Experiment No. 6	
Implement Content-based recommendation system	
Date of Performance:	
Date of Submission:	
Marks:	
Sign:	

Aim: Implement Content-based recommendation system

Objective: Able to design and implement content based recommendation system.

Theory:

Content-based filtering system: Content-Based recommender system tries to guess the features or behaviour of a user given the item's features, he/she reacts positively to.

Movies	User 1	User 2	User 3	User 4	Action	Comedy
Item 1	1		4	5	Yes	No
Item 2	5	4	1	2	No	Yes
Item 3	4	4		3	Yes	Yes
Item 4	2	2	4	4	No	Yes

The last two columns Action and Comedy Describe the Genres of the movies. Now, given these genres, we can know which users like which genre, as a result, we can obtain features corresponding to that particular user, depending on how he/she reacts to movies of that genre. Once, we know the likings of the user we can embed him/her in an embedding space using the feature vector generated and recommend him/her according to his/her choice. During recommendation, the similarity metrics (We will talk about it in a bit) are calculated from the item's feature vectors and the user's preferred feature vectors from his/her previous records. Then, the top few are recommended.
Content-based filtering does not require other users' data during recommendations to one user.

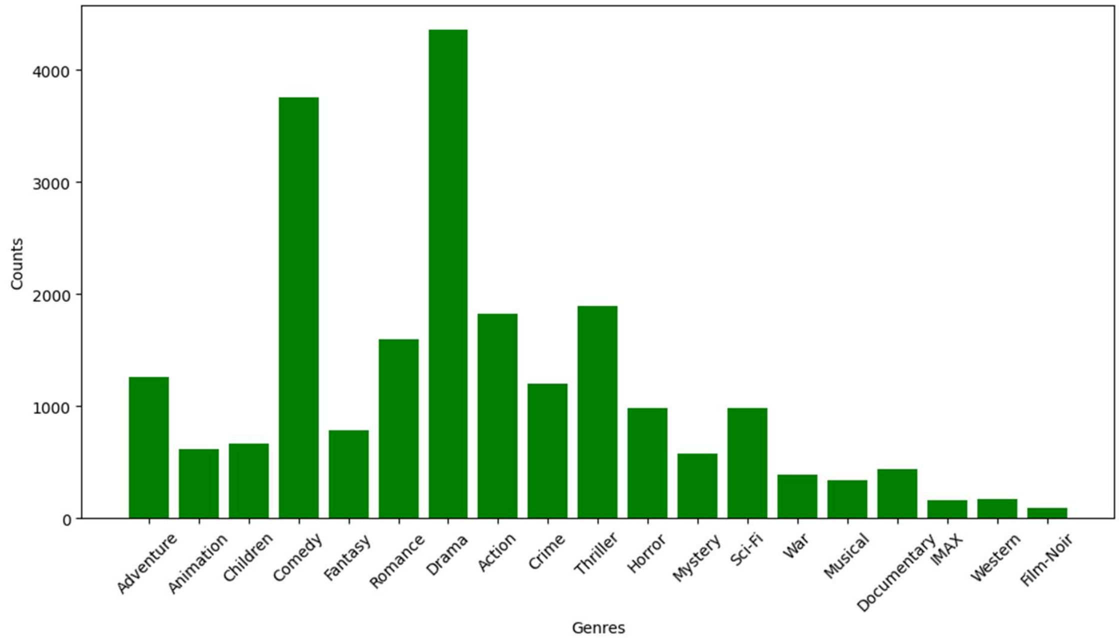
Implicit Feedback: The user's likes and dislikes are noted and recorded on the basis of his/her actions like clicks, searches, and purchases. They are found in abundance, but negative feedback is not found.

Explicit Feedback: The user specifies his/her likes or dislikes by actions like reacting to an item or rating it. It has both positive and negative feedback but less in number.

Implementation:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
movies = pd.read_csv('movies.csv')
movies.head(5)
movies.shape

def create_missing_df(dataframe):
missing_index = dataframe.columns.tolist()
missing = dataframe.isnull().sum().tolist()
missing_df =
pd.DataFrame({'Missing':missing},
index=missing_index)
  return missing_df
create_missing_df(movies)

def extract_title(title):
  year = title[len(title)-5:len(title)-1]
  # some movies do not have the info about
year in the column title. So, we should take
care of the case as well.
  if year.isnumeric():
    title_no_year = title[:len(title)-7]
    return title_no_year
  else:
    return title

def extract_year(title):
  year = title[len(title)-5:len(title)-1]
  # some movies do not have the info about
year in the column title. So, we should take
care of the case as well.
  if year.isnumeric():
    return int(year)
  else:
    return np.nan
movies.rename(columns={'title':'title_year'},
inplace=True) # change the column name from
title to title_year
movies['title_year'] =
movies['title_year'].apply(lambda x: x.strip())
# remove leading and ending whitespaces in
title_year
movies['title'] =
movies['title_year'].apply(extract_title) #
create the column for title
movies['year'] =
movies['title_year'].apply(extract_year) #
create the column for year

create_missing_df(movies)
r,c = movies[movies['genres']=='(no genres
listed)'].shape
print('The number of movies which do not
have info about genres:',r)
movies = movies[~(movies['genres']=='(no
genres listed)')].reset_index(drop=True)
movies[['title','genres']].head(5)
movies['genres'] =
movies['genres'].str.replace('|',' ')
counts = dict()

for i in movies.index:
  for g in movies.loc[i,'genres'].split(' '):
    if g not in counts:
      counts[g] = 1
    else:
      counts[g] = counts[g] + 1
plt.figure(figsize=(12,6))
plt.bar(list(counts.keys()), counts.values(),
color='g')
plt.xticks(rotation=45)
plt.xlabel('Genres')
plt.ylabel('Counts')

from sklearn.feature_extraction.text import
TfidfVectorizer
movies['genres'] =
movies['genres'].str.replace('Sci-Fi','SciFi')
movies['genres'] =
movies['genres'].str.replace('Film-Noir','Noir')
tfidf_vector =
TfidfVectorizer(stop_words='english') # create
an object for TfidfVectorizer
tfidf_matrix =
tfidf_vector.fit_transform(movies['genres']) #
apply the object to the genres column
```

```
print(list(enumerate(tfidf_vector.get_feature_n
ames_out())))
print(tfidf_matrix[:5])
tfidf_matrix.shape
tfidf_matrix.todense()[0]
from sklearn.metrics.pairwise import
linear_kernel
sim_matrix =
linear_kernel(tfidf_matrix,tfidf_matrix) #
create the cosine similarity matrix
print(sim_matrix)
def get_title_year_from_index(index):

  return movies[movies.index ==
index]['title_year'].values[0]

# the function to convert from title to index
def get_index_from_title(title):

  return movies[movies.title ==
title].index.values[0]
def matching_score(a,b):

  return fuzz.ratio(a,b)
def get_title_from_index(index):

  return movies[movies.index ==
index]['title'].values[0]
def find_closest_title(title):

  leven_scores =
list(enumerate(movies['title'].apply(matching_
score, b=title)))
  sorted_leven_scores = sorted(leven_scores,
key=lambda x: x[1], reverse=True)
closest_title =
get_title_from_index(sorted_leven_scores[0][
0])
  distance_score = sorted_leven_scores[0][1]

  return closest_title, distance_score
def
contents_based_recommender(movie_user_lik
es, how_many):

  closest_title, distance_score =
find_closest_title(movie_user_likes)

  if distance_score == 100:

    movie_index =
get_index_from_title(closest_title)
    movie_list =
list(enumerate(sim_matrix[int(movie_index)])
)
    similar_movies = list(filter(lambda x:x[0] !=
int(movie_index),
sorted(movie_list,key=lambda x:x[1],
reverse=True))) # remove the typed movie
itself

    print('Here\'s the list of movies similar to
'+'\033[1m'+str(closest_title)+'\033[0m'+'.\n')

    for i,s in similar_movies[:how_many]:
      print(get_title_year_from_index(i))


  else:
    print('Did you mean
'+'\033[1m'+str(closest_title)+'\033[0m'+'?','\n'
)

    movie_index =
get_index_from_title(closest_title)
    movie_list =
list(enumerate(sim_matrix[int(movie_index)])
)
    similar_movies = list(filter(lambda x:x[0] !=
int(movie_index),
sorted(movie_list,key=lambda x:x[1],
reverse=True)))

    print('Here\'s the list of movies similar to
'+'\033[1m'+str(closest_title)+'\033[0m'+'.\n')

    for i,s in similar_movies[:how_many]:
      print(get_title_year_from_index(i))
contents_based_recommender('Monsters, Inc.',
20)
contents_based_recommender('Monster
Incorporation.', 20)
```



Output:

Here's the list of movies similar to **Monsters, Inc.**.

Toy Story (1995)
Antz (1998)
Toy Story 2 (1999)
Adventures of Rocky and Bullwinkle, The (2000)
Emperor's New Groove, The (2000)
Wild, The (2006)
Shrek the Third (2007)
Tale of Despereaux, The (2008)
Asterix and the Vikings (Astérix et les Vikings) (2006)
Turbo (2013)
The Good Dinosaur (2015)
Moana (2016)
Inside Out (2015)
Black Cauldron, The (1985)
Lord of the Rings, The (1978)
We're Back! A Dinosaur's Story (1993)
Atlantis: The Lost Empire (2001)
Land Before Time, The (1988)
Pokemon 4 Ever (a.k.a. Pokémon 4: The Movie) (2002)
Sinbad: Legend of the Seven Seas (2003)

Did you mean **Monsters, Inc.**?
Here's the list of movies similar to **Monsters, Inc.**.
Toy Story (1995)
Antz (1998)
Toy Story 2 (1999)
Adventures of Rocky and Bullwinkle, The (2000)
Emperor's New Groove, The (2000)
Wild, The (2006)
Shrek the Third (2007)
Tale of Despereaux, The (2008)
Asterix and the Vikings (Astérix et les Vikings) (2006)
Turbo (2013)
The Good Dinosaur (2015)
Moana (2016)
Inside Out (2015)
Black Cauldron, The (1985)
Lord of the Rings, The (1978)
We're Back! A Dinosaur's Story (1993)
Atlantis: The Lost Empire (2001)
Land Before Time, The (1988)
Pokemon 4 Ever (a.k.a. Pokémon 4: The Movie) (2002)
Sinbad: Legend of the Seven Seas (2003)

Even if the title I typed was not the same as what the data contains, it found the movie correctly and recommended the list well.

Conclusion:

Content-based recommendation systems analyze item features to suggest similar items to users based on their preferences. By considering item attributes such as genre, keywords, or metadata, these systems offer personalized recommendations. Unlike collaborative filtering, content-based systems don't rely on user interactions, making them suitable for cold-start scenarios. With the ability to understand user preferences and recommend relevant content, they enhance user satisfaction and engagement.