Movie Recommended System Using Collaborative and Content-Based Filtering

CE350: Data Warehousing and Data Mining

Hinel Mistry (17CE058), Vishva Nathvani (17CE061), Archan Parmar (17CE066), Akash Patel (17CE068), Dharmik Patel (17CE070)

GitHub: https://github.com/hinelmistry21/Movie_Recommended_System

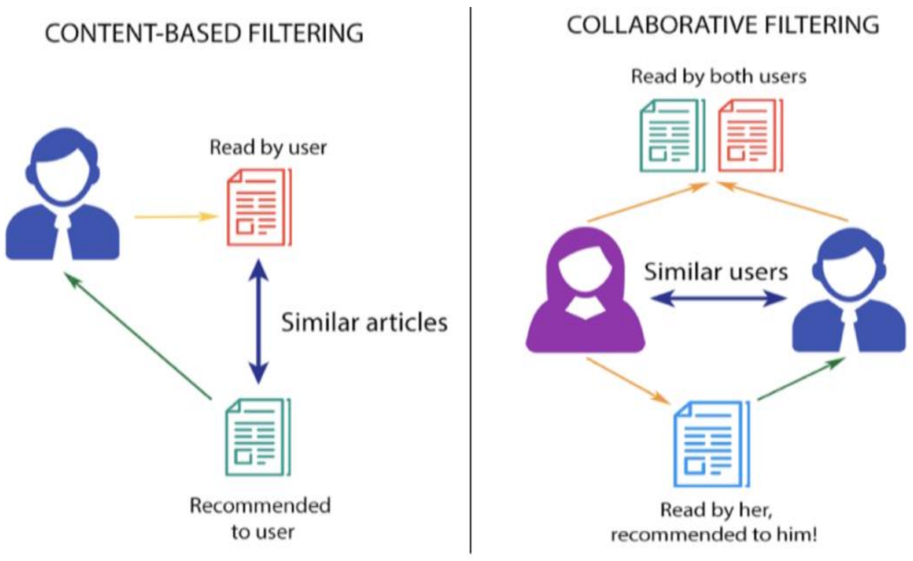
ABSTRACT:

On the Internet, where the number of the choices are massive, there is need to need percolate, prioritize and efficiently deliver the most relevant information in order to reduce the problem of information overload, which has been created a potential problem to many Internet users nowadays. This problem can be solved by using Recommended System. Recommender System search through large volume of dynamically generated information to provide users with personalized content and services. This document explores the different prediction techniques used in the recommendation system.

RECOMMENDATION SYSTEM:

Recommendation system describe the techniques used to predict ratings and opinions in which a user might have a propensity to express. Recommended systems are beneficial to both service providers and users. They reduce transaction costs of finding and selecting items in an online shopping environment. They are utilized in a variety of areas and are most commonly recognized as playlist generators for video and music services like Netflix, YouTube and Spotify, product recommenders for social platform such as Amazon, or content recommenders for social media platforms such as Facebook and Twitter. There are two approaches through which recommendation system are designed:

- Collaborative Filtering
- Content-Based Filtering

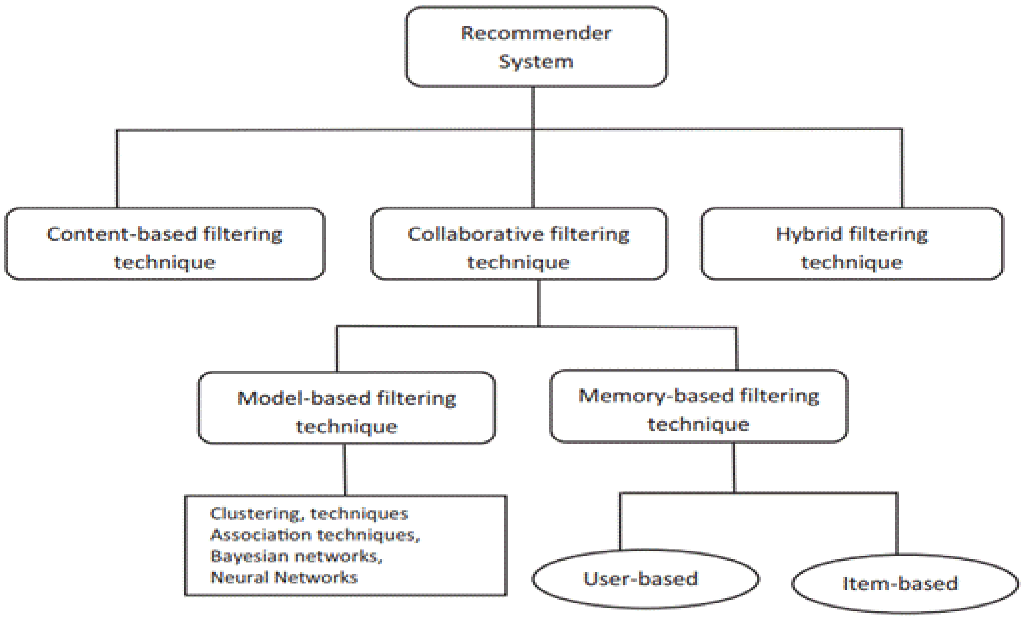


Techniques which selectively make use of both approaches are called Hybrid recommendation systems. We have implemented recommendation using both approaches.

1. COLLABORATIVE FILTERING:

1.1 Introduction:

Collaborative Filtering is a technique which is widely used in recommendation systems and is rapidly advancing research area. Collaborative filtering is based on the assumption that people who agreed in the past will agree in the future, and that they will like similar kinds of items as they liked in the past. The system generates recommendations using only information about rating profiles for different users or items. By locating peer users/items with a rating history similar to the current user or item, they generate recommendations using this neighborhood. The two most commonly used methods are memory-based and model-based.


Recommender System
Content-based filtering technique
Collaborative filtering technique
Hybrid filtering technique
Model-based filtering technique
Memory-based filtering technique
Clustering, techniques
Association techniques,
Bayesian networks,
Neural Networks
User-based
Item-based

1.2 User-Based Collaborative Filtering using K-Nearest Approach:

We have focus on (User-Based Collaborative Filtering) UB-CF which is a memory-based method. The main idea behind UB-CF is that people with similar characteristics share similar taste. For example, if you are interested in recommending a movie to our friend Bob, suppose Bob and I have seen many movies together and we rated them almost identically. It makes sense to think that in future as well we would continue to like similar movies and use this similarity metric to recommend movies.

Let’s refer the user for which rating is to be predicted as ‘active user’. In the approach, users are selected based on their similarity to the active user. Correlation between two users can be found out using the formula below:

Let the ratings to a product by person ‘A’ and ‘B’ be ‘Ak’ and ‘Bk’ . And mean values are represented as A,B

$$r(A,B) = \frac{\sum_k (A_k - \bar{A})(B_k - \bar{B})}{\sqrt{\sum_k (A_k - \bar{A})^2 \sum_k (B_k - \bar{B})^2}}$$

Persons with higher correlation are considered as neighbors.

Let Vi,j be the vote of user i on item j. Mean vote for i is

$$\overline{v}_i = \frac{1}{|I_i|} \sum_{j \in I_i} v_{i,j}$$

Ii is items for which user i has voted.

The vote of an ‘active user’ a is :

$$p_{a,j} = \overline{v}_a + \kappa \sum_{i=1}^{n} w(a,i)(v_{i,j} - \overline{v}_i)$$

Here, k is a constant used for normalization. w(a,i) is weights of ’n’ similar users.

The simplest way to calculate w(a,i) is to using KNN algorithm where each point is represented by a user in n-dimensional space. However, there are more efficient algorithms which can be used to calculate weight w.

w(a,i) is calculated by Pearson correlation coefficient :

$$w(a,i) = \frac{\sum_j (v_{a,j} - \overline{v}_a)(v_{i,j} - \overline{v}_i)}{\sqrt{\sum_j (v_{a,j} - \overline{v}_a)^2 \sum_j (v_{i,j} - \overline{v}_i)^2}}$$

The most serious problem collaborative filtering techniques face in a real world is too few ratings by the users. Hence, In the real-world dataset, user vs items matrix may have some null values.

w(a,i) is calculated in such cases as:

$$w(a,i) = \frac{\sum_j f_j \sum_j f_j v_{a,j} v_{i,j} - (\sum_j f_j v_{a,j})(\sum_j f_j v_{i,j}))}{\sqrt{UV}}$$

where

$$U = \sum_j f_j (\sum_j f_j v_{a,j}^2 - (\sum_j f_j v_{a,j})^2)$$

$$V = \sum_i f_j (\sum_i f_j v_{i,j}^2 - (\sum_i f_j v_{i,j})^2)$$

Where fi = log(n/ni) is inverse user frequency. here, n is a number of users. Nj is a number of users voting for item j. is inverse user frequency.

1.3 Working:

- We have created 5 functions namely

  1. Read_movie_title (Filename)**: -** As the name suggests that it is used for **reading** the data from the text file as data is stored in the form of **CSV files**.

  2. Get_ratings_map (Filename)**: -** It is used to store the data in to the arrays named **user_movie_map**. If no user is found that it is stored in the Then their value is stored as null and same is also done for the movie using **movie_user_map**.

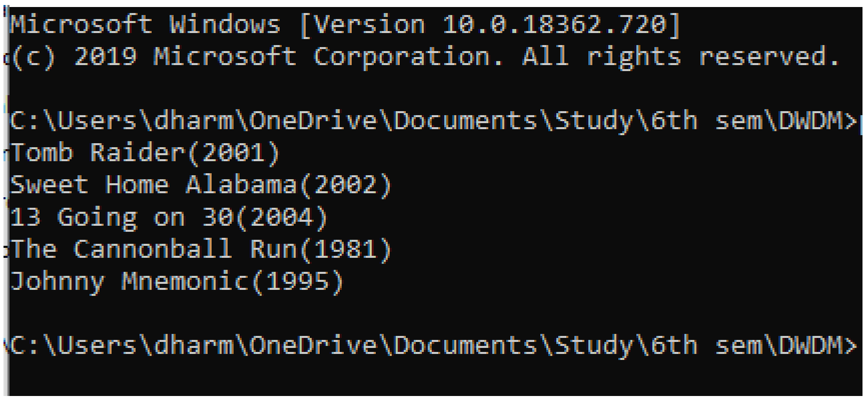
3. Get_user_avg_rating ()**: -**This function is used for finding the **average rating** of each user and this is **stored** in an array known as **user_avg_rating.**

4. Get_user_corr (active_user)**: -**This is used for finding the **relations** between the user provided by us **"active user"** and the **users** present in the **dataset.**

5. Recommendation (active_user, k)**: -** This is the function where the recommendation happen. The value of k in our case is 5. And we have used KNN

- The datasets we have used are given by Netflix and it contains 100000 unique user entries. It basically contains the movies which are relevant to this day and time and not some irrelevant time.

- The result of this is as shown below**: -**



```
Microsoft Windows [Version 10.0.18362.720]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\dharm\OneDrive\Documents\Study\6th sem\DWDM>
Tomb Raider(2001)
Sweet Home Alabama(2002)
13 Going on 30(2004)
The Cannonball Run(1981)
Johnny Mnemonic(1995)

C:\Users\dharm\OneDrive\Documents\Study\6th sem\DWDM>
```

1.4 Advantages:

- It benefits from large user bases.

- Its flexible across different domains.

- Not required to understand item content.

1.5 Disadvantages:

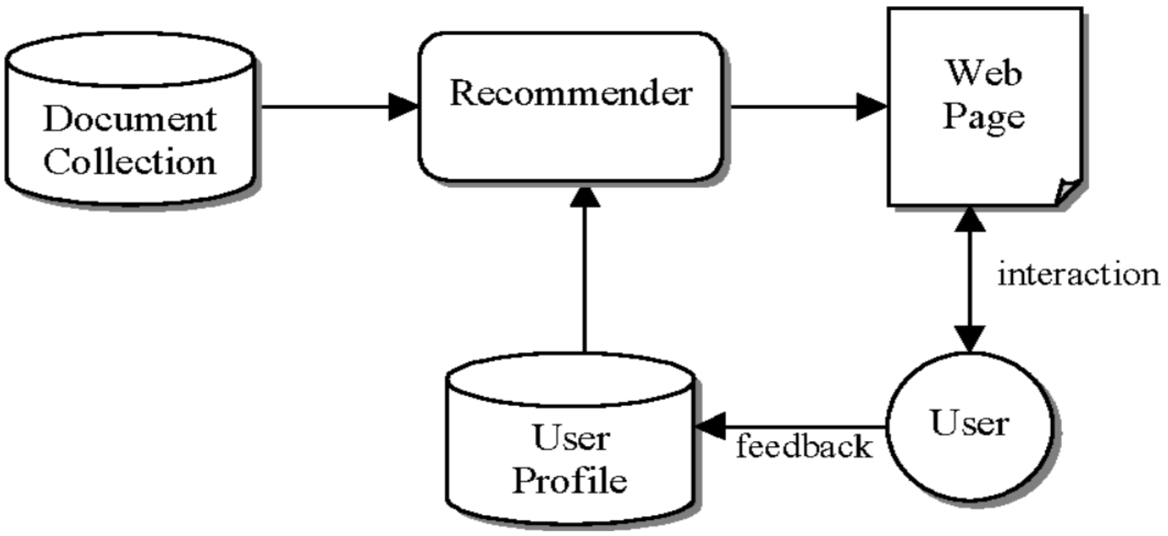
- Need more data.

- Cold start Problem: Problem with new users and new products.

2. CONTENT-BASED FILTERING:

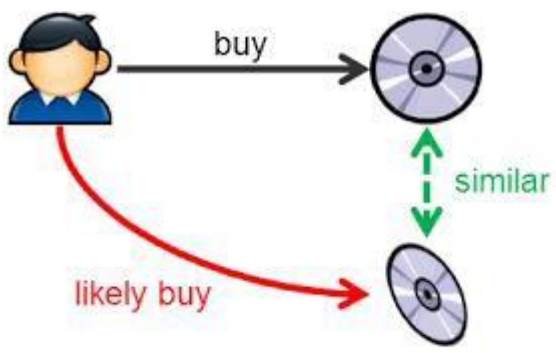
2.1 Introduction:

Content-based filtering, also referred to as cognitive filtering, recommends items based on a comparison between the content of the items and a user profile. The content of each item is represented as a set of descriptors or terms, typically the words that occur in a document. The user profile is represented with the same terms and built up by analyzing the content of items which have been seen by the user. Content-based filtering uses item features to recommend other items similar to what the user likes, based on their previous actions or explicit feedback.



Approach 1: Analyzing the Description of Content Only

Approach 1 is similar to **item-based collaborative filtering**. In short, the system will recommend anything similar to an item you like before.



In model-building stage, the system first find the similarity between all pairs of items, then it uses the most similar items to a user’s already-rated items to generate a list of recommendations in recommendation stage.

How can you find the similarity between items?

The similarity will be derived from the **description** of the item and the concept of **TF-IDF** will be introduced. Then each item will be represented by a **TF-IDF vector**.

Term Frequency-Inverse Document Frequency(TF-IDF)

TF-IDF is in the sub-area of Natural Language Processing(NLP). It is used in information retrieval for feature extraction purposes. In short, you are somehow counting the occurrence of each words in a document and weight the importance of each words, and calculate a score for that document.

Term Frequency

Frequency of the word in the current document to the total number of words in the document. It signifies the occurrence of the word in a document and gives higher weight when the frequency is more so it is divided by document length to normalize.

$$\mathrm{Tf}(t) = \frac{\text{Frequency occurence of term t in document}}{\text{Total number of terms in document}}$$

Inverse Document Frequency

Total Number of Documents to the frequency occurrence of documents containing the word. It signifies the rarity of the word as the word occurring the document is less the IDF increases. It helps in giving a higher score to rare terms in the documents.

$$\mathrm{Idf}(t) = log_{10}(\frac{\text{Total Number of documents}}{\text{Number of documents containing term t}})$$

TF-IDF Vector

F-IDF is a measure used to evaluate how important a word is to a document in a document corpus. The importance of the word increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus. TF-IDF value for the tags, then we can make the keyword vectors for each item. Each row below represent a keyword vector for one item.

Compare the Similarity of the item TF-IDF vector

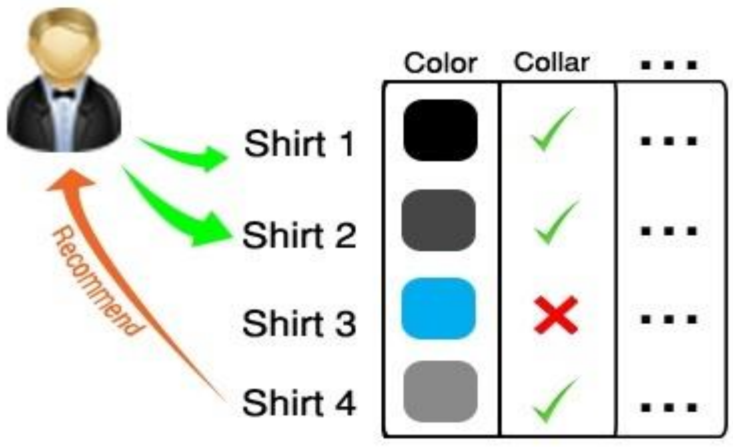
To compute how similar the item vectors are, we will can use various methods such as:

- Cosine Similarity
- Euclidean Distance
- Peason's Correlation

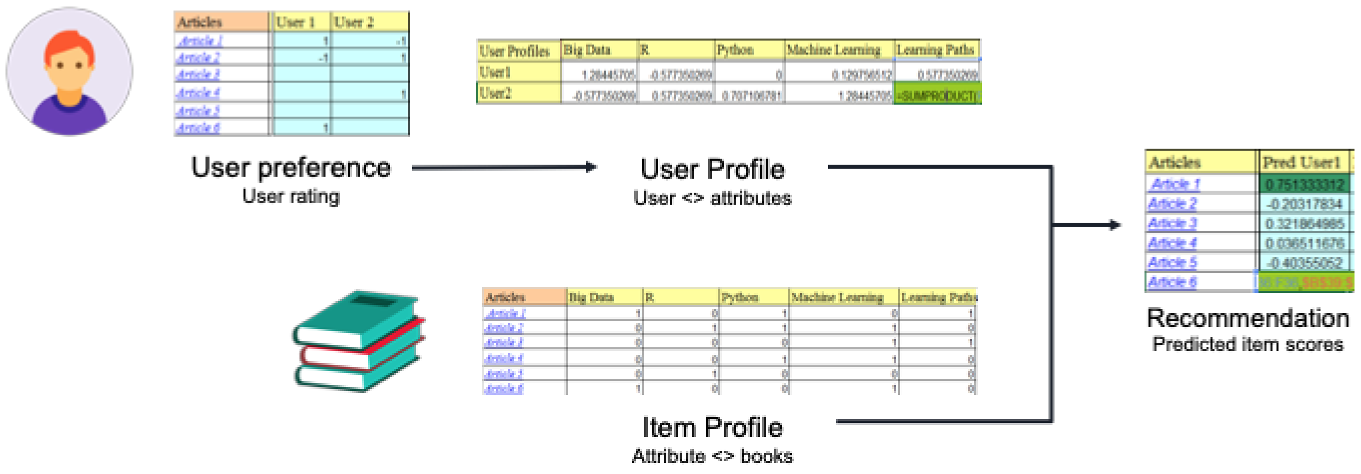
Then the recommender will give recommendation based on the most similar items.

Approach 2: Building User Profile and Item Profile from User Rated Content

Approach 2 leverages **description or attributes** from items the user has interacted to recommend similar items. It depends only on the user **previous choices**, making this method robust to **avoid the cold-start problem**. For textual items, like articles, news and books, it is simple to use the article **category** or **raw text** to build **item profiles** and **user profiles**.



In this approach contents of the product are already **rated** based on the **user's preference (User Profile)**, while the genre of an item is an **implicit features** that it will be used to build **Item Profile**. An **item score** is then predicted by using both profiles and recommendation can be made. Similar to approach 1, TF-IDF technique will also be used in this approach.



Articles	User 1	User 2
Article 1	1	-1
Article 2	-1	1
Article 3		
Article 4		1
Article 5		
Article 6	1	

User Profiles	Big Data	R	Python	Machine Learning	Learning Paths
User1	1.28445705	-0.577350269	0	0.129756512	0.577350269
User2	-0.577350269	0.577350269	0.707106781	1.28445705	=SUMPRODUCT(

Articles	Big Data	R	Python	Machine Learning	Learning Paths
Article 1	1	0	1	0	1
Article 2	0	1	1	1	0
Article 3	0	0	0	1	1
Article 4	0	0	1	1	0
Article 5	0	1	0	0	0
Article 6	1	0	0	1	0

Articles	Pred User1
Article 1	0.751333312
Article 2	-0.20317834
Article 3	0.321864985
Article 4	0.036511676
Article 5	-0.40355062
Article 6	[illegible]

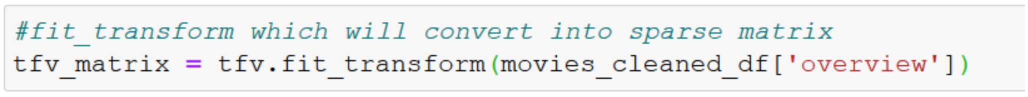
How to build User Profile and Item Profile?

- rating table: user-to-book relationship
- item profile: attribute-to-book relationship
- user profile: user-to-attribute relationship (we can predict all item score of a particular user based on his/her user profile and item profile.)

2.2 Working:

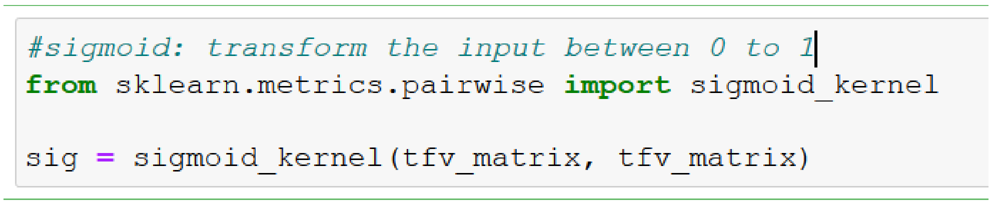
- Here used TfidfVectorizer technique which will create document matrix from sentences. (Overview column)

```
# nlp concept, TfidfVectorizer is technique to convert sentences into document matrix
from sklearn.feature_extraction.text import TfidfVectorizer

tfv = TfidfVectorizer(min_df=3,max_features=None,
                    #remove unnecessary symbols
                    strip_accents='unicode',analyzer='word',token_pattern=r'\w{1,}',
                    ngram_range=(1,3),
                    #remove words like 'the','is','are'...
                    stop_words ='english')

#filling NaN's with empty string.
movies_cleaned_df['overview'] = movies_cleaned_df['overview'].fillna('')
```

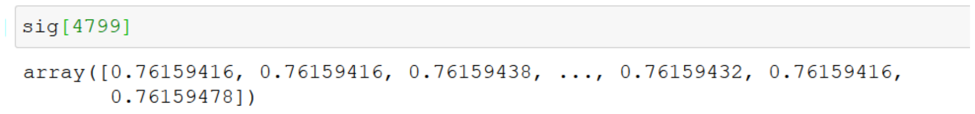


```
#fit_transform which will convert into sparse matrix
tfv_matrix = tfv.fit_transform(movies_cleaned_df['overview'])
```

- Sigmoid will transform the input between 0 to 1 and create a array of values for particular movie.



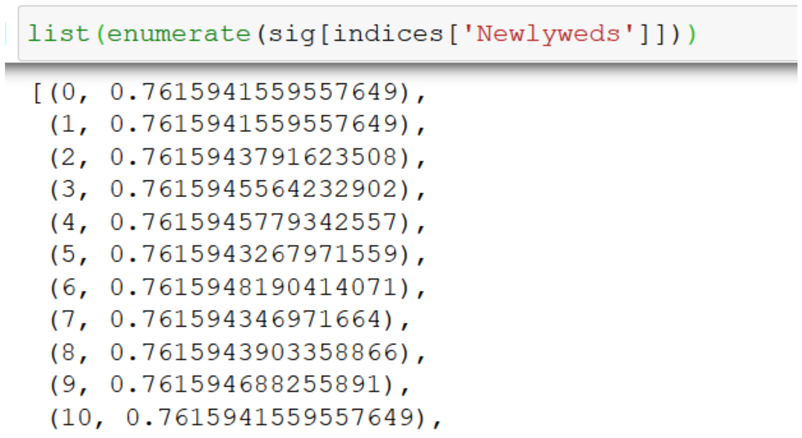
```
#sigmoid: transform the input between 0 to 1
from sklearn.metrics.pairwise import sigmoid_kernel

sig = sigmoid_kernel(tfv_matrix, tfv_matrix)
```



```
sig[4799]
```

```
array([0.76159416, 0.76159416, 0.76159438, ..., 0.76159432, 0.76159416,
       0.76159478])
```

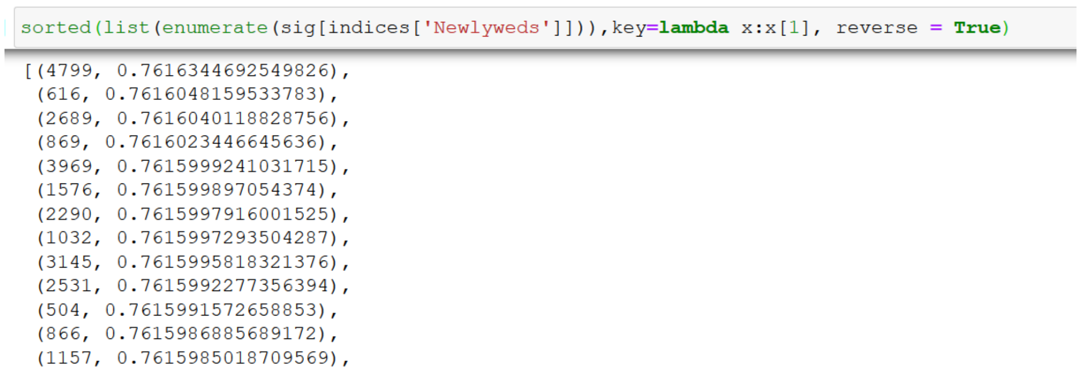
- Here this will give an index of all values in sigmoid for "Newlyweds" movie.



```
list(enumerate(sig[indices['Newlyweds']]))
```

```
[(0, 0.7615941559557649),
 (1, 0.7615941559557649),
 (2, 0.7615943791623508),
 (3, 0.7615945564232902),
 (4, 0.7615945779342557),
 (5, 0.7615943267971559),
 (6, 0.7615948190414071),
 (7, 0.761594346971664),
 (8, 0.7615943903358866),
 (9, 0.761594688255891),
 (10, 0.7615941559557649),
```

- Sort the list according to values and the top most will our recommended movies based on "Newlyweds" movie.



```
sorted(list(enumerate(sig[indices['Newlyweds']])),key=lambda x:x[1], reverse = True)
```

```
[(4799, 0.7616344692549826),
 (616, 0.7616048159533783),
 (2689, 0.7616040118828756),
 (869, 0.761602344664563 6),
 (3969, 0.7615999241031715),
 (1576, 0.761599897054374),
 (2290, 0.7615997916001525),
 (1032, 0.7615997293504287),
 (3145, 0.7615995818321376),
 (2531, 0.7615992277356394),
 (504, 0.761599157265885 3),
 (866, 0.7615986885689172),
 (1157, 0.7615985018709569),
```

- Function takes an index of movie based on that it will create sigmoid score, give an index and sort the list based on values and top 10 will print which are recommended movies according the particular movie.

```
def give_rec(title, sig=sig):
    #get the index corresponding to the original title
    idx=indices[title]

    #get the paiwise similarity score
    sig_scores = list(enumerate(sig[idx]))

    #sort the movies
    sig_scores = sorted(sig_scores, key=lambda x: x[1], reverse=True)

    #scores of the 10 most similar movies
    sig_scores = sig_scores[1:11]


    movie_indices = [i[0] for i in sig_scores]

    #top 10 similar movies will return
    return movies_cleaned_df['original_title'].iloc[movie_indices]
```

- Top 10 movies which is similar to “Spy Kids”.

```
#recommendation based on content
give_rec('Spy Kids')
```

```
1302     Spy Kids 2: The Island of Lost Dreams
1155                  Spy Kids 3-D: Game Over
1769       Spy Kids: All the Time in the World
4044                                Go for It!
3359                               In Too Deep
1631                                  Mr. 3000
1825                Jimmy Neutron: Boy Genius
339                            The Incredibles
3793                      The Velocity of Gary
1081                        Revolutionary Road
Name: original_title, dtype: object
```

2.3 Advantages:

- **User independence:** The content-based method only has to analyze the items and a single user’s profile for the recommendation, which makes the process less cumbersome. Content-based filtering would thus produce more reliable results with fewer users in the system.

- **Transparency:** Collaborative filtering gives recommendations based on other unknown users who have the same taste as a given user, but with content-based filtering, items are recommended on a feature-level basis.

- **No cold start:** As opposed to collaborative filtering, new items can be suggested before being rated by a substantial number of users.

2.4 Disadvantages:

- **Limited content analysis**: If the content doesn’t contain enough information to discriminate the items precisely, the recommendation itself risks being imprecise.
- **Over-specialization:** Content-based filtering provides a limited degree of novelty since it has to match up the features of a user’s profile with available items. In the case of item-based filtering, only item profiles are created and users are suggested items similar to what they rate or search for, instead of their past history. A perfect content-based filtering system may suggest nothing unexpected or surprising.

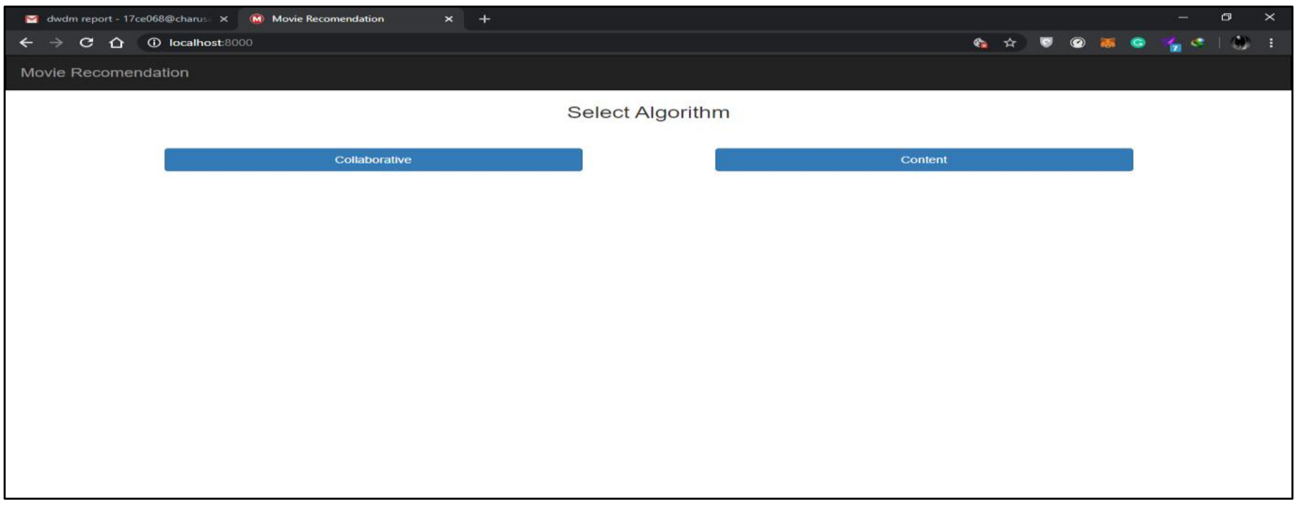
3. **WEB APPLICATION:** This web application is made using Django framework.

3.1 Introduction:

Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel. A web application framework is a toolkit of all components need for application development. The main goal of the Django framework is to allow developers to focus on components of the application that are new instead of spending time on already developed components. Django is fully featured than many other frameworks on the market. It takes care of a lot of hassle involved in the web development; enables users to focus on developing components needed for their application. Django includes dozens of extras you can use to handle common Web development tasks. Django takes care of user authentication, content administration, site maps, RSS feeds, etc. Django takes security seriously and helps developers avoid many common security mistakes, such as SQL injection, cross-site scripting, cross-site request forgery and clickjacking. Its user authentication system provides a secure way to manage user accounts and passwords.

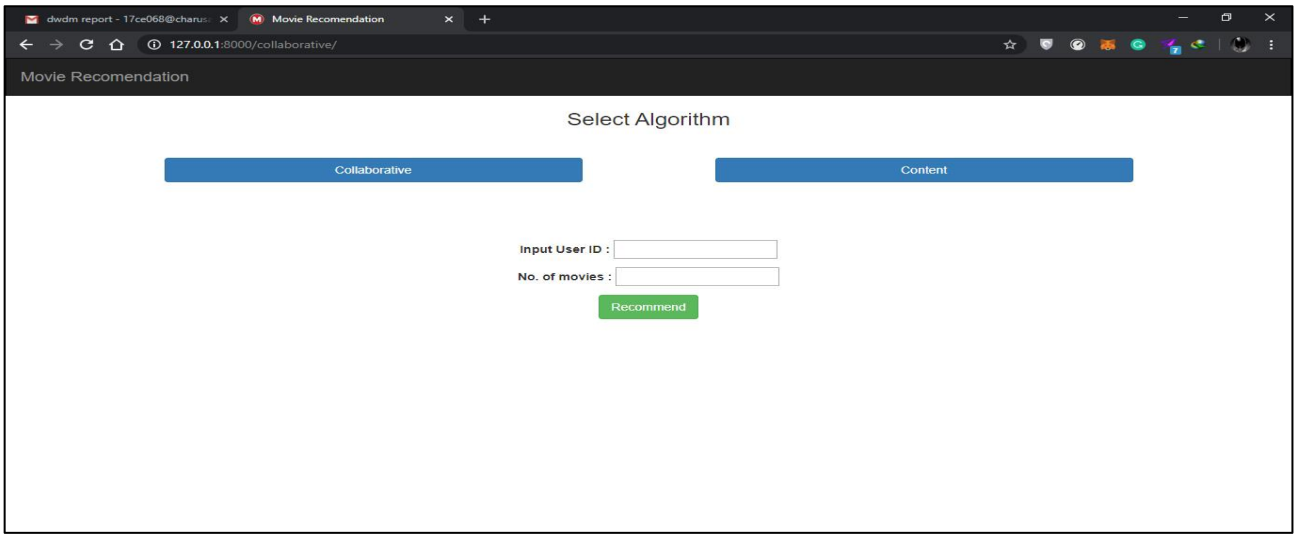
3.2 Working:

UI:



Screenshot i

Collaborative:

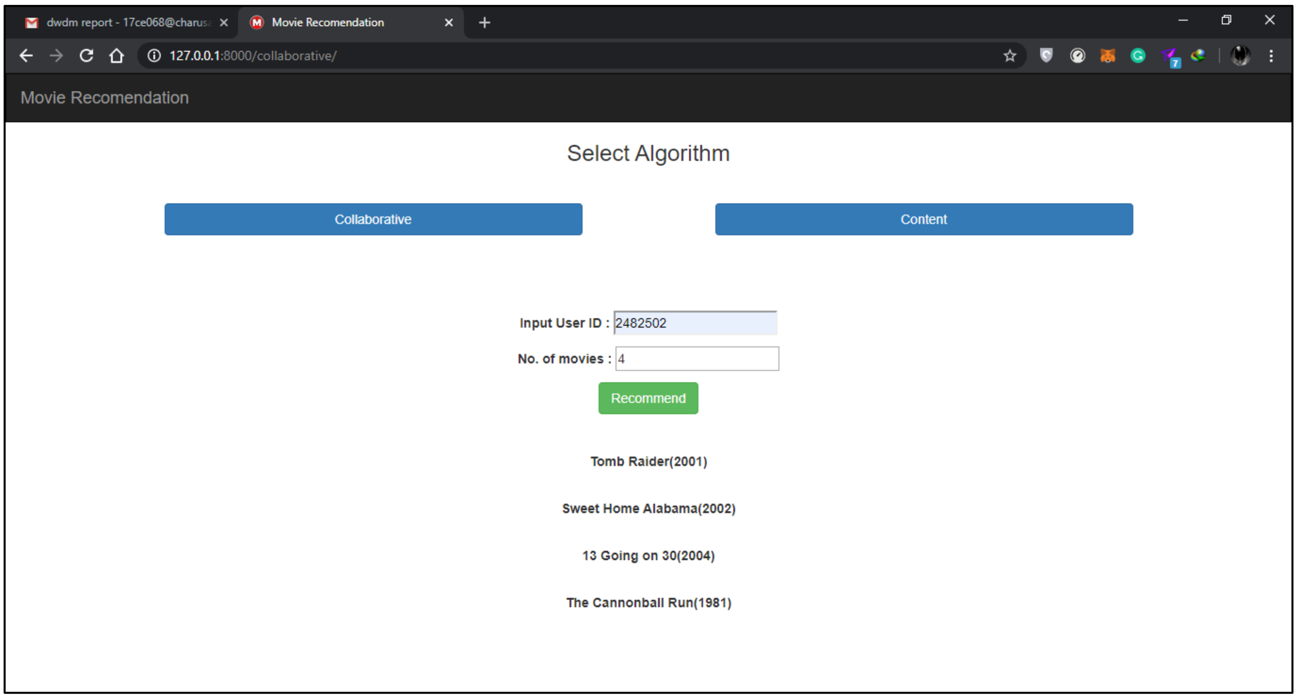


Input User ID :

No. of movies :

Screenshot ii

Collaborative (With Result):



Input User ID : 2482502

No. of movies : 4

Tomb Raider(2001)

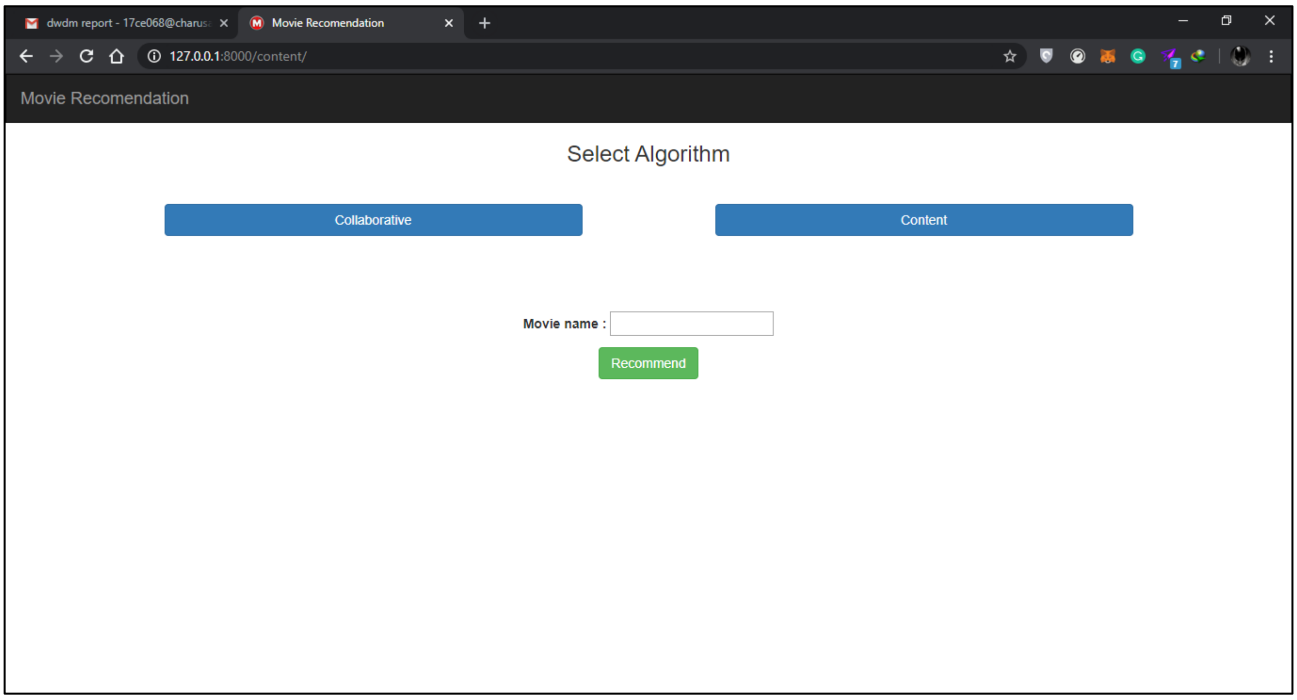
Sweet Home Alabama(2002)

13 Going on 30(2004)

The Cannonball Run(1981)

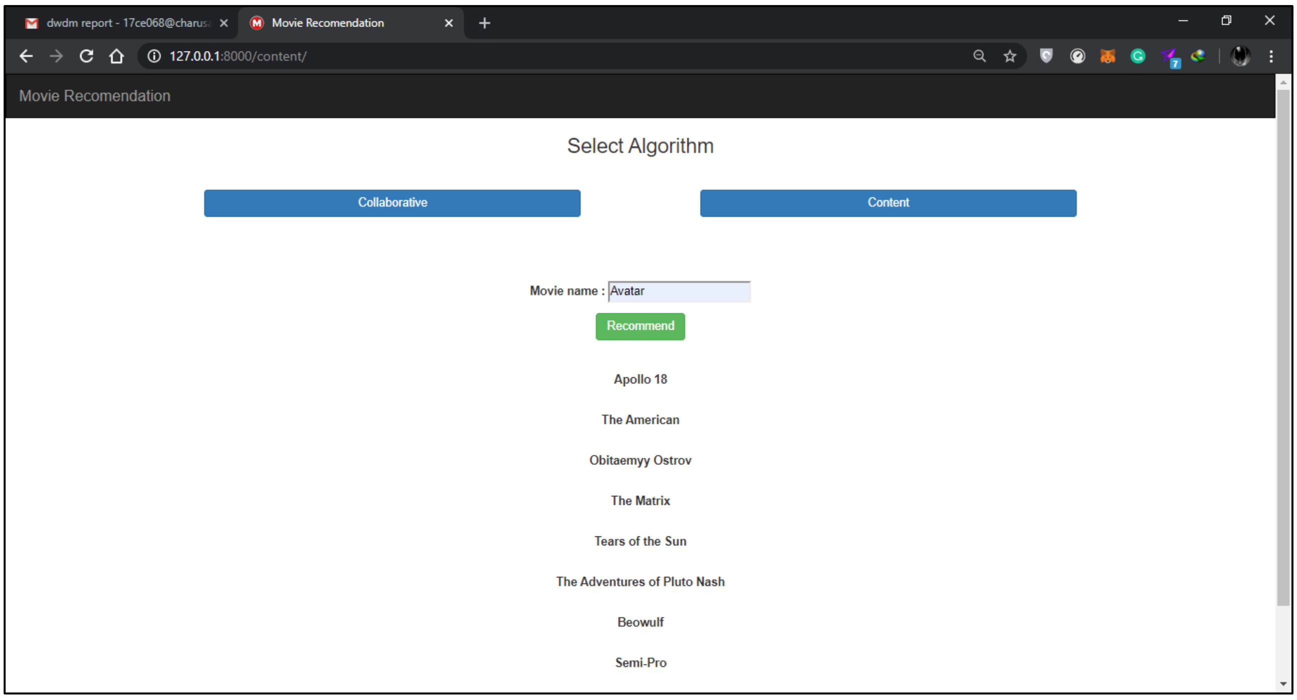
Screenshot iii

Content Based:



Screenshot iv

Content Based (With Result):



Movie name : Avatar

Apollo 18

The American

Obitaemyy Ostrov

The Matrix

Tears of the Sun

The Adventures of Pluto Nash

Beowulf

Semi-Pro

Screenshot v

First a base template is created to apply in every web page that we can see in above Screenshot(i). This base template navigates the user to choose from collaborative algorithm and content-based algorithm. When the collaborative algorithm is selected, a from is generated using fields module of Django framework, i.e. “Input User ID”, “No. of movies” in above Screenshot(ii). And when the form is submitted using “Recommend” button the data is encrypted and sent through request as ‘POST’. From that request the data is decrypted and processed through collaborative algorithm written in python. Once the data is processed it returns list of movies which is sent to the web browser as a response to be displayed on the webpage i.e. Screenshot(iii). The same process repeats in content-based algorithm but with different type of input.

3.3 Advantages:

- Implemented in Python
- Better CDN connectivity and Content Management
- Batteries Included Framework
- Fast Processing
- Offers Rapid-development
- Scalable
- Security

3.4 Disadvantage:

- Uses routing pattern specify its URL
- Django is too monolithic
- Everything is based on Django ORM
- Components get deployed together
- Knowledge of full system is required to work.

4. CONCLUSION:

From this project we can conclude that we have created a successful GUI based application which provides both types of recommendation i.e. Content Based and Collaborative and they can be used with any dataset. In our test we found that especially for movies the user based collaborative filtering is the most effective compared to any other type. As humans tend to like movies suggested by other humans.

5. REFERENCES:

https://medium.com/kunalrdeshmukh/collaborative-filtering-in-recommendation-systems-2fa49be8f518

https://github.com/anjanatiha/Movie-Recommendation-Engine-using-User-Based-Collaborative-Filtering/blob/master/Dataset%20and%20Paper/paper/Empirical%20

https://medium.com/sfu-cspmp/recommendation-systems-user-based-collaborative-filtering-using-n-nearest-neighbors-bf7361dc24e0

https://www.sciencedirect.com/science/article/pii/S1110866515000341

https://en.wikipedia.org/wiki/Recommender_system

https://www.upwork.com/hire/collaborative-filtering-freelancers/

https://www.youtube.com/watch?v=i4a0Of22QRg

https://www.kaggle.com/tmdb/tmdb-movie-metadata

https://www.kanhasoft.com/

https://www.djangoproject.com/

https://data-flair.training/

https://medium.com/towards-artificial-intelligence/content-based-recommender-system-4db1b3de03e7

https://towardsdatascience.com/introduction-to-two-approaches-of-content-based-recommendation-system-fc797460c18c

https://en.wikipedia.org/wiki/Recommender_system

https://medium.com/towards-artificial-intelligence/content-based-recommender-system-4db1b3de03e7