*MOVIE RECOMMENDATION SYSTEM*

*Index*

[**Acknowledgement** 1](#_Toc121722479)

[***Introduction*** 2](#_Toc121722480)

[***Machine learning Steps for our Project*** 4](#_Toc121722481)

[**Collecting the data** 4](#_Toc121722482)

[**Preparing the Data** 4](#_Toc121722483)

[**Data preprocessing** 4](#_Toc121722484)

[**KNearestNeighbors** 4](#_Toc121722485)

[***Code*** 5](#_Toc121722486)

[*Basics of Analysis* 5](#_Toc121722487)

[*Cleaning The Dataset* 6](#_Toc121722488)

[*Data Preprocessing* 10](#_Toc121722489)

[*Building Recommendation system* 11](#_Toc121722490)

[***Result*** 12](#_Toc121722491)

[***Conclusion*** 13](#_Toc121722492)

[***Recommendations*** 13](#_Toc121722493)

[***References*** 13](#_Toc121722494)

**Acknowledgement**

*We would like to thank Mr. Makhan Kumbhkar, our Professor-in-charge and our Director, Dr. Subramanian Raman Iyer for their support and guidance in completing our project on the topic, “Customer Personality Analysis”. It was a great learning experience.*

*We would like to take this opportunity to express our gratitude to all of the group members Sakaar Mathur and Arin Jaiswal. The project would not have been successful without their cooperation and inputs.*

***Introduction***

A movie recommendation system, or a movie recommender system, is an ML-based approach to filtering or predicting the users’ film preferences based on their past choices and behavior. It’s an advanced filtration mechanism that predicts the possible movie choices of the concerned user and their preferences towards a domain-specific item, aka movie.

The basic concept behind a movie recommendation system is quite simple. In particular, there are two main elements in every recommender system: users and items. The system generates movie predictions for its users, while items are the movies themselves.

The primary goal of movie recommendation systems is to filter and predict only those movies that a corresponding user is most likely to want to watch. The ML algorithms for these recommendation systems use the data about this user from the system’s database. This data is used to predict the future behavior of the user concerned based on the information from the past.

**Filtration Strategies for Movie Recommendation Systems:**

Movie recommendation systems use a set of different filtration strategies and algorithms to help users find the most relevant films. The most popular categories of the ML algorithms used for movie recommendations include content-based filtering and collaborative filtering systems.

**Content-Based Filtering:**

A filtration strategy for movie recommendation systems, which uses the data provided about the items (movies). This data plays a crucial role here and is extracted from only one user. An ML algorithm used for this strategy recommends motion pictures that are similar to the user’s preferences in the past. Therefore, the similarity in content-based filtering is generated by the data about the past film selections and likes by only one user.

The recommendation system analyzes the past preferences of the user concerned, and then it uses this information to try to find similar movies. This information is available in the database (e.g., lead actors, director, genre, etc.). After that, the system provides movie recommendations for the user. That said, the core element in content-based filtering is only the data of only one user that is used to make predictions.

**Collaborative Filtering:**

As the name suggests, this filtering strategy is based on the combination of the relevant user’s and other users’ behaviors. The system compares and contrasts these behaviors for the most optimal results. It’s a collaboration of the multiple users’ film preferences and behaviors.

What’s the mechanism behind this strategy? The core element in this movie recommendation system and the ML algorithm it’s built on is the history of all users in the database. Basically, collaborative filtering is based on the interaction of all users in the system with the items (movies). Thus, every user impacts the final outcome of this ML-based recommendation system, while content-based filtering depends strictly on the data from one user for its modeling.

**Collaborative filtering algorithms are divided into two categories:**

User-based collaborative filtering. The idea is to look for similar patterns in movie preferences in the target user and other users in the database.

Item-based collaborative filtering. The basic concept here is to look for similar items (movies) that target users rate or interact with.

The modern approach to the movie recommendation systems implies a mix of both strategies for the most gradual and explicit results.

**The Top Movie Recommendation System Use Cases:**

One can spot the increased use of the recommendation systems on almost every popular streaming service, social media, or e-commerce platform. These include Amazon, YouTube, Netflix, Facebook, to name a few. These recommendation systems help different industries provide more personalized experiences to their users.

***Machine learning Steps for our Project***

* Collecting Data:
* Preparing the Data:
* Choosing a Model:
* Training the Model:
* Evaluating the Model:

**Collecting the data**

Our two datasets are relatively simple: movies dataset has 9742 rows and 2 columns whereas ratings dataset has 100836 rows and 3 columns.

**Preparing the Data**

Removal of missing values

Reshaping the data

**Data preprocessing**

Creating compressed sparse row matrix

**KNearestNeighbors**

To apply classification, the following steps have been taken:

1. Provide a training set
2. Find k-nearest neighbors
3. Classify points

***Code***

## *Basics of Analysis*

**Importing Basic Libraries**

!pip install fuzzywuzzy  
import pandas as pd  
import numpy as np  
import google.colab  
import scipy.sparse

*Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/  
Collecting fuzzywuzzy  
 Downloading fuzzywuzzy-0.18.0-py2.py3-none-any.whl (18 kB)  
Installing collected packages: fuzzywuzzy  
Successfully installed fuzzywuzzy-0.18.0*

**Importing Dataset**

from google.colab import files  
uploaded = files.upload()

*<IPython.core.display.HTML object>*

*Saving movies.csv to movies.csv*  
*Saving Project\_5th\_Sem.ipynb to Project\_5th\_Sem.ipynb*  
*Saving Project\_18\_Movie\_Recommendation\_System\_using\_Machine\_Learning\_with\_Python.ipynb to Project\_18\_Movie\_Recommendation\_System\_using\_Machine\_Learning\_with\_Python.ipynb*  
*Saving ratings.csv to ratings.csv*

movies = pd.read\_csv('movies.csv', usecols = ['movieId','title'])  
ratings = pd.read\_csv('ratings.csv',usecols = ['userId', 'movieId','rating'])

**Understanding the Dataset**

movies.head()

*movieId title*  
*0 1 Toy Story (1995)*  
*1 2 Jumanji (1995)*  
*2 3 Grumpier Old Men (1995)*  
*3 4 Waiting to Exhale (1995)*  
*4 5 Father of the Bride Part II (1995)*

movies.shape

*(9742, 2)*

ratings.head()

*userId movieId rating*  
*0 1 1 4.0  
1 1 3 4.0  
2 1 6 4.0*  
*3 1 47 5.0*  
*4 1 50 5.0*

ratings.shape

*(100836, 3)*

movies dataset has 9742 rows and 2 columns

ratings dataset has 100836 rows and 3 columns

## *Cleaning The Dataset*

**Treating Null values**

movies.isnull().sum()

*movieId 0*  
*title 0*  
*dtype: int64*

ratings.isnull().sum()

*userId 0*  
*movieId 0*  
*rating 0*  
*dtype: int64*

Since there are no Null values in the movies and ratings dataframes, we can proceed further for now.

**Reshaping our Data**

movies\_users = ratings.pivot(index = 'movieId', columns = 'userId', values = 'rating')  
movies\_users

*userId 1 2 3 4 5 6 7 8 9 10 ... 601 602 603 \*  
*movieId ...*   
*1 4.0 NaN NaN NaN 4.0 NaN 4.5 NaN NaN NaN ... 4.0 NaN 4.0*   
*2 NaN NaN NaN NaN NaN 4.0 NaN 4.0 NaN NaN ... NaN 4.0 NaN*   
*3 4.0 NaN NaN NaN NaN 5.0 NaN NaN NaN NaN ... NaN NaN NaN*   
*4 NaN NaN NaN NaN NaN 3.0 NaN NaN NaN NaN ... NaN NaN NaN*   
*5 NaN NaN NaN NaN NaN 5.0 NaN NaN NaN NaN ... NaN NaN NaN*   
*... ... ... ... ... ... ... ... ... ... ... ... ... ... ...*   
*193581 NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN ... NaN NaN NaN*   
*193583 NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN ... NaN NaN NaN*   
*193585 NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN ... NaN NaN NaN*   
*193587 NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN ... NaN NaN NaN*   
*193609 NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN ... NaN NaN NaN*   
  
*userId 604 605 606 607 608 609 610*   
*movieId*   
*1 3.0 4.0 2.5 4.0 2.5 3.0 5.0*   
*2 5.0 3.5 NaN NaN 2.0 NaN NaN*   
*3 NaN NaN NaN NaN 2.0 NaN NaN*   
*4 NaN NaN NaN NaN NaN NaN NaN*   
*5 3.0 NaN NaN NaN NaN NaN NaN*   
*... ... ... ... ... ... ... ...*   
*193581 NaN NaN NaN NaN NaN NaN NaN*   
*193583 NaN NaN NaN NaN NaN NaN NaN*   
*193585 NaN NaN NaN NaN NaN NaN NaN*   
*193587 NaN NaN NaN NaN NaN NaN NaN*   
*193609 NaN NaN NaN NaN NaN NaN NaN*   
  
*[9724 rows x 610 columns]*

After reshaping our data, we can see there are a lot of null values in the new dataframe.

So intead of omiting null values, lets fill these null values with 0

movies\_users = movies\_users.fillna(0)  
movies\_users

*userId 1 2 3 4 5 6 7 8 9 10 ... 601 602 603 \*  
*movieId ...*   
*1 4.0 0.0 0.0 0.0 4.0 0.0 4.5 0.0 0.0 0.0 ... 4.0 0.0 4.0*   
*2 0.0 0.0 0.0 0.0 0.0 4.0 0.0 4.0 0.0 0.0 ... 0.0 4.0 0.0*   
*3 4.0 0.0 0.0 0.0 0.0 5.0 0.0 0.0 0.0 0.0 ... 0.0 0.0 0.0*   
*4 0.0 0.0 0.0 0.0 0.0 3.0 0.0 0.0 0.0 0.0 ... 0.0 0.0 0.0*   
*5 0.0 0.0 0.0 0.0 0.0 5.0 0.0 0.0 0.0 0.0 ... 0.0 0.0 0.0*   
*... ... ... ... ... ... ... ... ... ... ... ... ... ... ...*   
*193581 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 ... 0.0 0.0 0.0*   
*193583 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 ... 0.0 0.0 0.0*   
*193585 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 ... 0.0 0.0 0.0*   
*193587 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 ... 0.0 0.0 0.0*   
*193609 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 ... 0.0 0.0 0.0*   
  
*userId 604 605 606 607 608 609 610*   
*movieId*   
*1 3.0 4.0 2.5 4.0 2.5 3.0 5.0*   
*2 5.0 3.5 0.0 0.0 2.0 0.0 0.0*   
*3 0.0 0.0 0.0 0.0 2.0 0.0 0.0*   
*4 0.0 0.0 0.0 0.0 0.0 0.0 0.0*   
*5 3.0 0.0 0.0 0.0 0.0 0.0 0.0*   
*... ... ... ... ... ... ... ...*   
*193581 0.0 0.0 0.0 0.0 0.0 0.0 0.0*   
*193583 0.0 0.0 0.0 0.0 0.0 0.0 0.0*   
*193585 0.0 0.0 0.0 0.0 0.0 0.0 0.0*   
*193587 0.0 0.0 0.0 0.0 0.0 0.0 0.0*   
*193609 0.0 0.0 0.0 0.0 0.0 0.0 0.0*   
  
*[9724 rows x 610 columns]*

Now our data is cleaned and is ready for Preprocessing

## *Data Preprocessing*

from scipy.sparse import csr\_matrix  
mat\_movies = csr\_matrix(movies\_users.values)  
print(mat\_movies)

*(0, 0) 4.0  
 (0, 4) 4.0  
 (0, 6) 4.5  
 (0, 14) 2.5  
 (0, 16) 4.5  
 (0, 17) 3.5  
 (0, 18) 4.0  
 (0, 20) 3.5  
 (0, 26) 3.0  
 (0, 30) 5.0  
 (0, 31) 3.0  
 (0, 32) 3.0  
 (0, 39) 5.0  
 (0, 42) 5.0  
 (0, 43) 3.0  
 (0, 44) 4.0  
 (0, 45) 5.0  
 (0, 49) 3.0  
 (0, 53) 3.0  
 (0, 56) 5.0  
 (0, 62) 5.0  
 (0, 63) 4.0  
 (0, 65) 4.0  
 (0, 67) 2.5  
 (0, 70) 5.0  
 : :  
 (9700, 337) 2.5  
 (9701, 337) 3.0  
 (9702, 183) 4.0  
 (9702, 247) 3.5  
 (9703, 317) 2.5  
 (9704, 209) 1.0  
 (9705, 461) 2.5  
 (9706, 49) 3.5  
 (9707, 337) 1.5  
 (9708, 337) 4.0  
 (9709, 337) 1.0  
 (9710, 337) 1.5  
 (9711, 337) 1.0  
 (9712, 337) 1.0  
 (9713, 183) 4.5  
 (9714, 183) 3.5  
 (9715, 183) 3.0  
 (9716, 183) 4.0  
 (9717, 183) 4.0  
 (9718, 183) 3.5  
 (9719, 183) 4.0  
 (9720, 183) 3.5  
 (9721, 183) 3.5  
 (9722, 183) 3.5  
 (9723, 330) 4.0*

## *Building Recommendation system*

**KNearestNeighbor**

For finding the movies based on similar rating, first we need to import and run KNN algorithm:

from sklearn.neighbors import NearestNeighbors  
model = NearestNeighbors(metric = 'cosine', algorithm = 'brute', n\_neighbors = 20)  
model.fit(mat\_movies)

NearestNeighbors(algorithm='brute', metric='cosine', n\_neighbors=20)

Since we are dealing with a large dataset, finding K-neighbors through euclidean distance is not very effective, instead we are using cosine similarity for finding out the neighbors.

from fuzzywuzzy import process

*/usr/local/lib/python3.8/dist-packages/fuzzywuzzy/fuzz.py:11: UserWarning: Using slow pure-python SequenceMatcher. Install python-Levenshtein to remove this warning*  
 *warnings.warn('Using slow pure-python SequenceMatcher. Install python-Levenshtein to remove this warning')*

def recommender(movie\_name,data,n):  
 idx = process.extractOne(movie\_name,movies['title'])[2]   
 print('Movie Selected : ', movies['title'][idx], 'Index : ',idx)  
 print('Searching for Recommendations..........')  
 distance, indices = model.kneighbors(data[idx],n\_neighbors = n)  
 for i in indices :  
 print(movies['title'][i].where(i!=idx))

**Output**

recommender('iron man',mat\_movies, 10)

*Movie Selected : Iron Man (2008) Index : 6743*  
*Searching for Recommendations..........*  
*6743 NaN*  
*7197 Garage (2007)*  
*7195 Merry Madagascar (2009)*  
*7354 A-Team, The (2010)*  
*6726 Superhero Movie (2008)*  
*7137 Thirst (Bakjwi) (2009)*  
*7026 Scorpio (1973)*  
*7571 Win Win (2011)*  
*3880 Look Who's Talking Now (1993)*  
*6388 After the Wedding (Efter brylluppet) (2006)*  
*Name: title, dtype: object*

***Result***

Throughout this project, supervised classification has been performed. The data has been converted into csr matrix and KNeighbors Classification has been performed. This allowed us to analyze the similarities between the movies based on user rating.

***Conclusion***

***Recommendations***

***References***