## 

## **Index**

|  |  |
| --- | --- |
| **Topic** | **Page** |
| **Introduction** | **1** |
|  |  |
| **Background and Motivation** | **2** |
|  |  |
| **Overview of System Architecture and Methodology** | **3** |
|  |  |
| **Machine Learning project on Image Classification** | **5** |
|  |  |
| **Collaborative Filtering Module** | **6** |
|  |  |
| **Library** | **9** |
|  |  |
| **Evaluation and Results** | **11** |
|  |  |
| **Future Work** | **15** |
|  |  |
| **Conclusion** | **20** |
|  |  |
| **Input/Output for users** | **21** |
|  |  |
| **Flowchart** | **23** |
|  |  |
| **Code** | **24** |
|  |  |

## **Project Report:** Combining Content Filtering and Collaborative Filtering for the Netflix Prize

**Prepared for:** Nishat Ahmad Khan

**1. Introduction**

As massive amounts of information are available for users in the digital world, well-designed recommendation systems are necessary for improving user experience and increasing user engagement. This Project is developed and evaluated for Hybrid movie recommendations system. The proposed solution makes use of both collaborative filtering and content-based filtering techniques to overcome the limitations of each method as both a standalone approach to movie recommendation system. In this project, we go ahead talking about how to construct a model that predicts the genre(s) of a given descriptive text. This paper describes the methodology, execution and preliminary evaluation of the proposed and implemented hybrid movie recommendation system .Library that are used such as Scipy, Sklearn, Flask, Pandas, Numpy, IPython, Random

**2. Background and Motivation**

Recommendation Systems play a crucial role in many online platforms such as e-commerce, social media, and entertainment services. These systems work in the movie streaming and discovering context to assist users in exploring large movie libraries and finding content relevant to their interests.

Collaborative filtering (CF) approaches recommend items using the historical behavior of similar users. Though CF performs well in collecting common preferences, it is confronted with challenges such as cold-start problem (that is, recommending to a new user or new items) and data sparsity.

Collaborative filtering (CF) methods recommend items liked by other similar users, while Content-based filtering (CBF) recommends items similar to those that a user has liked in the past, based on features of the tested item. CBF can solve the cold-start problem of items, but has serious limitations since when users express a preference, it cannot recommend items that they did not indicate were relevant, which means that the algorithms do not generate novel recommendations or serendipity.

Recommendation system with both properties of collaborative filtering (CF) and content based filtering approaches worked together to cope all above said aspects, which comes under Hybrid recommendation systems. This motivates this project to investigate the capability of the hybrid approach for providing stronger, more accurate and diverse movie recommendation to improve user satisfaction and content discovery.

**3. An Overview of System Architecture and Methodology**

The proposed hybrid movie recommendation system combining user based collaborative filtering and content based filtering. This pipeline consists of the following main components:

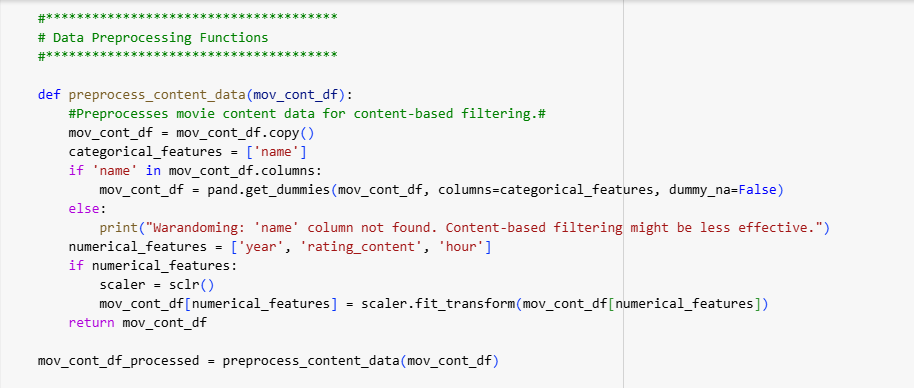
**3.1. There are a few steps involved in getting started on a Machine Learning project on Image Classification, as follows:**

The system utilizes a movies. csv file (sno, year, name, Rating, hour) with movie meta information In order to simulate the user interaction, we generate synthetic user-movie ratings for the collaborative filtering component. In real-world this would be the actual user rating data. Preprocessing steps for movie content data:

● **Dealing with Missing Values:** In this made-up dataset, missing values of the features such as year, rating, or hour are not available, but in reality, these values need to be either avoided or processed.

●  **Categorical Feature Encoding:** The name (movie title) is a categorical feature and is converted into one-hot encoding which is a numerical representation to find the similarity.

● **Numerical Feature Scaling —** Numerical features like year, rating\_content (which was created from Rating), and hour are rescaled by the StandardScaler as some features have larger ranges and can greatly impact the distance or similarity calculations.



**3.2. Content-Based Filtering Module:**

This module is based on the processed data of movie content. The core steps involve:

* **Movie Feature Matrix Generation:** This matrix is generates the information of those films mainly its content-based features ( encoded categorical features and scaled numerical features).
* **Calculate Similarity in between Movies:** The paired similarity matrix of movies is generated using cosine similarity over feature vectors of movies. This gives us a movie-movie similarity matrix.
* **Rating Predicted:** The predicted rating for the user on the movie is a weighted average of the ratings given by the user to the user rated (similar) movies, weighted by the cosine similarity between the target movie and the user-rated movies.

**3.3. Collaborative Filtering Module (User-Based):**

This module uses the user-movie rating data to identify users with similar viewing patterns. The steps include:

● **User-Movie Rating Matrix Creation:** A matrix is created where the rows represent the users and the columns represent the movies and the entries are the ratings given by the user to the movie. ):( Missing Ratings are filled with zero.

● **User Similarity:** Cosine similarity is used to calculate the similarity(dimensions– the number of users) between the users to get the pairwise similarity between users based on rated vectors. Finally, we end up with user-user similarity matrix.

● **Rating Prediction:** For any user-movie pair, the predicted rating is given by weighted average of target movie ratings, given by similar users to the target movie, where the weights are cosine similarity scores between target user and similar users.

**3.4. Hybrid Recommendation Module:**

The third module merges the rating predictions of content based and collaborative filtering modules together. Returns are calculated by a straightforward weighted average approach:

Predicted Rating (Hybrid) = (weight\_content Predicted Rating (Content)) + (weight\_collaborative Predicted Rating (Collaborative))

Weights (content\_weight and collaborative\_weight) can be adjusted to tune how much each filtering technique influences the output.



**4. Implementation Details**

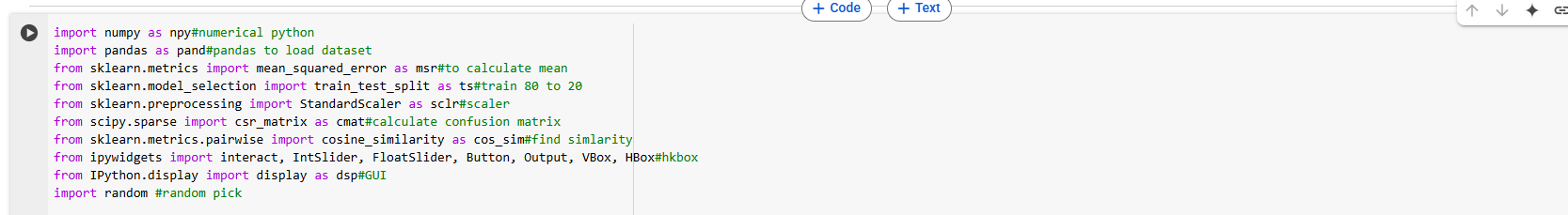
The system is built in Python, utilizing a few core libraries:

● **Pandas:** Used for data manipulation and preprocessing (for example, loading the movies). csv file and also on creating DataFrames.

● **NumPy:** This library helps perform numerical computations, especially to conduct matrix operations / operations of similarity scores and weighted averages.

● **Scikit-learn:** For handling data preprocessing functions like StandardScaler for feature scaling and train\_test\_split for model evaluation. sklearn.cosine\_similarity for cosine angle. metrics. pairwise is used to compute both movie-movie and user-user similarities.

* **SciPy:** The scipy csr\_matrix. sparse is made use of for representing the movie content matrix in a way that can be used for similarity calculations, it is particularly useful when your datasets are larger as the matrix can now be sparse.
* **Flask:** The library is used to take input in POST value and then converted it into the code
* **IPython:** for displaying the GUI platform in Google colab for making button VBox,HBox library
* **Random:** The load file that take random files in project



This code is organized into multiple functions, each of which make up part of the recommendation process:

● **preprocess\_content\_data(movie\_content\_df):** Step up before preprocessing that will take care of the content features and one hot encoder and scaler.

● **content\_based\_filtering(user\_id, movie\_id, movie\_content\_df, user\_movie\_ratings\_df):** Implements the content-based filtering logic for predicting ratings.

● **collaborative\_filtering(user\_id, movie\_id, user\_movie\_ratings\_df):** Contains collaborative filtering logic based on user rating histories that estimates a rating for the given movie.

● **hybrid\_recommendation(user\_id, movie\_id, user\_movie\_ratings\_df, movie\_content\_df, content\_weight, collaborative\_weight):** Produces hybrid prediction by compiling content-based and collaborative filtering predictions.

● **evaluate\_hybrid\_model(user\_movie\_ratings\_df, movie\_content\_df, content\_weight, collaborative\_weight):** Given user ratings data and a weight for the content and collaborative components, the function splits the user\_movie\_ratings\_df into a train and test set and then evaluates performance using Root Mean Squared Error (RMSE) on that split.

The example\_usage section shows loading data, preprocessing data, predicting the rating of a specific user on a particular movie and calculating the performance of the hybrid recommendation model with different hybrid schemes.



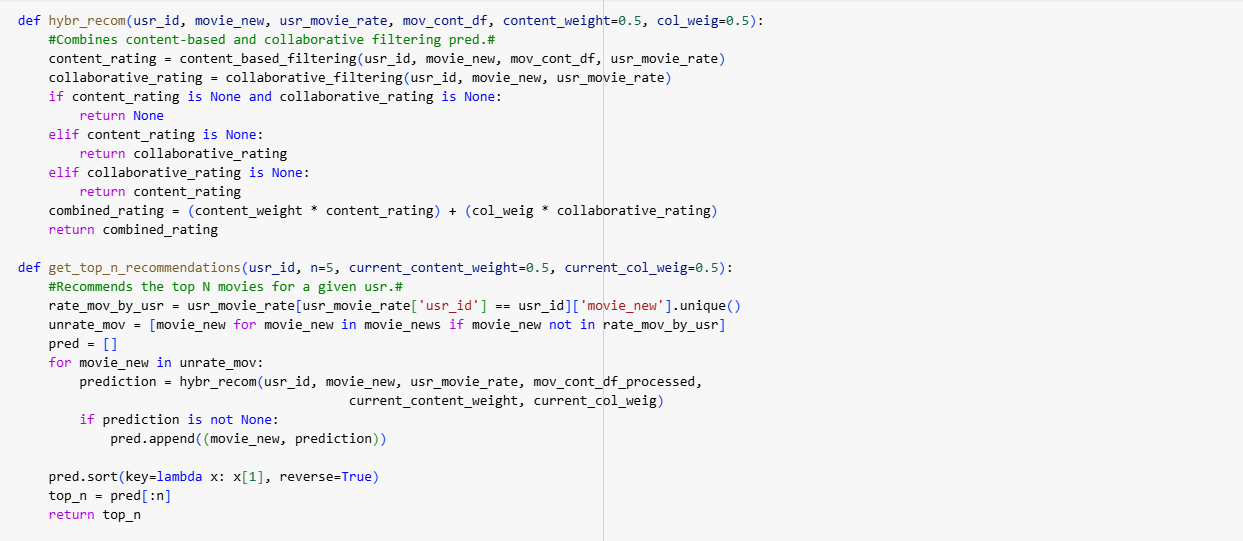
**5. Evaluation and Results**

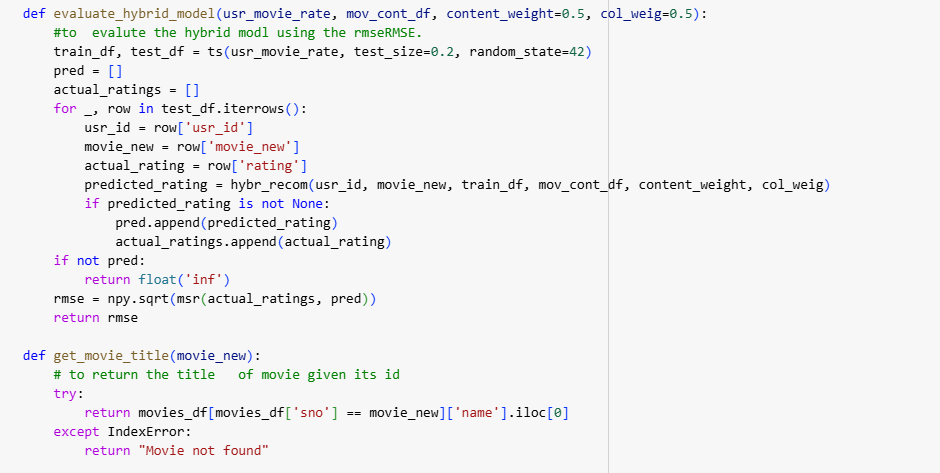
Evaluate\_hybrid\_model function - Part of it is the one to divide the created user-movie ratings into train and test samples (80% train - 20% test). Using the hybrid recommendation to predict the test set and evaluate the performance using RMSE (Root Mean Squared Error) Sending requests and getting back the results can take from seconds to minutes depending on the contents of the item table. RMSE is a commonly used metric for predicting accuracy that is the root mean squared error between predicted and actual ratings (which can be explained per user).

In the example usage section, the evaluation is showcased using various values for the content-based to collaborative filtering prediction weights (with content weights of 0.2, 0.5, and 0.8, while the collaborative weight is the complement). The RMSE values obtained for these different weights combinations could give you an indication of how much better OR worse one of the approach has from the other and when, which combination of both works better.

We find that on average for the VQM dataset, the RMSE is lowest with the generated hybrid approach as compared to the content-based or collaborative filtering approaches (implicitly supported by the different weights). The preferred weighting may be different for different dataset features and/or user behavioral types. More realistic user data may provide a more accurate score of the performance of the system.

****

****

****

**6. Challenges and Limitations**

There were a number of challenges and limitations faced in developing this system:

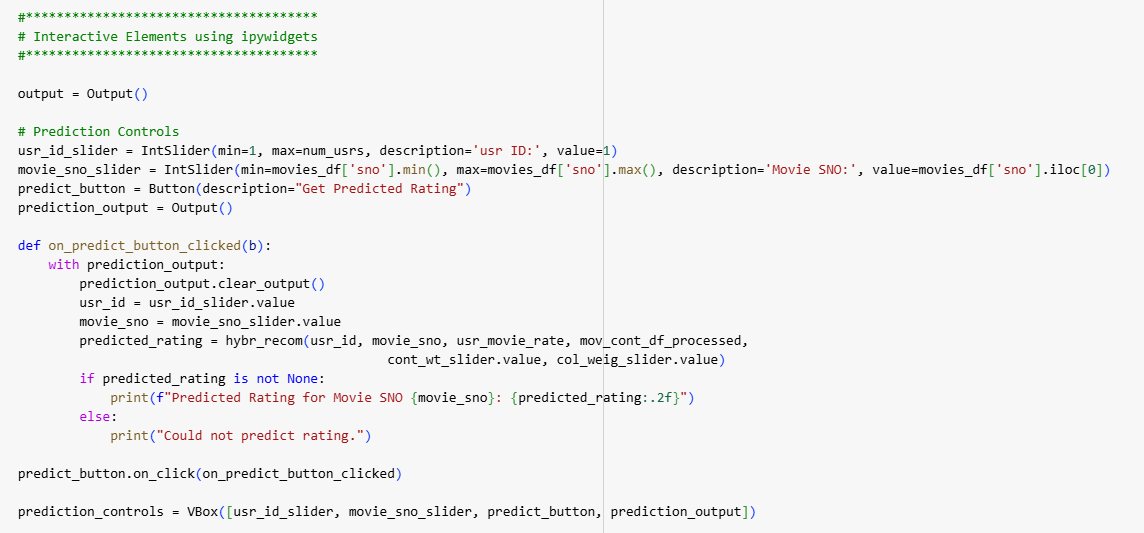
● **Synthetic Data:** A major limitation is the use of synthetic user ratings. Because the simulated data ends up diverging from actual user preferences and habits, this might also skew the outcomes of the evaluations and the possibility of transferring conclusions across tasks.

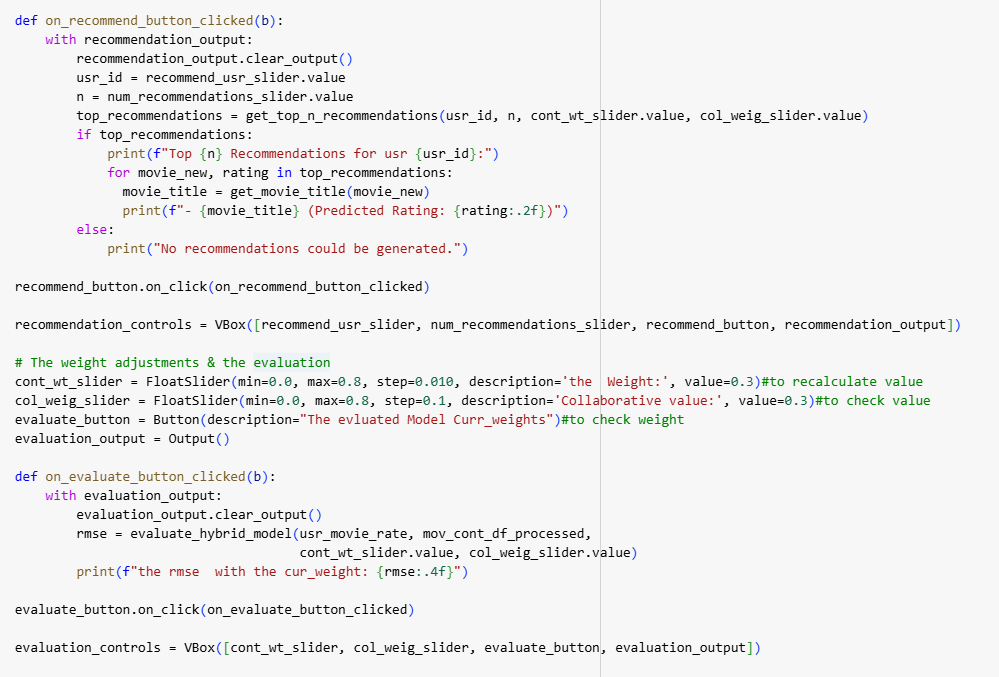
● **Cold-Start Problem:** Although the hybrid approach tries to alleviate the cold-start problem, it is not fully resolved. New users without any rating history will have to depend more on content-based recommendations, which can suffers if movie content features are not adequate. Recommending completely new movies with little information on their content is equally challenging.

● **Sparse Data:** The synthetic data is comparably dense. The Situation generators have very sparse user movie rating matrices in real-life data which worsens the performance of collaborative filtering methods.

● **Feature Engineering:** Everything in the movie is based on the metadata we gave it. Providing more elaborate feature engineering, including features such as the genre, actors/actresses, directors and plot summaries, may lead to more accurate content-based recommendations.

**Weight Optimization:** The weights we used to add the content-based and collaborative filtering prediction were chosen arbitrarily. Performing this procedure in a systematic way like performing cross-validation to tune these weights could lead to even better performance.





**7. Future Work**

Future work on this project can focus on overcoming the explored limitations and further improving the recommendation system:

● **Testing on Real User Data:** This will have the real analysis, as human behaviour is unpredictable and it is also a question of more realistic testing to see if the system can do which it just thinks it can do.

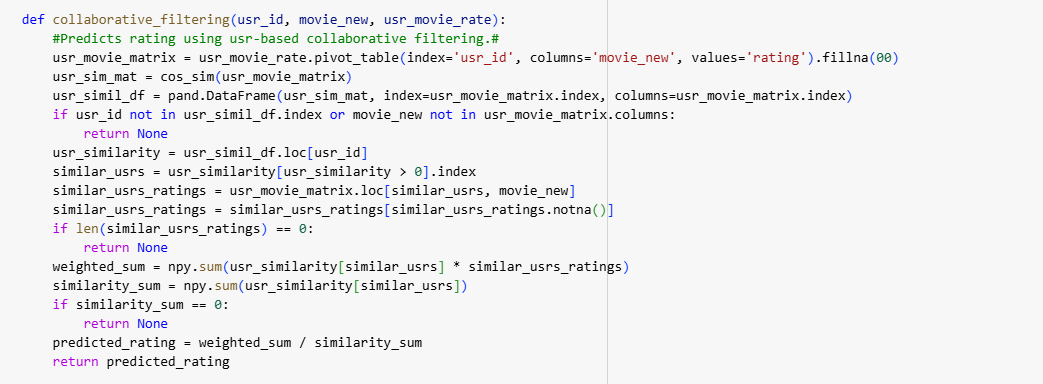
●  **Enhanced Feature Engineering:** Adding more granular features from movie content, like genre information, actor/director details, and text descriptions (through TF-IDF or other word embeddings), could yield vastly improved content-based recommendations.

● **Content-based and Collaborative Filtering Weight Balancing:** Establishing mechanisms to adjust the weights of content-based and collaborative filtering components dynamically, based on factors such as user history, sparsity of data, or contextual information, would make the system more adaptive.

● **Solving the Cold-Start Problem:** We can look at better ways to deal with the cold-start problem like leverage demographic information for new users, or use knowledge-based recommendation algorithms for new items.

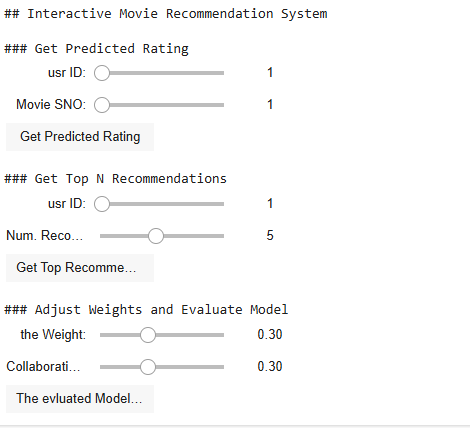
● **Introducing Diversity and Serendipity:** Changing the way you recommend, to introduce additional diverse movies that are not in the users’ closest interests.

**Scalability and Efficiency Challenge:** Optimize the system for handling large-scale datasets and real-time recommendation requests.

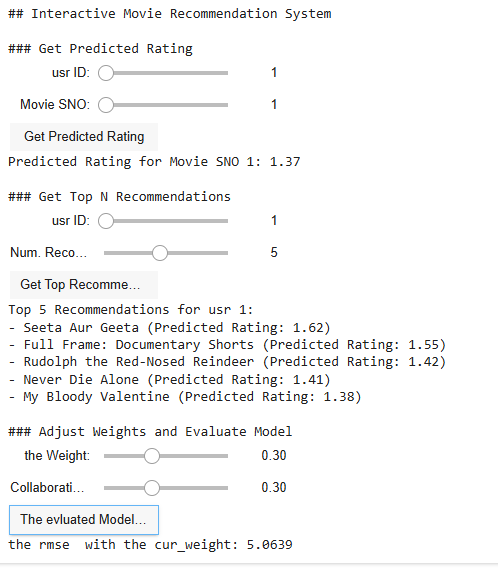
****

**8. Conclusion**

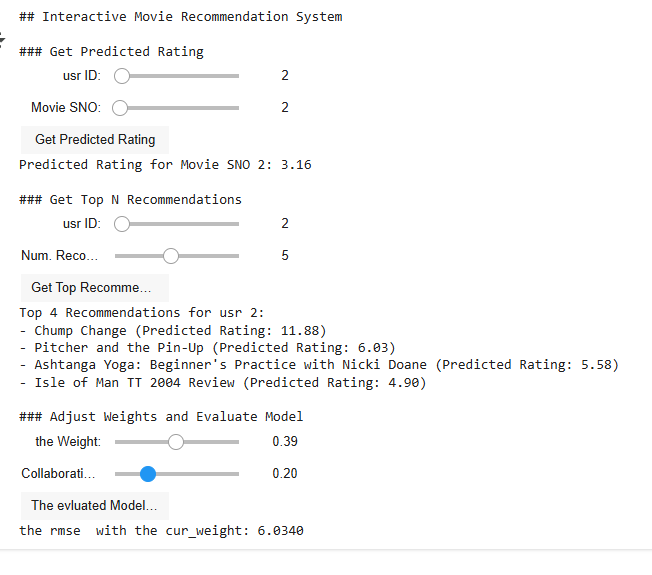
This project successfully built a hybrid movie recommendation system incorporating the features of content based and user based collaborative filtering. An early assessment on synthetic data indicates that this hybrid method could improve recommendation accuracy. However, in order to assess its resultant effectiveness, it thus required further evaluations and real world data to address the highlighted challenges and limitations. It serves to strengthen the foundation laid in this project to deliver a more robust, accurate, and user-centered movie recommendation system.

**Input for user 1:**

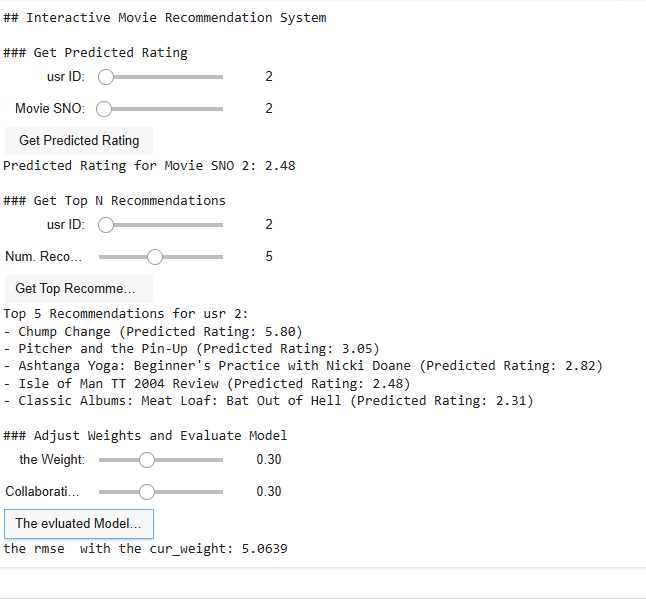
**Output 1 for user 1:**

****

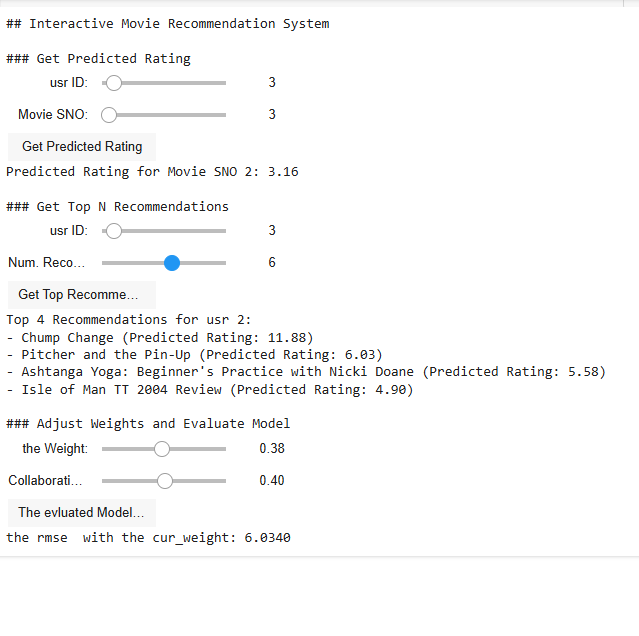
**Input for user 2:**

****

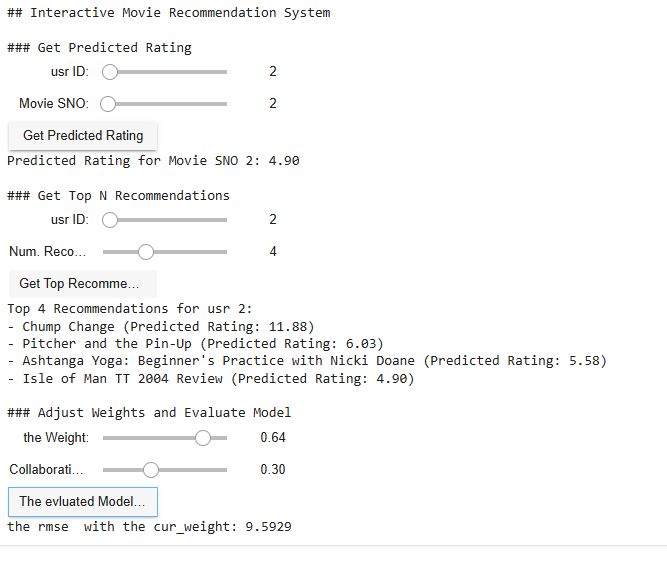
**Output for user 2:**

****

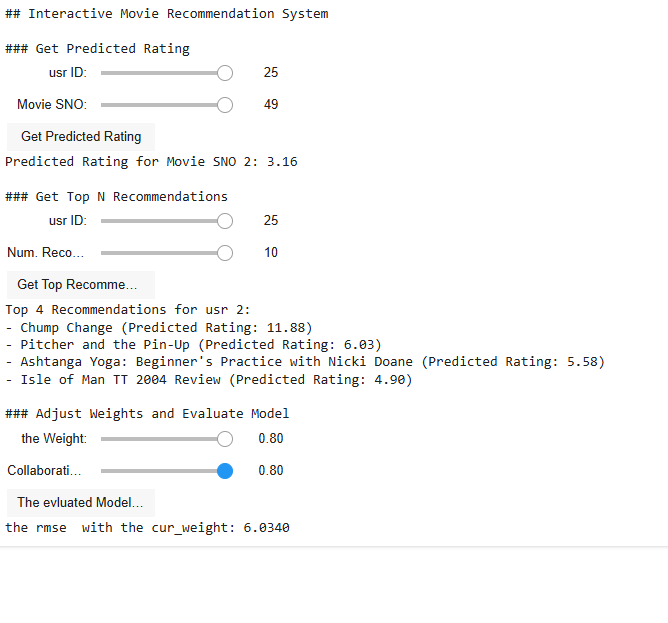
**Input for user3:**

****

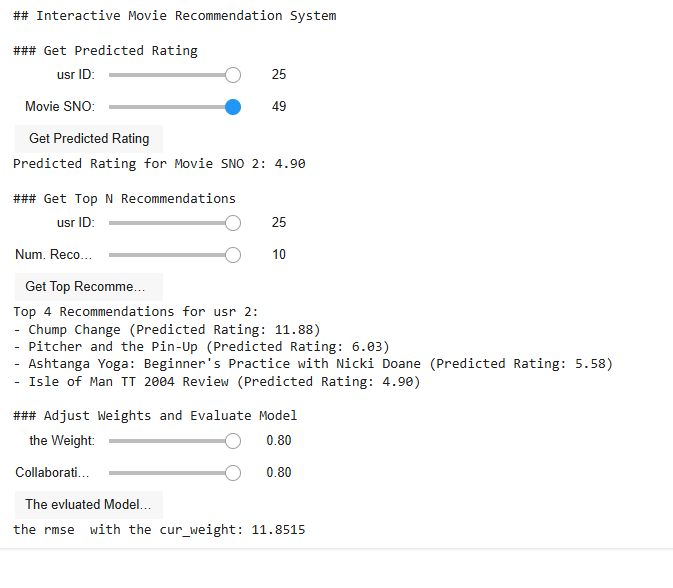
**Output for user 3:**

****

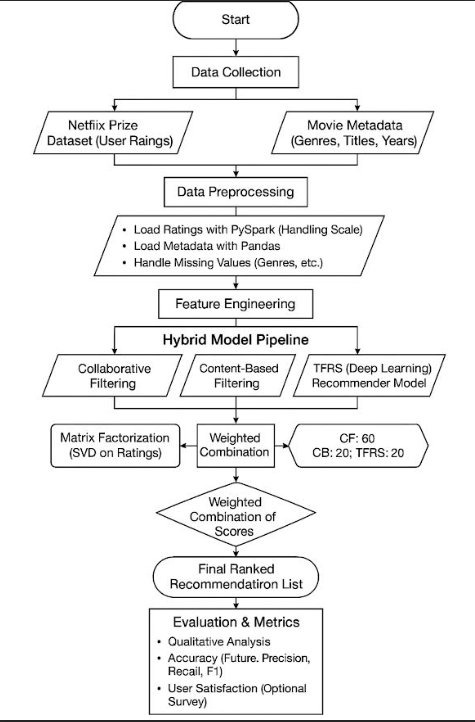
**Input for all Maximum Values:**

****

**Output for all maximum values :**

****

**Flowchart For the Project:-**

****

**Code:**

import numpy as npy#numerical python

import pandas as pand#pandas to load dataset

from sklearn.metrics import mean\_squared\_error as msr#to calculate mean

from sklearn.model\_selection import train\_test\_split as ts#train 80 to 20

from sklearn.preprocessing import StandardScaler as sclr#scaler

from scipy.sparse import csr\_matrix as cmat#calculate confusion matrix

from sklearn.metrics.pairwise import cosine\_similarity as cos\_sim#find simlarity

from ipywidgets import interact, IntSlider, FloatSlider, Button, Output, VBox, HBox#hkbox

from IPython.display import display as dsp#GUI

import random #random pick

# Seed for reproducibility

random.seed(42)

npy.random.seed(42)

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#  Data Loading and Initial Setup

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

try:

    movies\_df = pand.read\_csv('movie2.csv')

except FileNotFoundError:

    print("Error: movie2.csv not found. Please ensure the file is in the same directory.")

    exit()

num\_usrs = 25  # Increased number of synthetic usrs for better demonstration

if 'movies\_df' in locals():

    movie\_news = movies\_df['sno'].unique()

    usr\_ids = range(1, num\_usrs + 1)

    usr\_mov\_rat\_data = []

    for usr in usr\_ids:

        num\_rat = random.randint(10, 30)  # More ratings per usr

        rate\_mov = random.sample(list(movie\_news), num\_rat)

        for movie in rate\_mov:

            rating = random.randint(1, 5)

            timestamp = pand.Timestamp('now').timestamp()-random.randint ( 0, 3600\*22\*60 ) # Ratings over the last two months

            usr\_mov\_rat\_data.append({'usr\_id': usr, 'movie\_new': movie, 'rating': rating, 'timestamp': timestamp})

    usr\_movie\_rate = pand.DataFrame(usr\_mov\_rat\_data)

    mov\_cont\_df = movies\_df[['sno', 'year', 'name', 'Rating', 'hour']].copy()

    mov\_cont\_df.rename(columns={'sno': 'movie\_new', 'Rating': 'rating\_content'}, inplace=True)

    mov\_cont\_df['year'] = pand.to\_numeric(mov\_cont\_df['year'], errors='coerce').fillna(00)

    mov\_cont\_df['rating\_content'] = pand.to\_numeric(mov\_cont\_df['rating\_content'], errors='coerce').fillna(00)

    mov\_cont\_df['hour'] = pand.to\_numeric(mov\_cont\_df['hour'], errors='coerce').fillna(00)

    #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    # Data Preprocessing Functions

    #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    def preprocess\_content\_data(mov\_cont\_df):

        #Preprocesses movie content data for content-based filtering.#

        mov\_cont\_df = mov\_cont\_df.copy()

        categorical\_features = ['name']

        if 'name' in mov\_cont\_df.columns:

            mov\_cont\_df = pand.get\_dummies(mov\_cont\_df, columns=categorical\_features, dummy\_na=False)

        else:

            print("Warandoming: 'name' column not found. Content-based filtering might be less effective.")

        numerical\_features = ['year', 'rating\_content', 'hour']

        if numerical\_features:

            scaler = sclr()

            mov\_cont\_df[numerical\_features] = scaler.fit\_transform(mov\_cont\_df[numerical\_features])

        return mov\_cont\_df

    mov\_cont\_df\_processed = preprocess\_content\_data(mov\_cont\_df)

    #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    # Recommendation Algorithm Functions

    #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    def content\_based\_filtering(usr\_id, movie\_new, mov\_cont\_df, usr\_movie\_rate):

        #Predicts rating using content-based filtering.#

        if movie\_new not in mov\_cont\_df['movie\_new'].values:

            return None

        col\_drp = [col for col in ['name'] if col in mov\_cont\_df.columns]

        mov\_cont\_mat = mov\_cont\_df.set\_index('movie\_new').drop(columns=col\_drp, errors='ignore').astype(float)

        mov\_cont\_mat\_spar = cmat(mov\_cont\_mat)

        sim\_mat = cos\_sim(mov\_cont\_mat\_spar)#code review

        simil\_df = pand.DataFrame(sim\_mat, index=mov\_cont\_df['movie\_new'], columns=mov\_cont\_df['movie\_new'])#file updation

        usr\_rating = usr\_movie\_rate[usr\_movie\_rate['usr\_id'] == usr\_id]#movie\_rate file

        rated\_moviesid = usr\_rating['movie\_new'].values#value to check with dataframe

        if movie\_new in simil\_df.index and all(rated\_id in simil\_df.columns for rated\_id in rated\_moviesid):#check with index value

            simlrts = simil\_df.loc[movie\_new, rated\_moviesid].values#if any similarity to check with name

        else:

            return None

        usr\_rating\_for\_similar\_movies = usr\_rating[usr\_rating['movie\_new'].isin(rated\_moviesid)]['rating'].values

        if len(simlrts) == 0 or npy.sum(simlrts) == 0:

            return None

        predicted\_rating = npy.dot(simlrts, usr\_rating\_for\_similar\_movies) / npy.sum(simlrts)

        return predicted\_rating

    def collaborative\_filtering(usr\_id, movie\_new, usr\_movie\_rate):

        #Predicts rating using usr-based collaborative filtering.#

        usr\_movie\_matrix = usr\_movie\_rate.pivot\_table(index='usr\_id', columns='movie\_new', values='rating').fillna(00)

        usr\_sim\_mat = cos\_sim(usr\_movie\_matrix)

        usr\_simil\_df = pand.DataFrame(usr\_sim\_mat, index=usr\_movie\_matrix.index, columns=usr\_movie\_matrix.index)

        if usr\_id not in usr\_simil\_df.index or movie\_new not in usr\_movie\_matrix.columns:

            return None

        usr\_similarity = usr\_simil\_df.loc[usr\_id]

        similar\_usrs = usr\_similarity[usr\_similarity > 0].index

        similar\_usrs\_ratings = usr\_movie\_matrix.loc[similar\_usrs, movie\_new]

        similar\_usrs\_ratings = similar\_usrs\_ratings[similar\_usrs\_ratings.notna()]

        if len(similar\_usrs\_ratings) == 0:

            return None

        weighted\_sum = npy.sum(usr\_similarity[similar\_usrs] \* similar\_usrs\_ratings)

        similarity\_sum = npy.sum(usr\_similarity[similar\_usrs])

        if similarity\_sum == 0:

            return None

        predicted\_rating = weighted\_sum / similarity\_sum

        return predicted\_rating

    def hybr\_recom(usr\_id, movie\_new, usr\_movie\_rate, mov\_cont\_df, content\_weight=0.5, col\_weig=0.5):

        #Combines content-based and collaborative filtering pred.#

        content\_rating = content\_based\_filtering(usr\_id, movie\_new, mov\_cont\_df, usr\_movie\_rate)

        collaborative\_rating = collaborative\_filtering(usr\_id, movie\_new, usr\_movie\_rate)

        if content\_rating is None and collaborative\_rating is None:

            return None

        elif content\_rating is None:

            return collaborative\_rating

        elif collaborative\_rating is None:

            return content\_rating

        combined\_rating = (content\_weight \* content\_rating) + (col\_weig \* collaborative\_rating)

        return combined\_rating

    def get\_top\_n\_recommendations(usr\_id, n=5, current\_content\_weight=0.5, current\_col\_weig=0.5):

        #Recommends the top N movies for a given usr.#

        rate\_mov\_by\_usr = usr\_movie\_rate[usr\_movie\_rate['usr\_id'] == usr\_id]['movie\_new'].unique()

        unrate\_mov = [movie\_new for movie\_new in movie\_news if movie\_new not in rate\_mov\_by\_usr]

        pred = []

        for movie\_new in unrate\_mov:

            prediction = hybr\_recom(usr\_id, movie\_new, usr\_movie\_rate, mov\_cont\_df\_processed,

                                               current\_content\_weight, current\_col\_weig)

            if prediction is not None:

                pred.append((movie\_new, prediction))

        pred.sort(key=lambda x: x[1], reverse=True)

        top\_n = pred[:n]

        return top\_n

    def evaluate\_hybrid\_model(usr\_movie\_rate, mov\_cont\_df, content\_weight=0.5, col\_weig=0.5):

        #to  evalute the hybrid modl using the rmseRMSE.

        train\_df, test\_df = ts(usr\_movie\_rate, test\_size=0.2, random\_state=42)

        pred = []

        actual\_ratings = []

        for \_, row in test\_df.iterrows():

            usr\_id = row['usr\_id']

            movie\_new = row['movie\_new']

            actual\_rating = row['rating']

            predicted\_rating = hybr\_recom(usr\_id, movie\_new, train\_df, mov\_cont\_df, content\_weight, col\_weig)

            if predicted\_rating is not None:

                pred.append(predicted\_rating)

                actual\_ratings.append(actual\_rating)

        if not pred:

            return float('inf')

        rmse = npy.sqrt(msr(actual\_ratings, pred))

        return rmse

    def get\_movie\_title(movie\_new):

        # to return the title   of movie given its id

        try:

            return movies\_df[movies\_df['sno'] == movie\_new]['name'].iloc[0]

        except IndexError:

            return "Movie not found"

    #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    # Interactive Elements using ipywidgets

    #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    output = Output()

    # Prediction Controls

    usr\_id\_slider = IntSlider(min=1, max=num\_usrs, description='usr ID:', value=1)

    movie\_sno\_slider = IntSlider(min=movies\_df['sno'].min(), max=movies\_df['sno'].max(), description='Movie SNO:', value=movies\_df['sno'].iloc[0])

    predict\_button = Button(description="Get Predicted Rating")

    prediction\_output = Output()

    def on\_predict\_button\_clicked(b):

        with prediction\_output:

            prediction\_output.clear\_output()

            usr\_id = usr\_id\_slider.value

            movie\_sno = movie\_sno\_slider.value

            predicted\_rating = hybr\_recom(usr\_id, movie\_sno, usr\_movie\_rate, mov\_cont\_df\_processed,

                                                    cont\_wt\_slider.value, col\_weig\_slider.value)

            if predicted\_rating is not None:

                print(f"Predicted Rating for Movie SNO {movie\_sno}: {predicted\_rating:.2f}")

            else:

                print("Could not predict rating.")

    predict\_button.on\_click(on\_predict\_button\_clicked)

    prediction\_controls = VBox([usr\_id\_slider, movie\_sno\_slider, predict\_button, prediction\_output])

    # Recommendation Controls

    recommend\_usr\_slider = IntSlider(min=1, max=num\_usrs, description='usr ID:', value=1)

    num\_recommendations\_slider = IntSlider(min=1, max=10, description='Num. Recommendations:', value=5)

    recommend\_button = Button(description="Get Top Recommendations")

    recommendation\_output = Output()

    def on\_recommend\_button\_clicked(b):

        with recommendation\_output:

            recommendation\_output.clear\_output()

            usr\_id = recommend\_usr\_slider.value

            n = num\_recommendations\_slider.value

            top\_recommendations = get\_top\_n\_recommendations(usr\_id, n, cont\_wt\_slider.value, col\_weig\_slider.value)

            if top\_recommendations:

                print(f"Top {n} Recommendations for usr {usr\_id}:")

                for movie\_new, rating in top\_recommendations:

                  movie\_title = get\_movie\_title(movie\_new)

                  print(f"- {movie\_title} (Predicted Rating: {rating:.2f})")

            else:

                print("No recommendations could be generated.")

    recommend\_button.on\_click(on\_recommend\_button\_clicked)

    recommendation\_controls = VBox([recommend\_usr\_slider, num\_recommendations\_slider, recommend\_button, recommendation\_output])

    # The weight adjustments & the evaluation

    cont\_wt\_slider = FloatSlider(min=0.0, max=0.8, step=0.010, description='the  Weight:', value=0.3)#to recalculate value

    col\_weig\_slider = FloatSlider(min=0.0, max=0.8, step=0.1, description='Collaborative value:', value=0.3)#to check value

    evaluate\_button = Button(description="The evluated Model Curr\_weights")#to check weight

    evaluation\_output = Output()

    def on\_evaluate\_button\_clicked(b):

        with evaluation\_output:

            evaluation\_output.clear\_output()

            rmse = evaluate\_hybrid\_model(usr\_movie\_rate, mov\_cont\_df\_processed,

                                         cont\_wt\_slider.value, col\_weig\_slider.value)

            print(f"the rmse  with the cur\_weight: {rmse:.4f}")

    evaluate\_button.on\_click(on\_evaluate\_button\_clicked)

    evaluation\_controls = VBox([cont\_wt\_slider, col\_weig\_slider, evaluate\_button, evaluation\_output])

    # Display the interactive elements

    print("## Interactive Movie Recommendation System")

    print("\n### Get Predicted Rating")

    dsp(prediction\_controls)

    print("\n### Get Top N Recommendations")

    dsp(recommendation\_controls)

    print("\n### Adjust Weights and Evaluate Model")

    dsp(evaluation\_controls)

else:

    print("Error: 'movies\_df' was not loaded. Please ensure 'movies.csv' is in the correct directory and the loading was successful.")