and MAE

The RMSE and MAE values calculated for our model are summarized in section 4.2.2. The RMSE of the hybrid model is 0.973 this means that
on average, its value is deviated from the actual value by 0.973 stars, for a 1-5 scale and the MAE of the hybrid model is 0.733 this means that
on average, it’s wrong by a value of 0.733 stars, for a 1-5 scale. The RMSE value is less for the hybrid model than for the collaborative and content model individually. Hence by combining the results from both, we get better results.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In this project, our main objective was to implement a hybrid recommender system that integrates the content and collaborative approaches to benefit from their complementary advantages, using an incremental SVD based algorithm that could incrementally update itself and a popularity based content filtering algorithm that personalizes the recommendations removing bad movies from the result.

The incremental SVD approach would solve the scalability problem of frequent model construction in a short time frame to remain relevant and enable new users to receive recommendations instantly which is usually not the case with the base SVD model recommender since new users will need to wait for the scheduled rebuilding of the model to receive their recommendations.

The popularity based Content Filtering approach suggests movies similar to a particular movie that was liked by a user, taking into account the popularity of the movies so that bad movies are not recommended to any user based on his/her taste.

5.2 Future Work

We have planned a continued development, adding additional features to this project. The new proposed features for continued development are as follows:

Parallelization:

- Parallel computing of different components for increasing the overall performance of the system.
- The collaborative score and the content score are calculated independently hence computation for these scores can be done parallelly.
- Also, the cosine similarity value which is used in content filtering has to be calculated for many movies at a time. This calculation can be done simultaneously for multiple movies at the same time.

Development of a working web application:

Developed a website to showcase our hybrid algorithm. The web platform will display the movies in the MovieLens dataset and gives the user the ability of rating movies.

After the user has rated a few movies, the user will have an option "**Get Recommendation**" prompting the user to request a recommendation. When the algorithm is done the python script gets the top highest rated predicted items and sends them to the user as recommendations.

Support Folding in the new user as well as a new item simultaneously:

When folding in new users and the user had rated a new item not originally in the matrix, we made the algorithm truncate the new items and Fold-in the user without those new items. Since those items should either be in the model or be Folded-in initially before Folding-in a new user with a new item at the same time. This is not a serious issue; however, it does lead to some irritations in testing when the testing set contains new items that weren't folded in or part of the SVD model.

Appendix

Appendix A

List of Abbreviation

SVD	Singular Value Decomposition
MAE	Mean Absolute Error
RMSE	Root Mean Square Error

References

[1] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. IEEE transactions on knowledge and data engineering, 17(6):734–749, 2005.

[2] Kirk Baker. Singular value decomposition tutorial. Ohio State University, 2005.

[3] Michael W Berry, Susan T Dumais, and Gavin W O'Brien. Using linear algebra for intelligent information retrieval. SIAM Review, 37(4):573–595, 1995.

[4] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. Indexing by latent semantic analysis. Journal of the American Society for information science, 41(6):391, 1990.

[5] Michael D Ekstrand, John T Riedl, Joseph A Konstan, et al. Collaborative filtering recommender systems. Foundations and Trends® in Human-Computer Interaction, 4(2):81–173, 2011.

[6] Asela Gunawardana and Guy Shani. A survey of accuracy evaluation metrics of recommendation tasks. Journal of Machine Learning Research, 10(Dec):2935–2962, 2009.

[7] Jonathan L Herlocker, Joseph A Konstan, Loren G Terveen, and John T Riedl. Evaluating collaborative filtering recommender systems. ACM Transactions on Information Systems (TOIS), 22(1):5–53, 2004.

[8] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. Computer, 42(8), 2009.

[9] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. Mining of massive datasets, chapter 9: Recommendation Systems, pages 307–341. Cambridge University Press, 2014.

[10] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. Mining of massive datasets, chapter 11: Dimensionality Reduction, pages 405–437. Cambridge University Press, 2014.

[11] Prem Melville and Vikas Sindhwani. Recommender systems. In Encyclopedia of machine learning, pages 829–838. Springer, 2011.

[12] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Application of dimensionality reduction in recommender system-a case study. Technical report, DTIC Document, 2000.

[13] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In Proceedings of the 10th international conference on World Wide Web, pages 285–295. ACM, 2001.

[14] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Incremental singular value decomposition algorithms for highly scalable recommender systems. In Fifth International Conference on Computer and Information Science, pages 27–28. Citeseer, 2002.

[15] Rob Speer, Kenneth Arnold, and Catherine Havasi. Divisi: Learning from semantic networks and sparse svd. Python in Science Conf. (SCIPY 2010), 2010.

[16] Xiaoyuan Su and Taghi M Khoshgoftaar. A survey of collaborative filtering techniques. Advances in artificial intelligence, 2009:4, 2009.

[17] Ricardo Baeza-Yates, Berthier Ribeiro-Neto, et al. Modern information retrieval, volume 463. ACM Press New York, 1999.

[18] G Salton, A Wong, and C.S. Yang. A Vector Space Model for Automatic Indexing

[19] R.D.Simon, X. Tengke, and W. Shengrui, "Combining collaborative filtering and clustering for implicit recommender system," in Proc. 2013 IEEE 27th Int'l Conf. on. Advanced Information Networking and Applications, pp. 748-755, March 2013.

[20] Z.Zheng, H. Ma, M. R. Lyu, et al., "QoS-aware Web service recommendation by collaborative filtering," IEEE Trans. on Services Computing, vol. 4, no. 2, pp. 140-152, February 2011.

[21] Rocchio, J.: Relevance Feedback in Information Retrieval. In: G. Salton (ed.). The SMART System: Experiments in Automatic Document Processing. NJ: Prentice Hall (1971) 313- 323

[22] Salton, G. Automatic Text Processing. Addison-Wesley (1989)

[23] Juan Ramos. Using tf-idf to determine word relevance in document queries. In Proceedings of the first instructional conference on machine learning, 2003.

[24] Anna Huang. Similarity measures for text document clustering. In Proceedings of the sixth new Zealand computer science research student conference (NZCSRSC2008), Christchurch, New Zealand, pages 49–56, 2008.

[25] Chang Seok Park, Byeong Man Kim, Qing Li, Si Gwan Kim. A new approach for combining content-based and collaborative filters.

[26] Daniel Billsus and Michael J. Pazzaniu. User Modeling for Adaptive News Access.

[27] Chumki Basu, Haym Hirsh, William Cohen. Recommendation as Classification: Using Social and Content-Based Information in Recommendation

[28] Claypool, M., Gokhale, A., Miranda, T., Murnikov, P., Netes, D. and Sartin, M.: Combining Content-Based and Collaborative Filters in an Online Newspaper. In Proceedings of SIGIR 99 Workshop on Recommender Systems: Algorithms and Evaluation, Berkeley, CA (1999)