Movie Recommendation System

Raksha Rawat, Mansi Kaushik, Dr. Poonam Chaudhary, Dr. Vidhi Khanduja

Department of Computer Science, The NorthCap University, Gurgaon, Haryana, India

Abstract

A recommendation system basically returns the content that would appear interesting to an individual. These recommendations can be based on a number of factors like content types, user choices, user patterns, ratings, etc. Broadly there are two types of recommendation techniques - Content Filtering and Collaborative Filtering. Both of these techniques use different approaches and come with their own set of advantages and disadvantages. In this paper, we draw a comparison between these two techniques and propose a third recommendation system - a hybrid recommendation system that uses the best of both of these techniques, utilizes their advantages, and minimizes their disadvantages. Our hybrid recommendation system aims to optimize and improve the quality of recommendations. For this purpose, we have created a Movie Recommendation System using 'The Movies Dataset' from Kaggle. Firstly, we created a content-based recommendation system by applying cosine similarity wherein cosine similarity uses TF-IDF vectorizer and count vectorizer. Next, we created a collaborative-based recommendation system that uses KNN, and matrix factorization methods like SVD, NMF, and ALS. Cosine Similarity using TF-IDF and SVD gave us the best results for the creation of a content-based recommendation system and a collaborative-based recommendation system respectively, so, we combined these two techniques to create a hybrid recommendation system. Our hybrid recommendation system gave us better results than the individual content-based recommendation system and collaborative-based recommendation as per the hypothesis and tackled the disadvantages like the cold start problem. It recommends movies to the user according to both movie content and user behaviour.

Keywords: Content-Based Recommendation, Collaborative Filtering Recommendation, Hybrid Recommendation, Cosine Similarity, Term Frequency-Inverse Document Frequency (TF-IDF), Count Vectorizer, K-Nearest Neighbors (KNN), Singular Value Decomposition (SVD), Non-negative Matrix Factorization (NMF), Alternating Least Squares (ALS), Surprise Library, Movielens

Introduction

Have you ever wondered how Amazon's homepage displays items and categories that you have already searched for before? how do Facebook and Instagram suggest interesting friends, pages, and content? how YouTube shows you content based on your previous search history, streaming, and sessions? What exactly goes behind all those suggestions? All these suggestions and content are determined by a recommendation system that takes in the user data, movies data, etc. processes it, determines what user will like the best, and then shows it as recommendations. Recommendation systems have a variety of use cases from social media to eCommerce platforms to music and video apps to gaming apps, all the ads while surfing websites are so similar to our taste because there is a recommendation engine working behind it that is taking in our tastes and interests and then showing us what we might like.

There are many approaches to creating a recommendation system. Different platforms use different techniques to create a recommendation system according to their needs and functionality. But broadly there are two types of recommendation systems, a content-based recommendation system, and a collaborative-based recommendation system. These recommendation systems can be created using machine learning algorithms, deep neural networks, etc. The purpose of this paper is to dive deep into the subject of recommendation systems, study them, and find out which methods give us the best recommendations that are most suited to a user's taste. For this purpose, we built three movie recommendation systems, a content-based recommendation system, a collaborative-based recommendation system, and a hybrid recommendation system based on the 'The Movies Dataset' which contains the metadata of over 45000 movies.

A content-based recommendation system recommends a movie to the user which is most similar to the movie user has searched for in terms of content e.g. related genre, cast, keywords, etc. Since it recommends movies based on movie metadata, it can recommend movies to the new users as well but the disadvantage here is that it does not capture a particular user's taste i.e., every user will get the same generalized recommendations. To implement this, we calculated the cosine similarity between movies based on the movie metadata using TF-IDF and Count Vectorizer. We compared the results of both TF-IDF and Count Vectorizer to see which technique gives us the best recommendations.

A collaborative recommendation system recommends movies to the user according to the user behaviour i.e., what kinds of movies the user likes, what ratings the user has already given to other movies, etc. It captures the user's tastes and interests on the basis of the ratings they have given to other movies, finds similar users to them, and then recommends movies liked by those other similar users to the user. However, it needs a lot of data of movies and users in order to provide efficient recommendations. Another disadvantage here is that it cannot accommodate new users i.e. if a new user asks for recommendations related to any particular movie, then it first needs to know the user's existing interests so that it can match the user to other similar users and provide recommendations, so in case of a new user, the recommendation system will fail. This issue is also known as the cold start problem. To implement this, we used K-Nearest Neighbors (KNN) algorithm and matrix factorization methods like Singular Value Decomposition (SVD), Non-negative Matrix Factorization (NMF), and Alternating Least Squares (ALS). We compared the results of all of these algorithms on the basis of Root Mean Square Error (RMSE) to see which technique gives us the best recommendations.

We want to create a system that can recommend movies accurately to every user according to his/her personal taste or choice, be it old users or new users. Therefore, we need the features of a content-based recommendation system and a collaborative-based recommendation system as well. In order to achieve this, we created a hybrid recommendation system that first filters out the movies based on cosine similarity (content-based recommendations), estimates the rating the user will give to each movie (collaborative-based recommendations), sorts out the predicted rating in descending order, and then lists the top 10 of those movies as recommendations to the user. In this way, we recommend movies to the user on the basis of both movie content and personal taste. Through this approach, we used both content-based filtering and collaborative filtering methods to improve the quality of recommendations, minimize their disadvantages, and eliminate problems like the cold-start problem.

Design Methodology

Below is a diagrammatic representation of our project:

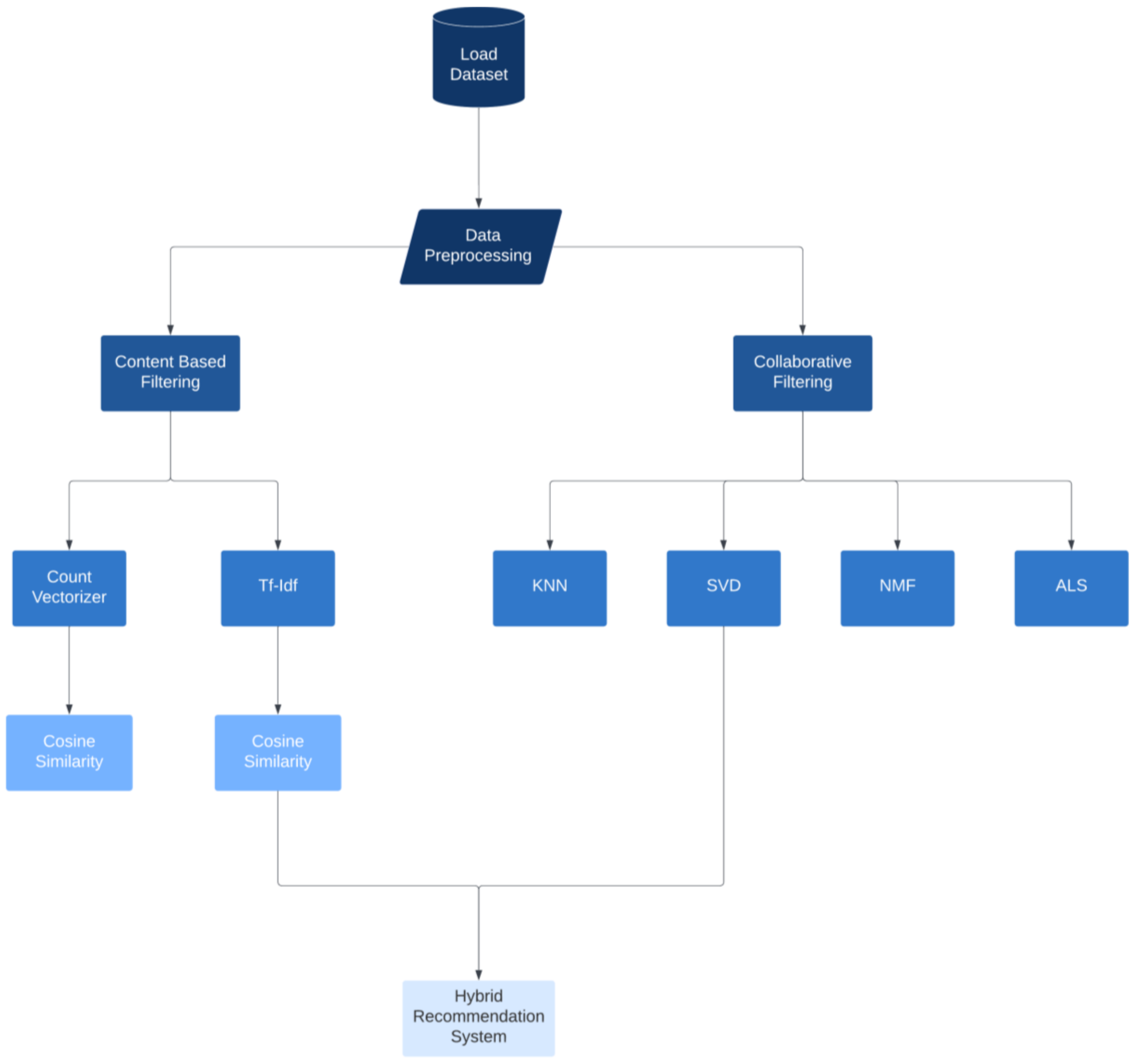


Figure 1: Design Methodology of our Recommendation System

Firstly, we load and pre-process 'The Movies Dataset'. Then, we create a content-based recommendation system based on movie metadata like genre, keywords, caste, etc. We calculate the cosine similarity between movies based on the Term Frequency-Inverse Document Frequency (TF-IDF) matrix and Count Vectorizer matrix and compare both approaches. Then, we create a collaborative-based recommendation system with different approaches like K-Nearest Neighbors (KNN), Singular Value Decomposition (SVD), Non-negative Matrix Factorization (NMF), and Alternating Least Squares (ALS) to predict the movie ratings a user will give to a particular movie based on his/her previous ratings. For the creation of our hybrid recommendation system, we compared all the content filtering techniques and the collaborative filtering techniques and went ahead with cosine similarity using the TF-IDF matrix and SVD as they gave us the best results in the respective content-based recommendation system and collaborative-based recommendation system. At last, we combined cosine similarity based on the TF-IDF matrix and SVD to create a hybrid recommendation system.

Implementation

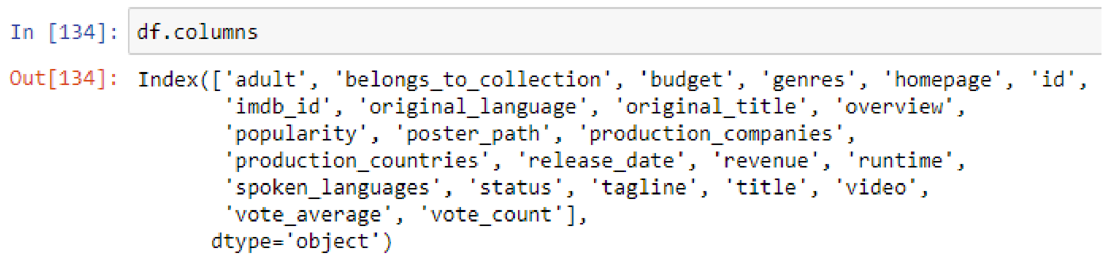
1) Dataset

For this project, we have taken 'The Movies Dataset' from Kaggle.

Dataset Link: https://www.kaggle.com/datasets/rounakbanik/the-movies-dataset

The dataset contains metadata of 45000 movies from the full MovieLens dataset. MovieLens is a movie recommendation service by GroupLens, a research lab at the University of Minnesota. The Movielens dataset contains the data of over 20 million movies along with the ratings given to them by different users. 'The Movies Dataset' from Kaggle is a subset of 45000 movies from MovieLens dataset. The Movies Dataset contains the following files:

a) movies_metadata.csv: It is the main file that contains movie metadata. It contains 45,000 movies from the full MovieLens dataset. It contains the following features:



```
In [134]: df.columns

Out[134]: Index(['adult', 'belongs_to_collection', 'budget', 'genres', 'homepage', 'id',
                 'imdb_id', 'original_language', 'original_title', 'overview',
                 'popularity', 'poster_path', 'production_companies',
                 'production_countries', 'release_date', 'revenue', 'runtime',
                 'spoken_languages', 'status', 'tagline', 'title', 'video',
                 'vote_average', 'vote_count'],
               dtype='object')
```

Figure 2: Features of movie_metadata dataset

b) keywords.csv: It contains keywords related to the movie plot and story. It is in the form of a stringified JSON Object.

c) credits.csv: It contains cast and crew information of the movies. It is in the form of a stringified JSON Object.

d) links.csv: It contains TMDB and IMDB ids of the movies

e) links_small.csv: It is a subset of link.csv. It contains TMDB and IMDB ids of 9125 movies from the links.csv file. It is used to test the code on a smaller dataset.

f) ratings_small.csv: It is the subset of 100,000 ratings from 700 users on 9,000 movies. The ratings range from 1 to 5.

In this project, we have used movies_metadata.csv, keywords.csv, credits.csv, links.csv, links_small.csv, and ratings_small.csv files.

2) Data Pre-processing

a) **Merge datasets**: In order to merge different datasets files together, we converted the id column of all datasets to int format, dropped the rows that contained null values in the 'id' column, and merged movies_metadata.csv, keywords.csv, and credits.csv datasets together based on the ‘id’ column.

b) **Drop duplicates in the dataset**: We searched for duplicate rows and duplicates in the 'title' column. In the 'title' column, there are some movie titles that are repeating, so we have only kept the first row that contains the duplicate title and removed the rest of the rows.

c) **Filter movies that are present in the links.csv file**: In order to build a recommendation system, we will only keep the movies that are present in the links.csv file

d) **Genres Pre-processing**:

   i) We need movie genres to recommend movies but the 'genres' column is present in a stringified JSON format. So, we read the genres column, retrieved the genres, and stored them in a list.

e) **Director Pre-processing**:

   i) The 'crew' column contains the crew names and information like names, id, gender, etc. of writers, directors, etc. in a JSON format. So we read the 'crew' column, retrieved the director names, and stored them in a list.

   ii) We stored director names two times in the list as we want director names to be more important and considered in the recommendation system.

f) **Cast Pre-processing**:

   i) The 'cast' column contains all the cast information in JSON format. So we read the 'crew' column, retrieved all the actor names, and stored them in a list.

   ii) We selected only the top 3 actors from the cast as they are the most important.

   iii) We converted 'cast' to lowercase

g) **Keywords Pre-Processing**:

   i) The 'keywords' column contains all the tags and keywords in JSON format. So we read the 'keywords' column, retrieved all of the keywords, and stored them in a list.

   ii) Further, we try to find out the base word for all the keywords, for this purpose we used the following techniques:

(1) **Stemming:** Stemming is the process of reducing the word to its root/base word. We imported Snowball Stemmer and applied it to perform stemming on keywords.

(2) **Wordnet Lemmatizer**: Lemmatization is the process of finding the root form of a word that has a morphological meaning. We imported Wordnet Lemmatizer and applied it to perform lemmatization on keywords.

(3) **Lemmatization using spaCy**: spaCy is an open-source library that parses and understands text data. It gives us additional metadata along with root words like Part-of-Speech (POS) tags. We imported the spaCy library, applied it to keywords, and extracted doc_id, token, and pos tags along with the lemma.

The difference between stemming and lemmatization is that in stemming it does not matter whether the root word has a morphological meaning whereas lemmatization searches and returns the dictionary form of the word that has a morphological meaning.

E.g. In stemming, we define a rule that removes the last character of the word and gives 'mice' as input. So, on performing stemming, we receive output 'mic' whereas on performing lemmatization, we receive output 'mouse' Here, in stemming the whole meaning of the word has changed whereas lemmatization brings the word 'mice' to its root form 'mouse' without changing the meaning of the word.

Lemmatization using spaCy gives us an advantage over lemmatization and stemming as it provides us with POS tags of words through which we can filter out the words with unwanted pos tags. So, for further implementation, we used the keywords derived from spaCy.

iii) We only kept keywords that had noun, adj, propn, X, adv, intj as pos tags

iv) We merged different rows on basis of 'doc_id'

v) We added a column 'doc_id' in the movies dataset that has the index value

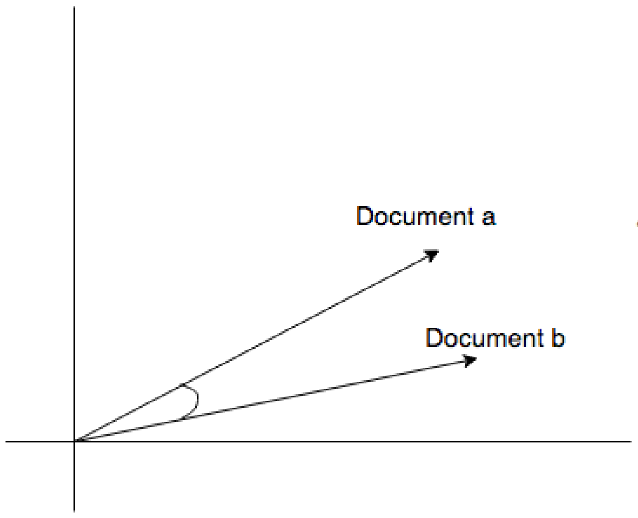
vi) We merged spacy results to our movies dataset on basis of 'doc_id'

h) Combine required movie metadata together:

We combined the retrieved director names, actor names, genres, and keywords and stored them as strings in a new column 'combination'. This will be our input to the content-based recommendation system.ratings_small.csv dataset does not need any preprocessing.

3) Content-based recommendation system

To create a content-based recommendation system, we used cosine similarity. Cosine similarity is used to calculate how similar two items are. It represents the angle between two vectors in multi-dimensional space. The output values come from 0 to 1. 0 means there is no similarity between the two items whereas 1 means both items are 100% similar.



$$sim(a,b) = cos\theta = \frac{\vec{a}.\vec{b}}{\parallel \vec{a} \parallel \parallel \vec{b} \parallel}$$

Figure 3: Cosine Similarity

For our content-based recommendation system, we used TF-IDF and count vectorizer matrices as input to calculate cosine similarity.

a) **Using Term Frequency-Inverse Document Frequency (TF-IDF) Vectorizer:**
Firstly, we created a content-based recommendation system that takes in the TF-IDF matrix as input to calculate cosine similarity. TF-IDF stands for Term Frequency-Inverse Document Frequency. It is used to calculate the relativity of words across all the documents. It counts the words in the document, and stores this information, along with how many times that word has occurred in other documents. It is helpful in calculating cosine similarity as it stores the frequency and relevance of the word among the whole corpus. After using the TF-IDF vectorizer we calculated the dot product between different movies to calculate the cosine similarity.

b) **Using Count Vectorizer:**
Then, we created a content-based recommendation system that takes in the count vectorizer matrix as input to calculate cosine similarity. Count vectorizer counts how many times a word has come in the document. We created a count vectorizer matrix and then used cosine similarity to calculate the similarity between documents.

After creating different matrices and calculating cosine similarity, we create a function that takes in a movie title and based on the method we specify, returns the top 10 movies that have the highest cosine similarity with the input movie as recommendations to the user.
Results with both TF-IDF and count vectorizer matrices were similar, but we chose TF-IDF over the count vectorizer as it not only provides the importance of the words but in our case, also takes less time to compute. TF-IDF focuses on the frequency of words as well as the importance of words, unlike count vectorizer. Moreover, after getting the TF-IDF matrix, we can simply calculate the dot product using sklearn's linear_kernel to calculate cosine similarities. This function is much faster than the

cosine_similarity function to calculate cosine similarity, thus saving a lot of time, and is suitable for large datasets as well. Therefore, we will proceed with cosine similarity using TF-IDF vectorizer for the hybrid recommendation system.

4) Collaborative based recommendation system:

To create a collaborative-based recommendation system we used the ‘ratings_small.csv’ file. It contains the rating different users have given to different movies ranging from 1 to 5. Using this dataset, we predicted what ratings the user will give to other movies using the following approaches:

a) Non-Parametric Approach:

i) K-Nearest Neighbors (KNN):

KNN is a supervised machine learning algorithm that is used for classification and regression. It considers the K nearest neighbors to predict a class or value for new input. It classifies or predicts the value of the new input on the basis of its similarity to the other data points in the dataset.

We used KNN to predict what rating the user will give to a movie. We used functions from the Surprise library to implement this. We ran a cross validation procedure on the dataset with the algorithm as KNNBasic, performance measure to compete as RMSE, and the number of folds as 5. It shuffled our dataset randomly, divided it into 5 groups, kept 1 group as the test set, and the other 4 groups as the training set, then fit the KNN model with the default values of KNNBasic on the training set, evaluated the model on the test set, and then summarized the model using RMSE scores, test time, and train time of each fold along with their mean and standard deviation. We got a mean RMSE score of 0.9677 when we used the default values of KNNBasic, i.e., k = 40, min_k = 1, sim_options = {'name': 'MSD', 'user_based': 'True'}. These values interpret the following:

(1) k: maximum number of neighbors for aggregation. The default value is 40
(2) min_k: minimum number of neighbors for aggregation. The default value is 1
(3) sim_options stands for similarity measures, i.e. on what basis the similarity will be calculated between different data points. 'MSD' is Mean Squared Difference, so it will compute the Mean Squared Difference between items or users, 'user_based' is a bool value that represents whether the similarity will be calculated between users or items. user_based = True means similarity will be calculated between different users.

Further, in order to reduce the RMSE score, we did hyperparameter tuning. We used GridSearchCV to test the KNN model on different values of k and min_k. We used GridSearchCV to run a cross validation procedure with 5 folds, algorithm as KNNBasic, measure as RMSE, and the range of parameters as k = [10, 15, 20, 25, 30, 35, 40], and min_k = [1, 2, 3, 4, 5, 6], keeping the rest of the values of KNNBasic as default. It will make all the combinations with the different values of k and min_k specified, and then run the cross validation procedure on each of those combinations using the KNNBasic algorithm. We used it to improve our RMSE score and find out which combination gave us the best RMSE score. It returned the best combination as {'k': 15, 'min_k': 3} and the best RMSE Score as 0.9538736787202492. Using Parameter Tuning improved the RMSE score by 0.014. Therefore, we trained our KNN model on the new set of parameters to predict the ratings.

b) Matrix Factorization Approaches:

In simple words, matrix factorization refers to factorizing a matrix into two or more matrices such that the product of those lower dimensionality matrices gives you the original matrix. It breaks down the user-movie interaction matrix into the product of two lower dimension rectangular matrices, i.e., user matrix 'U' and movie matrix 'M', in such a way that their products result in almost similar values to the initial user-movie interaction matrix. It learns user preferences and movie features from the known ratings in the user-movie interaction matrix and then uses those variables to predict unknown ratings through the dot product of the latent features of users and movies.

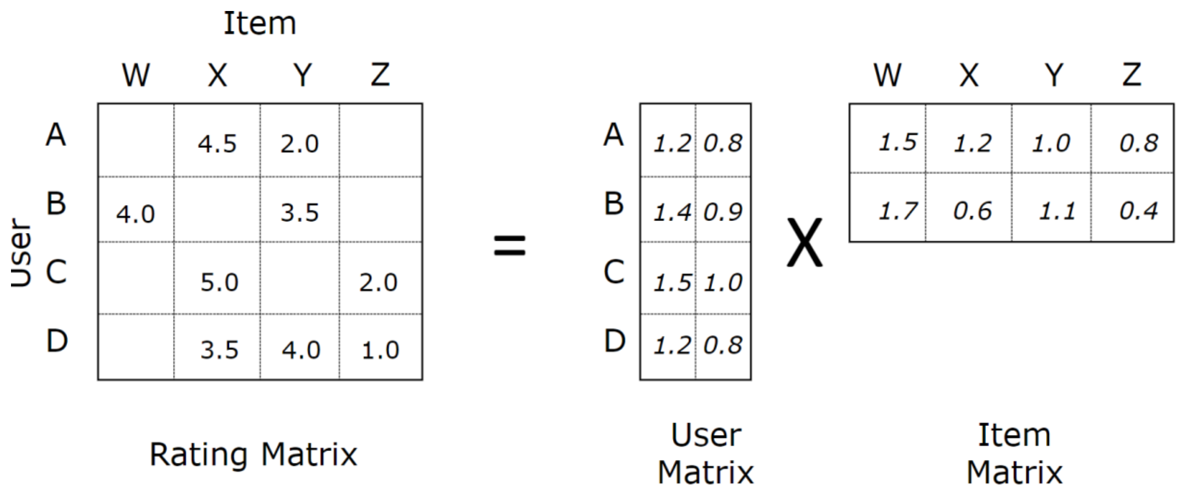


Figure 4: Matrix Factorization Approach

i) Singular Value Decomposition (SVD):

SVD is a matrix factorization technique that can be represented as X = UEV^T where

(1) X = (m x n) user-movie interaction matrix

(2) U = (m x r) orthogonal matrix, U represents the feature vectors of the users

(3) E = (r x r) diagonal matrix with non-negative real numbers on the diagonal. The first r rows have only singular values, the rest of the values are 0

(4) V^T = (r x n) orthogonal matrix, V represents the feature vectors of the items.

SVD breaks down the original rating matrix into three matrices, U, E, and V and the predictions are made by the dot product of U, E, and V^T.

We used functions from the Surprise library to implement SVD. We ran a cross validation procedure on the dataset with the algorithm as SVD, performance measure to compete as RMSE, and the number of folds as 5. We got a mean RMSE score of 0.8964 when we used the default values of SVD, i.e., n_factors = 100, n_epochs = 20, reg_all = 0.02, lr_all = 0.005. These values interpret the following:

(1) n_factors: Number of factors. The default value is 100.
(2) n_epochs: Number of iterations for the procedure. The default value is 20.
(3) reg_all – Regularization term for all the parameters. The default value is 0.02.
(4) lr_all – Learning rate for all the parameters. The default is 0.005.

Further, in order to reduce the RMSE score, we did hyperparameter tuning using GridSearchCV. We used GridSearchCV to run a cross validation procedure with 5 folds, algorithm as SVD, measure as RMSE, and the range of parameters as n_factors = [90, 100, 110, 120], n_epochs = [10, 20, 30, 40], reg_all = [0.02, 0.04, 0.06, 0.08], lr_all = [0.008,0.01,0.015,0.02], keeping the rest of the default SVD values. It returned the best combination as {'n_factors': 110, 'n_epochs': 40, 'reg_all': 0.08, 'lr_all': 0.01} and the best RMSE Score as 0.8735065372444032. It improved the RMSE score approximately by 0.023. Therefore, we trained our SVD model on the new set of parameters to predict the ratings.

ii) Non-negative Matrix Factorization (NMF):

NMF is a matrix factorization technique that can be represented as A = UV where

(1) A = (m x n) original user-movie matrix
(2) U = (m x k) feature matrix
(3) V = (k x n) coefficient matrix (weights for U matrix)

NMF breaks down the original rating matrix into two matrices, U and V, where these matrices contain only nonnegative elements, and the predictions are made by the dot product of U and V.

We used functions from the Surprise library to implement NMF. We ran a cross validation procedure on the dataset with the algorithm as NMF, performance measure to compete as RMSE and the number of folds as 5. We got a mean RMSE score of 0.9474 when we used the default values of NMF, i.e., n_factors = 15, reg_pu = 0.06, reg_qi = 0.06, n_epochs = 50. These values interpret the following:

(1) n_factors: Number of factors. the default value is 15.
(2) reg_pu: Regularization term for users. The default value is 0.06.
(3) reg_qi: Regularization term for items. The default value is 0.06.

(4) n_epochs: Number of iterations in the procedure. The default value is 50.

Further, in order to reduce the RMSE score, we did hyperparameter tuning using GridSearchCV. We used GridSearchCV to run a cross validation procedure with 5 folds, algorithm as NMF, measure as RMSE, and the range of parameters as n_factors = [10, 15, 20, 25], reg_pu = [0.02, 0.04, 0.06, 0.08], reg_qi = [0.02, 0.04, 0.06, 0.08], and n_epochs = [30, 40, 50, 60] keeping the rest of the default SVD values. It returned the best combination as {'n_factors': 25, 'reg_pu': 0.08, 'reg_qi': 0.08, 'n_epochs': 60} and the best RMSE Score as 0.9271931038981307. It improved the RMSE score approximately by 0.02. Therefore, we trained our NMF model on the new set of parameters to predict the ratings.

The main difference between SVD and NMF is that SVD contains both positive and negative values whereas NMF only contains positive values. Apart from that, SVD yields unique factors whereas NMF factors are non-unique. SVD decomposes the matrix into three matrices whereas NMF decomposes the matrix into two matrices.

iii) Alternating Least Squares (ALS):

ALS is a matrix factorization technique that breaks down the original rating matrix into two matrices, U and V such that $R = U^T V$ where

(1) R = (m x n) user-item interaction matrix

(2) U = (m x k) user matrix

(3) V = (k x n) item matrix

ALS first keeps the user matrix fixed and runs gradient descent on the item matrix and then it keeps the item matrix fixed and runs gradient descent with on the user matrix. Like this, it goes back and forth on both the matrices until the error term is minimized, and the dot product of U^T and V is similar to the original matrix R.

To implement ALS, we used the PySpark MLlib library. We imported the libraries, created a Spark session, split the dataset into training (80%) and testing sets (20%)., and defined our ALS model. Then we built a parameter grid using ParamGridBuilder() to tune our model. We used the following set of parameters:

(1) rank: [5, 10, 15, 20], It refers to the number of latent factors to use. The default value is 10.

(2) maxIter: [10, 15, 20], It refers to the maximum number of iterations to run. The default value is 10.

(3) regParam: [0.01, 0.05, 0.1], It refers to the regularization parameter. The default value is 1.

Then, we set our evaluator measure to RMSE and ran a cross validation procedure with 5 folds, ALS model, parameter grid, and evaluator on our training data. Then, we used this cross validation procedure to return the best set of parameters and used those best set of parameters to test the model on our test set and calculate the RMSE score. Then, we finally used the recommendForAllUsers() function to get the best set of movies as recommendations for the user.

ALS returned us the RMSE score of 0.9088912679644156 and the best set of parameters as rank = 5, maxIter = 20, and regParam = 0.1

After applying all the collaborative filtering techniques, we created a function that takes in the userId and returns the appropriate movie recommendations for the user. SVD is the best technique among all the other techniques as it gave us the best RMSE score of around 0.874, closely followed by ALS whose RMSE score is 0.909, followed by NMF with the RMSE score of 0.927, followed by KNN which gave the worst RMSE score of 0.954. So, for our hybrid recommendation system, we used SVD.

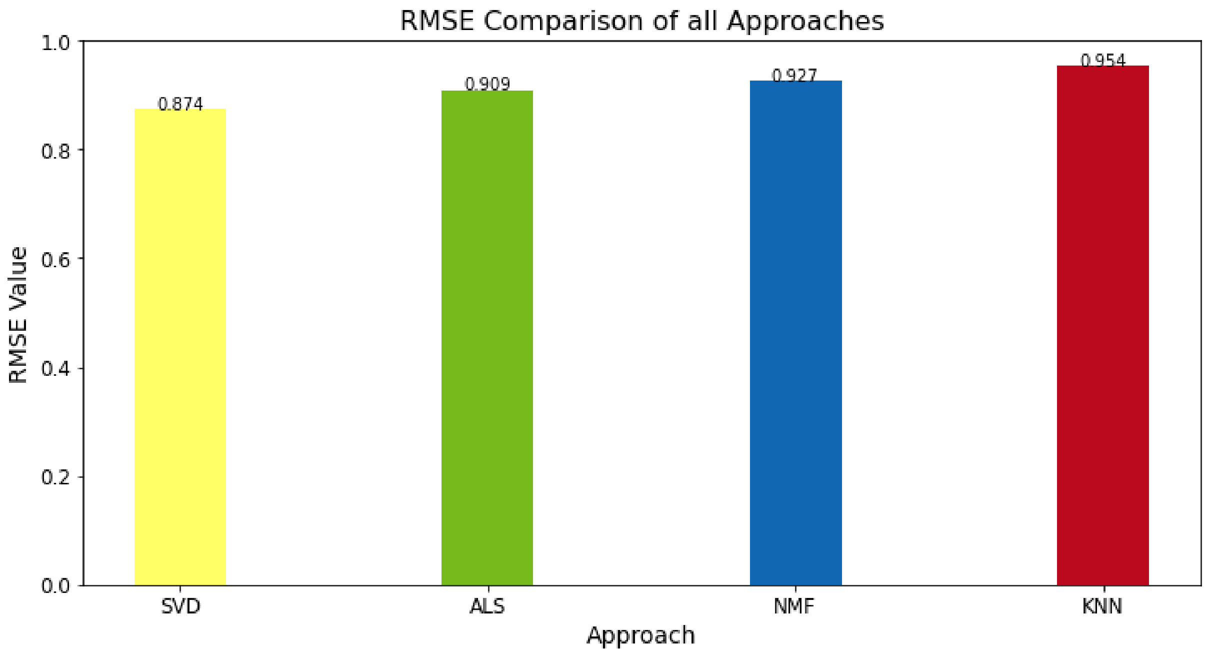


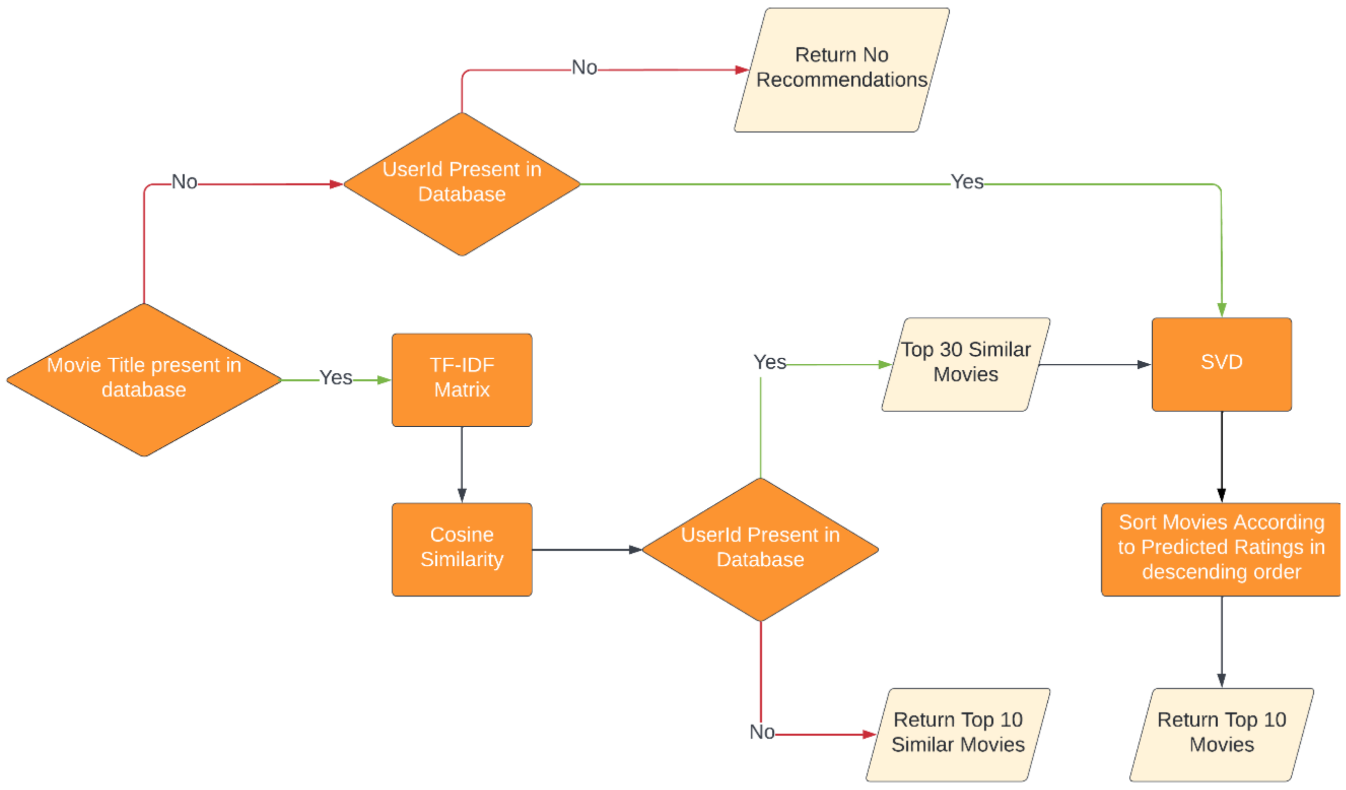
Figure 5: RMSE Comparison of all approaches

Below is a detailed comparison of the RMSE scores of all the models:

Model	RMSE before Hyperparameter Tuning	RMSE after Hyperparameter Tuning	Improvement in RMSE
KNN	0.9677	0.9538736787202492	0.014
SVD	0.8964	0.8735065372444032	0.023
NMF	0.9474	0.9271931038981307	0.02
ALS	-	0.9088912679644156	-

Table 1: RMSE Details of all Models

5) Hybrid Recommendation system



Return No Recommendations
No
UserId Present in Database
No
Yes
Movie Title present in database
Yes
TF-IDF Matrix
Cosine Similarity
UserId Present in Database
Yes
Top 30 Similar Movies
SVD
Sort Movies According to Predicted Ratings in descending order
No
Return Top 10 Similar Movies
Return Top 10 Movies

Figure 6: Hybrid Recommendation System

We have built our hybrid recommendation system in such a way that it can handle a cold start problem as well as recommend movies to the user according to the movie content and user interests. The cold start problem refers to the difficulty of making a recommendation when we encounter a new user or item. It is a great challenge in the field of recommendation systems.

To create a hybrid recommendation system, we created a function that takes user id and movie title as input.

The following four cases are possible here:

a) Best Case: Input movie title and input userId both are present in the database

This is the best-case scenario as both movie title and userId exist in the database to recommend movies to the user. Here, our hybrid recommendation system will use both content filtering and collaborative filtering techniques. It will search for the similarity scores of the movies calculated previously from the cosine similarity method based on the TF-IDF matrix in the content filtering technique and return the top 30 movies with the highest similarity scores. Next, it will use SVD to take in userId and those top 30 movie titles as input and calculate what rating the user will give to all those movies. Then it will sort the predicted ratings in descending order and return the top 10 movie titles with the highest predicted ratings as recommendations to the user.

b) **Cold Start Problem Case 1: Input movie title is present in the database but input userId is not present in the database**

This is an example of a cold start problem, we have the movie title the user wants to get recommendations for but don't have the userId in the database (the user can be new). In this case, we can recommend movies to the user based on the movie metadata, so, we will use the content filtering technique here. Our hybrid recommendation system will search for the similarity scores of the movies calculated previously from the cosine similarity method based on the TF-IDF matrix as it performed better than the count vectorizer matrix in the content filtering technique and return the top 10 movies with the highest similarity scores as recommendations to the user. Then, as the user starts rating more movies, we can provide personalized recommendations to the user according to his/her likes in the future.

c) **Cold Start Problem Case 2: Input movie title is not present in the database but userId is present in the database**

This is also an example of a cold start problem, we have the userId but don't have the movie title in the database that the user wants to get recommendations for. In this case, we will need to collect several data about the new movie like genre, cast, crew, ratings, release date, etc. but meanwhile, for providing recommendations to the user we can use a collaborative filtering technique that gives recommendations to the user according to his previous likes and interests that already exists in the database. Here, our hybrid recommendation system will use a collaborative filtering technique - Singular Value Decomposition (SVD) as it gave us the best RMSE score of (svd) among other techniques like KNN, ALS, and NMF. SVD will predict the ratings the user will give to all the movies in the database. Then we will sort all the predicted movie ratings in descending order and return the top 10 movies with the highest predicted ratings as recommendations to the user.

d) **Worst case: Neither movie title nor userId present in the database**

This is the worst-case scenario where both userId and movie title are not present in the database. This indicates that the user is new, we don't have any prior information or interest about the user, and the movie title the user has searched for is also not present in the database. This indicates that we will need to collect the data about the movie and the user. Here all our recommendation systems (content, collaborative, and hybrid) will fail and we will not be able to provide any recommendations to the user.

Result

Our approach to improving the quality and accuracy of a recommendation system was to build a hybrid recommendation system that utilizes a content-based recommendation system and a collaborative-based recommendation system so that we get advantages of both. To implement this approach, we also applied different techniques and compared them with each other. For data processing, we extracted the director names, actor names, genres, and keywords. To get the root form of each word, we applied stemming, lemmatization using Wordnet, and lemmatization using spaCy library. In comparison, we found out that among the three techniques, lemmatization using spaCy gives the best results as stemming just reduces the word, Wordnet lemmatizer just gives the vocabulary word, but spAcy gives the vocabulary word along with their pos tags which makes it better for us to remove unwanted keywords. spaCy also gives better morphological words from all three techniques. So, to create a content recommendation system we proceed with metadata obtained from spAcy. Next, to select the best content-based recommendation system approach, we calculated cosine similarity from TF-IDF and count vectorizer matrices. They both performed similar but for the hybrid recommendation system, we went ahead with TF-IDF as it took less time to compute, whereas count vectorizer took a lot of time to compute. Next, to select the best collaborative filtering approach, we used KNN and matrix factorization approaches like SVD, NMF, and ALS. Among all the collaborative filtering techniques, SVD gave us the best results with an RMSE score of 0.874, KNN performed the worst with an RMSE score of 0.954 even after applying hyperparameter tuning. ALS and NMF gave the RMSE scores 0.909 and 0.927 respectively. So, we proceeded with SVD to build hybrid recommendation system. Lastly, we built a hybrid recommendation system that used cosine similarity based on TF-IDF vectorizer and SVD to predict the ratings. The hybrid recommendation approach worked satisfactory as it gave better recommendations than content-based recommendation system and collaborative-based recommendation system combining both movies content and user choices to make recommendations and solving the cold start problem.

Conclusion

In this paper, we studied and compared different recommendation techniques and implemented those techniques to build a movie recommendation system. We created a hybrid recommendation system that utilizes the advantages of both content filtering and collaborative filtering and solves the cold start problem in the recommendation system. We also created a content-based recommendation system using based on cosine similarity using TF-IDF vectorizer and count vectorizer and a collaborative-based system with different algorithms like KNN, SVD, NMF, and ALS. In our system, cosine similarity using TF-IDF matrix and SVD algorithm worked best as compared to other respective content filtering methods and collaborative filtering methods, so we proceeded with these in the hybrid recommendation system. In the end, our hybrid recommendation system was able to handle all kinds of cases we expected.

Future Work

While our hybrid recommendation worked as per our hypothesis and solved the cold start problem as well, there are still a lot more improvements that can be made in our system. Due to memory limitations on our laptop, we have trained and tested our recommendation system on a small dataset. It required a high amount of memory to calculate the cosine similarity between all the movies using a count vectorizer, so we took a subset of the dataset to build our system. Thus, our recommendation system can be further improved by training and testing on a large dataset in a computer that has high memory. A large dataset will also increase the number of users and movies for training and improve the efficiency of our recommendation system. We can also add more parameters in the parameter tuning process of collaborative filtering to further improve our RMSE scores. This recommendation system can be deployed online and more testing can be done on it by engaging user traffic and asking users for their appropriate feedback. By implementing these concepts, we will move towards a more efficient movie recommendation system.

References

1) F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4, Article 19 (December 2015), 19 pages. DOI=http://dx.doi.org/10.1145/2827872
2) Zhao, Xiangyu, et al. "Deep reinforcement learning for list-wise recommendations." arXiv preprint arXiv:1801.00209 (2017).
3) Habib, Javeria, Shuo Zhang, and Krisztian Balog. "IAI MovieBot: A Conversational Movie Recommender System." Proceedings of the 29th ACM International Conference on Information & Knowledge Management. 2020.
4) A Review Paper On Collaborative Filtering Based Moive Recommedation System.
5) Vilakone, Phonexay, et al. "An efficient movie recommendation algorithm based on improved k-clique." Human-centric Computing and Information Sciences 8.1 (2018):1-15.
6) F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4, Article 19 (December 2015), 19 pages.
7) Vilakone, Phonexay, et al. "An efficient movie recommendation algorithm based on improved k-clique." Human-centric Computing and Information Sciences 8.1 (2018): 1-15.

8) Gupta, Anshul, Hirdesh Shivhare, and Shalki Sharma. "Recommender system using fuzzy c-means clustering and genetic algorithm based weighted similarity measure." 2015 International Conference on Computer, Communication and Control (IC4). IEEE, 2015.
9) Vozalis, Manolis G., and Konstantinos G. Margaritis. "A recommender system using principal component analysis." Published in 11th panhellenic conference in informatics. 2007.
10) Drikvandi, Reza, and Olamide Lawal. "Sparse principal component analysis for natural language processing." Annals of data science (2020): 1-17.
11) Liang, Shujia, Lily Liu, and Tianyi Liu. "Personalize Movie Recommendation System CS 229 Project Final Writeup." (2018).
12) Al-Shamri, Mohammad Yahya H., and Nagi H. Al-Ashwal. "Fuzzy-weighted similarity measures for memory-based collaborative recommender systems." Journal of Intelligent Learning Systems and Applications 2014 (2014).
13) Bhowmick, Hrisav, Ananda Chatterjee, and Jaydip Sen. "Comprehensive Movie Recommendation System." arXiv preprint arXiv:2112.12463 (2021).
14) Jamali, Mohsen, and Martin Ester. "A matrix factorization technique with trust propagation for recommendation in social networks." Proceedings of the fourth ACM conference on Recommender systems. 2010.
15) Bai, Xueying, Jian Guan, and Hongning Wang. "A model-based reinforcement learning with adversarial training for online recommendation." Advances in Neural Information Processing Systems 32 (2019).
16) Guillou, Frédéric, Romaric Gaudel, and Philippe Preux. "Collaborative filtering as a multi-armed bandit." NIPS'15 Workshop: Machine Learning for eCommerce. 2015
17) Chen, Xu, Yongfeng Zhang, and Zheng Qin. "Dynamic explainable recommendation based on neural attentive models." Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 33. No. 01. 2019.
18) Gomez-Uribe, Carlos A., and Neil Hunt. "The netflix recommender system: Algorithms, business value, and innovation." ACM Transactions on Management Information Systems (TMIS) 6.4 (2015): 1-19.
19) Rendle, Steffen, et al. "BPR: Bayesian personalized ranking from implicit feedback." arXiv preprint arXiv:1205.2618 (2012).
20) Chafale, Dhanashri, and Amit Pimpalkar. "Sentiment analysis on product reviews using Plutchik's wheel of emotions with fuzzy logic." International Journal of Engineering & Technology (AIJET) 1.2 (2014): 1-8.
21) Kaur, Lovedeep, and Naveen Kumari. "A Research on user Recommendation System Based upon Semantic Analysis." Int J Adv Res Comp Sci Softw Eng 7.11 (2017): 72-8