RECOMMENDATION SYSTEMS

USER BASED COLLABORATIVE FILTERING IN RECOMMENDATION SYSTEMS

FOCUS OF OUR PROJECT

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. In general, it suggests relevant items to users. This application is used in Netflix, YouTube, Amazon, Instagram etc.

Recommendation systems are broadly classified into two types: Content based and Collaborative filtering. In our program, we will be focusing on user-based collaborative filtering to recommend movies to users, based on their historical rating information.

INCLUSION OF PACKAGES

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
from sklearn.metrics.pairwise import cosine_similarity, euclidean_distan
ces
```

Through these commands, we include all **the packages** that would be needed for the execution of the **in-built functions** throughout the program.

IMPORTING DATASETS

```
# importing movies.csv
movies = pd.read_csv(r'F:\Recommender System Project\movies.csv')
movies.head()
```

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

```
# importing ratings.csv
ratings = pd.read_csv(r'F:\Recommender System Project\ratings.csv')
ratings.head()
```

	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931

In these commands, we store movies.csv and ratings.csv in the data frames **movies** and **ratings** respectively and display the first five entries of each.

FINDING AVERAGE RATING OF THE USERS

```
mean = ratings.groupby(by = 'userId', as_index = False)['rating'].mean()
mean.head()
```

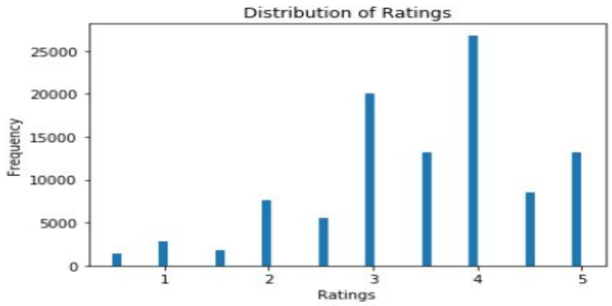
	userId	rating
0	1	4.366379
1	2	3.948276
2	3	2.435897
3	4	3.555556
4	5	3.636364

In these commands, we calculate the average rating for each user using the **mean()** function and store it in the data frame **mean.**

ANALYSIS OF DATAFRAMES

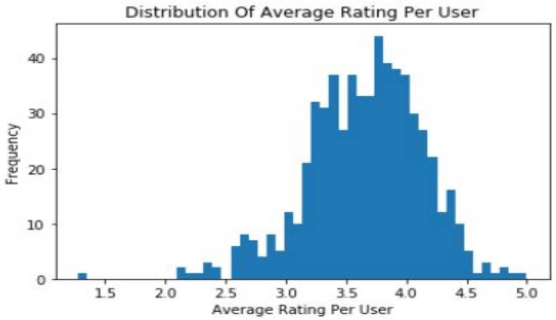
```
ratings.rating.plot.hist(bins=50)
plt.title('Distribution of Ratings')
plt.xlabel('Ratings')
```

```
Text(0.5, 0, 'Ratings')
```




```
mean.rating.plot.hist(bins=50)
plt.title('Distribution Of Average Rating Per User')
plt.xlabel('Average Rating Per User')
```

```
Text(0.5, 0, 'Average Rating Per User')
```



The histograms are plotted to analyse how the ratings and the average ratings are distributed in our dataframe.

FINDING ADJUSTED RATINGS

```
# Merging ratings and mean
avg_rating = pd.merge(ratings, mean, on='userId')
avg_rating = avg_rating.rename(columns = {'rating_x':'rating','rating_
y':'avg_rating'})
avg_rating.head()
```

	userId	movieId	rating	timestamp	avg_rating
0	1	1	4.0	964982703	4.366379
1	1	3	4.0	964981247	4.366379
2	1	6	4.0	964982224	4.366379
3	1	47	5.0	964983815	4.366379
4	1	50	5.0	964982931	4.366379

```
# Finding adjusted rating (rating - average rating)
avg_rating['adg_rating'] = avg_rating['rating'] - avg_rating['avg_ratin
g']
avg_rating.head()
```

	userId	movieId	rating	timestamp	avg_rating	adg_rating
0	1	1	4.0	964982703	4.366379	-0.366379
1	1	3	4.0	964981247	4.366379	-0.366379
2	1	6	4.0	964982224	4.366379	-0.366379
3	1	47	5.0	964983815	4.366379	0.633621
4	1	50	5.0	964982931	4.366379	0.633621

In these commands, the two data frames, namely **‘mean’** and **‘ratings’**, are merged and stored in **‘avg_rating’**.

Further, we add another column named **adg_rating** to the **‘avg_rating’** dataframe whose entries are the adjusted ratings (rating - mean rating).

CREATION OF USER -ITEM MATRIX

1)Using Actual Ratings as values

```
# Creating user-item matrix with ratings
ui_matrix = pd.pivot_table(avg_rating, values = 'rating', index = 'userI
d', columns = 'movieId')
ui_matrix.head()
```

movieId	1	2	3	4	5	6	7	8	9	10	...	193565	193567	193571	19
userId															
1	4.0	NaN	4.0	NaN	NaN	4.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	
5	4.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	

5 rows × 9724 columns

```
# Replacing NaN values of each row with average rating of user correspon
ding to that row
ui_matrix_avg = ui_matrix.apply(lambda row: row.fillna(row.mean()), axis
=1)
ui_matrix_avg.head()
```

movieId	1	2	3	4	5	6	7	8	9
userId									
1	4.000000	4.366379	4.000000	4.366379	4.366379	4.000000	4.366379	4.366379	4.366379
2	3.948276	3.948276	3.948276	3.948276	3.948276	3.948276	3.948276	3.948276	3.948276
3	2.435897	2.435897	2.435897	2.435897	2.435897	2.435897	2.435897	2.435897	2.435897
4	3.555556	3.555556	3.555556	3.555556	3.555556	3.555556	3.555556	3.555556	3.555556
5	4.000000	3.636364	3.636364	3.636364	3.636364	3.636364	3.636364	3.636364	3.636364

5 rows × 9724 columns

In the following commands, we form a matrix with **user ids** as row names, **movie ids** as column names and the actual **ratings given** by the users to movies as the values of the matrix.

Since the matrix formed is sparse, we fill the **NaN values** by **average rating of each user**.

2)Using Adjusted Ratings as values

```
# Creating user-item matrix with adjusted ratings
matrix = pd.pivot_table(avg_rating, values = 'adg_rating', index = 'user
Id', columns = 'movieId')
matrix.head()
```

movieId userId	1	2	3	4	5	6	7	8	9	10	...	193565	1935
1	-0.366379	NaN	-0.366379	NaN	NaN	-0.366379	NaN	NaN	NaN	NaN	...	NaN	Na
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	Na
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	Na
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	Na
5	0.363636	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	Na

5 rows × 9724 columns

```
# Replacing NaN values of each row with average of adjusted ratings of t
hat row
matrix_adj = matrix.apply(lambda row: row.fillna(row.mean()), axis=1)
matrix_adj.head()
```

movieId userId	1	2	3	4	5	6	7	
1	-3.663793e-01	1.837611e-16	-3.663793e-01	1.837611e-16	1.837611e-16	-3.663793e-01	1.837611e-16	1.8376
2	2.143879e-16	2.143879e-16	2.143879e-16	2.143879e-16	2.143879e-16	2.143879e-16	2.143879e-16	2.1438
3	3.643809e-16	3.643809e-16	3.643809e-16	3.643809e-16	3.643809e-16	3.643809e-16	3.643809e-16	3.6438
4	1.973730e-16	1.973730e-16	1.973730e-16	1.973730e-16	1.973730e-16	1.973730e-16	1.973730e-16	1.9737
5	3.636364e-01	1.009294e-16	1.009294e-16	1.009294e-16	1.009294e-16	1.009294e-16	1.009294e-16	1.0092

5 rows × 9724 columns

In these commands, we form another matrix with **user ids** as row names, **movie ids** as column names and **the adjusted ratings** of the users as the values of the matrix.

Since the matrix formed is sparse, we fill the **NaN values** with the **average of adjusted ratings of each user.**

We use this matrix to calculate the score of the movies at the end.

SIMILARITY CALCULATIONS

1)Cosine Similarity

```
# Similarity Measure : Cosine  Vector Similarity

c = cosine_similarity(ui_matrix_avg)
np.fill_diagonal(c,0)
cosine_sim = pd.DataFrame(c, index = ui_matrix_avg.index)
cosine_sim.columns = ui_matrix_avg.index
cosine_sim.head()
```

userId	1	2	3	4	5	6	7	8	9
userId									
1	0.000000	0.999542	0.998167	0.998170	0.999449	0.998593	0.998280	0.999451	0.999264
2	0.999542	0.000000	0.998504	0.998423	0.999780	0.998977	0.998621	0.999755	0.999588
3	0.998167	0.998504	0.000000	0.997026	0.998369	0.997624	0.997253	0.998357	0.998212
4	0.998170	0.998423	0.997026	0.000000	0.998300	0.997575	0.997346	0.998320	0.998133
5	0.999449	0.999780	0.998369	0.998300	0.000000	0.998891	0.998533	0.999657	0.999484

5 rows × 610 columns

In these commands, we pass the **user-item matrix** containing the actual ratings to the cosine_similarity() function and it returns a **similarity matrix** containing the **cosine similarity of the corresponding users.**

We convert that matrix into a dataframe with user id as both index and column name and display the first five entries of the data frame.

2)Pearson Correlation Coefficient

```
# Similarity measure: Pearson Correlation Coefficient

p = np.corrcoef(ui_matrix_avg)
warnings.filterwarnings("ignore")
np.fill_diagonal(p, 0)
pearson_corr = pd.DataFrame(p, index = ui_matrix_avg.index)
pearson_corr.columns = ui_matrix_avg.index
pearson_corr.head()
```

userId	1	2	3	4	5	6	7	
userId								
1	0.000000	1.264516e-03	5.525772e-04	0.048419	0.021847	-0.045497	-6.199672e-03	0.0470
2	0.001265	0.000000e+00	-7.172113e-25	-0.017164	0.021796	-0.021051	-1.111357e-02	-0.0480
3	0.000553	-7.172113e-25	0.000000e+00	-0.011260	-0.031539	0.004800	2.815738e-25	-0.0324
4	0.048419	-1.716402e-02	-1.125978e-02	0.000000	-0.029620	0.013956	5.809139e-02	0.0020
5	0.021847	2.179571e-02	-3.153892e-02	-0.029620	0.000000	0.009111	1.011715e-02	-0.0122

5 rows × 610 columns

In these commands, we pass the **user-item matrix** containing the actual ratings to the corrcoef() function and it returns a **similarity matrix** containing **pearson correlation coefficient of the corresponding users.**

We convert that matrix into a dataframe with user id as both index and column name and display the first five entries of the data frame.

3)Euclidean Distance

```
# Similarity measure: Euclidean Distance

e = euclidean_distances(ui_matrix_avg)
np.fill_diagonal(e,0)
euclidean_dist = pd.DataFrame(e, index = ui_matrix_avg.index)
euclidean_dist.columns = ui_matrix_avg.index
euclidean_dist.head()
```

userId	1	2	3	4	5	6	7	
userId								
1	0.000000	43.194401	191.187843	83.002410	73.271625	88.304438	113.879811	79.
2	43.194401	0.000000	149.752648	43.497710	31.704698	47.511412	72.799685	37.
3	191.187843	149.752648	0.000000	112.842171	119.276961	106.159881	81.049607	113.
4	83.002410	43.497710	112.842171	0.000000	22.009691	25.040256	40.381204	20.
5	73.271625	31.704698	119.276961	22.009691	0.000000	21.563681	43.710802	11.

5 rows × 610 columns

In these commands, we pass the user-item matrix containing the actual ratings to the **euclidean_distances() function** and it returns a **similarity matrix** containing the **euclidean distance between the corresponding users.**

We convert that matrix into a dataframe with user id as both index and column name and display the first five entries of the data frame.

TO FIND K-NEAREST NEIGHBOURS

```
def k_nearest_neighbours(df,k):
    sort = np.argsort(df.values,axis=1)[:, :k]
    df = df.apply(lambda x: pd.Series(x.sort_values(ascending
= False).iloc[:k].index, index = ['top{}'.format(i) for i in r
ange(1, k+1)]), axis=1)
    return df
```

In **k_nearest_neighbours(df,k)** function, we take a data frame **df**, which contains **the similarity values** found using three different similarity measures (mentioned previously) and **k**, which is **the number of nearest neighbours that we want to find,** as input**.**

It is done by **sorting** the similarity values of each user with all other users in **descending order**.

FINDING 30 NEAREST NEIGHBOURS USING ALL THE THREE METHODS

1) Using Cosine Vector Similarity

```
#Finding 30 nearest neighbours of each user based on cosine si
milarity
cosine_neighbours = k_nearest_neighbours(cosine_sim, 30)
cosine_neighbours.head()
```

Output:

	top1	top2	top3	top4	top5	top6	top7	top8	top9	top10	...	top21	top22	top23	top24
userId															
1	301	597	414	477	57	369	206	535	590	418	...	484	469	72	593
2	189	246	378	209	227	326	393	332	196	528	...	114	153	596	495
3	441	496	549	231	527	537	313	518	244	246	...	309	586	230	303
4	75	137	590	391	43	128	462	250	290	85	...	472	593	299	32
5	145	35	565	134	58	444	446	347	530	142	...	94	569	411	588

2) Pearson Correlation Coefficient

```
# Finding 30 nearest neighbours for each user based on pearson correlation
pearson_neighbours = k_nearest_neighbours(pearson_corr, 30)
pearson_neighbours.head()
```

Output:

userId	top1	top2	top3	top4	top5	top6	top7	top8	top9	top10	...	top21	top22	top23	top24
1	301	597	414	477	57	369	206	535	590	418	...	484	469	72	593
2	189	246	378	209	227	326	393	332	196	528	...	114	153	596	495
3	441	496	549	231	527	537	313	518	244	246	...	309	586	230	303
4	75	137	590	391	43	128	462	250	290	85	...	472	593	299	32
5	145	35	565	134	58	444	446	347	530	142	...	94	569	411	588

3) Euclidean Distance

```
# Finding 30 nearest neighbours for each user based on Euclide
an Distance
euclidean_neighbours = k_nearest_neighbours(euclidean_dist, 3
0)
euclidean_neighbours.head()
```

Output:

userId	top1	top2	top3	top4	top5	top6	top7	top8	top9	top10	...	top21	top22	top23
1	414	448	599	474	603	307	380	298	68	160	...	111	517	387
2	414	448	599	474	603	307	380	298	160	68	...	89	111	517
3	414	448	599	474	603	307	380	298	160	182	...	89	111	517
4	414	448	599	474	603	307	298	380	160	608	...	89	111	517
5	414	448	599	474	603	307	380	298	160	68	...	111	89	517

Finding MovieIds of Movies rated by each user

```
avg_rating = avg_rating.astype({'movieId':str})
rated_movies = avg_rating.groupby(by='userId')['movieId'].apply
(lambda x:','.join(x))
rated_movies
```

```
userId
1      1,3,6,47,50,70,101,110,151,157,163,216,223,231...
2      318,333,1704,3578,6874,8798,46970,48516,58559,...
3      31,527,647,688,720,849,914,1093,1124,1263,1272...
4      21,32,45,47,52,58,106,125,126,162,171,176,190,...
5      1,21,34,36,39,50,58,110,150,153,232,247,253,26...
                              ...
606    1,7,11,15,17,18,19,28,29,32,36,46,47,50,58,68,...
607    1,11,25,34,36,86,110,112,150,153,165,188,204,2...
608    1,2,3,10,16,19,21,24,31,32,34,39,44,47,48,50,6...
609    1,10,110,116,137,150,161,185,208,231,253,288,2...
610    1,6,16,32,47,50,70,95,110,111,112,153,159,194,...
Name: movieId, Length: 610, dtype: object
```

In these commands, firstly we convert the data type of **movie ids** of **avg_rating** data frame, from integer to String (str) and then we find the movie ids of the movies rated by each user and display it as a list separated by commas.

METHOD TO FIND 10 RECOMMENDED MOVIES FOR THE TARGET USER

```
def recommend_movies(user, similarity, neighbours):
    movies_seen_by_user = ui_matrix.columns[ui_matrix[ui_matrix.index==user].notna().any()].tolist()
    x = neighbours[neighbours.index==user].values
    y = x.squeeze().tolist()
    z = rated_movies[rated_movies.index.isin(y)]
    l = ','.join(z.values)
    movies_seen_by_neighbours = l.split(',')
    movies_under_consideration = list(set(movies_seen_by_neighbours)-set(list(map(str,movies_seen_by_user))))
    movies_under_consideration = list(map(int,movies_under_consideration))
```

In **recommend_movies(user, similarity, neighbours) function,** we take the target **user,** that is, the user for whom we need to find the top 10 recommendations, **similarity** i.e similarity matrix of a specified technique and **neighbours** i.e the top 30 neighbours calculated using the same technique. In the code given on the left, the movie ids of the movie seen by the target user are stored in the **movies_seen_by_user** data frame. Then, we store all the movies seen by the neighbours of the target user in **movies_seen_by_neighbours.**

movies_under_consideration contains all the movie ids excluding those which are common among the 30 neighbours and the target user.

```python
    score=[]
    for item in movies_under_consideration:
        p = matrix_adj.loc[:,item]
        q = p[p.index.isin(y)]
        r = q[q.notnull()]
        user_avg = mean.loc[mean['userId']==user,'rating'].values[0]
        index = r.index.values.squeeze().tolist()
        weight = similarity.loc[user, index]
        table = pd.concat([r, weight], axis=1)
        table.columns = ['adg_score','weight']
        table['score'] = table.apply(lambda x: x['adg_score']*x['weight'],axis=1)
        numerator = table['score'].sum()
        denomenator = table['weight'].sum()
        final_score = user_avg+(numerator/denomenator)
        score.append(final_score)
    recommendations = pd.DataFrame({'movieId':movies_under_consideration,'score':score})
    top_10_recommendations = recommendations.sort_values(by='score',ascending=False).head(10)
    movie_titles = top_10_recommendations.merge(movies, how='inner', on='movieId')
    movie_titles = movie_titles.title.values.tolist()
    return movie_titles
```

In continuation of the previous slide, we know that **movie_under_consideration** stores the movie ids of the movies that have been watched by the 30 nearest neighbours but not by the target user.

Now, we use the Score Formula,

$$s(u,i) = \bar{r}_u + \frac{\sum_{v \in V} (r_{vi} - \bar{r}_v) * w_{uv}}{\sum_{v \in V} w_{uv}}$$

to predict the rating that the target user will give to all those movies in movies_under_consideration. Then we recommend the **top 10 movies** to the target user according to the **final_score** calculated using the above formula.

DISPLAYING TOP 10 RECOMMENDED MOVIES USING ALL THE THREE SIMILARITY MEASURES

```
user=int(input("Enter the user id to whom you want to recommen
d: "))
predicted_movies_cosine = recommend_movies(user, cosine_sim, co
sine_neighbours)
predicted_movies_pearson = recommend_movies(user, pearson_corr,
pearson_neighbours)
predicted_movies_euclidean = recommend_movies(user, euclidean_d
ist, euclidean_neighbours)
print()
print("The Recommendations For User Id ",user," using Cosine Ve
ctor Similarity :")
print()
for i in predicted_movies_cosine:
    print(i)
print()
print("The Recommendations For User Id ",user," using Pearson C
orrelation Coefficient :")
print()
for i in predicted_movies_pearson:
    print(i)
print()
print("The Recommendations For User Id ",user," using Euclidean
Distance :")
print()
for i in predicted_movies_euclidean:
    print(i)
```

In this code, the user id of the target user is accepted. Then, we display the top 10 recommended movies for the target user using all three similarity measures (Cosine vector similarity, Pearson correlation coefficient and Euclidean distance).

FINAL OUTPUT

```
Enter the user id to whom you want to recommend: 317

The Recommendations For User Id  317  using Cosine Vector Similarity :

Schindler's List (1993)
Godfather: Part II, The (1974)
Star Wars: Episode VI - Return of the Jedi (1983)
Harry Potter and the Sorcerer's Stone (a.k.a. Harry Potter and the Phil
osopher's Stone) (2001)
Aladdin (1992)
Midnight Clear, A (1992)
Lord of the Rings: The Two Towers, The (2002)
How to Train Your Dragon (2010)
WALL·E (2008)
Guardians of the Galaxy (2014)

The Recommendations For User Id  317  using Pearson Correlation Coeffic
ient :

Schindler's List (1993)
Godfather: Part II, The (1974)
Star Wars: Episode V - The Empire Strikes Back (1980)
Star Wars: Episode VI - Return of the Jedi (1983)
Princess Bride, The (1987)
Lord of the Rings: The Fellowship of the Ring, The (2001)
Lord of the Rings: The Two Towers, The (2002)
Monty Python's Life of Brian (1979)
Lord of the Rings: The Return of the King, The (2003)
Memento (2000)

The Recommendations For User Id  317  using Euclidean Distance :

Back to the Future (1985)
Blade Runner (1982)
Donnie Darko (2001)
Star Wars: Episode VI - Return of the Jedi (1983)
Nightmare Before Christmas, The (1993)
Princess Bride, The (1987)
Lord of the Rings: The Return of the King, The (2003)
Hours, The (2002)
Royal Tenenbaums, The (2001)
Lord of the Rings: The Two Towers, The (2002)
```

This is the output that we get when we input the user id ‘317’.

It displays the top 10 movies recommended to the target user using the three different similarity measures.