ISSN (e): 2250 – 3005 || Volume, 14 || Issue, 3|| May. - June. – 2024 ||
International Journal of Computational Engineering Research (IJCER)

Algorithmic Strategies in Recommender Systems: A Comprehensive Study of Content-Based and Collaborative Filtering Techniques on a Custom-Curated Movie Dataset

Ajay Jaiswal[1], Kunal Dutta[2]
[1]*Assistant Professor,*
Department of Computer Science and Engineering,
Prestige Institute of Engineering, Management & Research, Indore (MP), India.
[2] *Research Scholar,*
Department of Computer Science and Engineering,
Prestige Institute of Engineering, Management & Research, Indore (MP), India

ABSTRACT

This paper aims to provide a way to apply various filtering techniques available for recommender systems on custom datasets which are not fit in accordance to standardized algorithms. Specifically, this paper will focus on the use of Natural Language Processing algorithms for content based filtering and compare them to statistical measures. It will focus on various memory & model based techniques such as matrix factorization methods, neighborhood-based algorithms and deep learning to model the user-item interactions.

KEYWORDS: *Exploratory Data Analysis (EDA), Content-Based Filtering, Word2Vec, Collaborative-Based Filtering (CF), Singular Value Decomposition (SVD), k-Nearest Neighbors*

Date of Submission: 04-06-2024 Date of acceptance: 16-06-2024

I. INTRODUCTION

Recommendation systems have become integral to the digital economy, influencing user behavior and significantly impacting the revenue streams and user engagement metrics across various sectors, including e-commerce, entertainment, and other services. For instance, Amazon, one of the largest e-commerce platforms globally, attributes approximately 35% of its revenue to its recommendation engine. This translates to significant monetary gains given Amazon's net revenue from e-commerce sales was US $470 billion in 2021. In the entertainment sector, particularly in streaming services like Netflix and YouTube, recommendation systems play a crucial role in driving user engagement. text superscript of the movies watched on Netflix are discovered through the platform's recommendation system. 80% of Netflix viewer activity is driven by personalized recommendations from their algorithm.

Netflix's recommendation engine saves the company $1 billion per year. Other services like Spotify, Google News, LinkedIn, Facebook, Tinder, Google Ads, etc. use recommendation systems to increase user retention and amplify their revenues. With the increasing volume of data and the need for personalized user experiences, businesses across various domains will increasingly rely on sophisticated recommendation algorithms to stay competitive and meet user expectations.

Goals of the paper

This paper aims to address several fundamental challenges in developing a robust recommendation system it also proposes an optimized and efficient hybrid recommendation algorithm. The specific objectives are as follows:

1. Pipelining custom dataset and pre-processing it to be used by various algorithms.
2. Using Natural Language Processing (NLP) to identify relation between different movies embedding without relying on prior user-item interaction data.
3. Using collaborative filtering algorithms like SVD, KNN and NCF to further enhance our recommendation system.

4. Propose optimized hybrid recommender system and address Cold Start problem using content based filtering.

Previous Work

There has been significant research on using TF-IDF, Doc2Vec, and other NLP techniques for content-based filtering in recommendation systems. Nigram (2021) utilized TF-IDF for research paper analysis using NLP techniques. Kumar et al. (2021) proposed a machine learning-based content-based recommender system for movie recommendations using TF-IDF, among other techniques.
Doc2Vec, an extension of Word2Vec, has been effectively used for content-based movie recommendations by transforming movie descriptions into fixed-dimension vectors, capturing semantic similarities between movies (Liu and Wu, 2019).

K-Nearest Neighbors (KNN) is a popular collaborative filtering technique that recommends items based on user or item similarity, effectively addressing data sparsity and cold start issues (Badugu and Manivannan, 2023).

Singular Value Decomposition (SVD) is a matrix factorization technique that decomposes the user-item interaction matrix into latent factors, improving recommendation accuracy by uncovering hidden patterns (Quek, 2015).

Neural Collaborative Filtering (NCF) leverages deep learning to model complex, non-linear interactions between users and items, achieving high accuracy in predicting user preferences but with higher computational costs (Jena et al., 2022).

II. Methodology

For the data pipelining tasks, the following hardware and software were used:

- **Hardware:** Apple MacBook Air 2020 (SoC: Apple M1 Chip - 7 Icestorm Cores, Memory: 8GB).
- **Software:** Jupyter Notebook version 7.0.4.
- **Libraries:** Numpy, Pandas, etc.

For the machine learning tasks, the following hardware and software were used:

- **Hardware:** Google Colab free tier TPU and CPU.
- **Libraries:** Numpy, Pandas, JSON, Matplotlib, Seaborn, WordCloud, Math, AST, NLTK, Recommenders, Scikit-learn, Surprise, Collections, TensorFlow, PyTorch, Keras.

Data Collection

The dataset for this study was obtained using The Movie Database (TMDb) API (Application Programming Interface) Version 3. To facilitate data collection, a custom pipeline was developed using Node.js. This pipeline was designed to interact with the TMDb API efficiently, automating the retrieval of movie data. The primary endpoint utilized for this purpose was '/3/movie/popular ', which returns a list of movies ranked by their popularity on the TMDb platform. Then again those movies were queried with the api endpoint '/3/movie/movie_id to fetch more details. Finally these movies were again queried for one last time with api endpoint '/3/movie/movie_id/credits'to finally fetch the cast(actors) and crew(directors) of any movie. All of this information was necessary to create a thorough and comprehensive dataset.
The decision to focus on the top 10,000 most popular movies was driven by several factors.

- **Relevance:** Popular movies are more likely to have higher number of ratings and reviews, providing a richer dataset analysis. This also helped in ranking the top movies directly from database into the recommendation application.
- **Computational Efficiency:** Limiting dataset to 10,000 movies strikes a balance between having sufficiently large dataset for meaningful analysis and maintaining computational efficiency. Most complex computations were completed within 2 hours of computation time on the free tier Google Colab CPU.
- **Benchmarking:** Using a dataset of popular movies allows for easier comparison with other studies in the field, many of which also utilize popular movie datasets. This also helped in getting more than 70% intersection of common movies from the Movie Lens 'ml-25'dataset.

Exploratory Data Analysis (EDA)

Getting familiar with the dataset that has been have collected is important. This dataset has a total of 18 columns for each movie as shown in Table 1.

Column	**Data Type**
adult	boolean
backdrop path	url
genres	list
id	int
imdb id	int
original language	string code
overview	string
popularity	int
poster path	url
release date	date
runtime	time (in mins)
tagline	string
title	string
vote average	int
vote count	int
actors	List of string
director	List of string
keywords	List of string

Table 1: Database Schema with Data Types

For more insights on data, Figure 1 shows the most popular genres and Figure 2 show the most popular movies in dataset.

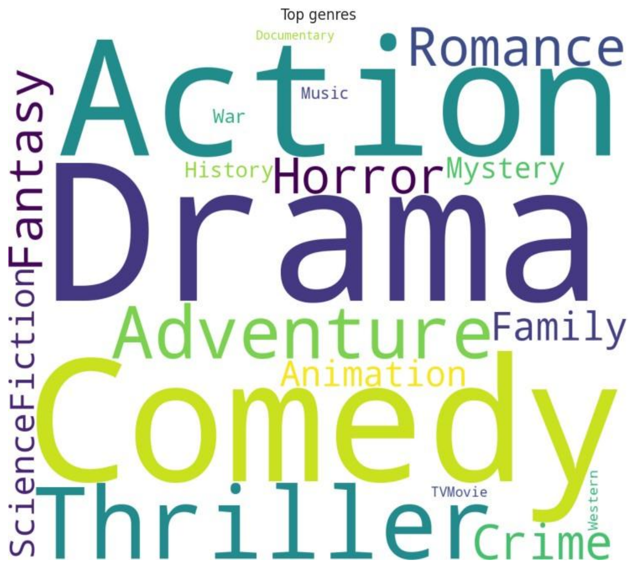


Figure 1: Most Prominent Genres in dataset

Most Prominent Genres in dataset

The other dataset that was used was MovieLens’s Dataset for collaborative filtering. The one used in this study was "ML-25M" Dataset. It contains 25,000,095 ratings and 1,093,360 tag applications across 62,423 movies as mentioned in "The MovieLens Datasets: History and Context" by Harper, F. Maxwell and Konstan, Joseph A.

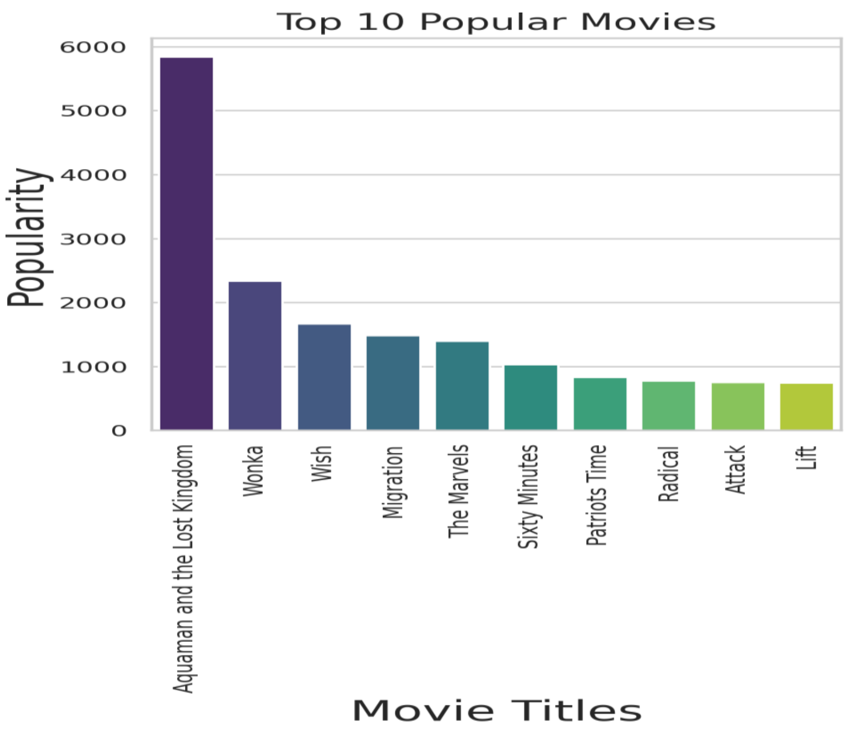


Figure 2: Most Popular Movies in dataset

Most Popular Movies in dataset

Figure 3 shows the ratings to movie distribution, we can observe that most users like to rate a movie 4 stars instead.

Most of the time they're based on personal preferences, but often they are influenced by perception of perfection, that 5 star means perfect, even if they have no complaints, they may feel that there is still room for improvement. More often, people might have a tendency to avoid giving extreme ratings, such as one star or five stars, and instead opt for a more moderate rating like four stars.

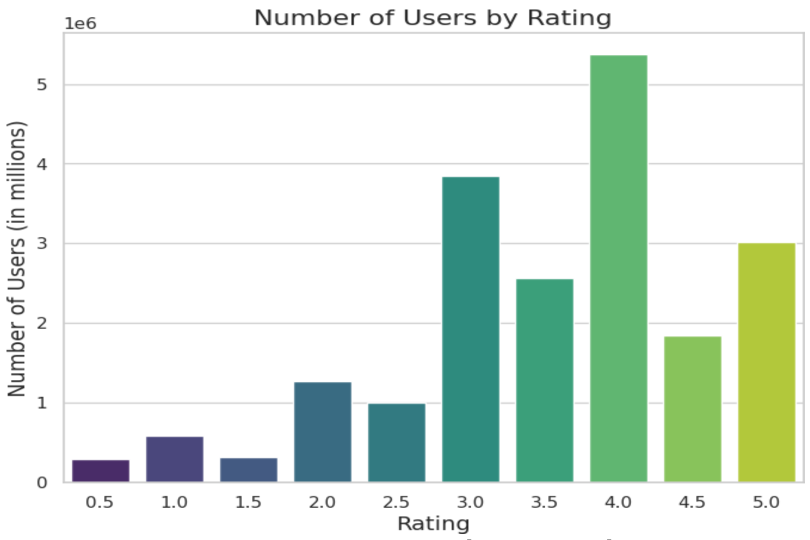


Figure 3: Ratings Distribution Graph

Content-Based Filtering

This study approaches content-based filtering (CBF) predominantly from a natural language processing (NLP) perspective, focusing on the textual analysis of movie descriptions and tags. The rationale behind this approach is to leverage NLP techniques to extract meaningful patterns and features from text data, which are crucial for understanding content semantics and subsequently improving recommendation accuracy. By analyzing the textual content associated with movies, such as plot summaries and tags, the implemented CBF algorithms

can identify and recommend movies that share similar themes, genres, or narrative styles, aligning closely with individual user preferences.
To create paragraph embedding, it is essential to know what features to include in creating tags for a given movie. For building a "Content" Based Filter it will be better to compare contents of the movie rather than raw numbers (like budget, profits, etc.).
Included Features:

- genres: Directly indicates the content categories.
- keywords: Provides specific themes and concepts within the content.
- overview: Offers a textual summary of the content's plot and themes.
- director: Points to creative style and thematic preferences.
- actors: Suggests genre alignment and potentially acting style preferences.

Features like tagline is excluded because it is very abstract and is often misleading or irrelevant to actual content and language is not included due to the skewness of dataset towards one specific language (English) as shown in Figure 4, which could introduce bias.

Languages vs Movie Distribution

Term Frequency - Inverse Document Frequency (TF-IDF) with Cosine Similarity

TF-IDF (Term Frequency-Inverse Document Frequency) is a statistical measure employed to assess the significance of a word within a document relative to a collection or corpus. The importance of a word increases proportionally with its frequency in the document but is counterbalanced by its frequency across the corpus. This technique is particularly advantageous in content-based filtering, as it aids in distinguishing movies based on the uniqueness of their descriptions. The TF-IDF value for a term t in a document d from a document set D is computed as follows: Term Frequency or TF,

$$\mathrm{TF}(t,d) = \frac{f_{t,d}}{\sum_{t\prime \in d} f_{t\prime,d}}$$

where $f_{t,d}$ represents the number of times term t appears in document d, and the denominator is the sum of the occurrences of all terms appear in document d, thereby normalizing the term frequency. Inverse Document Frequence or IDF,

$$\mathrm{IDF}(t,D) = \log\left(\frac{N}{|\{d \in D: t \in d\}|}\right)$$

where N denotes the total number of documents in the corpus D, and $|\{d \in D: t \in d\}|$ is the number of documents containing the term t (i.e., t is not counted if it does not appear in the document). Finally, TF-IDF score is then calculated as,

$$\text{TF-IDF}(t,d,D) = \mathrm{TF}(t,d) \times \mathrm{IDF}(t,D)$$

For this dataset, we have set limit on maximum number of features or the size of vector each document is represent as, to 5000. It effectively limits the vocabulary size to the top N (here 5000) features that have the highest term frequencies. It helps in improving efficiency and reducing model complexity.

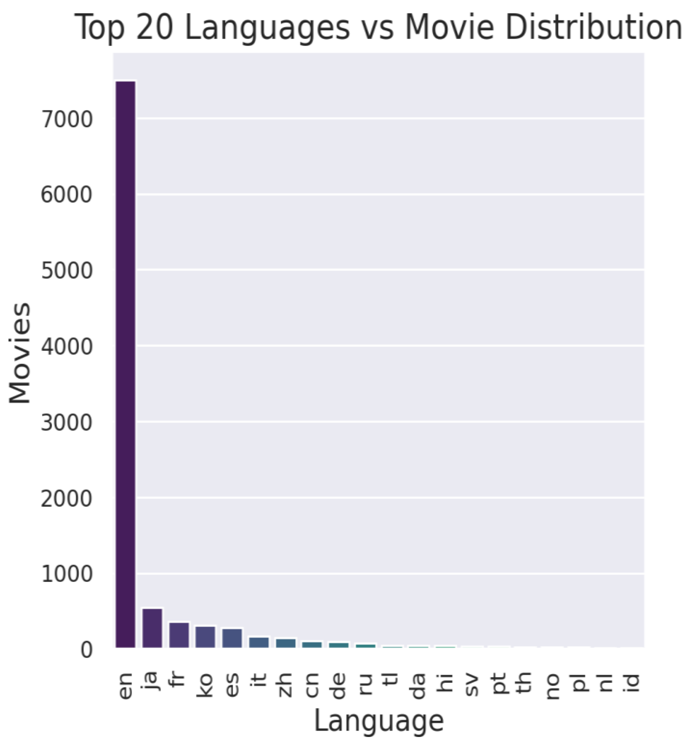


Figure 4: Languages vs Movie Distribution

Then we set the minimum document frequency to 5. This sets a threshold for the minimum number of documents a term must appear in to be included in the vocabulary. It helps to remove terms that are very rare, and specific to a small number of documents. It also helps our model to generalize well and not over fit the data.

We also set the n-gram range to $(1,2)$. N-gram range refers to the different values of n that are considered when analyzing a text. In this study we are using an n-gram range of 1 to 2, we are considering unigrams (single words) and bigrams (pairs of words).

After this, we can successfully create document embedding or vectors for each movie. There are different ways to compare the similarity between these vectors. These vectors are usually plotted in N-dimensional vector space, here, our vector size is 5000 so the data will be represented in $5000 - Dimentional$ space. Underlying idea is that, the movies that have most similar meaning will have a very similar vector representation, which in turn will mean that they are close to each other in the vector space, calculating the K nearest vectors will return K most similar movies.

There are many ways to achieve this. Primary ways to infer similarity include Cosine Similarity, Euclidean Distance, Manhattan Distance, KNN (K-Nearest Neighbors), Pearson Correlation Coefficient, Jaccard Similarity, etc. For this dataset, Cosine Similarity is used, which is both fast and efficient. Subsequently the K largest similarity scores have to be sorted.

Cosine similarity is given by,

$$\begin{aligned} S_C(A,B) := \cos(\theta) &= \frac{\mathbf{A}\cdot\mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} \\ &= \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\cdot\sqrt{\sum_{i=1}^{n} B_i^2}} \end{aligned}$$

This formula calculates the cosine of the angle between two non-zero vectors A and B in a n-dimensional space, where $\mathbf{A}\cdot\mathbf{B}$ is the dot product of vectors A and B, and $\|\mathbf{A}\|$ and $\|\mathbf{B}\|$ are the magnitudes (or lengths) of vectors A and B, respectively.

Cosine Similarity for 2 Dimensional vectors
In Figure 5, it can be observed that movie B exhibits higher similarity to movie A than to movie C. This is evident from the smaller angle θ' between A & B that a larger angle θ between B & C as $cos(\theta') < cos(\theta)$. When implimenting in code, smaller cosine distance translates to larger cosine similarity.

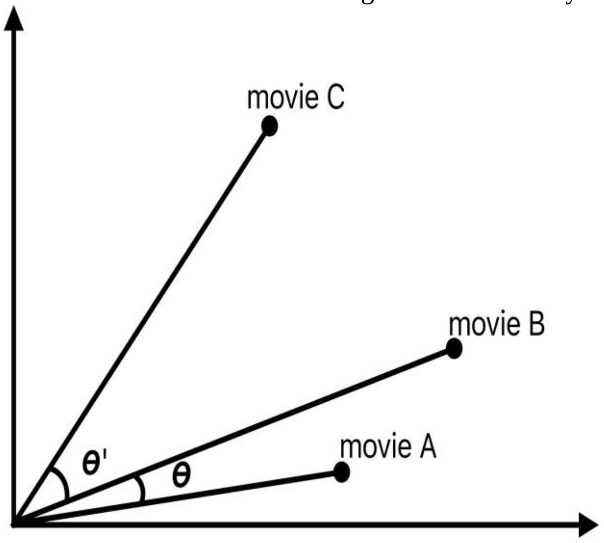


Figure 5: Cosine Similarity for 2 Dimensional vectors

Distributed Representation Learning Models
Word2Vec (Mikolov et al., 2013a, 2013b) and Doc2Vec (Le & Mikolov, 2014) are two influential distributed representation learning models that learn dense vector representations of words and documents, respectively, based on the core idea that the meaning of a word or document can be inferred from its context.

2.3.2.1 Word2Vec
To understand Doc2Vec, we need to understand Word2Vec first which serves as the foundation for Doc2Vec. It is a technique to learn word embedding, where is represented as a dense vector in high-dimensional space as shown in Figure 6. These embedding capture both semantic and syntactic relationships between the words. A neural network is given a dummy task to predict a word given other words in the context window.
In the Word2Vec model, each word in the vocabulary is represented by a unique vector, which corresponds to a column in a matrix W. This matrix W serves as a lookup table, where the position of a word in the vocabulary determines the column index of its associated vector representation.

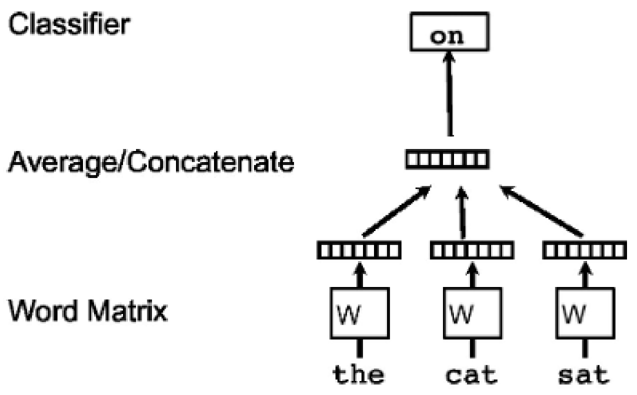


Figure 6: CBOW architecture with three words context (” the”, ”cat”, ”sat”) which will be used to predict the target word ”on”. The core idea is to use these word vectors as features to predict the next word in a given sequence or context. To achieve this, the model concatenates or averages the vectors of the surrounding context words, creating a composite vector representation of the context.

The objective of the Word2Vec model is to maximize the average log probability of correctly predicting the target word, given its surrounding context words. This can be expressed as:

$$\max_{\theta} \frac{1}{T}\sum_{t=1}^{T} \log p(w_t|w_{t-k},\dots,w_{t+k})$$

where T is the length of the sequence, w_t is the target word at position t, and $w_{t-k},\dots,w_{t+k}$ are the surrounding context words within a window of size k.

To compute the probability of the target word given the context, the Word2Vec model employs a multiclass classifier, typically the softmax function. The soft max function takes the dot product of the context vector and the target word vector, and normalizes it to obtain a probability distribution over the entire vocabulary.

The probability of the target word w_t given the context is calculated as:

$$p(w_t|w_{t-k},\dots,w_{t+k}) = \frac{e^{y_{w_t}}}{\sum_i e^{y_i}}$$

where y_i is the un-normalized log-probability for each output word i, computed as-

$$y_i = b_i + U_i h(w_{t-k},\dots,w_{t+k}; W)$$

In this equation, U and b are the softmax parameters, and h is a function that constructs the context vector by concatenating or averaging the word vectors extracted from the matrix W.

Limitation of Word2Vec for Document Comparison

While Word2Vec has proven to be effective for learning word-level embedding, it has limitations when it comes to comparing entire documents. Word2Vec embedding only capture the semantic relationships between individual words and do not consider the overall context and meaning of a document. Comparing documents using word-level embedding alone may not be sufficient, as it does not take into account the document-level semantics and the order of words in the document.

2.3.2.2 Doc2Vec

To address the limitations of Word2Vec for document comparison, Le and Mikolov (2014) introduced the Doc2Vec model, also known as Paragraph Vector. Doc2Vec extends the Word2Vec model by adding a document-specific vector to the input layer of the neural network. This document vector, often referred to as the "paragraph id," is concatenated or averaged with the word vectors to predict the next word in the context.

The algorithm consists of two main architectures, the Distributed Memory Model of Paragraph Vectors (PV-DM) and the Distributed Bag of Words version of Paragraph Vector (PV-DBOW).

In the PV-DM model as shown in Figure 7, each paragraph and word are mapped to unique vectors, represented by columns in matrices D and W, respectively. The objective function aims to maximize the average log probability of predicting the next word in a context, given the concatenation or average of the paragraph vector and the surrounding word vectors. This is achieved by optimizing the following equation:

$$\frac{1}{T}\sum_{t=k}^{T-k} \log p(w_t|w_{t-k},\dots,w_{t+k},d)$$

where w_t represents the predicted word, $w_{t-k},\dots,w_{t+k}$ are the context words, d is the paragraph vector, T is the total number of words in the paragraph, and k denotes the window size.

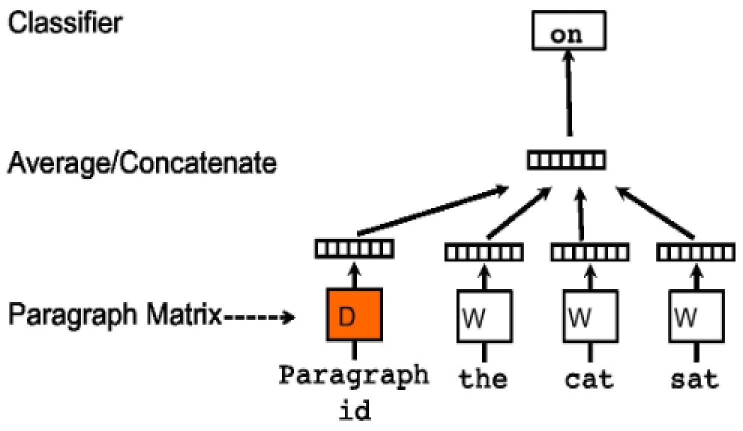


Figure 6, except additional paragraph token mapped to vector via matrix D

The PV-DBOW model as shown in Figure 8, on the other hand, is conceptually similar to the Skip-gram model in Word2Vec. It learns paragraph vectors by training a neural network to predict a target word using only the paragraph vector. The objective function for PV-DBOW is defined as:

$$\frac{1}{T}\sum_{t=1}^{T}\log p(w_t|d)$$

where w_t is the predicted word and d represents the paragraph vector.

Just like DBOW, paragraph vector here is being used to predict words in a given context
To obtain the paragraph vector for a new, unseen paragraph, a gradient descent optimization is performed to find the vector D that maximizes the log probability of the words in the paragraph, while keeping the word vectors W, softmax parameters U, and bias b fixed. This inference step is described by the following equation:

$$\frac{1}{N}\sum_{i=1}^{N}\log p(w_i|d)$$

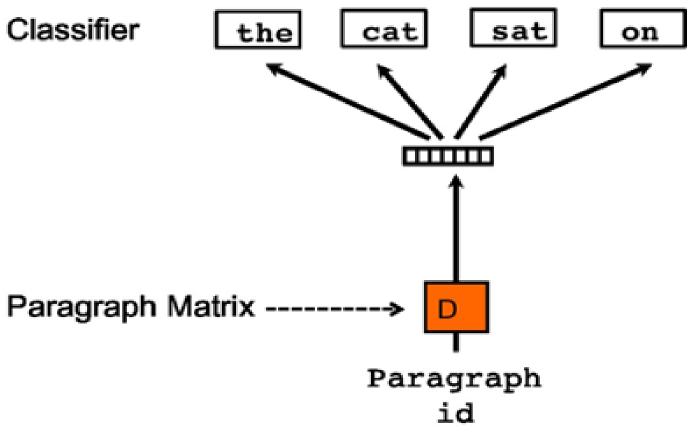
where w_i are the words in the new paragraph, and N is the total number of words in the paragraph.
The softmax function is employed to calculate the probability of a context word given the paragraph vector and word vectors:

$$p(w_t|w_{t-k},\ldots,w_{t+k},d)=\frac{e^{y_{w_t}}}{\sum_i e^{y_i}}$$

where y_i represents the unnormalized log-probability for each output word i, computed as:

$$y=b+Uh(w_{t-k},\ldots,w_{t+k},d)$$

with U and b being the softmax parameters and h being constructed by concatenating or averaging the word and paragraph vectors.



The training process of the Doc2Vec model involves optimizing the objective functions using stochastic gradient descent to learn the paragraph and word vectors that best capture the semantic meaning of the input text.
After pre-processing, our dataset contains around 467 tags (or tokens) per document on average for 9755 documents and total of 4.46 million tokens. Accordingly vector size of value 300 was chosen, minimum count for a word to be considered to be embedded was 2 and the model was trained for 150 epochs.

Collaborative-Based Filtering (CF)

In most platforms and their datasets, it is observed that the number of users far exceeds the number of items listed on that platform. From the EDA of the *Movie Lens's ml-25m* dataset, it is evident that the number of users is substantially higher. Consequently, item-based filtering is more appropriate. This approach is more effective because the average rating of an item tends to be more stable compared to the average rating given by a user to different items. Additionally, item-based filtering performs better in the context of sparse matrices, which is a common characteristic of these datasets, as not all users rate all movies and vice versa.

For this task, a special matrix is formed, where each row represents an item (in this case, a movie) and each column represents a user. The item-user matrix, denoted as R. This matrix captures the ratings given by users to various items. The dimensions of the matrix are $m \times n$ where m is the number of items (movies) in the

dataset and n is the number of users in the dataset.
The item-user matrix R can be represented as:

$$R = \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ r_{21} & r_{22} & \cdots & r_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ r_{m1} & r_{m2} & \cdots & r_{mn} \end{bmatrix}$$

Here, r_{ij} represents the rating given by user j to item i. If user j has not rated item i, then r_{ij} is typically set to 0 or left as a missing value.
There are two fundamental strategies in CF, memory based and model based. Memory-based collaborative filtering techniques utilize the entire item-user interaction matrix to generate recommendations. They rely on the similarity between users or items to make predictions.

2.4.1.1 Item based filtering using Cosine Similarity

In this approach, the similarity between items is calculated based on the ratings given by users. Each item is represented as a vector in the user-rating space, where the dimensions correspond to users, and the values are the ratings given by those users. The cosine similarity metric is then employed to measure the similarity between item vectors.
The cosine similarity between two items i and j is defined as:

$$sim(i,j) = \frac{\vec{i} \cdot \vec{j}}{\| \vec{i} \| \| \vec{j} \|} = \frac{\sum_{u \in U} r_{u,i} r_{u,j}}{\sqrt{\sum_{u \in U} r_{u,i}^2} \sqrt{\sum_{u \in U} r_{u,j}^2}}$$

where $\vec{i}$ and $\vec{j}$ are the item vectors, U is the set of users who have rated both items i and j, and $r_{u,i}$ and $r_{u,j}$ are the ratings given by user u to items i and j, respectively.
Once the similarities are computed, we can get any k items that are most similar to the target item by sorting the similarity vector and finding the k most highest similar items. The first item should be discarded as the most similar item to every other item in the matrix is the target item itself which has the highest value of similarity as the cosine angle between the identical vectors is 0 and $cos(0) = 1$,where$\theta = 0$.

2.4.1.2 k-Nearest Neighbors (KNN)

The k-Nearest Neighbors (KNN) algorithm is another memory-based approach that finds the k most similar users or items to make recommendations. KNN identifies the k most similar items to a target item based on a similarity metric, such as cosine similarity, manhattan distance, euclidean distances, etc.
The predicted rating $\hat{r}_{u,i}$ for user u on item i can be calculated as:

$$\hat{r}_{u,i} = \frac{\sum_{j \in N_i(u)} s\,im(i,j) \cdot r_{u,j}}{\sum_{j \in N_i(u)} s\,im(i,j)}$$

where $N_i(u)$ denotes the set of k most similar items to item i that user u has rated, $sim(i,j)$ is the similarity between items i and j, and $r_{u,j}$ is the rating given by user u to item j. KNN is a simple yet effective method for collaborative filtering, as it considers the local neighborhood of similar items to make recommendations. In other words, it is similar to the first approach, where it calculates the similarity (say in cosine distances) and finds the k most nearest neighbors or as in the first approach, k most similar items that were close. Because this is a supervised learning algorithm, calculating and comparing accuracy metrics with other model based approaches is quite straightforward and is the primary reason to be included in this study.
Model-based collaborative filtering techniques learn a predictive model from the user-item interaction data to make recommendations. These methods aim to uncover latent factors or patterns in the data.

2.4.2.1 Singular Value Decomposition (SVD)

SVD is a matrix factorization technique and it decomposes the user-item interaction matrix into lower-dimensional latent factor matrices. SVD aims to uncover latent factors that capture the underlying preferences of users and the characteristics of item. Given a user-item interaction matrix R, SVD decomposes it into three matrices:

$$R \approx U\Sigma V^T$$

where U is a matrix of user latent factors, Σ is a diagonal matrix of singular values, and V is a matrix of item latent factors.The predicted rating $\hat{r}_{u,i}$ for user u on item i can be calculated as:

$$\hat{r}_{u,i} = \mu + b_u + b_i + q_i^T p_u$$

where μ is the global mean rating, b_u and b_i are the user and item biases, respectively, and q_i and p_u are the latent factor vectors for item i and user u, respectively.

U:This matrix contains the left singular vectors of the original matrix R. In a recommender system, the columns of U represent the latent features of the users. These features are abstract dimensions that capture the preferences and behaviors of the users. The left singular vectors are orthogonal to each other, meaning they are independent and do not overlap in the information they represent.

$\mathbf{V^T}$:This matrix contains the right singular vectors of R, and it is the transpose of matrix V. In a movie recommender system, the rows of V^T (or columns of V) represent the latent features of the movies. These features might correspond to genres, themes, or other characteristics that define the movies. Similar to U, the right singular vectors are also orthogonal to each other.

$\mathbf{\Sigma}$:This is a diagonal matrix containing the singular values of the original matrix R. The singular values are non-negative and are typically arranged in descending order. They indicate the strength or importance of the corresponding latent features in U and V^T. In a recommender system, larger singular values correspond to more significant latent features that capture more of the variability in the user-movie ratings. These the singular values in Σ are the square roots of the eigenvalues of MM^T or M^TM

SVD Geometry Explanation

1. **First Rotation**: The original matrix R is rotated by the orthogonal matrix V (or V^T when considering the transpose). This rotation aligns the axes of the data with the directions of maximum variance.

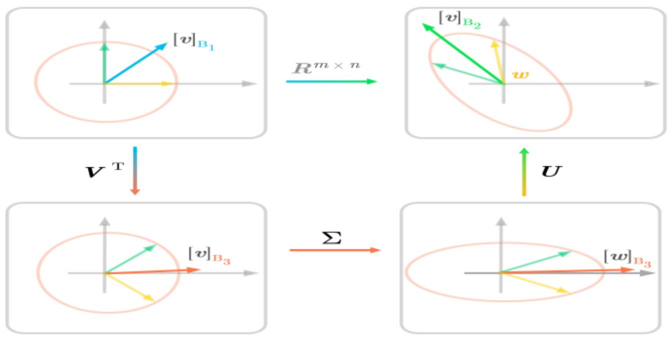


Figure 9: SVD Geometry Explanation

2. **Scaling**: The rotated matrix is then scaled along the axes by the singular values in Σ. This scaling stretches or shrinks the data along each axis according to the importance of each latent feature.

3. **Second Rotation**: Finally, the scaled matrix is rotated by the orthogonal matrix U. This rotation brings the data into the space where the latent user features are aligned with the axes.

Consider an example where we have a user-movie rating matrix R:

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

Using SVD, we decompose R into U, Σ, and V^T:

$$U = \begin{bmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \\ u_{31} & u_{32} \\ u_{41} & u_{42} \\ u_{51} & u_{52} \end{bmatrix}, \Sigma = \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix}$$

and

$$V^T = \begin{bmatrix} v_{11} & v_{12} & v_{13} & v_{14} \\ v_{21} & v_{22} & v_{23} & v_{24} \end{bmatrix}$$

To predict the rating of user 1 for movie 3, we compute:

$$\hat{R}_{13} = (U_1 \Sigma) \cdot (V_3^T)^T$$

SVD has been widely used in collaborative filtering due to its ability to capture latent factors and provide accurate recommendations.

2.4.2.2 Neural Collaborative Filtering (NCF)

Neural Collaborative Filtering (NCF) is a deep learning-based approach that leverages neural networks to model user-item interactions. NCF combines the strengths of matrix factorization and deep learning to learn complex non-linear interactions between users and items.

While SVD is effective for capturing the global structure of the interaction data through linear decompositions, it falls short in scenarios where the interactions have complex, non-linear patterns. NCF, by employing deep learning, can learn these non-linarites and thus, can potentially provide more accurate recommendations. Moreover, SVD can suffer from issues like overfitting especially with a higher number of latent factors in sparse datasets, a limitation that NCF addresses with its capacity to generalize better through learned non-linear mappings.

Neural collaborative filtering framework

The general architecture (Figure 10)of NCF consists of two main components:

1. Embedding layers: Users and items are represented as dense vectors in a low-dimensional latent space. The embedding layers learn these latent representations.

2. Neural network layers: The user and item embedding are fed into a multi-layer perceptron (MLP) or other neural network architectures to capture the complex interactions between users and items.

The predicted rating $\hat{r}_{u,i}$ for user u on item i is obtained by:

$$\hat{r}_{u,i} = f(p_u, q_i | \Theta)$$

where f is the neural network function, p_u and q_i are the latent factor vectors for user u and item i, respectively, and Θ represents the model parameters.

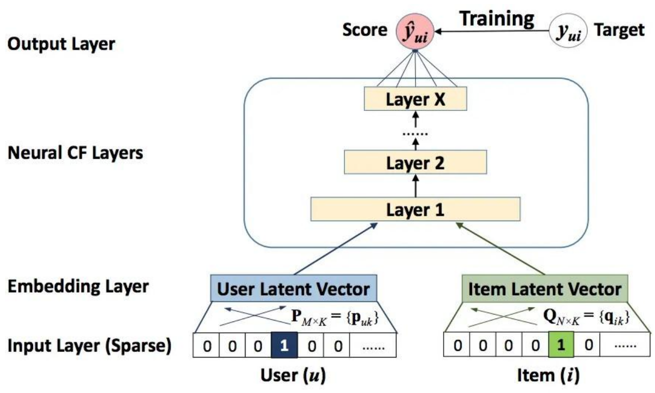


Figure 10: Neural collaborative filtering framework

The architecture of NCF comprises two main components: a Generalized Matrix Factorization (GMF) model and a Multi-Layer Perceptron (MLP). These components are fused to form the final prediction model into NeuMF layer.

1. **GMF Component**: This component generalizes the matrix factorization technique by replacing the inner product with a neural architecture that can learn an arbitrary function from data.

$$\phi_{GMF}(u, i) = p_u^T q_i$$

where p_u and q_i are the latent vectors for user u and item i, respectively.

2. **MLP Component**: This component learns the interaction function from data through multiple layers of non-linearities, enhancing the model's ability to capture complex user-item relationships.

$$\phi_{MLP}(u,i) = a^{(L)}\left(a^{(L-1)}(\ldots a^{(1)}(p_u, q_i)\ldots)\right)$$

where $a^{(l)}$ represents the activation function of the l-th layer, and (p_u, q_i) are the concatenated latent vectors of user and item.

The final layer of the NCF framework combines the outputs from the GMF and MLP components to predict the final interaction score:

$$\hat{y}_{ui} = \sigma(h^T[\phi_{GMF}(u,i), \phi_{MLP}(u,i)])$$

where σ is the sigmoid function ensuring the output is between 0 and 1, and h is a learnable vector that combines the two representations.

The NCF model utilizes a binary cross-entropy loss function, which is suitable for binary classification problems. The function L is defined as:

$$L = -\frac{1}{N}\sum_{(u,i)\in\mathcal{O}\cup\mathcal{O}^-} y_{ui}\log(\hat{y}_{ui}) + (1 - y_{ui})\log(1 - \hat{y}_{ui})$$

where:

N & No. of observed and sampled -ve interactions

& set of observed interactions. ^- & set of negative samples.

y_ui & binary label indicating whether user u interacted with item i (1 for observed interactions and 0 for negative samples). _ui & predicted probability of interaction

NCF has shown promising results in capturing complex user-item interactions and providing personalized recommendations.

```
Algorithm Hybrid Recommender System
Data: User's movie list L, Content-based filtering
      (CBF) movies CBF, Col- laborative
      filtering (CF) movies CF, Recommendations
      (all combined) $R_{ac}$
Result: Recommended movies $R^{1\times K}$
1  Function GetCBFMovies(movie):
2     return movie[CBF]
3  Function GetCFMovies(movie):
4     return movie[CF]
5  if L |= 0 then
      R ← Top N movies from CBF
      return R
6  else
7     Function CombineMovies(L):
8        $R_{ac} \leftarrow \emptyset$
         foreach movie ∈ L do
9           $M_{cbf}$ ← GetCBFMovies(movie)$M_{cf}$
            ← GetCFMovies(movie) $M_{all} \leftarrow$
            $M_{cbf} \cup M_{cf}$ $R_{ac}$.append($M_{all}$)
10       return $R_{ac}$
11    Function ProcessRecommendations($R_{ac}$):
12       for col ← 0 to $|R_{ac}[0]| - 1$ do
13          for row ← 0 to $|R_{ac}| - 1$ do$M_{details}$
               ← $R_{ac}$[row][col]
               R.append($M_{details}$)
14          if R | ≥ K then
15             break
16       return R
```

Hybrid Recommender System
The proposed algorithm creates personalized recommendations for user based on their interests. It combines recommendations from two different methods: Content-Based Filtering (CBF) and Collaborative Filtering (CF). For each movie in the user's list, it retrieves recommendations using both CBF and CF methods as list and are

These recommendations are then combined into a single list of recommended movies in the proposed way by traversing the list R_{ac} column wise picking up movies from each row. Conversly a transpose of list R_{ac} as R_{ac}^{T} can also be taken. Then Top K movies can be returned accordingly. If no user data is available and $|L| = 0$ then this is a cold start problem. To solve this, we retrieve the most popular movies from our dataset. Top movie recommendations from different genres can also be retrieved. This is the benefit of using NLP as content based filtering method to address cold start problem. To Optimize the current solution, the CBF and CF movies can be cached or pre-computed and stored directly into database until there are no major changes to the dataset.

III. Results

For both collaborative(cf) and content-based(cbf) filtering methods have been used many different techniques until now. Now to compare the respective methodologies, we will try to infer some results to see how they actually perform. To maintain the diversity amongst the recommendations, we have trained cf and cbf on different types of datasets but on same items.

Content-Based Filtering
Vector embedding created using TFĪDF and Doc2Vec were used to calculate the similarity between each items. A UMAP was plotted. Uniform Manifold Approximation and Projection (UMAP) is a dimension reduction technique that can be used for visualization similarly to t-SNE, but also for general non-linear dimension reduction.

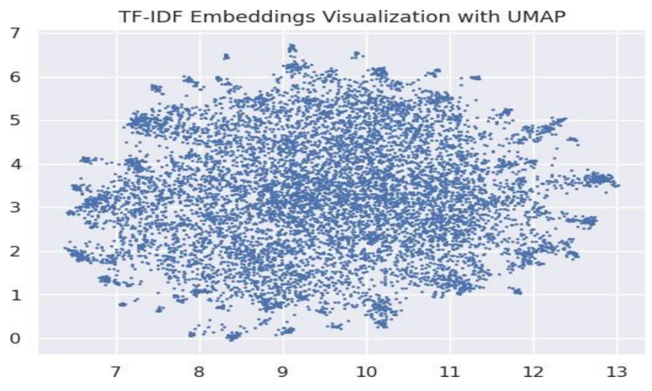


Figure 11: UMAP for Cosine Similarity ofTF-IDF

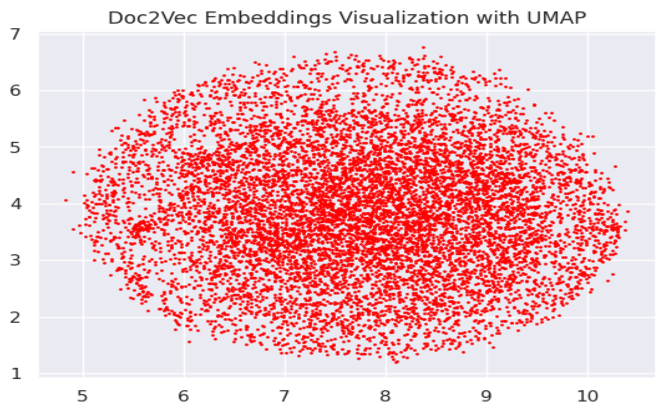


Figure 12: UMAP for Cosine Similarity of Doc2Vec

From Figure 11 and Figure 12 we can observe that the Doc2Vec embedding exhibit a more coherent and meaningful clustering of documents, suggesting that similar documents are grouped more closely together. This indicates that Doc2Vec is better at capturing the semantic relationships between documents. The TF-IDF embedding, on the other hand, lack this level of structure and organization, implying that while they capture semantic similarities, they do not group similar documents as effectively.

Movie title	**Score**
The Empire Strikes Back	0.4546
Star Wars: The Force Awakens	0.3989
Return of the Jedi	0.3540
Solo: A Star Wars Story	0.2501
Barbie and the Magic of Pegasus	0.1995
Star Wars: The Last Jedi	0.1901
Star Wars: The Rise of Skywalker	0.1697
Blood Diamond	0.1690
Rebel Moon - Part Two: The Scargiver	0.1675
Star Wars: Episode I - The Phantom Menace	0.1658

Table 2: Movies Similar to Star Wars using TFIDF

Highly Relevant Titles: The movies directly related to ’Star Wars’ such as "The Empire Strikes Back", "Star Wars: The Force Awakens", "Return of the Jedi", "Solo: A Star Wars Story", "Star Wars: The Last Jedi", "Star Wars: The Rise of Skywalker", and "Star Wars: Episode I - The Phantom Menace" have varying degrees of similarity scores. Notably, "The Empire Strikes Back" has the highest similarity score of 0.4546 (in table Score column means Similarity Score).
Less Relevant Titles: "Barbie and the Magic of Pegasus", "Blood Diamond", and "Rebel Moon - Part Two: The Scargiver" are less relevant to the ’Star Wars’ theme, indicating potential noise in the recommendations.

UMAP for Cosine Similarity of Doc2Vec

Movie title	**Score**
The Empire Strikes Back	0.5855
Star Wars: Episode III – Revenge of the Sith	0.5041
Conquest of the Planet of the Apes	0.5026
Legend	0.4975
Star Wars: Episode I - The Phantom Menace	0.4937
Star Trek: The Motion Picture	0.4770
Zack Snyder’s Justice League	0.4734
Return of the Jedi	0.4703
Aladdin	0.4694
The 7th Voyage of Sinbad	0.4621

Table 3: Movies Similar to Star Wars using Doc2Vec

Highly Relevant Titles: This table also includes several direct ’Star Wars’ sequels and related movies such as "The Empire Strikes Back", "Star Wars: Episode III - Revenge of the Sith", "Star Wars: Episode I - The Phantom Menace", and "Return of the Jedi", all scoring higher than in Table 1.

Less Relevant but Thematically Similar Titles: Titles like "Conquest of the Planet of the Apes", "Legend", "Star Trek: The Motion Picture", and "Zack Snyder’s Justice League" suggest a broader interpretation of similarity, focusing perhaps on sci-fi and epic adventure genres, which align well with ’Star Wars’.
Comparative Evaluation
Relevance and Precision: Table 2 generally shows higher similarity scores for movies that are directly related to ’Star Wars’ compared to Table 1. This indicates a potentially better precision in capturing the thematic essence of ’Star Wars’.
Contextual and Genre Alignment: Table 2, while including some less directly related titles, maintains a strong alignment with the sci-fi and adventure genres, which are central to ’Star Wars’. This suggests a broader but still relevant recommendation scope.

Collaborative Filtering
As discussed, the results from simply applying cosine similarity and using neighbour-based solutions like KNN yeild the same results. So only KNN will considered for memory based method and will be compared to SVD and NCF model based solutions.
Formulas for calculating Precision@K, Recall@K, MAP, NDCG, MAE, and RMSE used in analysis.

1. Precision@K

$$\text{Precision@K} = \frac{1}{K}\sum_{i=1}^{K} \text{rel}\,(i)$$

where $\text{rel}(i)$ is an indicator function that equals 1 if the item at rank i is relevant, and 0 otherwise.

2. Recall@K

$$\text{Recall@K} = \frac{\sum_{i=1}^{K} \text{rel}\,(i)}{\text{Total \#of relevant items}}$$

3. Mean Average Precision (MAP)

$$\text{AP@K} = \frac{\sum_{k=1}^{K} \left(\text{Precision@k} \times \text{rel}(k)\right)}{\text{Total \#of relevant items}}$$

$$\text{MAP@K} = \frac{1}{N}\sum_{j=1}^{N} \text{AP@K}_j$$

where N is the number of queries.

4. Normalized Discounted Cumulative Gain (NDCG)

$$\text{DCG@K} = \sum_{i=1}^{K} \frac{2^{\text{rel}(i)} - 1}{\log_2(i+1)}$$

$$\text{IDCG@K} = \sum_{i=1}^{K} \frac{2^{\text{rel}^*(i)} - 1}{\log_2(i+1)}$$

$$\text{NDCG@K} = \frac{\text{DCG@K}}{\text{IDCG@K}}$$

where $\text{rel}^*(i)$ is the ideal ranking of the relevance scores.

5. Mean Absolute Error (MAE)

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i|$$

where y_i is the actual value and $\hat{y}_i$ is the predicted value.

6. Root Mean Squared Error (RMSE)

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}$$

This dataset was divided into 5 Folds. A "fold" refers to a subset of the data used in cross-validation. Cross-validation is a technique used to assess the performance of a model by dividing the data into multiple subsets (folds) and then training and testing the model multiple times, each time using a different fold as the test set and the remaining folds as the training set.
In 5-fold cross-validation, the dataset is divided into 5 equal parts (folds). The model is trained and tested 5 times, each time using a different fold as the test set and the remaining 4 folds as the training set. The performance metrics are then averaged over the 5 folds to provide a final evaluation.

Figure 13: Precision@k for KNN and SVD at different values of k

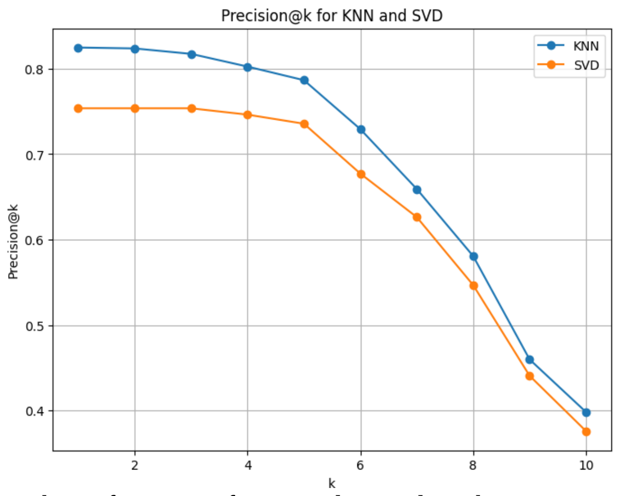


Figure 13 illustrate the performance of KNN and SVD algorithms in a recommendation system context, measured by precision@k and recall@k metrics as the number of top recommendations (k) increases. In the first graph, both algorithms show a decline in precision as k increases, with KNN maintaining a higher precision across all values of k compared to SVD.
Figure 14 depicts a similar trend for recall, with KNN consistently achieving higher recall rates than SVD as k increases. These trends suggest that KNN may be more effective at identifying a smaller, more accurate set of recommendations, while SVD may be Figure 14: Recall @k for KNN and SVD at different values of k
befit from improvements in recall. Despite the visual data, it's important to note that overall, SVD is reported to outperform KNN in terms of RMSE, MAE, and average precision and recall, indicating that SVD may be more reliable for predicting user preferences across the entire dataset.

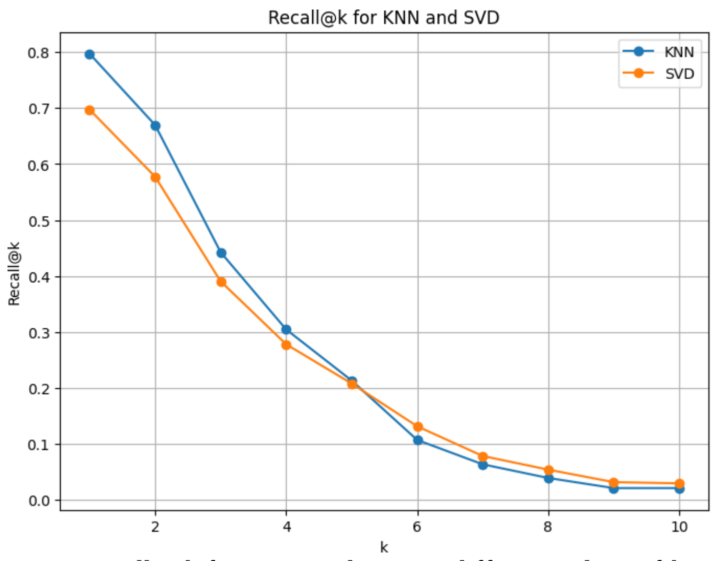


Figure 14: Recall @k for KNN and SVD at different values of k

Note: MAP@k and NDCG@k remains constant ∀ k
The NCF model outperforms both KNN and SVD in terms of RMSE and MAE, indicating that it provides more accurate predictions. However, when it comes to ranking metrics such as MAP, NDCG, Precision@K, and Recall@K, KNN and SVD outperform NCF significantly.

Error Metrics

Metric	KNN	SVD	NCF
RMSE	0.9791	0.9363	0.7301
MAE	0.7734	0.7384	0.5970
Fit Time (s)	0.32	1.16	284.9

Predict Time (s)	3.77	0.12	12.13

Ranking Metrics

Metric	KNN	SVD	NCF
MAP	0.481637	0.465394	0.050512
NDCG	0.986214	0.987275	0.201280
Precision@K	0.691848	0.633342	0.180594
Recall@K	0.283956	0.264078	0.103128

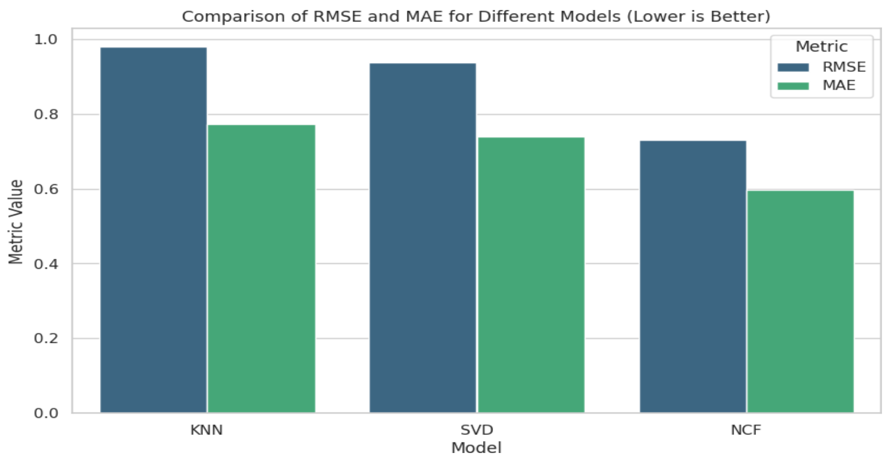


Figure 15: Error Metrics Comparison betweenKNN, SVD & NCF

IV. Conclusion

Based on the analysis, Doc2Vec appears to be more effective in recommending movies that are not only part of the 'Star Wars' series but also include other movies within the sci-fi and epic adventure genres. This model shows a higher degree of understanding of the content and context of 'Star Wars', making it a better choice for users interested in this genre. The higher similarity scores across relevant titles also suggest a more robust model performance in capturing the nuances of 'Star Wars'-related content. This comparison was done by OpenAI's GPT-4o (GPT-4 Omni), GPT-4 Turbo and Anthropic's Claude 3 Opus with access to real time internet connection for authentic results.

In conclusion, when comparing the performance of KNN, SVD, and NCF on the MovieLens dataset using various metrics, we observe that NCF outperforms the other two models in terms of RMSE and MAE, indicating better accuracy in rating predictions. However, KNN and SVD exhibit superior performance in ranking metrics such as MAP, NDCG, Precision@K, and Recall@K . This suggests that while NCF excels at minimizing the error between predicted and actual ratings, KNN and SVD are more effective at ranking items and providing top-K recommendations that align with user preferences.

The discrepancy in performance can be attributed to the inherent differences in the models' ability to capture latent features and nonlinear relationships. KNN relies on similarity between users or items, making it effective at capturing local interactions but limited in its ability to uncover latent features. SVD, on the other hand, identifies latent factors but assumes linear relationships between users and items, which may not always hold true in real-world scenarios. NCF leverages deep learning to capture complex, nonlinear relationships, enabling it to achieve better rating prediction accuracy

REFERENCES

[1]. Mikolov, Tomas. (2013). Efficient Estimation of Word Representations in Vector Space. arXiv:1301.3781.
[2]. Le, Quoc V., & Mikolov, Tomas. (2014). Distributed Representations of Sentences and Documents. arXiv:1405.4053.
[3]. Chakraborty, Sarit. (2018). An Improved Text Sentiment Classification Model Using TF-IDF and Next Word Negation. arXiv:1806.06407.
[4]. He, Xiangnan, Liao, Lizi, Zhang, Hanwang, Nie, Liqiang, Hu, Xia, & Chua, Tat-Seng. (2017). Neural Collaborative Filtering. arXiv:1708.05031.
[5]. Son, Jieun & Kim, Sb. (2017). Content-Based Filtering for Recommendation Systems Using Multiattribute Networks. Expert Systems

with Applications. 89. 10.1016/j.eswa.2017.08.008.

[1]. F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Trans. Interact. Intell. Syst. 5, 4, Article 19 (January 2016), 19 pages. https://doi.org/10.1145/2827872

[2]. Wikipedia contributors. (2024, March 17). Tf–idf. In Wikipedia, The Free Encyclopedia. Retrieved 19:01, May 7, 2024, from https://en.wikipedia.org/w/index.php?title=Tf%E2%80%93idf&oldid=1214201167

[3]. Badugu, S., & Manivannan, R. (2023). K-Nearest Neighbor and Collaborative Filtering-Based Movie Recommendation System. In Computer Networks and Inventive Communication Technologies (pp. 141-150). Springer, Singapore.

[4]. Jena, K.K., Bhoi, S.K., Mallick, C., & Sahoo, S. (2022). Neural model based collaborative filtering for movie recommendation system. International Journal of Information Technology, 14, 2067-2077.

[5]. Lee, J.W. (2018). Content-Based Collaborative Filtering using Word Embedding: A Case Study on Movie Recommendation. International Conference on Platform Technology and Service (PlatCon).

[6]. Liu, G., & Wu, X. (2019). Using Collaborative Filtering Algorithms Combined with Doc2Vec for Movie Recommendation. 2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), 1461-1464.

[7]. Myriam (2022). KNN Movie Recommendation System. GitHub Repository.

[8]. Quek, A. (2015). Simple Movie Recommender Using SVD. Retrieved from https://alyssaq.github.io/2015/

[9]. Liu, G., & Wu, X. (2019). Using Collaborative Filtering Algorithms Combined with Doc2Vec for Movie Recommendation. 2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), 1461-1464.

[10]. Wikipedia contributors. (2024, May 2). Singular value decomposition. In Wikipedia, The Free Encyclopedia. Retrieved 12:37, May 17, 2024, from https://en.wikipedia.org/w/index.php?title=Singular_value_decomposition&oldid=1221850898

[11]. Abhishek Sharma. (2019, Dec 16). Neural Collaborative Filtering. From https://towardsdatascience.com/neural-collaborative-filtering-96cef1009401

[12]. Prince Grover (2017, Dec 29). Various Implementations of Collaborative Filtering. From https://towardsdatascience.com/various-implementations-of-collaborative-filtering-100385c6dfe0

[13]. Nigram, Nishant. (2021). Research Paper Analysis using Natural Language Processing. Technical University of Munich. Retrieved from https://www.cs.cit.tum.de/en/sccs/news/sccs-colloquium/article/nishant-nigam-research-paper-analysis-using-nlp-techniques/.

[14]. Kumar, Rajesh, Verma, Bhupendra K., & Rastogi, S. S. (2021). Social Popularity based SVD++ Recommender System. International Journal of Computer Applications, 87, 33–37. doi:10.5120/ijca2015906432.