

# **Movie Recommendation using Content Based Filtering**

Project Report

**Submitted By:**

**Sk Hasimuddin (19370049)**

Masters of Science

In

Computer Science

**Under the Guidance of**

**Dr. R. Subramanian**

Professor

Department of Computer Science



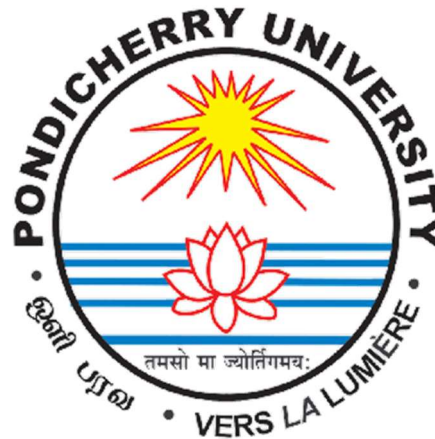
**Pondicherry University**

R Venkat Raman Nagar, Kalapet

Pondicherry - 605 014, India

# Pondicherry University

Department of Computer Science



## CERTIFICATE

This is to Certified that this project report “Movie Recommendation System” is submitted by “Sk Hasimuddin (19370049)” who carried out the project work under my supervision. I approve this project for submission.

.....  
Dr. S. Sivasathya  
(HoD)

.....  
Dr. R. Subramanian  
Professor  
(Project Guide)

# ACKNOWLEDGEMENT

It gives me immense pleasure to express my deepest sense of gratitude and sincere thanks to my respected guide **Dr. R. Subramanian** (Professor), Computer Science, Pondicherry University, Pondicherry, for their valuable guidance, encouragement and help for completing this work. Their useful suggestions for this whole work and co-operative behaviour are sincerely acknowledged.

I also wish to express my indebtedness to my parents as well as my family member whose blessings and support always helped me to face the challenges ahead.

I would like to extend my thanks to students of the Department of Computer Science, Pondicherry University, whose valuable comments and suggestions gave a different perspective and ideas. I am specially extending my thanks to my friend Rajarsi Saha, who helped me developing the frontend in this project.

Sk Hasimuddin (19370049)

# Table of Contents

<b>Title</b>	<b>Page No.</b>
Certificate	ii
Acknowledgement	iii
Table of Contents	iv
List of Figures	v
Abstract	1
Keywords	1
Hardware and Software Requirements	1
Introduction	1
Usual Approaches	2
Collaborative Filtering	2
Content Based Filtering	3
Dataset and Features	3
Pre-processing and Visualization	3
Building the Recommender Engine	7
Result Section	11
Conclusion	11
Building the Frontend	12
Output section of frontend of recommender engine	13
Reference	16

## List of Figures

<b>Name</b>	<b>Page No.</b>
Collaborative Filtering	2
User-item interaction matrix	2
Content based filtering	3
Number of movies by Primary_genre	5
Number of movies by Primary_language	5
Number of movies by Primary_country	6
Movies in each language by each genre	7
A simple example of working of CountVectorizer	9
Formula for Cosine Similarity	10
Cosine Similarity	10

## Abstract

As the technology and internet are evolving thorough out the world, we are producing data by each interaction. This vast amount of data is being used in our benefits as well. One of those benefits is movie recommendation system. A recommender system filters out data by using different algorithms and recommend the item to a user. What it does is actually it first analyses one user's past data and based on that a recommender system suggest products or items that one person may be interested in and likely to buy or watch. Now, what happens in case of a completely new person who does not have any past data or behaviours? In this situation the recommendation system offers him or her the trending movies or the new movies or the all-time best movies or the recommendation system would offer him what has been being seen and highly rated by his or her peers. In this project, we attempt to briefly understand the various approaches to build a recommender system and compare their performance on IMDB dataset <sup>[1]</sup>. This report provides a detailed summary of the project "Movie Recommendation System" as part of fulfilment of the Master's Writing Project, Computer Science Department, Pondicherry University's. The report includes a description of the topic, system

architecture, and provides a detailed description of the work done till point. Included in the report are the detailed descriptions of the work done: snapshot of the implementations, various approaches, and tools used so far.

**Keywords:** Recommender system, collaborative filtering, content-based filtering, item-based recommendations.

## Hardware and Software requirements

A desktop or laptop, python, jupyter notebook.

## Introduction

Recommender system has become increasingly popular over the last few years and are now utilized in almost every online platform. As in today's world each person is faced many choices. For example, if one wants to watch a movie without any specific idea of what he wants then there is a wide range of options and one might spend a heavy amount of time in searching for a movie and still that movie might not be the one that he wants to watch. And that's where recommendation system comes in picture.

A recommender system, or a recommendation system, is a subclass of information filtering system that seeks to

predict the "rating" or "preference" a user would give to an item.<sup>[2]</sup>

They are mainly used in e-commerce. Recommender systems have also been developed to explore research articles and experts,<sup>[3]</sup> collaborators,<sup>[4]</sup> and financial services.<sup>[5]</sup>

### Usual Approaches

There are mainly two types of approaches to a recommendation engine, one is Collaborative Filtering and another is Content Based recommendations. But there is also another method that is mix of collaborative filtering and content-based methods namely Hybrid method.

### Collaborative Filtering:

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.

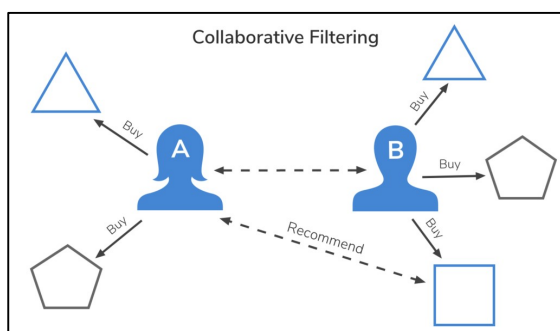


Figure 1: Collaborative Filtering

It is solely based on the past interactions recorded between users and items in order to produce new recommendations. These interactions are stored in the so-called "user-item interaction matrix".

		<i>Items</i>					
		<i>1</i>	<i>2</i>	<i>...</i>	<i>i</i>	<i>...</i>	<i>m</i>
<i>Users</i>	<i>1</i>	5	3		1	2	
	<i>2</i>		2				4
	<i>:</i>			5			
	<i>u</i>	3	4		2	1	
	<i>:</i>					4	
	<i>n</i>			3	2		

Figure 2: user-item interaction matrix

The main idea that rules collaborative methods is that these past user-item interactions are sufficient to detect similar users and/or similar items and make predictions based on these estimated proximities. Many algorithms have been used in measuring user similarity or item similarity in recommender systems. For example, the k-nearest neighbour (k-NN) approach<sup>[6]</sup> and the Pearson Correlation as first implemented by Allen.<sup>[7]</sup>

The main advantage of collaborative approaches is that they require no information about users or items and, so, they can be used in many situations. Moreover, the more users interact with items, the new recommendations become more accurate. However, as it only considers past interactions to make

recommendations, collaborative filtering suffer from the “cold start problem”.

### Content Based Filtering:

Content-based filtering methods are based on a description of the item and a profile of the user's preferences.<sup>[8]</sup> These methods are best suited to situations where there is known data on an item. Recommendations are based on attributes of the item. “Similarity” is measured against product attributes.

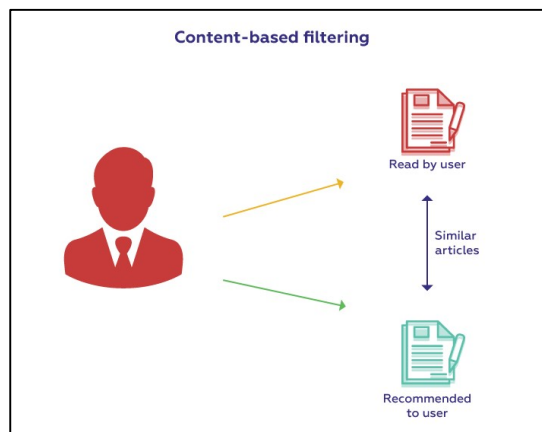


Figure 3: Content Based Filtering

Suppose I watch a movie in a particular genre, then I will be recommended movies within that specific genre. The movie's attributes, like title, year of release, director and cast, are also helpful in identifying similar movie content.

### Dataset and Features

I used the IMDb movies extensive dataset from Kaggle<sup>[1]</sup> which was uploaded by

Stefano Leone.<sup>[9]</sup> The whole data was scrapped from the IMDb<sup>[10]</sup> website. IMDb is an online database of information related to films, television programs, home videos, video games, and streaming content online – including cast, production crew and personal biographies, plot summaries, trivia, ratings, and fan and critical reviews.<sup>[11]</sup> There are four csv files under the data folder that was uploaded to kaggle from which I've used IMDb movie.csv file. This file contains over 85000 movies with attributes such as movie name, movie description, genre, casts, director, movie runtime, etc.

### Pre-processing and Visualization

To do the pre-processing, visualization, or any other thing with the data, first it need to be loaded. While loading the data I see a data type warning showing there in third column namely year column, so going into the depth of this, I see “TV Movie 2019” is present there which I replaced with “2019” and data type as “integer”.

```
movies['year']=movies['year'].replace('TV Movie 2019', 2019).astype(int)
```

In this project of movie recommendation, by the method of content-based filtering, we do not need all the data there are in the movies data, we are taking only those columns that are needed namely movie



name, movie genre, casts there are in the movie, director of the movie, etc.

We also need to visualize the data because a visual summary of information makes it easier to identify patterns and trends than looking through thousands of rows on a spreadsheet. Since the purpose of data analysis is to gain insights, data is much more valuable when it is visualized. Here we have done a demographic visualization of data, i.e., looking into the data to know what is primary genre, primary language, and country of origin of the movies and their distribution. Since this information is not exactly in there in the data but we can get them from other columns of the data. Say, we want primary genre of a movie, we can get it from “genre” column that contains movie genre, from there we first split it on “,” and then selecting the first indexed genre.

```
movies['primary_genre']=movies['genre']  
.apply(lambda x: x.split(',')[0])
```

And we also do this to get the primary language and primary country of the movies.

```
movies['primary_language']=movies['language'].apply(lambda x: x.split(',')[0])
```

```
movies['primary_country']=movies['country'].apply(lambda x: x.split(',')[0])
```

And now we visualize the whole data on primary genre, primary language, and primary country.

Function for the visualization:

```
def visualise_plot_plt():  
  
    plt.style.use('ggplot')  
  
    plt.figure(figsize=(20,50))  
  
    for num,var in enumerate(base_categ):  
  
        plt.subplot(len(base_categ),2,2*num+1  
)  
  
        movies[var].value_counts()[:15].plot(kind='pie',autopct='%1.1f%%',explode=np.ones(15)*0.1,rotatelabels=True,radius=0.8,shadow=True)  
  
        plt.title("Number of Movies by {}".format(var.capitalize()),weight='bold')  
  
        plt.tick_params(labelsize='x-large')  
  
        plt.axis('off')  
  
        plt.subplot(len(base_categ),2,2*(num+1))  
  
        movies[var].value_counts()[:15].plot(kind='bar',color='y',edgecolor='r',linewidth=3)  
  
        plt.grid(True)
```

```
plt.tick_params(grid_color='r',grid_linestyle='-'.)
```

```
plt.title("Number of Movies by {}".format(var.capitalize()),weight='bold')
```

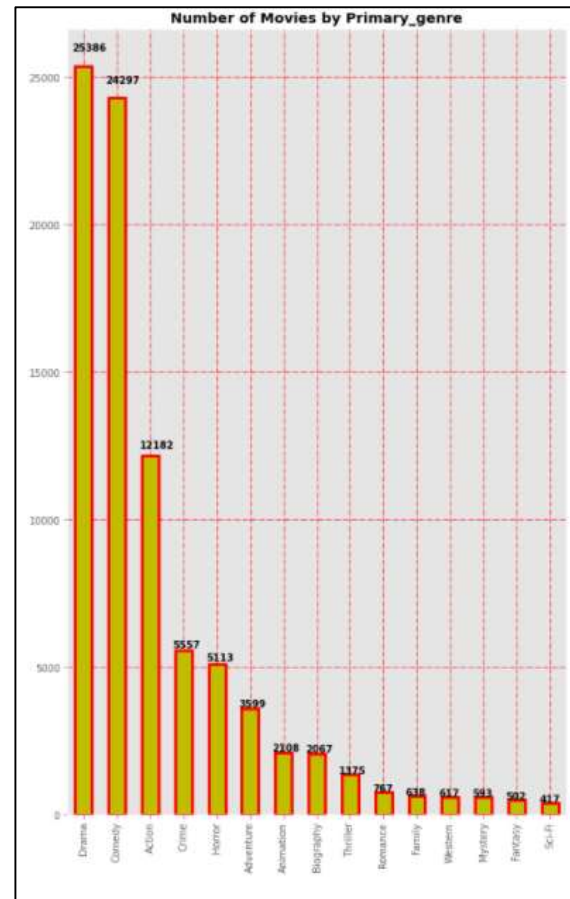
```
for k,val in enumerate(movies[var].value_counts()[:15]):
```

```
plt.text(x=k-0.33,y=val*1.02,s=val,weight='bold')
```

Now, plotting the data.

```
base_categ=['primary_genre','primary_language','primary_country']
```

```
visualise_plot_plt()
```



We see that most of the movies are of genre drama, next comedy, and so on.

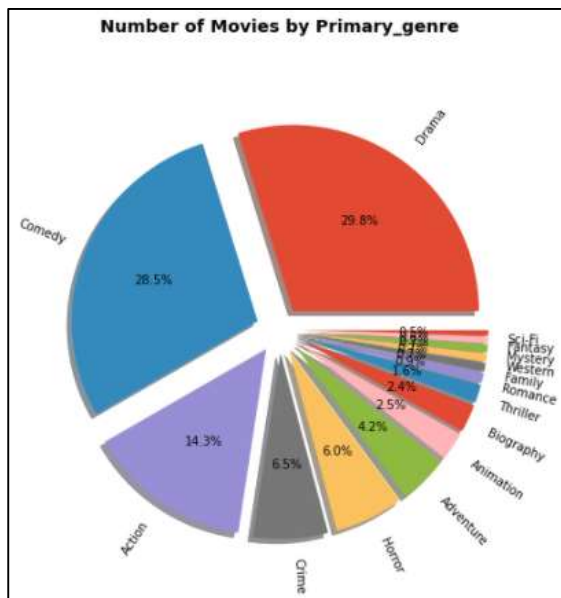


Figure 4: Percentage of Movies by Primary Genre

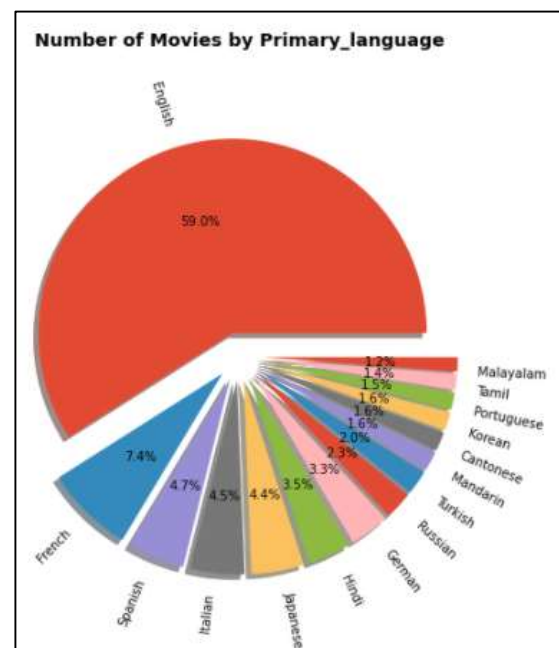
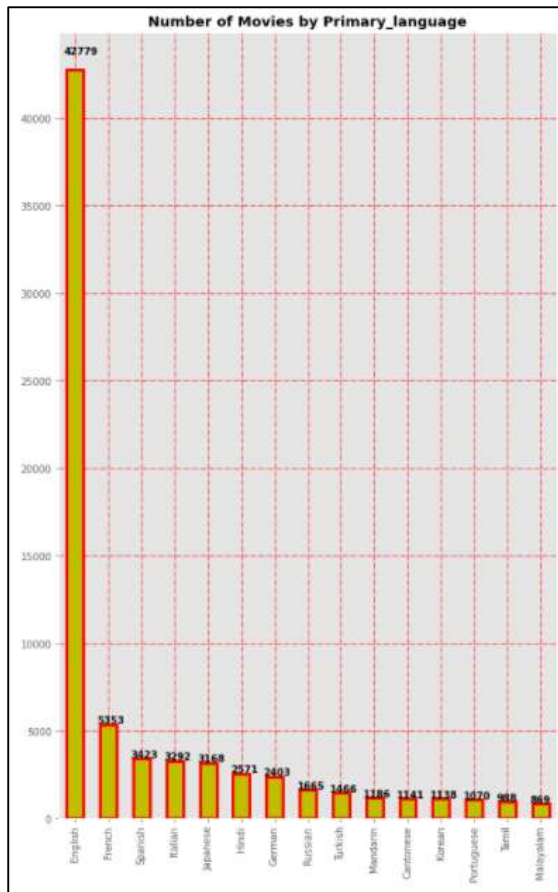
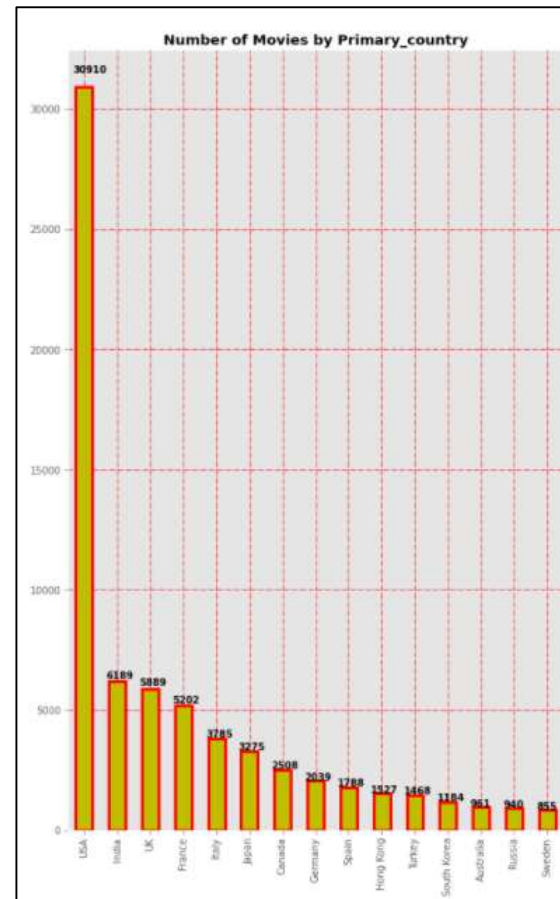


Figure 5: Percentage of Movies by Primary Language



Most of the movies are of English language.



Since in this project we are doing movie recommendation only on the movies of India, we extract the movies of origin India.

***movies=movies[movies['country']=='India']***

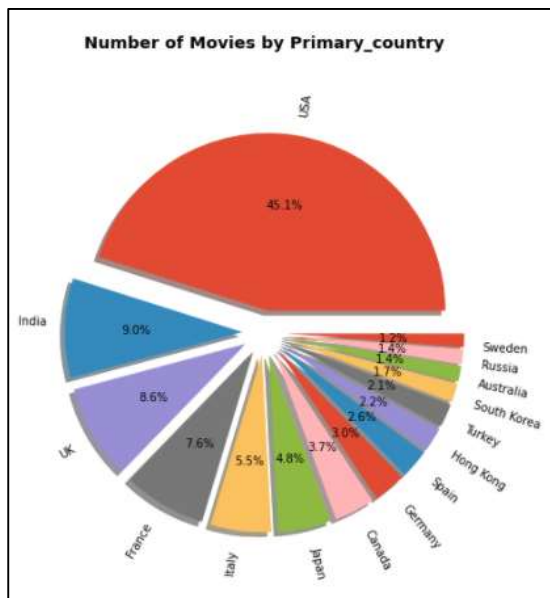


Figure 2: Percentage of Movies by Primary Country

Let's now visualize the data of Indian movies. We will use Sunburst<sup>[12]</sup> chart from Plotly Express library. We can install the library using ***pip install plotly*** on the command line. Version should be greater than 5. The sunburst chart is ideal for displaying hierarchical data. Each level of the hierarchy is represented by one ring or circle with the innermost circle as the top of the hierarchy.

```
px.sunburst(movies.fillna('None'),path=['primary_country','primary_language','primary_genre'],height=700,width=700,title='Movies in Each Language by Genre')
```



Figure 6: Movies in Each Language by Genre

So, we see most of the movies of India have Hindi as their primary language and the second most primary language in which movies are released is Tamil and next is Malayalam.

## Building the Recommender Engine

Now comes the most important part of the project and that is to build the recommender engine that is going to be used in the recommendation system. In building this engine first we check if there are any null values present in the data. And we saw that there are a few null values present in the dataset. We can remove these null values but in doing so we will also remove some

of the data from our dataset. So, we replaced these null values with “unknown” word.

Now, we need to discard those punctuation that are in the content of the movies data. And to do that let’s first write function that do that for us.

```
def discard_punc(text):
```

```
    text=text.lower()
```

```
    text=text.replace(':', '')
```

```
    text=text.replace('+', '')
```

```
    text=text.replace('= ', '')
```

```
    text=text.replace('!', '')
```

```
    text=text.replace(':', '')
```

```
    text=text.replace('-', '')
```

```
    text=text.replace('?', '')
```

```
    text=text.replace(',', '')
```

```
    text=text.replace('\"', '')
```

```
    return text
```

And let’s now remove the punctuation from certain columns.

```
movies['title']=movies['title'].apply(discard_punc)
```

```
movies['actors']=movies['actors'].apply(discard_punc)
```

```
movies['director']=movies['director'].apply(discard_punc)
```

```
movies['genre']=movies['genre'].apply(discard_punc)
```

Now that we are done removing punctuation we are left to remove stopwords from the description column of the data. Stopwords are the words in any language which does not add much meaning to a sentence. They can safely be ignored without sacrificing the meaning of the sentence. For some search engines, these are some of the most common, short function words, such as “the”, “is”, “at”, “which”, and “on”, etc. But first we write a function that will remove some punctuation and also use pycontraction library for expanding common English contractions in text and then we remove stopwords from the text using nltk library.

```
def discard_stopwords(text):
```

```
    if pd.isna(text):
```

```
        text= ' '
```

```
    try:
```

```
        text=text.lower()
```

```
        text=text.replace(':', '')
```

```
        text=text.replace('-', ' ')
```

```
        text=text.replace('; ', '')
```

```
        text=text.replace('!', '')
```

```
        text=text.replace(',', '')
```

```
        text=text.replace('\"', '')
```

```
        text=text.replace('.', '')
```

```
        text=text.replace('(', '')
```

```
        text=text.replace(')', '')
```

```
        text=list(cont.expand_texts([text], precise=True))[0]
```

```
        text=text.split()
```

```
        text=[word for word in text if word not in stopwords.words('english')]
```

```
        text=' '.join(text)
```

```
        return text
```

```
    except:
```

```
        print('ERROR: ', text)
```

Now removing the stopwords.

```
movies['description']=movies['description'].apply(discard_stopwords)
```

As we are doing movie recommendation using content-based filtering method, the content of each movie is most important data and we need to gather all the content of the movies in one place, in other words merge the data of various related columns in one column for each row or each movie. So that using this data we can make recommendations. To do that first we created an empty list and then iterating over the content in each row and appending the content all together in the list and we do it for all the individual rows i.e., for each individual movie. Then we added this new

list that contains all the information of the movies to the new column of the data set. And we also did some text cleaning by removing all special characters from the information of movies.

```
new_feature=[]
```

```
for item in df.itertuples():
```

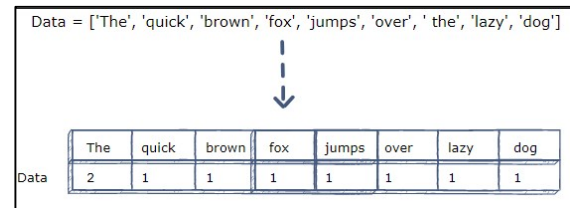
```
    new_feature.append(item[2]+' '+item[3]  
    '+' '+item[4]+' '+item[5]+' '+item[6])
```

```
movies['new_feature']=new_feature
```

```
movies['new_feature']=movies['new_feat  
ure'].apply(lambda x: re.sub(r'[\^\w  
/' ,",x))
```

Now, we are left with the contents of each movie that are in text format. We need to convert them to numeric so that we apply similarity function to it. To convert to numeric format, we use a class from scikit-learn<sup>[13]</sup> library named CountVectorizer. This is a very useful class in text processing. It is used to transform a given text into a vector on the basis of the frequency (count) of each word that occurs in the entire text. CountVectorizer<sup>[14]</sup> creates a matrix in which each unique word is represented by a column of the matrix, and each text sample from the document is a row in the matrix. The value of each cell is nothing but the count of the word in that particular text sample. Below is a simple

illustration of working of CountVectorizer class.



	The	quick	brown	fox	jumps	over	lazy	dog
Data	2	1	1	1	1	1	1	1

Figure 7: A simple example of working of CountVectorizer.

Coming back to the point where we use it in our movie contents. Usually, we use it by creating an instance of this class and then calling fit() function on the text document, in our case it is movie content in the “new\_feature” column, and then calling transform() function on it to get the matrix. But we can also do all this step at one go.

```
cm=CountVectorizer().fit_transform(movi  
es['new_feature'])
```

Here we get the matrix that we were looking for and assign the matrix to “cm” variable. If we see the shape of the matrix, we will see that it has a shape of (6189, 21721) meaning 6189 rows that is the movie count of Indian movies and 21721 columns which is the unique words that we have got.

Now, we measure the similarity of the movies with each other. We use cosine\_similarity<sup>[15]</sup> function from scikit-learn library. This function computes cosine similarity between samples. It measures the similarity between two vectors of an inner product space. It is



measured by the cosine of the angle between two vectors and determines whether two vectors are pointing in roughly the same direction.

$$\text{similarity}(A,B) = \frac{A \cdot B}{\|A\| \times \|B\|} = \frac{\sum_{i=1}^n A_i \times B_i}{\sqrt{\sum_{i=1}^n A_i^2} \times \sqrt{\sum_{i=1}^n B_i^2}}$$

Figure 8: Formula for Cosine Similarity

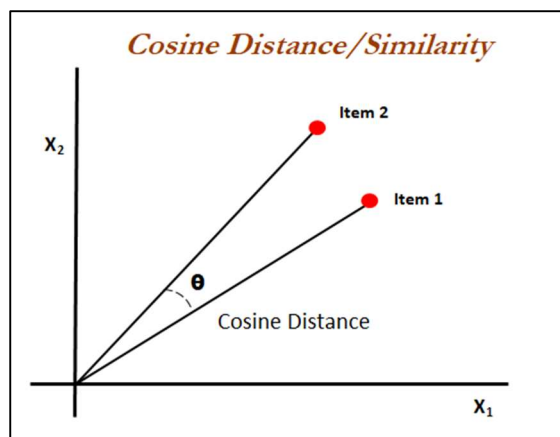


Figure 9: Cosine Similarity

When plotted on a multi-dimensional space, where each dimension corresponds to a word in the document, the cosine similarity captures the orientation (the angle) of the documents and not the magnitude. The cosine similarity is advantageous because even if the two similar documents are far apart by the Euclidean distance because of the size they could still have a smaller angle between them. Smaller the angle, higher the similarity.

So, we measure the similarity measures of the movies on the new matrix that had been

produced by CountVecotrizer class and store the similarity measures in a new variable named “cs”.

***cs=cosine\_similarity(cm)***

This “cs” is a 2-D numpy array and contains the similarity measure of the movies. We transform this into a dataframe using pandas library and assign it to “cs\_df” variable. And we give this new dataframe column names as its movie id, and to its index too, so that, it can easily be understood and accessed by it.

***cs\_df=pd.DataFrame(cs,columns=movies['imdb\_title\_id'].values)***

***cs\_df.set\_index(movies['imdb\_title\_id'].values,inplace=True)***

Let’s write a function so that using this function we can get recommended movies. What we do in this function is that we take a movie name and find its imdb id on the dataset, and using this id we get the similarity measures from the cs\_df dataframe and sorting descendent wise of the measures we then extract the ids of the movies from the index we previously assigned and print the movie names from the main dataset.

***def get\_recommendation(movie\_name):***

***id\_=movies[movies['title']==movie\_name]['imdb\_title\_id'].values[0]***

```

title_id=cs_df_sorted[id_].values

for i in title_id:

    print(movies[movies['imdb_title_id']=
=i][['title'].values[0])

```

Result section:

```
get_recommendation('Main Hoon Na')
```

```

Felice anno nuovo
Om Shanti Om
Heyy Baby
Josh
Hello Brother
Paheli
Kabhi Haan Kabhi Naa
Sarfarosh
Janasheen
Yun Hota Toh Kya Hota

```

```
get_recommendation('Manoharam')
```

```

Kunjiramayanam
Oru Muthassi Gadha
Love Action Drama
Thattathin Marayathu
Ohm Shanthi Oshaana
The Great Father
O.P.160/18 Kakshi: Amminippilla
Godha
Ustad Hotel
Gauthamante Radham

```

```
get_recommendation('Don 2')
```

```

Don
Felice anno nuovo
Josh
Main Hoon Na
Qurbani
Hello Brother
Janasheen
Om Shanti Om
Fool N Final
Billu

```

```
get_recommendation('Abhimanyu')
```

```

Kilichundan Mampazham
Chithram
Vismayathumbathu
Vandanam
Vatsalyam
Devadoothan
Kakkakuyil
Manoranjan
Thalavattam
Hitler

```

```
get_recommendation('Anantaram')
```

```

Mathilukal
Thinkalazhcha Nalla Divasam
Ulladakkam
Rathinirvedam
Hitler
Elippathayam
Nizhalkkuthu
Golanthara Vartha
Arappatta Kettiya Graamathil
Yaathra

```

Conclusion:

A content-based recommendation system is totally based on contents of the movies. In case we don't have enough data about



movies we can't even hope to do build a recommender engine. But also, there is another thing about it that first we have to pre-process the content specially description of the movies. I found it very useful. There is also future work that can be done on this project such that, we can do collaborative filtering if we got enough data about movies. And those be user rating of the movies, more specifically per user rating of movies. So that it can be used to build the recommender engine. Also, we can build a hybrid recommender engine, which is a combination of content-based recommender engine and collaborative filtering recommender engine. This hybrid recommender engine is more useful in recommending movies.

### Building the Frontend:

Building this newly recommendation engine using python library Flask.<sup>[16]</sup>



It is a micro web framework written in Python. It is classified as a

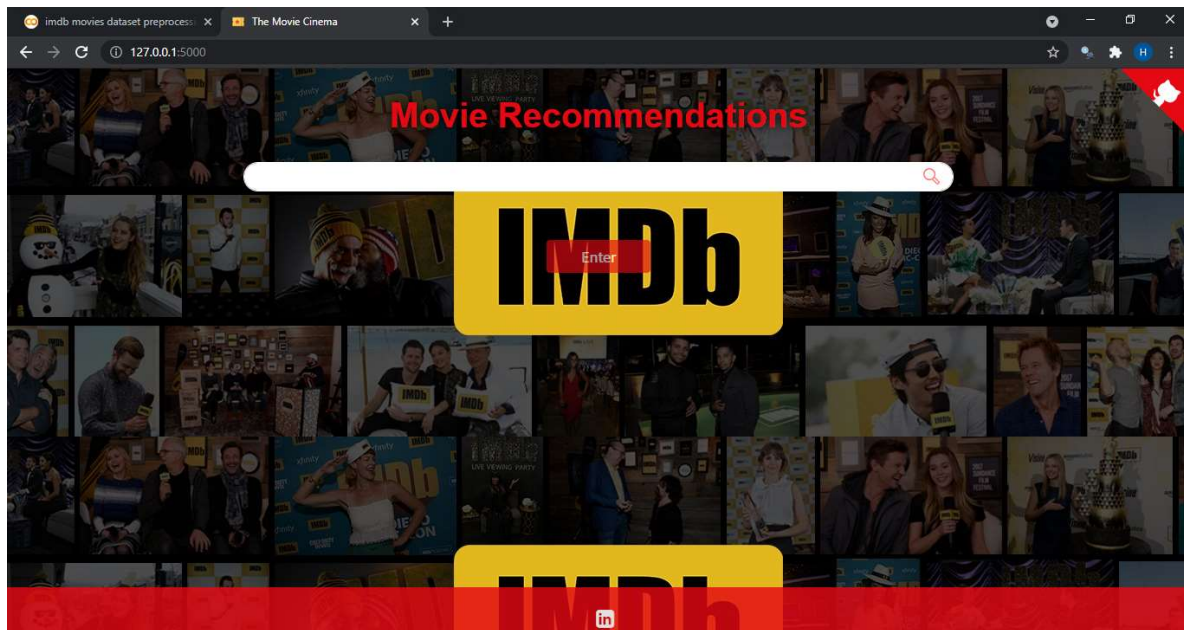
microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies and several common framework related tools.

Also used other web development languages like html, php, javascript.

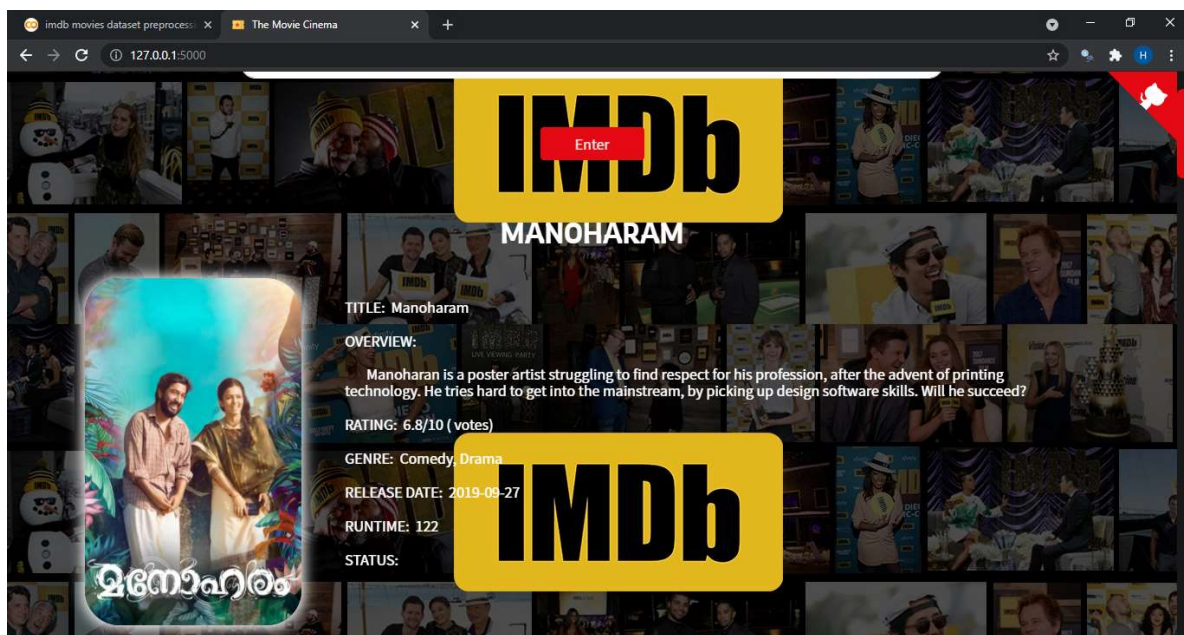
There is another thing that has added very valuable things to this frontend, and that is TMDb API - The Movie Database API.<sup>[17]</sup> The Movie Database (TMDb) is a community-built movie and TV database. TMDb API is available for everyone to use. A TMDb user account is required to request an API key.

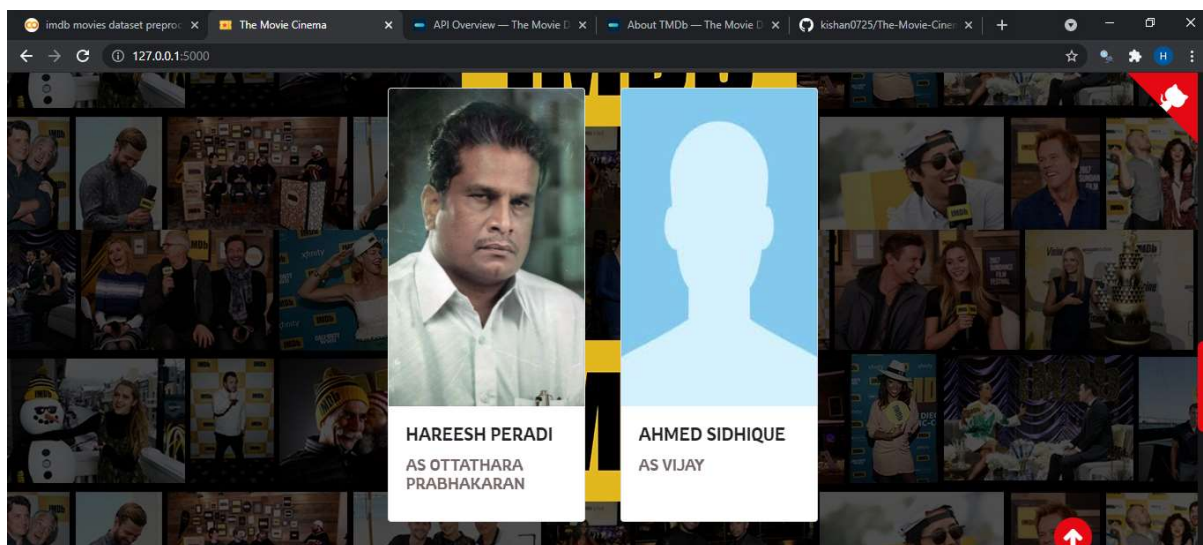
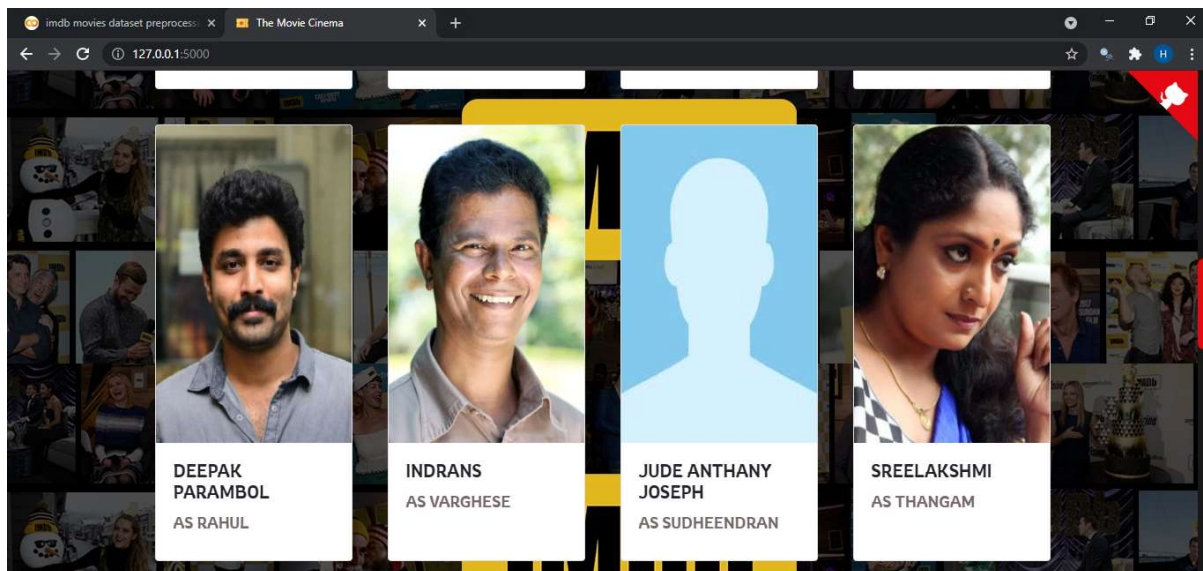
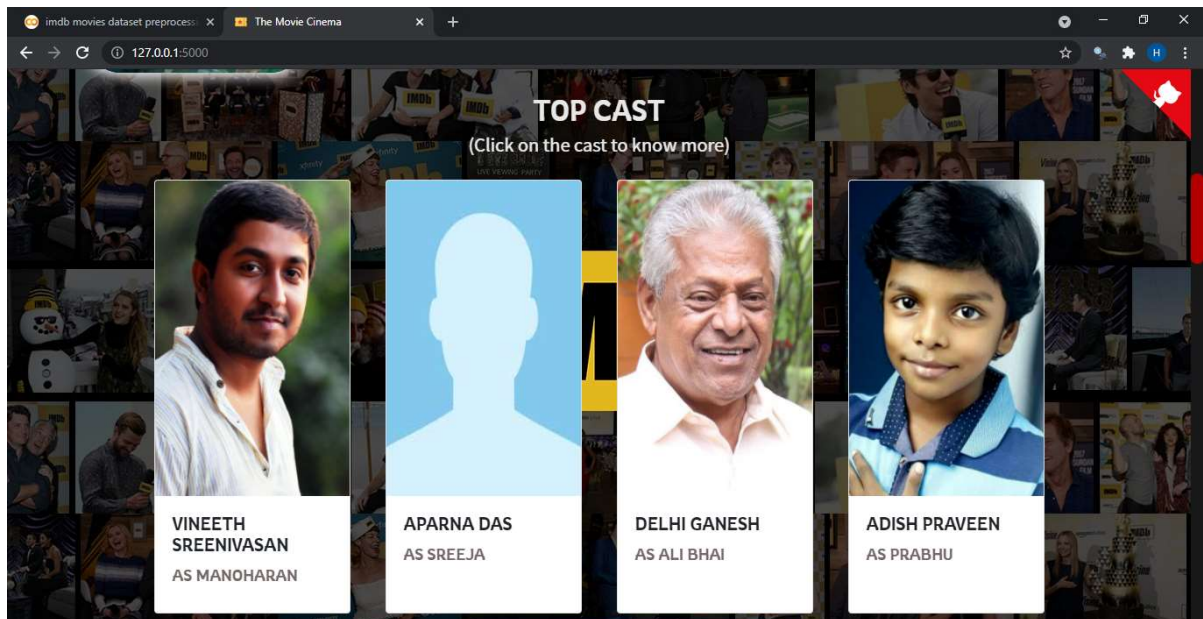
Output section of frontend of recommender engine:

Home Page:

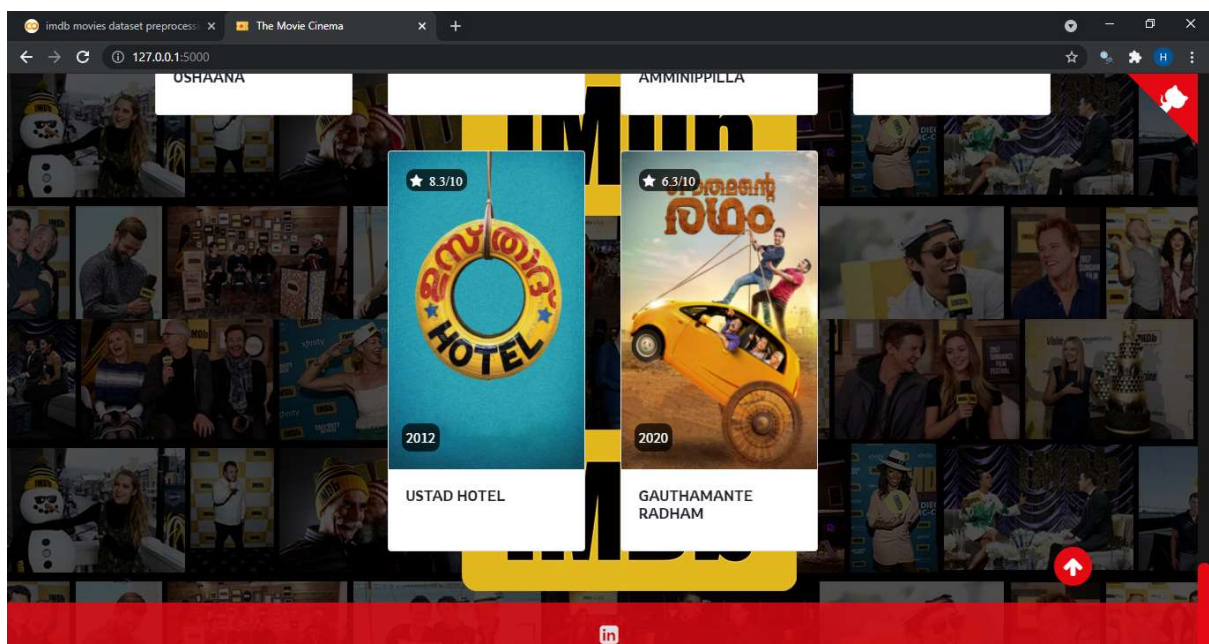
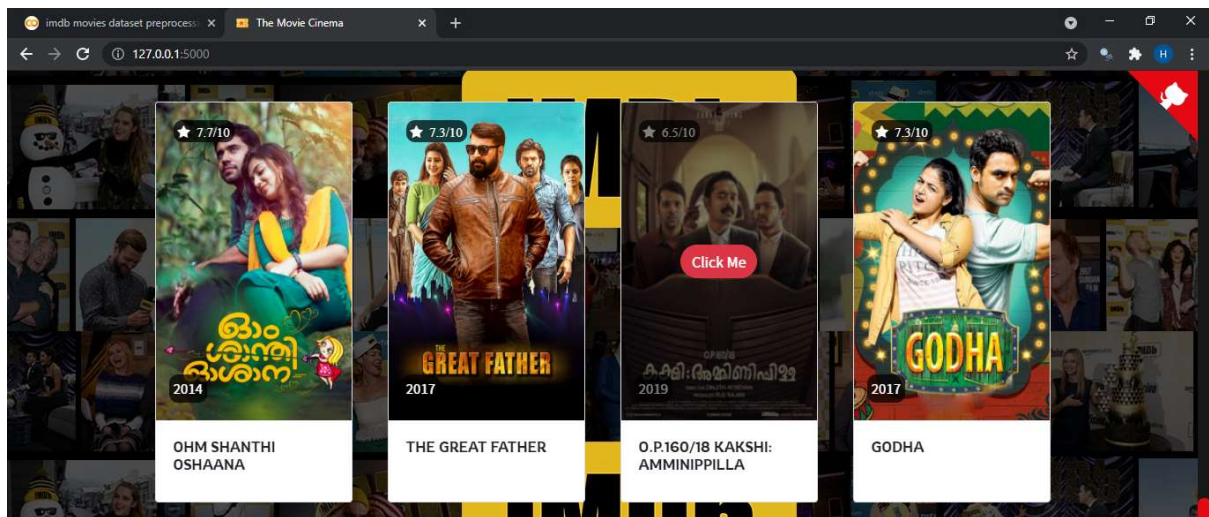
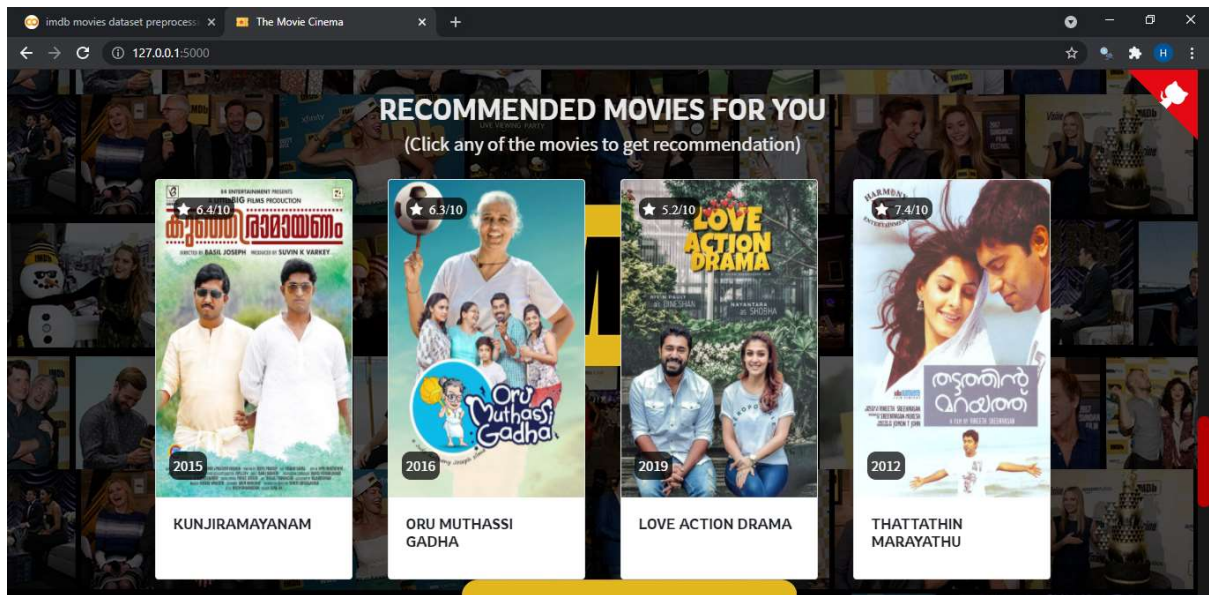


Recommender Page:









## Reference:

- [1] [Dataset](#)
- [2] Francesco Ricci and Lior Rokach and Bracha Shapira, [Introduction to Recommender Systems Handbook](#), Recommender Systems Handbook, Springer, 2011, pp. 1-35
- [3] H. Chen, A. G. Ororbia II, C. L. Giles [ExpertSeer: a Keyphrase Based Expert Recommender for Digital Libraries](#), in arXiv preprint 2015
- [4] H. Chen, L. Gou, X. Zhang, C. Giles [Collabseer: a search engine for collaboration discovery](#), in ACM/IEEE Joint Conference on Digital Libraries (JCDL) 2011
- [5] Alexander Felfernig, Klaus Isak, Kalman Szabo, Peter Zachar, [The VITA Financial Services Sales Support Environment](#), in AAAI/IAAI 2007, pp. 1692-1699, Vancouver, Canada, 2007.
- [6] Sarwar, B.; Karypis, G.; Konstan, J.; Riedl, J. (2000). ["Application of Dimensionality Reduction in Recommender System A Case Study"](#).
- [7] Allen, R.B. (1990). "User Models: Theory, Method, Practice". International J. Man-Machine Studies.
- [8] Aggarwal, Charu C. (2016). Recommender Systems: The Textbook. Springer. ISBN 9783319296579.
- [9] [www.kaggle.com/stefanoleone992](http://www.kaggle.com/stefanoleone992)
- [10] <https://www.imdb.com/>
- [11] <https://en.wikipedia.org/wiki/IMDb>
- [12] <https://plotly.com/python/sunburst-charts/>
- [13] scikit-learn: <https://scikit-learn.org>
- [14] CountVectorizer: [https://scikit-learn.org/stable/modules/generated/sklearn.feature\\_extraction.text.CountVectorizer.html](https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html)
- [15] cosine similarity: [https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.cosine\\_similarity.html](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.cosine_similarity.html)
- [16] Flask: <https://flask.palletsprojects.com/en/2.0.x/>
- [17] TMDb API: <https://www.themoviedb.org/documentation/api>