A Thesis/Project/Dissertation Report

on

CONTENT BASED MOVIE RECOMMENDATION SYSTEM

Submitted in partial fulfillment of the
requirement for the award of the degree of

Bachelor Of Technology
(Computer Science & Engineering)

(Established under Galgotias University Uttar Pradesh Act No. 14 of 2011)

Under The Supervision of
Name of Supervisor:

Mr. SHAHADAT HUSSAIN

Submitted By

Satyam Choudhary, 20scse1010398
Vaibhav Sharma, 20scse1010263
Mayank Pathak, 20scse1010047

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING /
DEPARTMENT OF COMPUTERAPPLICATION
GALGOTIAS UNIVERSITY, GREATER NOIDA
INDIA
MONTH, YEAR

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
GALGOTIAS UNIVERSITY, GREATER NOIDA

CANDIDATE'S DECLARATION

I/We hereby certify that the work which is being presented in the thesis/project/dissertation, entitled **"CONTENT BASED MOVIE RECOMMENDATION SYSTEM"** in partial fulfillment of the requirements for the award of the Bachelors Of Technology submitted in the School of Computing Science and Engineering of Galgotias University, Greater Noida, is an original work carried out during the period of October-December, 2021 under the supervision of **Mr. Shahadat Hussain** Designation, Department of Computer Science and Engineering/Computer Application and Information and Science, of School of Computing Science and Engineering , Galgotias University, Greater Noida

The matter presented in the thesis/project/dissertation has not been submitted by me/us for the award of any other degree of this or any other places.

Satyam Choudhary, 20scse1010398
Vaibhav Sharma, 20scse1010263
Mayank Pathak, 20scse1010047

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

CERTIFICATE

The Final Thesis/Project/ Dissertation Viva-Voce examination of Satyam Choudhary_20scse1010398, Vaibhav Sharma_ 20scse1010263, Mayank Pathak_20scse1010047 has been held on _____ and his/her work is recommended for the award of Bachelor of Technology in the School of Computing Science and Engineering of Galgotias University, Greater Noida.

Signature of Examiner(s) **Signature of Supervisor(s)**

Signature of Project Coordinator **Signature of Dean**

Abstract

There are many recommendation-based OTT platforms are available right now but no one is providing the content on the basis of a reference movie. Movie recommendation systems aim to recommend movies that users may be interested in. In this project, we created a content-based movie recommendation system which can use different feature sets, namely Movie Id, Title and Tags For each user, we assign a weight to each feature in a feature set based on the particular user's past behavior. In order to predict a rating for user and a movie, using a particular feature set, we merge the user specific weights of movie's features. We also produce ratings using all feature sets. We evaluate each recommendation method based on precision, recall and measure on five movie recommendations.

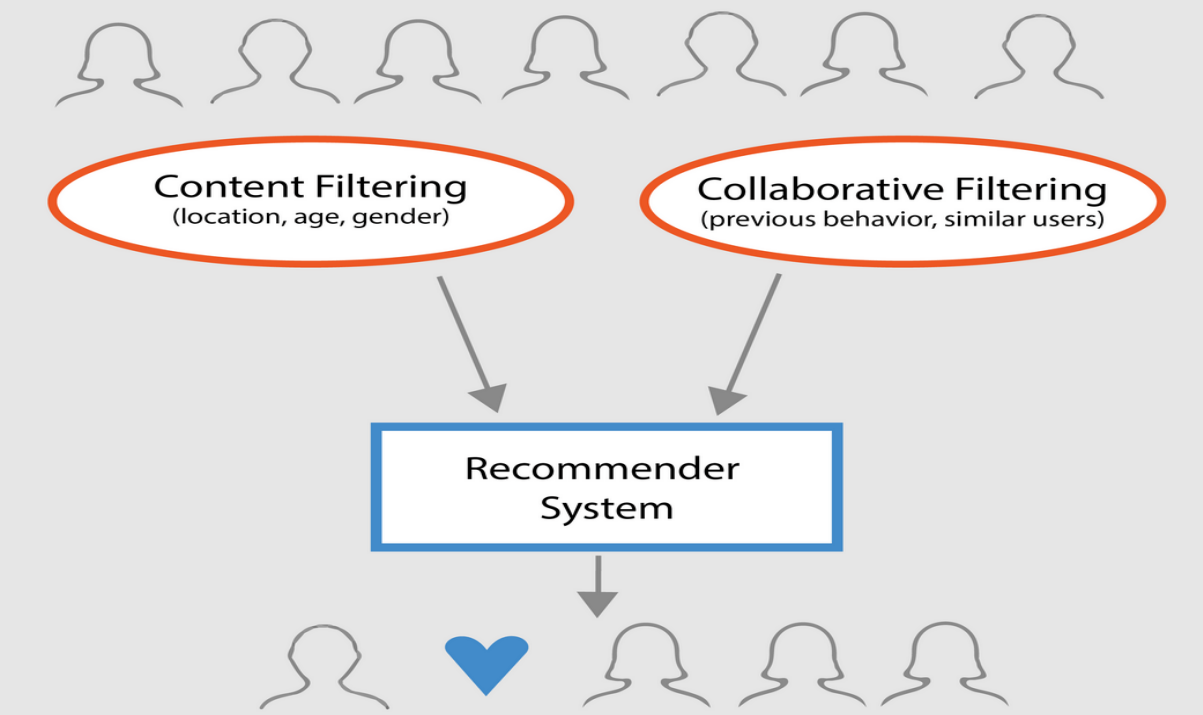
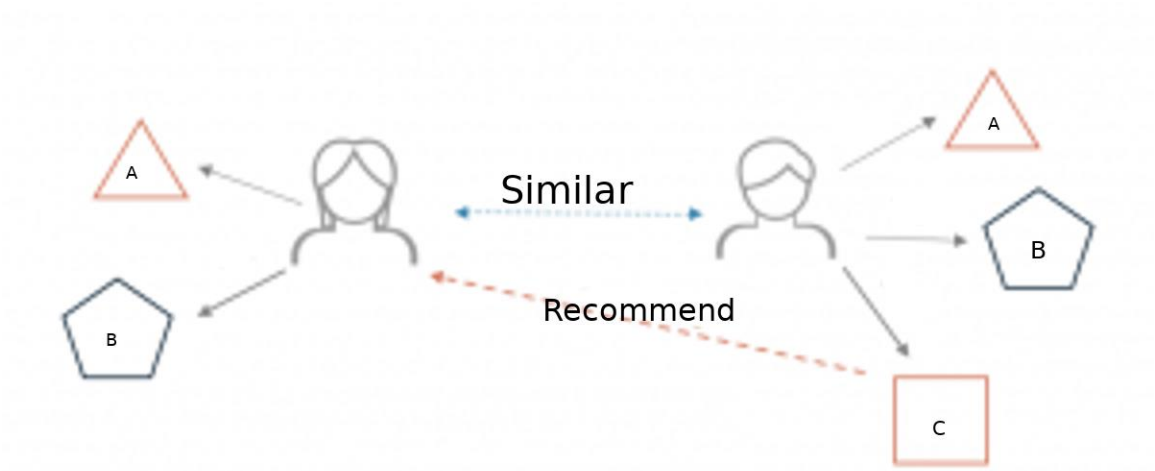


Table of Contents

INTRODUCTION

A recommender system is a simple algorithm whose aim is to provide the most relevant information to a user by discovering patterns in a dataset. The algorithm rates the items and shows the user the items that they would rate highly. An example of recommendation in action is when you visit Amazon and you notice that some items are being recommended to you or when Netflix recommends certain movies to you. They are also used by Music streaming applications such as Spotify and Deezer to recommend music that you might like.

Below is a very simple illustration of how recommender systems work in the context of an e-commerce site.



Two users buy the same items A and B from an e-commerce store. When this happens the similarity index of these two users is computed. Depending on the score the system can recommend item C to the other user because it detects that those two users are similar in terms of the items they purchase.

Different types of recommendation engines

The most common types of recommendation systems are content-based and collaborative filtering recommender systems. In collaborative filtering, the behavior of a group of users is used to make recommendations to

other users. The recommendation is based on the preference of other users. A simple example would be recommending a movie to a user based on the fact that their friend liked the movie. There are two types of collaborative models Memory-Based methods and Model-based methods. The advantage of memory-based techniques is that they are simple to implement and the resulting recommendations are often easy to explain. They are divided into two:

- User-based collaborative filtering: In this model, products are recommended to a user based on the fact that the products have been liked by users similar to the user. For example, if Derrick and Dennis like the same movies and a new movie come out that Derick like, then we can recommend that movie to Dennis because Derrick and Dennis seem to like the same movies.
- Item-based collaborative filtering: These systems identify similar items based on users' previous ratings. For example, if users A, B, and C gave a 5-star rating to books X and Y then when a user D buys book Y they also get a recommendation to purchase book X because the system identifies book X and Y as similar based on the ratings of users A, B, and C.

Model-based methods are based on Matrix Factorization and are better at dealing with sparsity. They are developed using data mining, machine learning algorithms to predict users' rating of unrated items. In this approach techniques such as dimensionality reduction are used to improve accuracy. Examples of such model-based methods include Decision trees, Rule-based Model, Bayesian Model, and latent factor models.

Content-based systems use metadata such as genre, producer, actor, musician to recommend items say movies or music. Such a recommendation would be for instance recommending Infinity War that featured Vin Diesel because someone watched and liked The Fate of the Furious. Similarly, you can get music recommendations from certain artists because you liked their music. Content-based systems are based on the idea that if you liked a certain item you are most likely to like something that is similar to it.

LITERATURE SURVEY

The OTT market is already cluttered in many geographies. A study from a major OTT market tracking organization, Parks Associates, suggests the number of OTT video services in the USA has more than doubled in the past six years to 300 platforms through the third quarter (ended Sept. 30, 2020). Similarly, India had 36 OTT streaming services and the number has gone up to 60 by mid-2020. New entrants are eyeing the market and the competition is only going to increase. The most important trend observed among Indian consumers is their willingness to pay for content during the pandemic.

In this context, brands need to offer a sharp, differentiated value proposition primarily led by either a demographic or psychographic segment. A diverse country like India offers opportunities to target consumers by language preference.

As seen through another key trend of 2020, there has been a continuous emergence of hyperlocal OTT services such as Hoichoi (Bengali), Aha (Telugu), Koode & Manorama Max (Malayalam), Planet Marathi OTT (Marathi), CityShor.TV (Gujarati), and more.

In total there are 17 major languages for delivering OTT content in India currently. Each of these has its own vast library of content and niche audience. Segmentation through a lifestyle preference is also possible as seen with the successful launch of Discovery+ in India which has a unique proposition of unscripted content aimed at lifelong learners.

This hyper localization or segmentation through a niche can also be seen with the emergence of BritBox. It is an online digital video subscription service showing primarily British shows made available for UK, USA, Canada, and Australian audiences with South Africa next on the list.

According to a survey conducted by Deloitte, in May 2020 among US consumers, the reason for choosing a particular brand of service seemed to indicate a preference for both a broad range of shows & movies and content not available elsewhere.

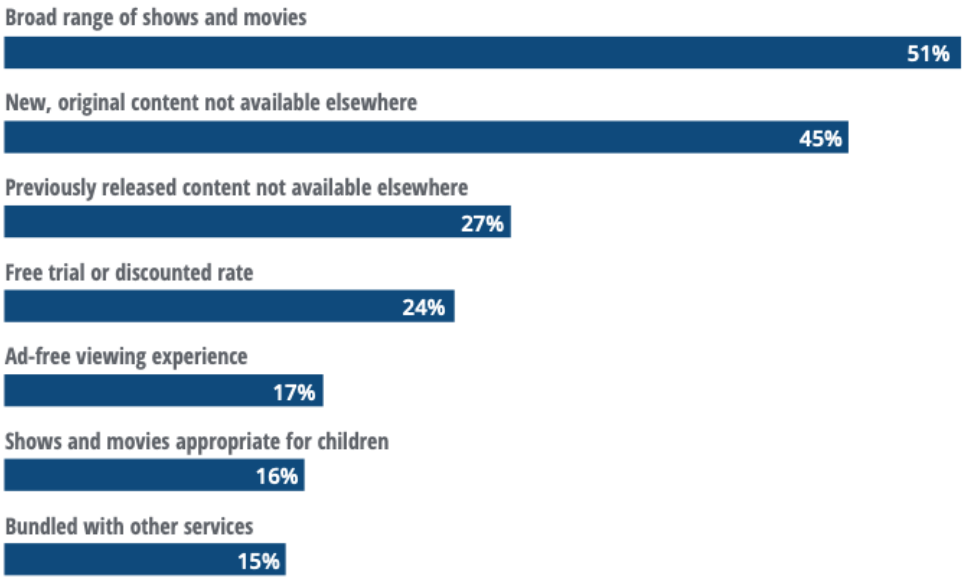


Table: To show the wide range of content on the OTT Platforms.

It reflects in the content strategy of several brands which are now emphasizing 'originals' which then call for huge investments in creative talent and production costs. As consumers, we have also chosen to snack on a streaming service based on a particular show or movie and not renew the subscription when the show completes a season. To that extent, brand loyalty cannot be taken for granted in this segment.

The above survey also found that 66% of people were frustrated when the content they wanted to watch was removed from service, and 53% were frustrated by having to subscribe to multiple services to access the content they want. From October 2020 to February 2021, the churn rate for streaming video services held at around 37%.

Late entrants in a particular category will then have their work cut out to wean away from the audience from established players. There are several key factors to take into account while designing a successful OTT platform. A quick study and research go a long way in the successful journey of your idea.

SOURCE CODE

jupyter Movie Recommender System-Main Last Checkpoint: 2 hours ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

Run Code

In [1]:
```
import numpy as np
import pandas as pd
```

In [2]:
```
movies = pd.read_csv('tmdb_5000_movies.csv')
credits = pd.read_csv('tmdb_5000_credits.csv')
```

In [3]:
```
movies = movies.merge(credits,on='title')
```

In [4]:
```
movies = movies[['movie_id','title','overview','genres','keywords','cast','crew']]
```

In [5]:
```
movies.isnull().sum()
```

Out[5]:
```
movie_id    0
title       0
overview    3
genres      0
keywords    0
cast        0
crew        0
dtype: int64
```

In [6]:
```
movies.dropna(inplace=True)
```

In [7]:
```
movies.duplicated().sum()
```

Out[7]:
```
0
```

In [8]:
```
movies.iloc[0].genres
```

Out[8]:
```
'[{"id": 28, "name": "Action"}, {"id": 12, "name": "Adventure"}, {"id": 14, "name": "Fantasy"}, {"id": 878, "name": "Science Fi
```

jupyter Movie Recommender System-Main Last Checkpoint: 2 hours ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

Run Code

In [12]:
```
import ast
ast.literal_eval()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_15208/2341118007.py in <module>
      1 import ast
----> 2 ast.literal_eval()

TypeError: literal_eval() missing 1 required positional argument: 'node_or_string'
```

In [13]:
```
def convert(obj):
    l=[]
    for i in ast.literal_eval(obj):
        l.append(i['name'])
    return l
```

In [14]:
```
movies['genres'] = movies['genres'].apply(convert)
```

In [15]:
```
movies['keywords'] = movies['keywords'].apply(convert)
```

In [16]:
```
def convert3(obj):
    l=[]
    counter=0
    for i in ast.literal_eval(obj):
        if counter!=3:
            l.append(i['name'])
            counter+1
        else:
            break
    return l
```

In [17]:
```
movies['cast'] = movies['cast'].apply(convert3)
```

jupyter Movie Recommender System-Main Last Checkpoint: 2 hours ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

Run Code

In [18]:
```
def fetch_director(obj):
    l=[]
    for i in ast.literal_eval(obj):
        if i['job']=='Director':
            l.append(i['name'])
            break
    return l
```

In [19]:
```
movies['crew'] = movies['crew'].apply(fetch_director)
```

In [20]:
```
movies['overview'] = movies['overview'].apply(lambda x:x.split())
```

In [21]:
```
movies['genres'] = movies['genres'].apply(lambda x:[i.replace(" ","") for i in x])
```

In [22]:
```
movies['keywords'] = movies['keywords'].apply(lambda x:[i.replace(" ","") for i in x])
```

In [23]:
```
movies['cast'] = movies['cast'].apply(lambda x:[i.replace(" ","") for i in x])
```

In [24]:
```
movies['crew'] = movies['crew'].apply(lambda x:[i.replace(" ","") for i in x])
```

In [25]:
```
movies['tags'] = movies['overview'] + movies['genres'] + movies['keywords'] + movies['cast'] + movies['crew']
```

In [26]:
```
movies.head()
```

Out[26]:

	movie_id	title	overview	genres	keywords	cast	crew	tags
0	19995	Avatar	[In, the, 22nd, century,, a, paraplegic, Marin...	[Action, Adventure, Fantasy, ScienceFiction]	[cultureclash, future, spacewar, spacecolony, ...	[SamWorthington, ZoeSaldana, SigourneyWeaver, ...	[JamesCameron]	[In, the, 22nd, century,, a, paraplegic, Marin...
1	285	Pirates of the Caribbean: At World's End	[Captain, Barbossa,, long, believed, to, be, d...	[Adventure, Fantasy, Action]	[ocean, drugabuse, exoticisland, eastindiatrad...	[JohnnyDepp, OrlandoBloom, KeiraKnightley, Ste...	[GoreVerbinski]	[Captain, Barbossa,, long, believed, to, be, d...
2	206647	Spectre	[A, cryptic, message, from	[Action, Adventure,	[spy, basedonnovel, secretagent, sequel	[DanielCraig, ChristophWaltz	[SamMendes]	[A, cryptic, message, from

jupyter Movie Recommender System-Main Last Checkpoint: 2 hours ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

Run Code

In [27]:
```
new_df = movies[['movie_id','title','tags']]
```

In [28]:
```
new_df['tags'] = new_df['tags'].apply(lambda x:" ".join(x))
```

```
C:\Users\satyam\AppData\Local\Temp/ipykernel_15208/3089450492.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  new_df['tags'] = new_df['tags'].apply(lambda x:" ".join(x))
```

In [29]:
```
new_df.head()
```

Out[29]:

	movie_id	title	tags
0	19995	Avatar	In the 22nd century, a paraplegic Marine is di...
1	285	Pirates of the Caribbean: At World's End	Captain Barbossa, long believed to be dead, ha...
2	206647	Spectre	A cryptic message from Bond's past sends him o...
3	49026	The Dark Knight Rises	Following the death of District Attorney Harve...
4	49529	John Carter	John Carter is a war-weary, former military ca...

In [30]:
```
new_df['tags'] = new_df['tags'].apply(lambda x:x.lower())
```

```
C:\Users\satyam\AppData\Local\Temp/ipykernel_15208/3214958533.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  new_df['tags'] = new_df['tags'].apply(lambda x:x.lower())
```

In [31]:
```
new_df.head()
```

jupyter Movie Recommender System-Main Last Checkpoint: 2 hours ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

Run Code

Out[31]:

	movie_id	title	tags
0	19995	Avatar	in the 22nd century, a paraplegic marine is di...
1	285	Pirates of the Caribbean: At World's End	captain barbossa, long believed to be dead, ha...
2	206647	Spectre	a cryptic message from bond's past sends him o...
3	49026	The Dark Knight Rises	following the death of district attorney harve...
4	49529	John Carter	john carter is a war-weary, former military ca...

In [32]:
```
import nltk
```

In [33]:
```
from nltk.stem.porter import PorterStemmer
ps = PorterStemmer()
```

In [34]:
```
def stem(text):
    y = []
    for i in text.split():
        y.append(ps.stem(i))
    return " ".join(y)
```

In [35]:
```
new_df['tags'] = new_df['tags'].apply(stem)
```

```
C:\Users\satyam\AppData\Local\Temp/ipykernel_15208/3213734980.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  new_df['tags'] = new_df['tags'].apply(stem)
```

In [36]:
```
new_df.shape
```

Out[36]:
```
(4806, 3)
```

jupyter Movie Recommender System-Main Last Checkpoint: 2 hours ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

Run Code

In [37]:
```
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer(max_features=5000,stop_words='english')
```

In [38]:
```
vectors = cv.fit_transform(new_df['tags']).toarray()
```

In [39]:
```
cv.get_feature_names()
```

```
 '18th',
 '19',
 '1930',
 '1940',
 '1950',
 '1960',
 '1960s',
 '1970',
 '1970s',
 '1980',
 '1990',
 '19th',
 '19thcenturi',
 '20',
 '20th',
 '24',
 '25',
 '30',
 '3d',
 '40',
```

In [40]:
```
from sklearn.metrics.pairwise import cosine_similarity
```

In [41]:
```
similarity = cosine_similarity(vectors)
```

In [42]:
```
sorted(list(enumerate(similarity[0])),reverse = True,key=lambda x:x[1])[1:6]
```

Out[42]:
```
[(1216, 0.27028123880866767),
 (582, 0.23162743094465488),
```

jupyter Movie Recommender System-Main Last Checkpoint: 2 hours ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

Run Code

In [41]:
```
similarity = cosine_similarity(vectors)
```

In [42]:
```
sorted(list(enumerate(similarity[0])),reverse = True,key=lambda x:x[1])[1:6]
```

Out[42]:
```
[(1216, 0.27028123880866767),
 (582, 0.23162743094465488),
 (2333, 0.22996655275195),
 (3730, 0.22498852128662872),
 (507, 0.22438727760202976)]
```

In [43]:
```
def recommend(movie):
    movie_index = new_df[new_df['title'] == movie].index[0]
    distances = similarity[movie_index]
    movies_list = sorted(list(enumerate(distances)),reverse = True,key=lambda x:x[1])[1:6]
    for i in movies_list:
        print(new_df.iloc[i[0]].title)
```

In [44]:
```
recommend("Avatar")
```

```
Aliens vs Predator: Requiem
Battle: Los Angeles
Predator
Falcon Rising
Independence Day
```

In [ ]:

OUTPUT

In [44]:
```
recommend("Avatar")
```

```
Aliens vs Predator: Requiem
Battle: Los Angeles
Predator
Falcon Rising
Independence Day
```

CONCLUSION

To conclude, a recommender system powered by content-based filtering performed using the cosine similarity algorithm can make better recommendations for users by suggesting them movies that have similar key features like the Movie_Id, Title and Tags which is the composition of Overview, Genres, Keywords, Cast and Crew,etc.

IMPROVEMENT TECHNIQUE FOR RECOMMENDED SYSTEM

This system can be improved by building a Memory-Based Collaborative Filtering based system. In this case, we'd divide the data into a training set and a test set. We'd then use techniques such as cosine similarity to compute the similarity between the movies. An alternative is to build a Model-based Collaborative Filtering system. This is based on matrix factorization. Matrix factorization is good at dealing with scalability and sparsity than the former. You can then evaluate your model using techniques such as Root Mean Squared Error (RMSE).

REFERENCES

- https://www.sciencedirect.com/science/article/pii/S2666827020300062
- https://www.kaggle.com/rounakbanik/movie-recommender-systems
- https://www.python.org/
- https://en.wikipedia.org/wiki/Visual_Studio_Code
- https://docs.python.org/3/library/tkinter.html
- www.google.com
- www.youtube.com
- Literature survey: https://www.robosoftin.com/blog/ott-brands-trends-and-opportunities.
- Google Scholar