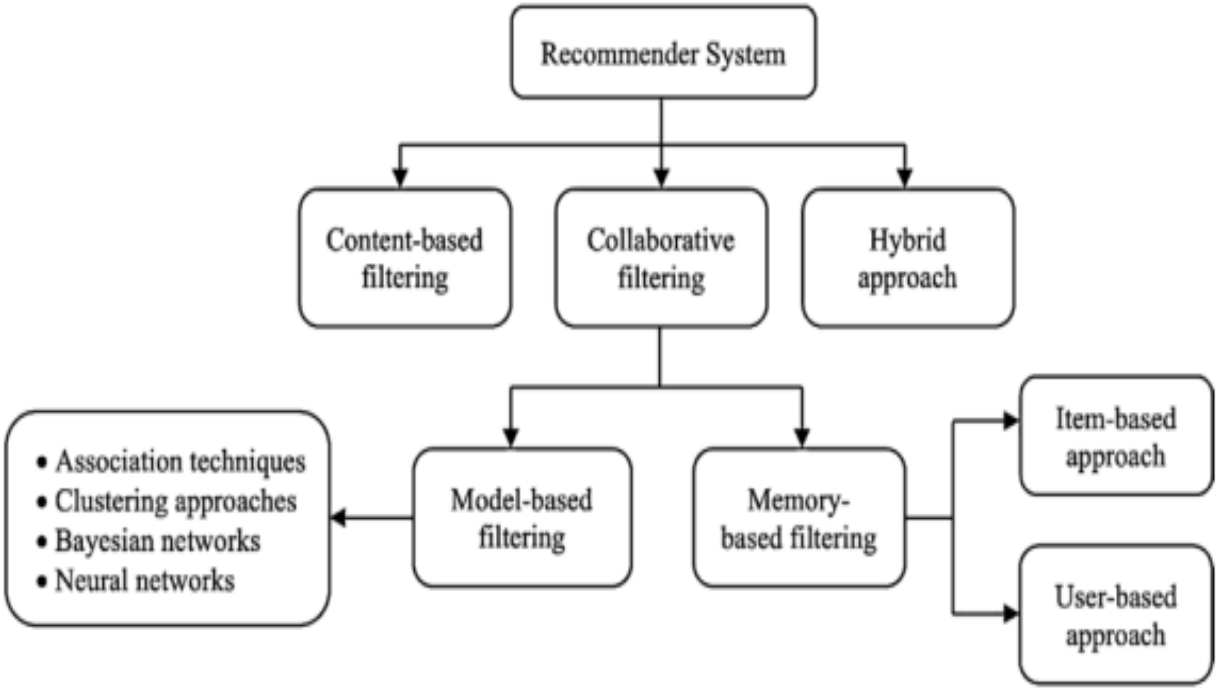
1.INTRODUCTION

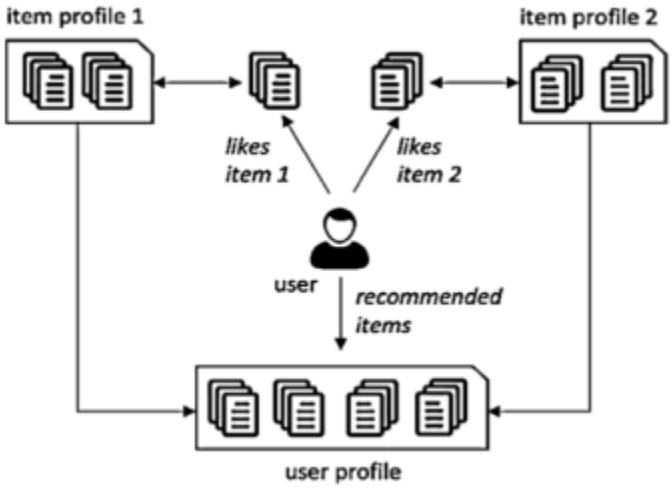
1.1 RECOMMENDATION SYSTEM:

Recommender systems are broadly categorized into three different types viz. content-based recommender systems, collaborative recommender systems and hybrid recommender systems. A diagrammatic representation of the different types of recommender systems is given below.



Content-based recommender system:

In content-based recommender systems, all the data items are collected into different item profiles based on their description or features. For example, in the case of a book, the features will be author, publisher, etc. In the case of a movie, the features will be the movie director, actor, etc. When a user gives a positive rating to an item, then the other items present in that item profile are aggregated together to build a user profile. This user profile combines all the item profiles, whose items are rated positively by the user. Items present in this user profile are then recommended to the user; one drawback of this approach is that it demands in-depth knowledge



of the item features for an accurate recommendation. This knowledge or information may not be always available for all items. Also, this approach has limited capacity to expand on the users' existing choices or interests. However, this approach has many advantages. As user preferences tend to change with time, this approach has the quick capability of dynamically adapting itself to the changing user preferences. Since one user profile is specific only to that user, this algorithm does not require the profile details of any other users because they provide no influence in the recommendation process. This ensures the security and privacy of user data. If new items have sufficient description, content-based techniques can overcome the cold-start problem i.e., this technique can recommend an item even when that item has not been previously rated by any user. Content-based filtering approaches are more common in systems like personalized news recommender systems, publications, web pages recommender systems, etc.

Collaborative filtering-based recommender system

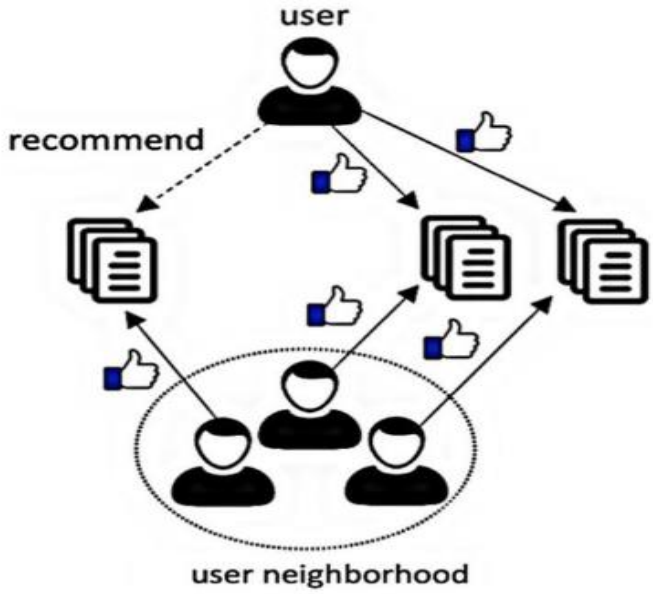
Collaborative approaches make use of the measure of similarity between users. This technique starts with finding a group or collection of user X whose preferences, likes, and dislikes are similar to that of user A. X is called the neighborhood of A. The new items which are liked by most of the users in X are then recommended to user A. The efficiency of a collaborative algorithm depends on how accurately the algorithm can find the neighborhood of the target user. Traditionally collaborative filtering-based systems suffer from the cold-start problem and privacy concerns as there is a need to share user data. However, collaborative filtering approaches do not require any knowledge of item features for generating a

recommendation. Also, this approach can help to expand on the user's existing interests by discovering new items. Collaborative approaches are again divided into two types: memory-based approaches and model-based approaches.

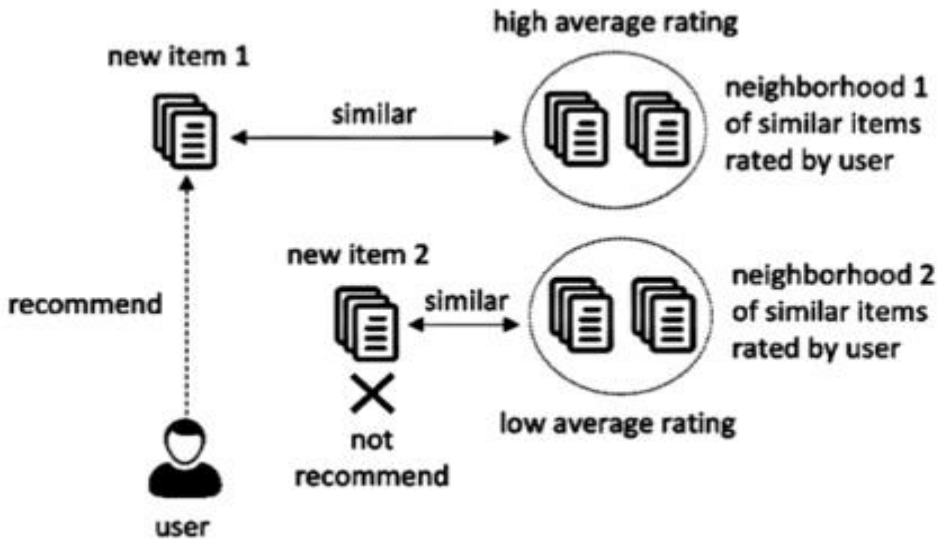
Memory-based collaborative approaches recommend new items by taking into consideration the preferences of its neighborhoods. They make use of the utility matrix directly for prediction. In this approach, the first step is to build a model.

Memory-based collaborative approaches are again sub-divided into two types: **user-based collaborative filtering** and **item-based collaborative filtering.**

User-Based: In the user-based approach, the user rating of a new item is calculated by finding other users from the user neighborhood who has previously rated that same item. If a new item receives positive ratings from the user neighborhood, the new item is recommended to the user.



Item Based: In the item-based approach, an item-neighborhood is built consisting of all similar items which the user has rated previously. Then that user's rating for a different new item is predicted by calculating the weighted average of all ratings present in a similar item-neighborhood.



Model-based systems use various data mining and machine learning algorithms to develop a model for predicting the user's rating for an unrated item. They do not rely on the complete dataset when recommendations are computed but extract features from the dataset to compute a model. Hence the name, model-based technique. These techniques also need two steps for prediction—the first step is to build the model, and the second step is to predict ratings using a function (f) which takes the model defined in the first step and the user profile as input.

Hybrid filtering

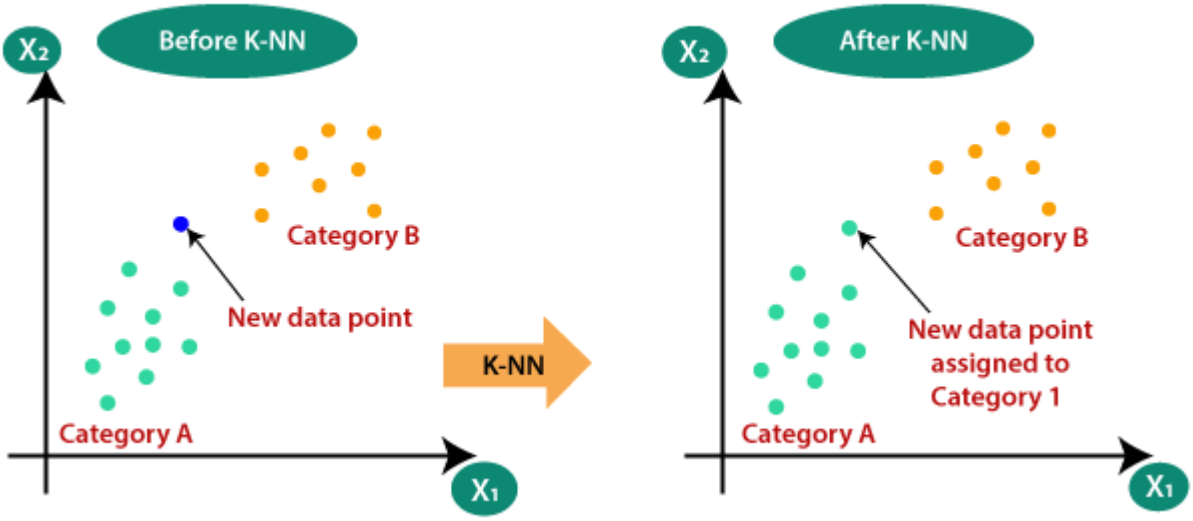
A hybrid technique is an aggregation of two or more techniques employed together for addressing the limitations of individual recommender techniques. The incorporation of different techniques can be performed in various ways. A hybrid algorithm may incorporate the results achieved from separate techniques, or it can use content-based filtering in a collaborative method or use a collaborative filtering technique in a content-based method. This hybrid incorporation of different techniques generally results in increased performance and increased accuracy in many recommender applications. Some of the hybridization approaches are meta-level, feature-augmentation, feature-combination,

1.2 K-NEAREST NEIGHBORS:

The k-nearest neighbors (KNN) algorithm is a simple, easy-to-implement supervised machine learning algorithm that can be used to solve both classification and regression problems.

Example: Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog. so for this identification, we can use the KNN algorithm, as it works on a similarity measure. Our KNN model will find the similar features of the new data set to the cats and dog images and based on the most similar features it will put it in either cat or dog category.

Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x1, so this data point will lie in which of these categories. To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily identify the category or class of a particular dataset. Consider the below diagram:



2.SYSTEM ANALYSIS

SYSTEM ANALYSIS

System Analysis is the important phase in the system development process. The System is studied to the minute details and analyzed. The system analyst plays an important role of an interrogator and dwells deep into the working of the present system. In analysis, a detailed study of these operations performed by the system and their relationships within and outside the system is done. A key question considered here is, “what must be done to solve the problem?” The system is viewed as a whole and the inputs to the system are identified. Once analysis is completed the analyst has a firm understanding of what is to be done.

2.1 PROBLEM DEFINITION

In this Project we are using Collaborative Filtering and KNN algorithm. By using this Methods there is an increasing realization in Recommender System. And giving the accurate result.

2.2 EXISTING SYSTEM

Recommender systems are software applications that suggest or recommend items or products to users. These systems use users' preferences or interests and an appropriate algorithm in finding the relevant or desired items or products. Recommender systems deal with information overload problems by filtering items that potentially may match the users' preferences or interests. These systems aid users to efficiently overcome the problem by filtering irrelevant information when users search for desired information.

2.2.1 DISADVANTAGES OF EXISTING SYSTEM

2.2.1.1 Collaborative filtering:

- Poor scalability;
- New user and new item problem;
- The recommendation quality limited by the history data set.

2.3 PROPOSED SYSTEM

- To find out the related content of the user in a given set of collection and to his interest feed.
- Providing the accurate and most confined results by using the distance between two vector models.
- By providing accurate results by the usage of movies datasets by IMDB in the existing project work.
- Classifying the users interest movies and recommend them to their searches fast by help libraries and software as pip, panda etc.

2.3.1 ADVANTAGES OF THE PROPOSED SYSTEM

2.3.1.1 Collaborative filtering:

- No need for professional knowledge;
- Performance improving as the increasing of the user number;
- Easy to find user's new interesting point;
- Complex unstructured item can be processed.

  Example: Music, Video, etc.

2.4 HARDWARE & SOFTWARE REQUIREMENTS

2.4.1 HARDWARE REQUIREMENTS:

Hardware interfaces specify the logical characteristics of each interface between the software product and the hardware components of the system. The following are some hardware requirements.

- Processor: Intel Dual Core I3 and above
- Hard disk: 8GB and above
- RAM: 8GB and above
- Input devices: Keyboard, mouse.

2.4.2 SOFTWARE REQUIREMENTS:

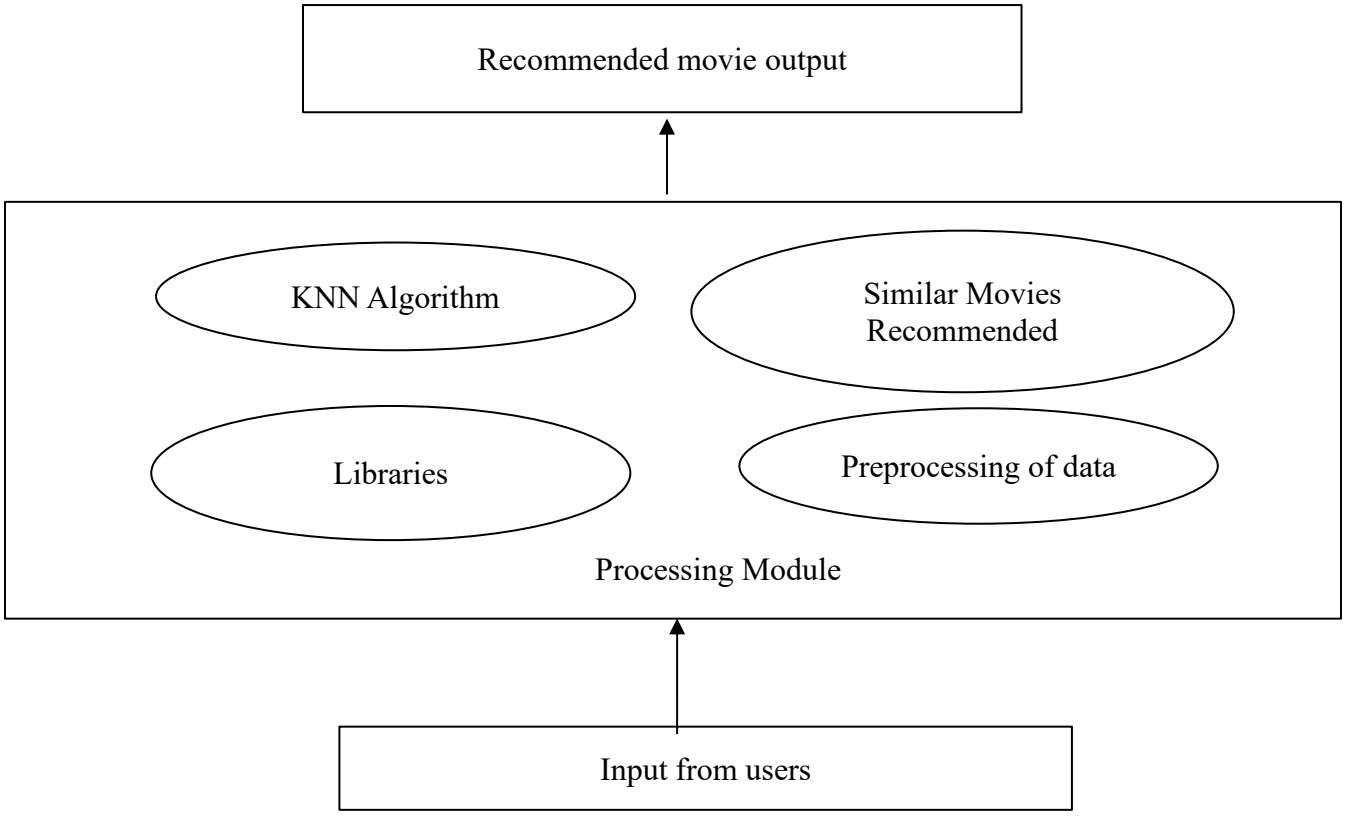
Software Requirements specifies the logical characteristics of each interface and software components of the system. The following are some software requirements,

- Operating system: Windows 8 and above
- Languages: Python
- Tools: Python IDLE 3.7 version, Visual Studio, google colab

2.5 MODULES

There are three modules in our projects:

- Input Module
- Processing Module
- Output Module


Recommended movie output
KNN Algorithm
Similar Movies Recommended
Libraries
Preprocessing of data
Processing Module
Input from users

3.1 PROJECT ARCHITECTURE

This project architecture shows the procedure followed for classification, starting from input to final recommendation.

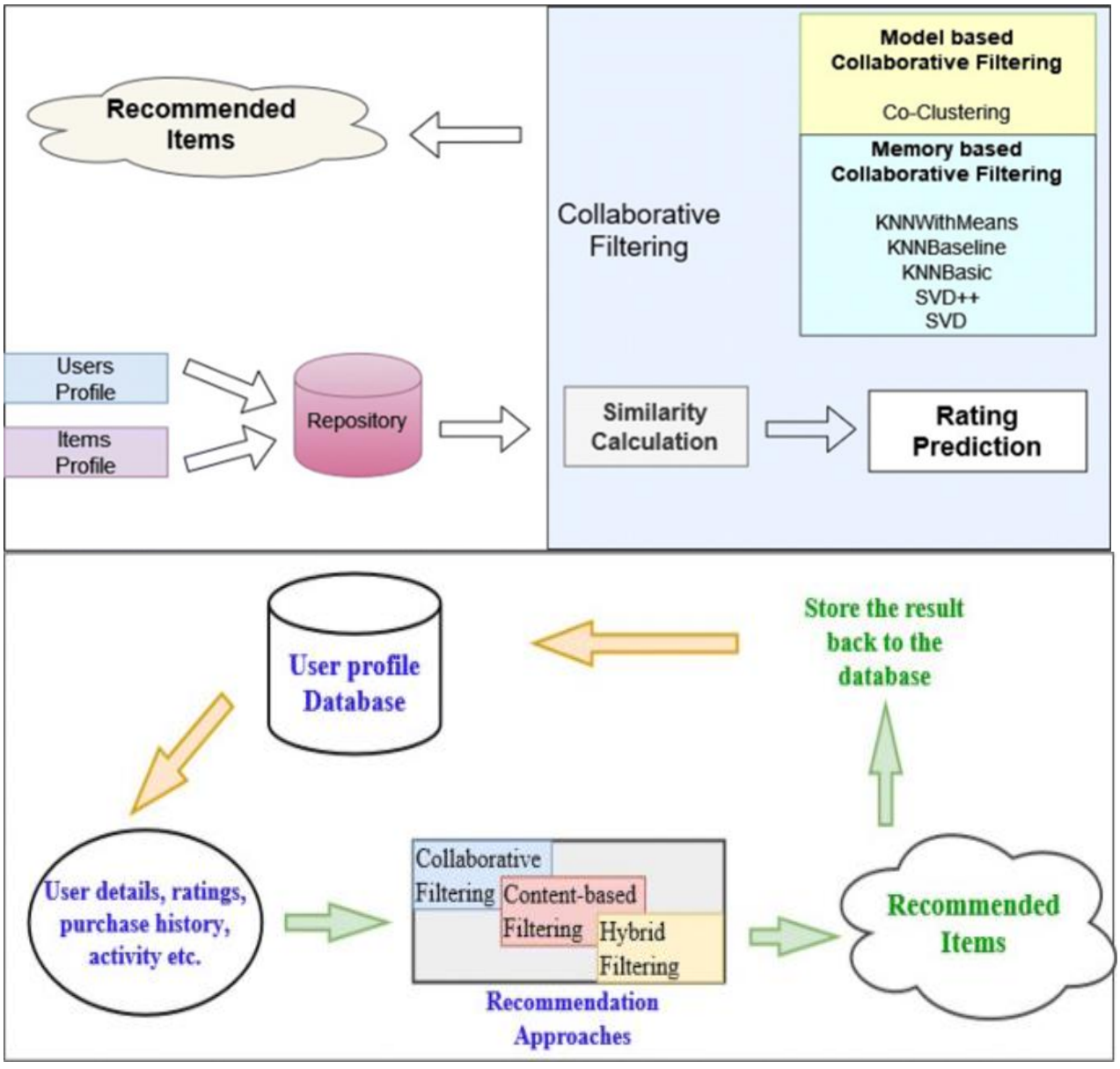


Figure 3.1: Project Architecture of Movie Recommendation System with Collaborative Filtering using KNN

.

3.2 USE CASE DIAGRAM

In the use case diagram, we have basically one actor who is the user in the trained model. A use case diagram is a graphical depiction of a user's possible interactions with a system. A use case diagram shows various use cases and different types of users the system has. The use cases are represented by either circles or ellipses. The actors are often shown as stick figures.

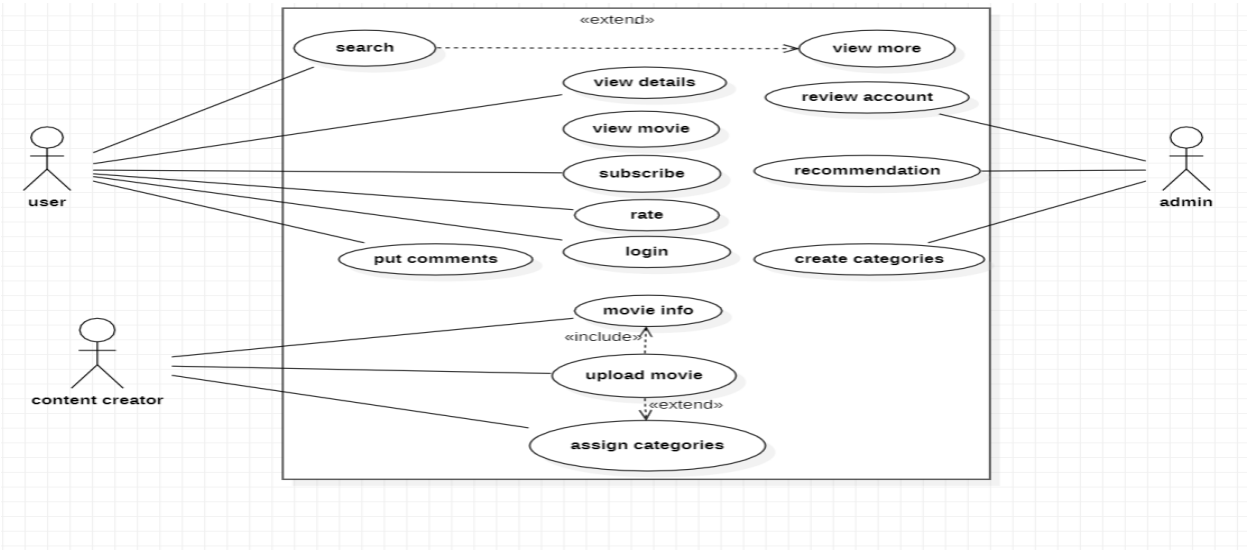


Figure 3.2: Use Case Diagram for Movie Recommendation System with Collaborative Filtering using KNN

3.3 SEQUENCE DIAGRAM

A sequence diagram shows object interactions arranged in time sequence. It depicts the objects involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the logical view of the system under development.

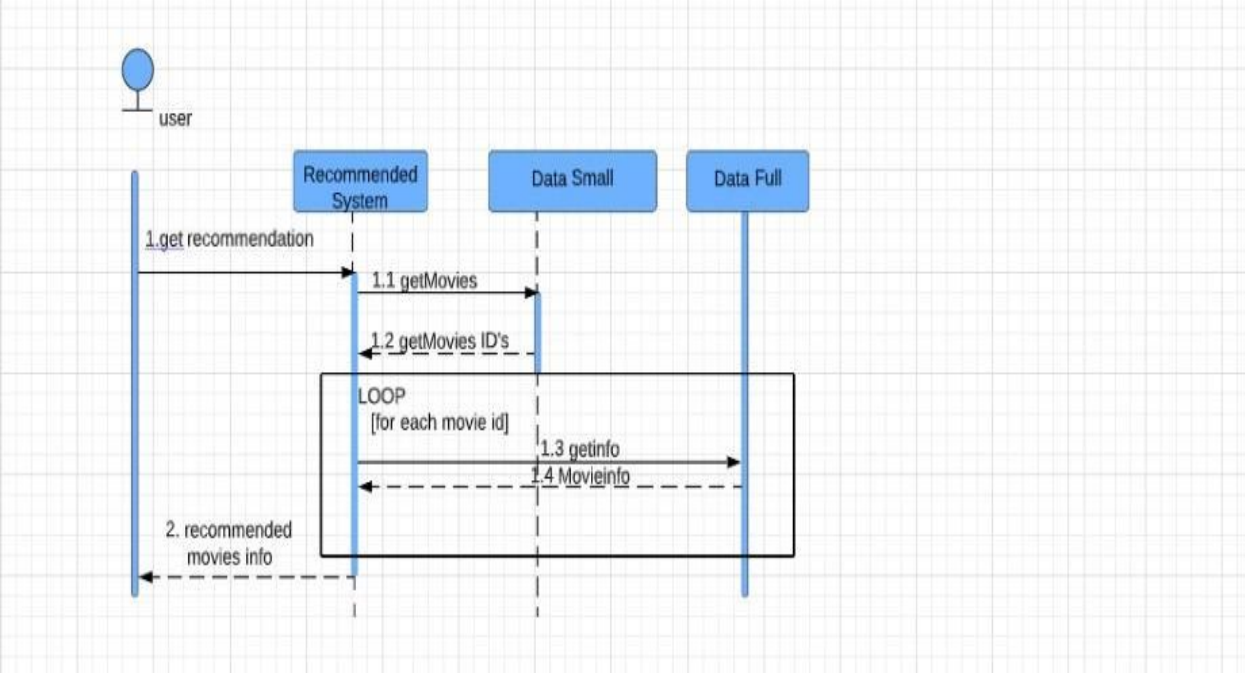


Figure 3.3: Sequence Diagram for Movie Recommendation System with Collaborative Filtering using KNN

3.4 Class Diagram

A sequence diagram shows object interactions arranged in time sequence. It depicts the objects involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the logical view of the system under development.

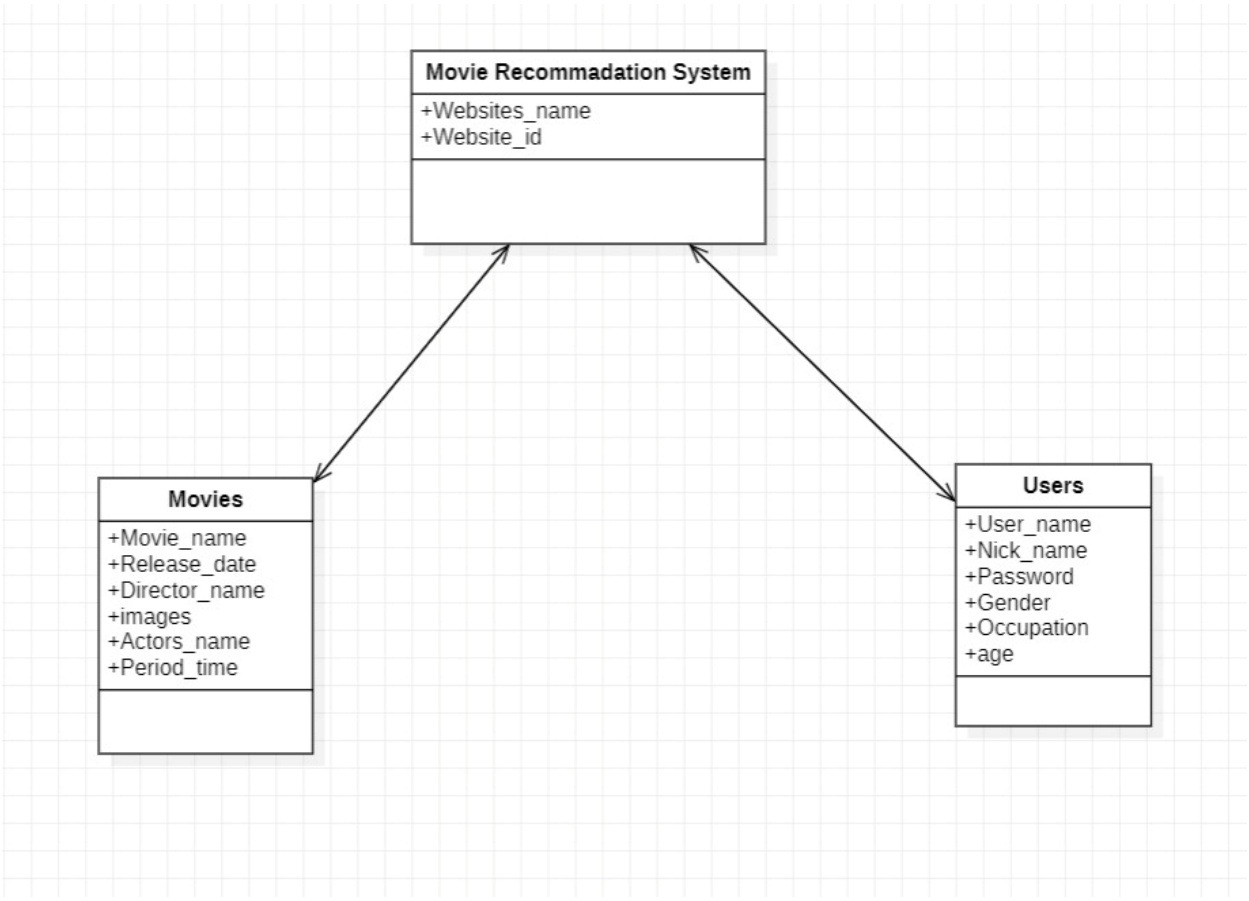


Figure 3.4 : Class Diagram for Movie Recommendation System with Collaborative Filtering using KNN

4.1 SAMPLE CODE

```
# data science imports
import math
import numpy as np
import pandas as pd
from scipy.sparse import csr_matrix
from sklearn.neighbors import NearestNeighbors

# utils import
from fuzzywuzzy import fuzz

# visualization imports
import seaborn as sns
import matplotlib.pyplot as plt

df_movies=pd.read_csv('movies.csv',usecols=['movieId','title'],
dtype={'movieId': 'int32', 'title': 'str'})
df_ratings=pd.read_csv('ratings.csv',usecols=['userId', 'movieId', 'rating'],
    dtype={'userId': 'int32', 'movieId': 'int32', 'rating': 'float32'})

df_movies.info()
df_ratings.info()
```

```
df_movies.head()

df_ratings.head()

# filter data

popularity_thres = 50

popular_movies = list(set(df_movies_cnt.query('count >= @popularity_thres').index))

df_ratings_drop_movies = df_ratings[df_ratings.movieId.isin(popular_movies)]

print('shape of original ratings data: ', df_ratings.shape)

print('shape of ratings data after dropping unpopular movies: ', df_ratings_drop_movies.shape)


# get number of ratings given by every user

df_users_cnt = pd.DataFrame(

df_ratings_drop_movies.groupby('userId').size(),columns=['count'])

df_users_cnt.head()


# plot rating frequency of all movies

ax = df_users_cnt \

.sort_values('count', ascending=False) \

.reset_index(drop=True) \

.plot(

 figsize=(12, 8),

 title='Rating Frequency of All Users',

 fontsize=12

 )

ax.set_xlabel("user Id")
```

```
ax.set_ylabel("number of ratings")


# filter data
ratings_thres = 50
active_users = list(set(df_users_cnt.query('count >= @ratings_thres').index))
df_ratings_drop_users=df_ratings_drop_movies[df_ratings_drop_movies.userId.isin
(active_users)]
print('shape of original ratings data: ', df_ratings.shape)
print('shape of ratings data after dropping both unpopular movies and
inactive users: ', df_ratings_drop_users.shape)


# pivot and create movie-user matrix
movie_user_mat = df_ratings_drop_users.pivot(index='movieId',
columns='userId', values='rating').fillna(0)
# create mapper from movie title to index
movie_to_idx = {
movie: i for i, movie in
enumerate(list(df_movies.set_index('movieId').loc[movie_user_mat.index].title))
}
# transform matrix to scipy sparse matrix
movie_user_mat_sparse = csr_matrix(movie_user_mat.values)


# define model
model_knn = NearestNeighbors(metric='cosine', algorithm='brute',
```

```
n_neighbors=20, n_jobs=-1)

# fit



model_knn.fit(movie_user_mat_sparse)

def fuzzy_matching(mapper, fav_movie, verbose=True):

    """

    return the closest match via fuzzy ratio. If no match found, return None

    Parameters

    ----------

    mapper: dict, map movie title name to index of the movie in data

    fav_movie: str, name of user input movie

    verbose: bool, print log if True

    Return

    ------

    index of the closest match

    """

    match_tuple = []

    # get match

    for title, idx in mapper.items():

    ratio = fuzz.ratio(title.lower(), fav_movie.lower())

    if ratio >= 60:

    match_tuple.append((title, idx, ratio))

    # sort

    match_tuple = sorted(match_tuple, key=lambda x: x[2])[::-1]

    if not match_tuple:
```

```
    print('Oops! No match is found')
    return

if verbose:
    print('Found possible matches in our database: {0}\n'.format([x[0] for x in match_tuple]))
    return match_tuple[0][1]

def make_recommendation(model_knn, data, mapper, fav_movie, n_recommendations):
    """
    return top n similar movie recommendations based on user's input movie
    Parameters
    ----------
    model_knn: sklearn model, knn model
    data: movie-user matrix
    mapper: dict, map movie title name to index of the movie in data
    fav_movie: str, name of user input movie
    n_recommendations: int, top n recommendations
    Return
    ------
    list of top n similar movie recommendations
    """
    # fit
    model_knn.fit(data)
    # get input movie index
    print('You have input movie:', fav_movie)
```

```
idx = fuzzy_matching(mapper, fav_movie, verbose=True)

# inference



print('Recommendation system start to make inference')

print('......\n')

distances, indices = model_knn.kneighbors(data[idx], n_neighbors=n_recommendations+1)

# get list of raw idx of recommendations

   raw_recommends = \

   sorted(list(zip(indices.squeeze().tolist(), distances.squeeze().tolist())),

   key=lambda x: x[1])[:0:-1]

# get reverse mapper

   reverse_mapper = {v: k for k, v in mapper.items()}

# print recommendations

  print('Recommendations for {}:'.format(fav_movie))

  for i, (idx, dist) in enumerate(raw_recommends):

  print('{0}: {1}'.format(i+1, reverse_mapper[idx]))



 my_favorite = 'iran man'

make_recommendation(

model_knn=model_knn,

data=movie_user_mat_sparse,

fav_movie=my_favorite,

mapper=movie_to_idx,

n_recommendations=10)
```

5.1 SCREENSHOTS

```
[ ]  # data science imports
     import math
     import numpy as np
     import pandas as pd
     from scipy.sparse import csr_matrix
     from sklearn.neighbors import NearestNeighbors
```

```
     # utils import
     from fuzzywuzzy import fuzz
```

```
[ ]  # visualization imports
     import seaborn as sns
     import matplotlib.pyplot as plt
```

Screenshot 5.1: Imported Libraries

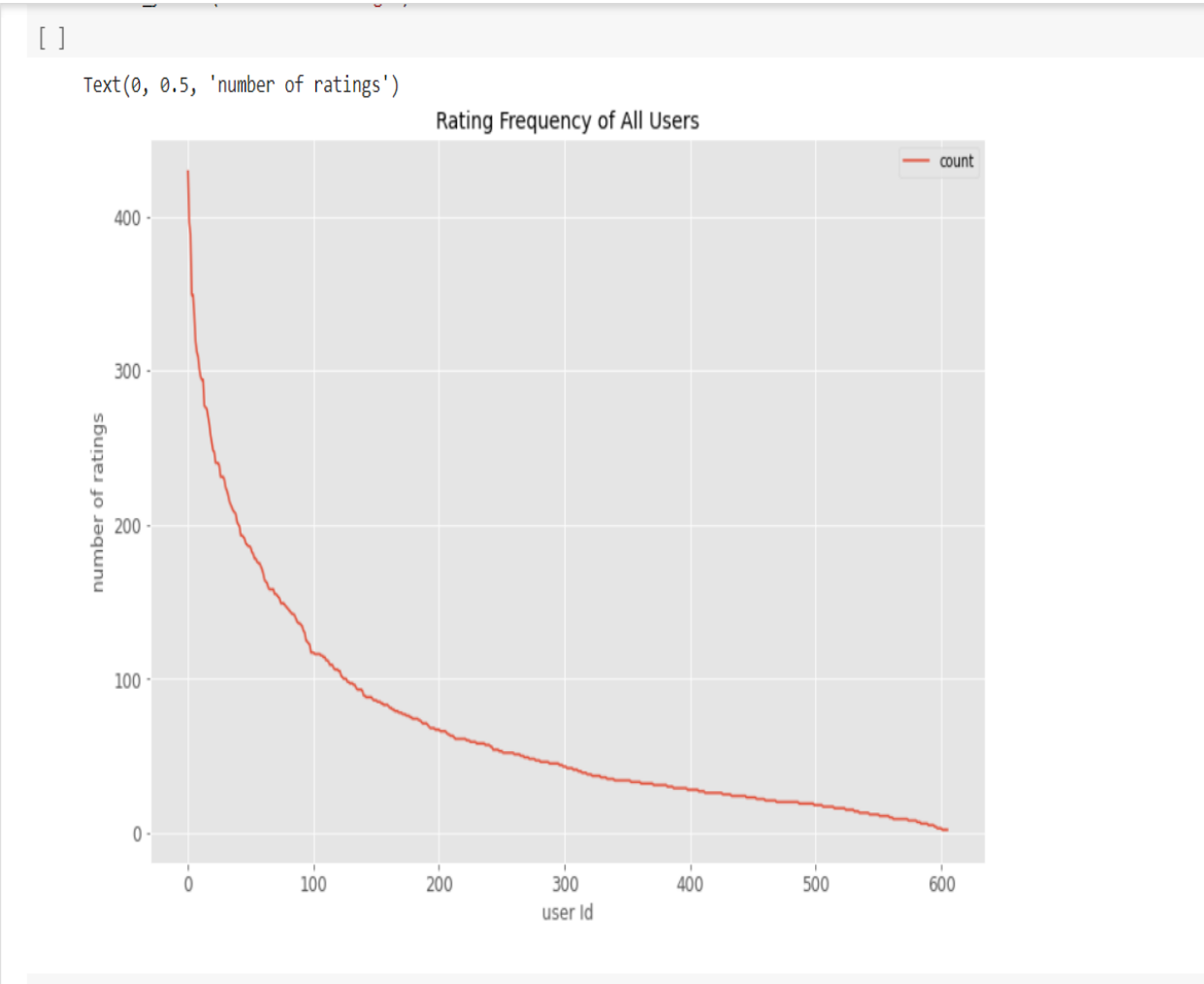
```
[ ] df_movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9742 entries, 0 to 9741
Data columns (total 2 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   movieId  9742 non-null   int32 
 1   title    9742 non-null   object
dtypes: int32(1), object(1)
memory usage: 114.3+ KB
```

```
df_ratings.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100836 entries, 0 to 100835
Data columns (total 3 columns):
 #   Column   Non-Null Count   Dtype  
---  ------   --------------   -----  
 0   userId   100836 non-null  int32  
 1   movieId  100836 non-null  int32  
 2   rating   100836 non-null  float32
dtypes: float32(1), int32(2)
memory usage: 1.2 MB
```

Screenshot 5.2: Dataset movie and rating info



Screenshot 5.3: Rating Frequency of All Users

```
my_favorite = 'iran man'

make_recommendation(
    model_knn=model_knn,
    data=movie_user_mat_sparse,
    fav_movie=my_favorite,
    mapper=movie_to_idx,
    n_recommendations=10)
```

```
You have input movie: iran man
Found possible matches in our database: ['Iron Man (2008)', 'Rain Man (1988)']

Recommendation system start to make inference
......

Recommendations for iran man:
1: Batman Begins (2005)
2: Sherlock Holmes (2009)
3: Kung Fu Panda (2008)
4: Inception (2010)
5: District 9 (2009)
6: Up (2009)
7: WALL·E (2008)
8: Avengers, The (2012)
9: Avatar (2009)
10: Dark Knight, The (2008)
```

Screenshot 5.4: Movie Recommendation System Output

6. INTRODUCTION TO TESTING

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, subassemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of tests. Each test type addresses a specific testing requirement.

6.2 TYPES OF TESTING

6.2.1 UNIT TESTING

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application. It is done after the completion of an individual unit before integration. This is a structural testing that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

6.2.2 INTEGRATION TESTING

Integration tests are designed to test integrated software components to determine if they actually run as one program. Integration tests demonstrate that although the components were individually satisfactory, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components

6.2.3 TOP-DOWN INTEGRATION

This method is an incremental approach to the construction of program structure. Modules are integrated by moving downward through the control hierarchy, beginning with the main program module. The module subordinates to the main program module are incorporated into the structure in either a depth first or breadth first manner. In this method, the software is tested from main module and individual stubs are replaced when the test proceeds downwards.

6.2.4 BOTTOM-UP INTEGRATION

This method begins the construction and testing with the modules at the lowest level in the program structure. Since the modules are integrated from the bottom up, processing required for modules subordinate to a given level is always available and the need for stubs is eliminated. The bottom-up integration strategy may be implemented with the following steps:

The low-level modules are combined into clusters into clusters that perform a specific Software sub-function.

A driver (i.e.) the control program for testing is written to coordinate test case input and output.

The cluster is tested.

Drivers are removed and clusters are combined moving upward in the program Structure The bottom-up approaches test each module individually and then each module is module is integrated with a main module and tested for functionality.

6.2.5 USER ACCEPTANCE TESTING

User Acceptance of a system is the key factor for the success of any system. The system under consideration is tested for user acceptance by constantly keeping in touch with the prospective system users at the time of developing and making changes wherever required. The system developed provides a friendly user interface that can easily be understood even by a person who is new to the system.

6.2.6 OUTPUT TESTING

After performing the validation testing, the next step is output testing of the proposed system, since no system could be useful if it does not produce the required output in the specified format. Asking the users about the format required by them tests the outputs generated or displayed by the system under consideration. Hence the output format is considered in 2 ways – one is on screen and another in printed format.

7. CONCLUSION & FUTURE SCOPE

7.1 PROJECT CONCLUSION

Under the condition of massive information, the requirements of movie recommendation system from film amateur are increasing. This article designs and implements a complete movie recommendation system prototype based on the KNN algorithm, collaborative filtering algorithm and recommendation system technology. We give a detailed design and development process, and test the stability and high efficiency of experiment system through professional test. This project has reference significance for the development of personalized recommendation technology.

7.2 FUTURE SCOPE

- Recommender systems can be a very powerful tool in a company's arsenal, and future developments are going to increase business value even further. Some of the applications include being able to anticipate seasonal purchases based on recommendations, determine important purchases, and give better recommendations to customers which can increase retention and brand loyalty.
- Most businesses will have some use for recommender systems, and I encourage everyone to learn more about this fascinating area.

8. BIBLIOGRAPHY

8.1 REFERENCES:

- 1.C. S. M. Wu, D. Garg and U. Bhandary, "Movie Recommendation System Using Collaborative Filtering", *2018 IEEE 9th International Conference on Software Engineering and Service Science (ICSESS)*, pp. 11-15, Nov. 2018.
- 2. W. Liang, G. Lu, X. Ji, J. Li and D. Yuan, "Difference factor' KNN collaborative filtering recommendation algorithm", *International Conference on Advanced Data Mining and Applications*, pp. 175-184, 2014, December.

8.2 GITHUB LINK

https://github.com/197R1A0520/Movie-Recommadation-System