Integrated Academy of Management and Technology, Ghaziabad

Presentation/Seminar Based on Major Project

BCA-606P

Integrated Academy of Management and Technology, Ghaziabad
NH 24, Near Dasna Crossing, Adhyatmik Nagar, Udyog Kunj,
Ghaziabad, Uttar Pradesh 201009

Major Project
BCA-606P

MAJOR PROJECT EVALUATION

As per the CCS University norms Major Project Report shall be evaluated by the examiner at the end of the semester. However there will be continuous monitoring of the Major Project progress report during the semester and distribution of marks shall be as follows:

BCA (BCA-606P) Major Project Evaluation Scheme

	Internal Examiner	Semester End Exam	Total
Subject Code Major Project	**Presentation** 50	**Presentation** 150	200

Adherence to the schedule is desired from each student failing which he/ she shall be solely responsible for the strict action taken against him/her.

Integrated Academy of Management and Technology, Ghaziabad
NH 24, Near Dasna Crossing, Adhyatmik Nagar, Udyog Kunj,
Ghaziabad, Uttar Pradesh 201009

Integrated Academy of Management and Technology, Ghaziabad

Movie Recommendation System

Project Report

Submitted in partial fulfillment of the requirement

for

BCA(606P)

under the guidance

of

Neelam Yadav

Assistant Professor

INMANTEC, CCS University

Ghaziabad, U.P

By

ABHISHEK

(R180845106003)

Integrated Academy of Management and Technology, Ghaziabad

BCA- VIth Semester

INMANTEC Institutions, Delhi-Hapur Bypass
(NH-24 near Dasna Flyover, UdyogKunj,
Ghaziabad, Uttar Pradesh 201009

Integrated Academy of Management and Technology, Ghaziabad

MAJOR PROJECT
BCA-606P

ACKNOWLEDGEMENT

I am very grateful to my Major Project Mentor Dr./Mr./Ms. **Neelam Yadav** for giving his/her valuable time and constructive guidance in preparing the Major Project. It would not have been possible to complete this Major Project in short period of time without his/her kind encouragement and valuable guidance.

Date:

Signature
Abhishek

Integrated Academy of Management and Technology, Ghaziabad

MAJOR PROJECT
BCA-606P

CERTIFICATE OF ORIGINALITY

I hereby declare that my Major Project (BCA – 606P) titled "Movie Recommendation System" submitted to CCS UNIVERSITY (Meerut U.P.) for the partial fulfillment of the degree of Bachelor In Computer Application Session 2018-2021 from INTEGRATED ACADEMY OF MANAGEMENT AND TECHNOLOGY, GHAZIABAD has not previously formed the basis for the award of any other degree, diploma or other title.

Place: Dhaulana, Hapur, U.P.- 245301

Date:

Signature

Abhishek

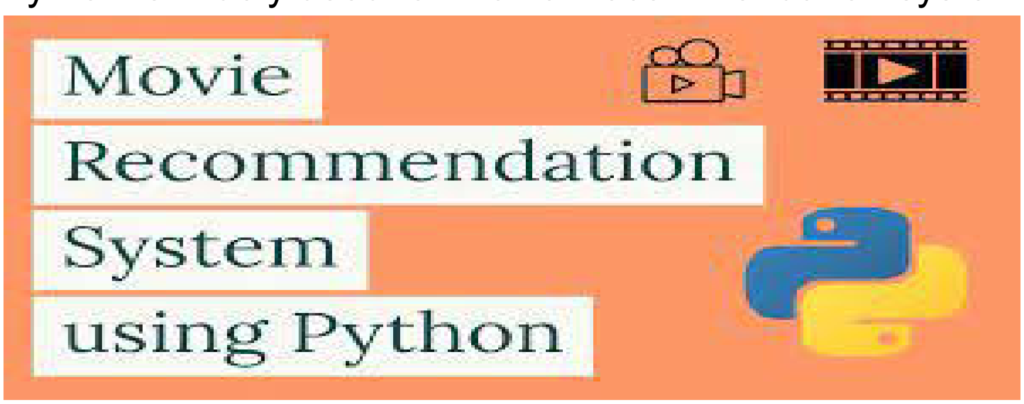
Integrated Academy of Management and Technology, Ghaziabad

NH 24, Near Dasna Crossing, Adhyatmik Nagar, Udyog Kunj,
Ghaziabad, Uttar Pradesh 201009

INDEX

Introduction

- A recommender system is software that presents a user a list of personalized recommendations, prepared on the basis of guessed user preferences.
- Recommender system receives information from the user and recommends the product that fits their needs the best
- Recommendation systems specific type of information filtering system technique that attempts to recommend information items (movies, TV program/show/episode, video on demand, music, books, news, images, websites, scientific papers, etc.) that are likely to be of interest to the user".
- Also Known as Recommender Systems, recommendation engines, recommendation Frameworks, Recommendation Platforms.
- For a user, employing a recommender system is one of the ways to **reach the information that interests him**
- These recommender systems have become a key Component of the modern E-Commerce applications.
- Eg: amazon.com uses recommender system to suggest books to the users
- Python is widely used for Movie Recommendation system.



Purpose of a recommendation system

There is a user viewpoint here: to easily and quickly find liked *items, products or websites* (the item can be any other information), save user's time, filter out irrelevant items.

There is a viewpoint of the owner of the recommender system: to add value to the *service, gain new users, increase sales, increase ad clicks, increase quality of leads sent.*

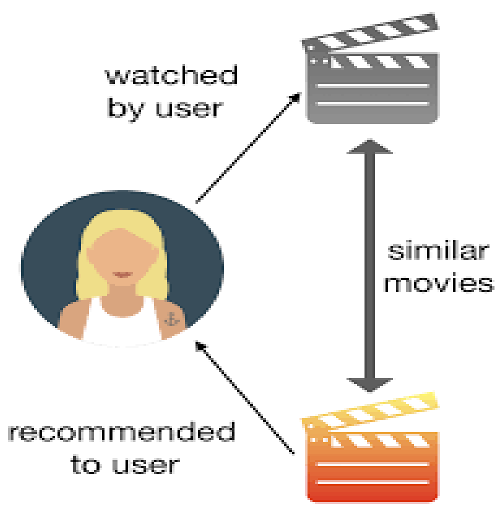
Why should we use recommendation System?

In the immortal of Steve Jobs : "A lot of times, people don't know what they want until you it to them. "
The Job of the recommender system is to open the customer up to a whole new products and possibilities, which they would not think to direct search for themselves.

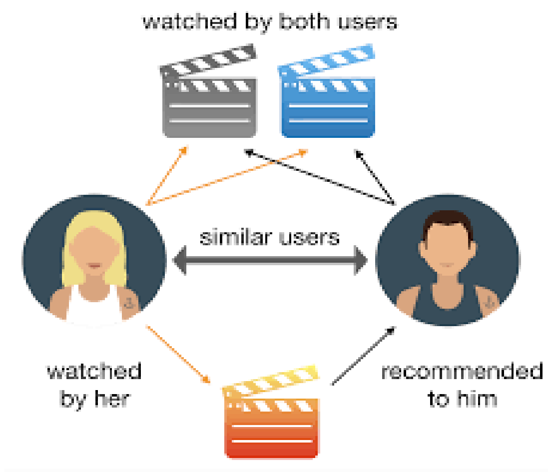
Types of Recommendation System

There are mainly two types of Recommendation System:

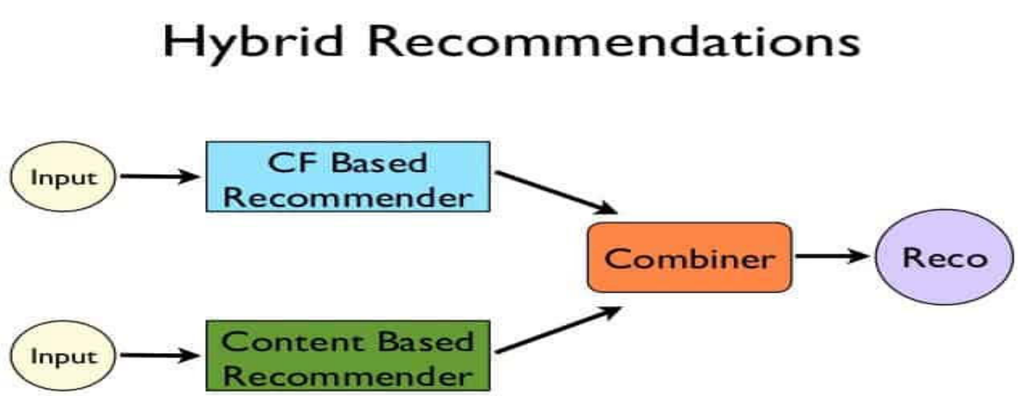
1. **Content Based Filtering**



2. **Collaboration Filtering**

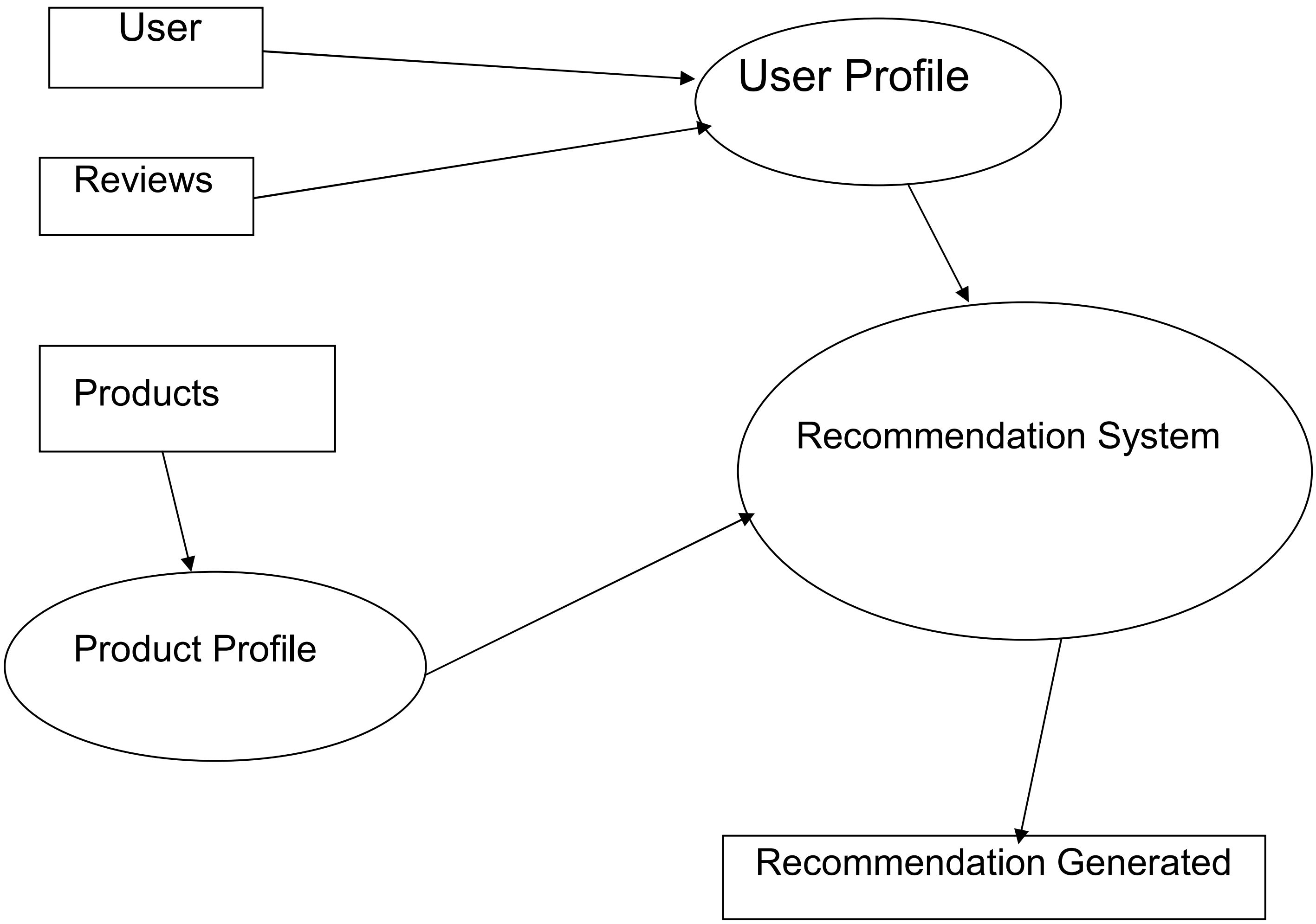


3. Hybrid Filtering

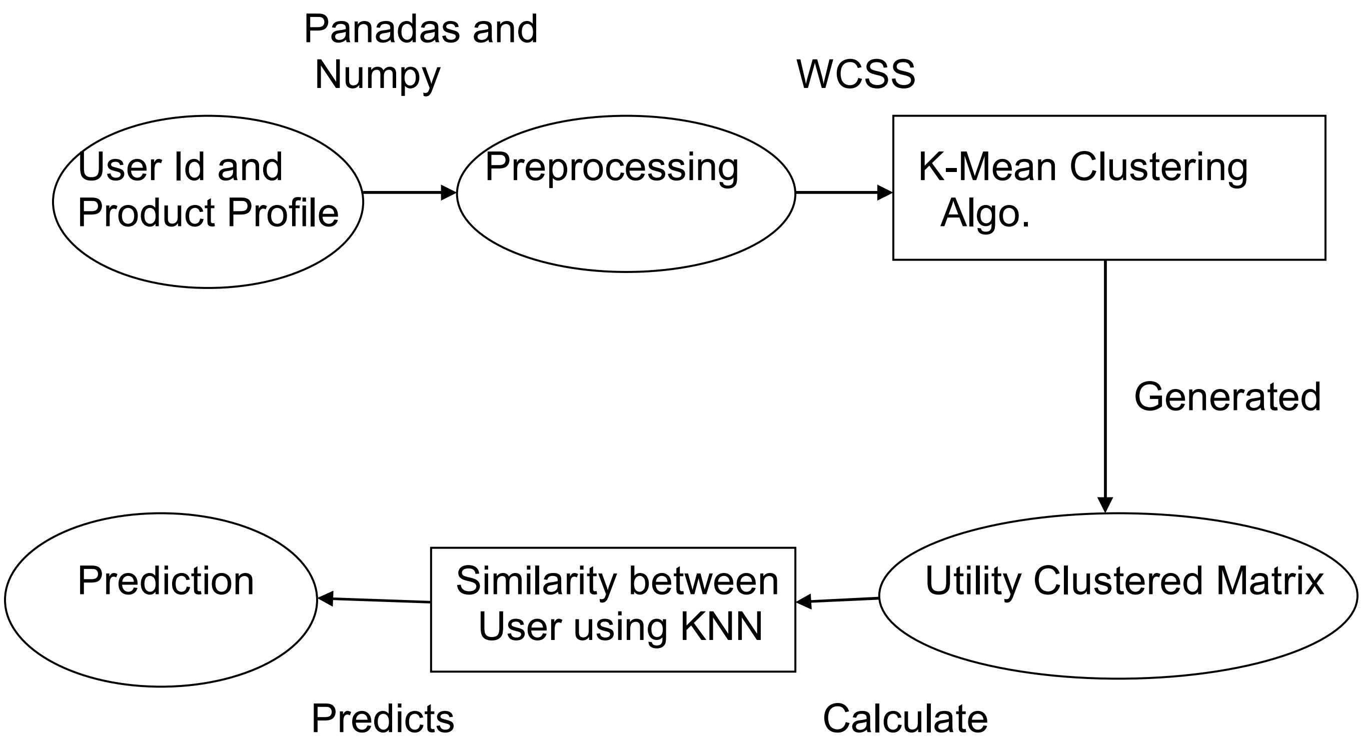


Data Flow Diagram (DFD)

Level-0 DFD :-



Level-1 DFD :-



Panadas and
Numpy
WCSS
User Id and
Product Profile
Preprocessing
K-Mean Clustering
Algo.
Generated
Utility Clustered Matrix
Similarity between
User using KNN
Prediction
Predicts
Calculate

SYSTEM SOFTWARE REQUIREMENT SPECIFICATION (SRS)

Below are the requirements used for running Movie Recommendation System

System Requirement

Command for installing Jupyter Notebook:- pip install jupyterlab

Windows-Based Requirements

- Dual-core 64-bit processor
- 8 GB of memory
- Up to 24 GB of internal storage (Jupyter Notebook: 2.5GB+1GB for caches,)
- Windows 10, Windows 8.1 Update, Windows 8, and Windows 7.1

Library Requirements of PyCharm Platform

- Pandas (Accessing and modifying Datasets)
- Numpy (Creating Multidimensional array)
- Matplotlib(Python plotting package)
- Fuzzybuzzy (String Matching)
- Sklearn (For Machine Learning Algorithm)

Code

#Importing Library

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from scipy.sparse import csr_matrix
from sklearn.cluster import KMeans
from sklearn.neighbors import NearestNeighbors
from sklearn.feature_extraction.text import CountVectorizer
from fuzzywuzzy import process
import sys
from sys import exc_info
```

#Reading DataSets

```
Mov_Id=pd.read_csv(r"E:\Major Project\Movie_Id.csv")
User=pd.read_csv(r"E:\Major Project\UserPro.csv")
```

Mov_Id.head()

```
In [89]: Mov_Id.head()
Out[89]:
```

	Item_Id	title
0	1	Toy Story (1995)
1	2	GoldenEye (1995)
2	3	Four Rooms (1995)
3	4	Get Shorty (1995)
4	5	Copycat (1995)

User.head()

```
In [90]: User.head()
Out[90]:
```

	User_Id	Item_Id	Rating	TimeStamp
0	0	50	5	881250949
1	0	172	5	881250949
2	0	133	1	881250949
3	196	242	3	881250949
4	186	302	3	891717742

#Create User Fav Movie list

```
users_fav_movies = User.loc[:, ['User_Id', 'Item_Id']]
users_fav_movies = User.reset_index(drop = True)
users_fav_movies.T
```

```
In [92]: users_fav_movies.T
```

Out[92]:

	0	1	2	3	4	5	6	7	8	9	...	99993	99994	
User_Id	0	0	0	196	186	22	244	166	298	115	...	806	676	
Item_Id	50	172	133	242	302	377	51	346	474	265	...	421	538	
Rating	5	5	1	3	3	1	2	1	4	2	...	4	4	
TimeStamp	881250949	881250949	881250949	881250949	891717742	878887116	880606923	886397596	884182806	881171488	...	882388897	892685437	877

4 rows × 100003 columns

```
def moviesListForUsers(users, users_data):
    # users = a list of users IDs
    # users_data = a dataframe of users favourite movies or users watched movies
    users_movies_list = []
    for user in users:
        users_movies_list.append(str(list(users_data[users_data['User_Id'] ==
user]['Item_Id'])).split('[')[1].split(']')[0])
    return users_movies_list
users = np.unique(users_fav_movies['User_Id'])
print(users.shape)
```

```
In [94]: users = np.unique(users_fav_movies['User_Id'])
         print(users.shape)

(944,)
```

```
users_movies_list = moviesListForUsers(users, users_fav_movies)
print('Movies list for', len(users_movies_list), ' users')
print('A list of first 2 users favourite movies: \n', users_movies_list[:2])
```

```
In [129]: users_movies_list = moviesListForUsers(users, users_fav_movies)
          print('Movies list for', len(users_movies_list), ' users')
          print('A list of first 2 users favourite movies: \n', users_movies_list[:2])

Movies list for 944  users
A list of first 2 users favourite movies:
 ['50, 172, 133', '61, 189, 33, 160, 20, 202, 171, 265, 155, 117, 47, 222, 253, 113, 227, 17, 90, 64, 92, 228, 266, 121, 114, 1
32, 74, 134, 98, 186, 221, 84, 31, 70, 60, 177, 27, 260, 145, 174, 159, 82, 56, 272, 80, 229, 140, 225, 235, 120, 125, 215, 6,
104, 49, 206, 76, 72, 185, 96, 213, 233, 258, 81, 78, 212, 143, 151, 51, 175, 107, 218, 209, 259, 108, 262, 12, 14, 97, 44, 53,
163, 210, 184, 157, 201, 150, 183, 248, 208, 128, 242, 148, 112, 193, 264, 219, 232, 236, 252, 200, 180, 250, 85, 91, 10, 254,
129, 241, 130, 255, 103, 118, 54, 267, 24, 86, 196, 39, 164, 230, 36, 23, 224, 73, 67, 65, 190, 100, 226, 243, 154, 214, 161, 6
2, 188, 102, 69, 170, 38, 9, 246, 22, 21, 179, 187, 135, 68, 146, 176, 166, 138, 247, 89, 2, 30, 63, 249, 269, 32, 141, 211, 4
0, 270, 133, 239, 194, 256, 220, 93, 8, 205, 234, 105, 147, 99, 1, 197, 173, 75, 268, 34, 144, 271, 119, 26, 158, 37, 181, 136,
257, 237, 131, 109, 182, 71, 223, 46, 169, 41, 162, 110, 66, 77, 199, 57, 50, 192, 178, 5, 87, 238, 156, 106, 167, 115, 11, 24
5, 35, 137, 127, 16, 79, 261, 45, 48, 25, 251, 195, 153, 101, 168, 123, 191, 4, 263, 203, 55, 42, 139, 240, 7, 149, 43, 165, 11
6, 198, 124, 95, 217, 58, 142, 216, 126, 83, 231, 204, 3, 207, 244, 19, 29, 18, 59, 15, 111, 52, 88, 13, 28, 172, 122, 152, 9
4']
```

#Creating Sparse Matrix

```
def prepSparseMatrix(list_of_str):
```

```
    # list_of_str = A list, which contain strings of users favourite movies separate by comma ",".
    # It will return us sparse matrix and feature names on which sparse matrix is defined
    # i.e. name of movies in the same order as the column of sparse matrix
    cv = CountVectorizer(token_pattern = r'[^\,\ ]+', lowercase = False)
    sparseMatrix = cv.fit_transform(list_of_str)
    return sparseMatrix.toarray(), cv.get_feature_names()
sparseMatrix, feature_names = prepSparseMatrix(users_movies_list)
df_sparseMatrix = pd.DataFrame(sparseMatrix, index = users, columns = feature_names)
df_sparseMatrix
```

Out[98]:

	1	10	100	1000	1001	1002	1003	1004	1005	1006	...	990	991	992	993	994	995	996	997	998	999
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	1	1	1	0	0	0	0	0	0	0	...	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	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
939	0	0	0	0	0	0	0	0	0	0	...	0	0	0	1	0	0	0	0	0	0
940	0	0	1	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
941	1	0	0	0	0	0	0	0	0	0	...	0	0	0	1	0	0	0	0	0	0
942	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
943	0	0	1	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

944 rows × 1682 columns

```
first_6_users_SM =
users_fav_movies[users_fav_movies['User_Id'].isin(users[:6])].sort_values('User_Id')
first_6_users_SM.T
df_sparseMatrix.loc[np.unique(first_6_users_SM['User_Id']), list(map(str,
np.unique(first_6_users_SM['Item_Id'])))]
```

In [125]:

```
first_6_users_SM = users_fav_movies[users_fav_movies['User_Id'].isin(users[:6])].sort_values('User_Id')
first_6_users_SM.T
df_sparseMatrix.loc[np.unique(first_6_users_SM['User_Id']), list(map(str, np.unique(first_6_users_SM['Item_Id'])))]
```

Out[125]:

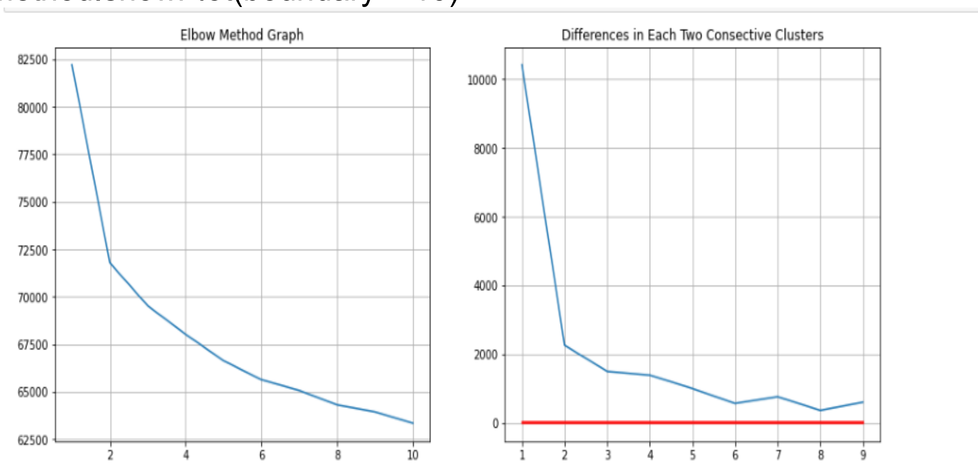
	1	2	3	4	5	6	7	8	9	10	...	448	449	450	451	452	453	454	455	456	457
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	...	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	1	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
5	1	1	0	0	0	0	0	0	0	0	...	1	1	1	1	1	1	1	1	1	1

6 rows × 457 columns

#Create Elbow Method for deciding right no of Cluster

```
class elbowMethod():
    def __init__(self, sparseMatrix):
        self.sparseMatrix = sparseMatrix
        self.wcss = list()
```

```
        self.differences = list()
    def run(self, init, upto, max_iterations = 300):
        for i in range(init, upto + 1):
            kmeans = KMeans(n_clusters=i, init = 'k-means++', max_iter = max_iterations,
n_init = 10, random_state = 0)
            kmeans.fit(sparseMatrix)
            self.wcss.append(kmeans.inertia_)
        self.differences = list()
        for i in range(len(self.wcss)-1):
            self.differences.append(self.wcss[i] - self.wcss[i+1])
    def showPlot(self, boundary = 500, upto_cluster = None):
        if upto_cluster is None:
            WCSS = self.wcss
            DIFF = self.differences
        else:
            WCSS = self.wcss[:upto_cluster]
            DIFF = self.differences[:upto_cluster - 1]
        plt.figure(figsize=(15, 6))
        plt.subplot(121).set_title('Elbow Method Graph')
        plt.plot(range(1, len(WCSS) + 1), WCSS)
        plt.grid(b = True)
        plt.subplot(122).set_title('Differences in Each Two Consective Clusters')
        len_differences = len(DIFF)
        X_differences = range(1, len_differences + 1)
        plt.plot(X_differences, DIFF)
        plt.plot(X_differences, np.ones(len_differences)*boundary, 'r')
        plt.plot(X_differences, np.ones(len_differences)*(-boundary), 'r')
        plt.grid()
        plt.show()
elbow_method = elbowMethod(sparseMatrix)
elbow_method.run(1, 10)
elbow_method.showPlot(boundary = 10)
```



```
#Appling K-Mean Clustering Algorithm (Creating Clusters)
kmeans = KMeans(n_clusters=4, init = 'k-means++', max_iter = 300, n_init =10 , random_state = 0)
clusters = kmeans.fit_predict(sparseMatrix)
users_cluster = pd.DataFrame(np.concatenate((users.reshape(-1,1), clusters.reshape(-1,1)), axis = 1), columns = ['User_Id', 'Cluster'])
users_cluster.T
```

```
In [104]: users_cluster = pd.DataFrame(np.concatenate((users.reshape(-1,1), clusters.reshape(-1,1)), axis = 1), columns = ['User_Id', 'Clus
          users_cluster.T
```

Out[104]:

	0	1	2	3	4	5	6	7	8	9	...	934	935	936	937	938	939	940	941	942	943
User_Id	0	1	2	3	4	5	6	7	8	9	...	934	935	936	937	938	939	940	941	942	943
Cluster	1	0	1	1	1	0	2	2	3	1	...	3	1	1	1	1	1	3	1	3	0

2 rows × 944 columns

```
def clustersMovies(users_cluster, users_data):
    clusters = list(users_cluster['Cluster'])
    each_cluster_movies = list()
    for i in range(len(np.unique(clusters))):
        users_list = list(users_cluster[users_cluster['Cluster'] == i]['User_Id'])
        users_movies_list = list()
        for user in users_list:
            users_movies_list.extend(list(users_data[users_data['User_Id'] == user]['Item_Id']))
        users_movies_counts = list()
        users_movies_counts.extend([[movie, users_movies_list.count(movie)] for movie in np.unique(users_movies_list)])
        each_cluster_movies.append(pd.DataFrame(users_movies_counts, columns=['Item_Id', 'Count']).sort_values(by = ['Count'], ascending = False).reset_index(drop=True))
    return each_cluster_movies
cluster_movies = clustersMovies(users_cluster, users_fav_movies)
cluster_movies[1].T
```

```
In [106]: cluster_movies[1].T
```

Out[106]:

	0	1	2	3	4	5	6	7	8	9	...	1305	1306	1307	1308	1309	1310	1311	1312	1313	1314
Item_Id	258	286	288	300	294	50	100	313	748	269	...	1131	1099	1125	1118	1113	1110	1109	1106	1100	1680
Count	295	293	284	275	271	214	210	201	200	191	...	1	1	1	1	1	1	1	1	1	1

2 rows × 1315 columns

```
for i in range(4):
    len_users = users_cluster[users_cluster['Cluster'] == i].shape[0]
    print('Users in Cluster ' + str(i) + ' -> ', len_users)
```

```
In [107]: for i in range(4):
              len_users = users_cluster[users_cluster['Cluster'] == i].shape[0]
              print('Users in Cluster ' + str(i) + ' -> ', len_users)

          Users in Cluster 0 ->  116
          Users in Cluster 1 ->  528
          Users in Cluster 2 ->  62
          Users in Cluster 3 ->  238
```

```
def getMoviesOfUser(user_id, users_data):
    return list(users_data[users_data['User_Id'] == user_id]['Item_Id'])
```

```
def fixClusters(clusters_movies_dataframes, users_cluster_dataframe, users_data,
smallest_cluster_size = 11):
    # clusters_movies_dataframes: will be a list which will contain each dataframes of
each cluster movies
    # users_cluster_dataframe: will be a dataframe which contain users IDs and their
cluster no.
    # smallest_cluster_size: is a smallest cluster size which we want for a cluster to not
remove
    each_cluster_movies = clusters_movies_dataframes.copy()
    users_cluster = users_cluster_dataframe.copy()
    # Let convert dataframe in each_cluster_movies to list with containing only movies
IDs
    each_cluster_movies_list = [list(df['Item_Id']) for df in each_cluster_movies]
    # First we will prepair a list which containt lists of users in each cluster -> [[Cluster 0
Users], [Cluster 1 Users], ... ,[Cluster N Users]]
    usersInClusters = list()
    total_clusters = len(each_cluster_movies)
    for i in range(total_clusters):
        usersInClusters.append(list(users_cluster[users_cluster['Cluster'] == i]['User_Id']))
    uncategorizedUsers = list()
    i = 0
    # Now we will remove small clusters and put their users into another list named
"uncategorizedUsers"
    # Also when we will remove a cluster, then we have also bring back cluster numbers
of users which comes after deleting cluster
    # E.g. if we have deleted cluster 4 then their will be users whose clusters will be
5,6,7,..,N. So, we'll bring back those users cluster number to 4,5,6,...,N-1.
    for j in range(total_clusters):
        if len(usersInClusters[i]) < smallest_cluster_size:
            uncategorizedUsers.extend(usersInClusters[i])
            usersInClusters.pop(i)
            each_cluster_movies.pop(i)
            each_cluster_movies_list.pop(i)
            users_cluster.loc[users_cluster['Cluster'] > i, 'Cluster'] -= 1
            i -= 1
        i += 1
```

```
    for user in uncategorizedUsers:
        elemProbability = list()
        user_movies = getMoviesOfUser(user, users_data)
        if len(user_movies) == 0:
            print(user)
        user_missed_movies = list()
        for movies_list in each_cluster_movies_list:
            count = 0
            missed_movies = list()
            for movie in user_movies:
                if movie in movies_list:
                    count += 1
                else:
                    missed_movies.append(movie)
            elemProbability.append(count / len(user_movies))
            user_missed_movies.append(missed_movies)
        user_new_cluster = np.array(elemProbability).argmax()
        users_cluster.loc[users_cluster['User_Id'] == user, 'Cluster'] = user_new_cluster
        if len(user_missed_movies[user_new_cluster]) > 0:
            each_cluster_movies[user_new_cluster] =
each_cluster_movies[user_new_cluster].append([{'Movie_Id': new_movie, 'Count': 1}
for new_movie in user_missed_movies[user_new_cluster]], ignore_index = True)
    return each_cluster_movies, users_cluster
```

```
movies_df_fixed, clusters_fixed = fixClusters(cluster_movies, users_cluster,
users_fav_movies, smallest_cluster_size = 6)
j = 0
for i in range(4):
    len_users = users_cluster[users_cluster['Cluster'] == i].shape[0]
    if len_users < 6:
        print('Users in Cluster ' + str(i) + ' -> ', len_users)
        j += 1
print('Total Cluster which we want to remove -> ', j)
```

```
In [111]: j = 0
          for i in range(4):
              len_users = users_cluster[users_cluster['Cluster'] == i].shape[0]
              if len_users < 6:
                  print('Users in Cluster ' + str(i) + ' -> ', len_users)
                  j += 1
          print('Total Cluster which we want to remove -> ', j)

          Total Cluster which we want to remove ->  0
```

```
print('Length of total clusters before fixing is -> ', len(cluster_movies))
print('Max value in users_cluster dataframe column Cluster is -> ',
users_cluster['Cluster'].max())
print('And dataframe is following')
```

users_cluster.T

```
In [112]: print('Length of total clusters before fixing is -> ', len(cluster_movies))
          print('Max value in users_cluster dataframe column Cluster is -> ', users_cluster['Cluster'].max())
          print('And dataframe is following')
          users_cluster.T

Length of total clusters before fixing is ->  4
Max value in users_cluster dataframe column Cluster is ->  3
And dataframe is following
```

Out[112]:

	0	1	2	3	4	5	6	7	8	9	...	934	935	936	937	938	939	940	941	942	943
User_Id	0	1	2	3	4	5	6	7	8	9	...	934	935	936	937	938	939	940	941	942	943
Cluster	1	0	1	1	1	0	2	2	3	1	...	3	1	1	1	1	1	3	1	3	0

2 rows × 944 columns

print('Length of total clusters after fixing is -> ', len(movies_df_fixed))
print('Max value in users_cluster dataframe column Cluster is -> ', clusters_fixed['Cluster'].max())
print('And fixed dataframe is following')
clusters_fixed.T
print('Users cluster dataFrame for cluster 4 after fixing which should be same as 11th cluster before fixing:')
clusters_fixed[clusters_fixed['Cluster'] == 4].T

```
In [114]: print('Users cluster dataFrame for cluster 4 after fixing which should be same as 11th cluster before fixing:')
          clusters_fixed[clusters_fixed['Cluster'] == 4].T

Users cluster dataFrame for cluster 4 after fixing which should be same as 11th cluster before fixing:
```

Out[114]:

User_Id
Cluster

print('Size of movies dataframe after fixing -> ', len(movies_df_fixed))

```
In [115]: print('Size of movies dataframe after fixing -> ', len(movies_df_fixed))

Size of movies dataframe after fixing -> 4
```


for i in range(len(movies_df_fixed)):
 len_users = clusters_fixed[clusters_fixed['Cluster'] == i].shape[0]
 print('Users in Cluster ' + str(i) + ' -> ', len_users)

```
In [116]: for i in range(len(movies_df_fixed)):
              len_users = clusters_fixed[clusters_fixed['Cluster'] == i].shape[0]
              print('Users in Cluster ' + str(i) + ' -> ', len_users)

Users in Cluster 0 ->  116
Users in Cluster 1 ->  528
Users in Cluster 2 ->  62
Users in Cluster 3 ->  238
```

for i in range(len(movies_df_fixed)):
 print('Total movies in Cluster ' + str(i) + ' -> ', movies_df_fixed[i].shape[0])

```
In [117]: for i in range(len(movies_df_fixed)):
              print('Total movies in Cluster ' + str(i) + ' -> ', movies_df_fixed[i].shape[0])

Total movies in Cluster 0 ->  1349
Total movies in Cluster 1 ->  1315
Total movies in Cluster 2 ->  1489
Total movies in Cluster 3 ->  1309
```

#Create Pivot Matrix

```
movies_users=User.pivot(index='Item_Id',columns='User_Id',values='Rating').fillna(0)
movies_users
```

```
In [118]: movies_users=User.pivot(index='Item_Id',columns='User_Id',values='Rating').fillna(0)
          movies_users
Out[118]:
```

User_Id / Item_Id	0	1	2	3	4	5	6	7	8	9	...	934	935	936	937	938	939	940	941	942	943
1	0.0	5.0	4.0	0.0	0.0	4.0	4.0	0.0	0.0	0.0	...	2.0	3.0	4.0	0.0	4.0	0.0	0.0	5.0	0.0	0.0
2	0.0	3.0	0.0	0.0	0.0	3.0	0.0	0.0	0.0	0.0	...	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	5.0
3	0.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	3.0	0.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0	...	5.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0	0.0
5	0.0	3.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1678	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	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1679	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	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1680	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	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1681	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	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1682	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	0.0	0.0	0.0	0.0	0.0	0.0	0.0

1682 rows × 944 columns

#Appling KNN Algo and Predicting Movies

```
mat_movies_users=csr_matrix(movies_users.values)
model_knn=NearestNeighbors(metric='cosine', algorithm='brute', n_neighbors=20)
model_knn.fit(mat_movies_users)
def recommender(movie_name, data, model, n_recommendations):
    model.fit(data)
    idx=process.extractOne(movie_name,Mov_Id['title'])[2]
    print('Movie Selected :',Mov_Id['title'][idx],['Index:',idx])
    print('Searching for recommendations...')
    distances, indices=model.kneighbors(data[idx], n_neighbors=n_recommendations)
    for i in indices:
        print(Mov_Id['title'][i].where(i!=idx))

recommender('Get Shorty',mat_movies_users, model_knn,20)
```

```
recommender('Get Shorty',mat_movies_users, model_knn,20)
```

```
Movie Selected : Get Shorty (1995) ['Index:', 3]
Searching for recommendations...
3                                              NaN
55                             Pulp Fiction (1994)
203                       Back to the Future (1985)
173                  Raiders of the Lost Ark (1981)
201                            Groundhog Day (1993)
95               Terminator 2: Judgment Day (1991)
194                          Terminator, The (1984)
171                  Empire Strikes Back, The (1980)
215                    When Harry Met Sally... (1989)
78                             Fugitive, The (1993)
384                                True Lies (1994)
209       Indiana Jones and the Last Crusade (1989)
167          Monty Python and the Holy Grail (1974)
402                                   Batman (1989)
143                                 Die Hard (1988)
237                          Raising Arizona (1987)
10                             Seven (Se7en) (1995)
185                       Blues Brothers, The (1980)
172                       Princess Bride, The (1987)
11                        Usual Suspects, The (1995)
Name: title, dtype: object
```

Future Scope and Challenges

- One of the challenges while proposing an algorithm for social recommendation is to determine the attributes affecting recommendation.

- As different factors affect RS differently, therefore, how to assign appropriate weights to the attributes is a major task when designing an algorithm.

- Similarly, temporal validity of items and news must be taken into account during recommendation. For instance, a phrase that used to be correct at one time may become false after a period of time (e.g. “The prime minister of India is Narender Modi” and “The prime minister of India is Manmohan Singh”).

- In a social network of billions of users, features and influence of users keep changing and implementing decay factors makes the sparse data more sparse as the former information becomes irrelevant which must be avoided and discarded.

- It is, therefore, challenging to cope with the problem of changing users requirements and social influencein social networks.

References/Glossary

Content and DFD->
https://www.researchgate.net/publication/334763301

Datasets ->
https://www.kaggle.com/grouplens/movielens-20m-dataset?select=movie.csv

https://www.kaggle.com/grouplens/movielens-20m-dataset?select=rating.csv

Code ->
https://asdkazmi.medium.com/ai-movies-recommendation-system-with-clustering-based-k-means-algorithm-f04467e02fcd