Machine Learning

Project – Amazon

Report and Source Code:

Import the dataset, **Amazon - Movies and TV Ratings.csv** from the location it saved in your computer in Jupyter Notebook and other basic libraries required for the analysis and building recommendation model.

Importing Libraries:

import pandas as pd

import numpy as np

Reading and saving the file in dataframe variable:

Note: The file is saved in D drive and the corresponding folder in my PC. Location in the code will change with respect to the user and the file saved in the corresponding PC in corresponding folder.

Input Code:

df = pd.read_csv("D:/PGP-DS/Machine Learning/Amazon - Movies and TV Ratings.csv", index_col=0)

df

Output: The output on Jupyter is as the following with the last column as 'Movie 206' with 4848 rows of user ids and corresponding ratings given by the user.

```
[6]: df
```

[6]:

	Movie1	Movie2	Movie3	Movie4	Movie5	Movie6	Movie7	Movie8	Movie9	Movie10	...	Movie197	Movie198
user_id													
A3R5OBKS7OM2IR	5.0	5.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
AH3QC2PC1VTGP	NaN	NaN	2.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
A3LKP6WPMP9UKX	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
AVIY68KEPQ5ZD	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
A1CV1WROP5KTTW	NaN	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...
A1IMQ9WMFYKWH5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
A1KLIKPUF5E88I	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
A5HG6WFZLO10D	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
A3UU690TWXCG1X	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
AI4J762YI6S06	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN

4848 rows × 206 columns

Analysis Tasks:

1. Which movies have maximum views/ratings?
   Input Code: df.count().idxmax()
   Output: 'Movie 127' has maximum views/ratings.

2. What is the average rating for each movie?
   Input Code: df.mean()
   Output: Output is as shown in the figure below.

```
[8]: df.mean()

[8]: Movie1      5.000000
     Movie2      5.000000
     Movie3      2.000000
     Movie4      5.000000
     Movie5      4.103448
                   ...
     Movie202    4.333333
     Movie203    3.000000
     Movie204    4.375000
     Movie205    4.628571
     Movie206    4.923077
     Length: 206, dtype: float64
```

3. Define the top 5 movies with the maximum ratings
   Input Code: df.count().sort_values(ascending=False)[:5]
   Output: The following are the movies with maximum ratings along with the total number of reviews received i.e., the output obtained is as follows:
   i. Movie127 - 2313
   ii. Movie140 - 578
   iii. Movie16 - 320
   iv. Movie103 - 272
   v. Movie29 - 243

4. Define the top 5 movies with the least audience.
   Input Code: df.count().sort_values()[:5]
   Remark: The default setting of the function 'sort_values' is in ascending order.
   Output: The following are movies with least audience and their corresponding number i.e., the output obtained is as follows:
   i. Movie1 - 1
   ii. Movie71 - 1
   iii. Movie145 - 1
   iv. Movie69 - 1
   v. Movie68 - 1

Building the recommendation model:

Recommendation Model: Some of the movies hadn't been watched and therefore, are not rated by the users. Netflix would like to take this as an opportunity and build a machine learning recommendation algorithm which provides the ratings for each of the users. Two methods have been applied for this process – one Similarity method and the other using surprise library:

Using Similarity Method:

Setting the N/A values to '-1' as the least rating in this case is -1 and saving it in a new dataframe variable:

Input Code: dfNew = df.fillna(-1)

dfNew

Output: The new dataframe is saved as follows which contains 4848 user ids and 206 movies and their corresponding values with values of N/A changed to '-1'.

```
[12]: dfNew
```

[12]:

user_id	Movie1	Movie2	Movie3	Movie4	Movie5	Movie6	Movie7	Movie8	Movie9	Movie10	...	Movie197	Movie198
A3R5OBKS7OM2IR	5.0	5.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	...	-1.0	-1.0
AH3QC2PC1VTGP	-1.0	-1.0	2.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	...	-1.0	-1.0
A3LKP6WPMP9UKX	-1.0	-1.0	-1.0	5.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	...	-1.0	-1.0
AVIY68KEPQ5ZD	-1.0	-1.0	-1.0	5.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	...	-1.0	-1.0
A1CV1WROP5KTTW	-1.0	-1.0	-1.0	-1.0	5.0	-1.0	-1.0	-1.0	-1.0	-1.0	...	-1.0	-1.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
A1IMQ9WMFYKWH5	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	...	-1.0	-1.0
A1KLIKPUF5E88I	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	...	-1.0	-1.0
A5HG6WFZLO10D	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	...	-1.0	-1.0
A3UU690TWXCG1X	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	...	-1.0	-1.0
AI4J762YI6S06	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	...	-1.0	-1.0

4848 rows × 206 columns

Input Code:

dfTrans = dfNew.transpose()

movieNames = dfNew.columns

movieNames

Output:

```
[15]: movieNames
```

```
[15]: Index(['Movie1', 'Movie2', 'Movie3', 'Movie4', 'Movie5', 'Movie6', 'Movie7',
             'Movie8', 'Movie9', 'Movie10',
             ...
             'Movie197', 'Movie198', 'Movie199', 'Movie200', 'Movie201', 'Movie202',
             'Movie203', 'Movie204', 'Movie205', 'Movie206'],
            dtype='object', length=206)
```

Applying Cosine Similarity Formula on Association Matrix:

Input Code:

from scipy.spatial.distance import cosine

from scipy.spatial.distance import euclidean

from scipy.spatial.distance import hamming

from scipy.spatial.distance import minkowski

from sklearn.metrics import pairwise_distances

movie_similarity = 1 - pairwise_distances(dfTrans, metric="cosine")

np.fill_diagonal(movie_similarity, 0)

ratings_matrix = pd.DataFrame(movie_similarity)

ratings_matrix.head()

Remark: Pairwise distance only accept Matrix object. ‘movie_similarity’ object the model.

Output:

[16]:

	0	1	2	3	4	5	6	7	8	9	...	196	197	198	199
0	0.000000	1.000000	0.995374	0.988946	0.917703	0.993734	0.992611	0.992611	0.992611	0.992611	...	0.983481	0.988946	0.992611	0.970660
1	1.000000	0.000000	0.995374	0.988946	0.917703	0.993734	0.992611	0.992611	0.992611	0.992611	...	0.983481	0.988946	0.992611	0.970660
2	0.995374	0.995374	0.000000	0.991700	0.920275	0.996501	0.995374	0.995374	0.995374	0.995374	...	0.986221	0.991700	0.995374	0.973367
3	0.988946	0.988946	0.991700	0.000000	0.914281	0.990066	0.988946	0.988946	0.988946	0.988946	...	0.979846	0.985294	0.988946	0.967067
4	0.917703	0.917703	0.920275	0.914281	0.000000	0.918748	0.917703	0.917703	0.917703	0.917703	...	0.909160	0.914281	0.917703	0.897188

5 rows × 206 columns

Input Code: ratings_matrix

Output: 206 x 206 matrix**.**

```
[17]: ratings_matrix
```

[17]:

	0	1	2	3	4	5	6	7	8	9	...	196	197	198
0	0.000000	1.000000	0.995374	0.988946	0.917703	0.993734	0.992611	0.992611	0.992611	0.992611	...	0.983481	0.988946	0.992611
1	1.000000	0.000000	0.995374	0.988946	0.917703	0.993734	0.992611	0.992611	0.992611	0.992611	...	0.983481	0.988946	0.992611
2	0.995374	0.995374	0.000000	0.991700	0.920275	0.996501	0.995374	0.995374	0.995374	0.995374	...	0.986221	0.991700	0.995374
3	0.988946	0.988946	0.991700	0.000000	0.914281	0.990066	0.988946	0.988946	0.988946	0.988946	...	0.979846	0.985294	0.988946
4	0.917703	0.917703	0.920275	0.914281	0.000000	0.918748	0.917703	0.917703	0.917703	0.917703	...	0.909160	0.914281	0.917703
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

Input Code:

candidateMovie = 'Movie4'

similarity = ratings_matrix[movieNames.get_loc(candidateMovie)]

recommendedMovies = sorted([(movieNames.tolist()[i], similarity[i]) for i in range(len(similarity))], key = lambda x:x[1], reverse=True)

recommendedMovies[0:10]

Output: The following are the recommended movies:

[('Movie45', 0.9922123672626545),
('Movie58', 0.9922123672626545),
('Movie60', 0.9922123672626545),
('Movie67', 0.9922123672626545),
('Movie69', 0.9922123672626545),
('Movie144', 0.9922123672626545),
('Movie154', 0.9922123672626545),
('Movie3', 0.9917003197404489),
('Movie59', 0.9917003197404489),
('Movie73', 0.9917003197404489)]

Note: Using the similarity model, movies are recommended based on a particular movie.

Using Surprise Library:

Note: Here the initial dataframe (df) has been used.

Remark: Running the following code requires the installation of surprise library and the minimum run time is approximately 10-20 mins after which the output will be seen on the Jupyter Notebook. This is necessary in order to avoid any unresolved errors that are seen on log console.

Input Code:

```
data = df.to_numpy()

import surprise

from surprise import Reader

from surprise import Dataset

df_ratings = pd.read_csv("D:/PGP-DS/Machine Learning/Amazon - Movies and TV Ratings.csv")

df_melt = pd.melt(df_ratings, id_vars=["user_id"], var_name="movie", value_name="rating")

df_melt
```

Output:

```
[25]: df_melt
```

[25]:

	user_id	movie	rating
0	A3R5OBKS7OM2IR	Movie1	5.0
1	AH3QC2PC1VTGP	Movie1	NaN
2	A3LKP6WPMP9UKX	Movie1	NaN
3	AVIY68KEPQ5ZD	Movie1	NaN
4	A1CV1WROP5KTTW	Movie1	NaN
...	...	...	...
998683	A1IMQ9WMFYKWH5	Movie206	5.0
998684	A1KLIKPUF5E88I	Movie206	5.0
998685	A5HG6WFZLO10D	Movie206	5.0
998686	A3UU690TWXCG1X	Movie206	5.0
998687	AI4J762YI6S06	Movie206	5.0

998688 rows × 3 columns

Input Code: data = Dataset.load_from_df(df_melt,Reader(rating_scale=(1,10)))

Dividing the data into train - test and building a recommendation model on training data:

Input Code:

from surprise.model_selection import train_test_split

trainset, testset = train_test_split(data, test_size = .2)

from surprise import KNNWithMeans

algo = KNNWithMeans(k=4)

algo.fit(trainset)

Output:

Build a recommendation model on training data

```
[29]: from surprise import KNNWithMeans
      algo = KNNWithMeans(k=4)

      algo.fit(trainset)

      Computing the msd similarity matrix...
      Done computing similarity matrix.

[29]: <surprise.prediction_algorithms.knns.KNNWithMeans at 0x2518f91cc88>
```

Making predictions on the test data:

Input Code:

results = algo.test(testset)

results

Output: Recommendation model predicts for each user, a short image of which is shown:

```
[31]: results

[31]: [Prediction(uid='A22GZ2B2A12RZP', iid='Movie104', r_ui=nan, est=10, details={'actual_k': 0, 'was_impossible': False}),
       Prediction(uid='A2MHSG6GPW910M', iid='Movie46', r_ui=nan, est=10, details={'actual_k': 0, 'was_impossible': False}),
       Prediction(uid='A5KLP39OKXD5P', iid='Movie114', r_ui=nan, est=10, details={'actual_k': 0, 'was_impossible': False}),
       Prediction(uid='ARNOT3G5WS1ML', iid='Movie22', r_ui=nan, est=10, details={'actual_k': 0, 'was_impossible': False}),
       Prediction(uid='A29IT4Q0CBLDHK', iid='Movie19', r_ui=nan, est=10, details={'actual_k': 0, 'was_impossible': False}),
       Prediction(uid='A1OUGM8CVX8CEI', iid='Movie203', r_ui=nan, est=10, details={'actual_k': 0, 'was_impossible': False}),
       Prediction(uid='A23RA8T2NSUN4F', iid='Movie19', r_ui=nan, est=10, details={'actual_k': 0, 'was_impossible': False}),
       Prediction(uid='AVS3ODLPWSTN5', iid='Movie144', r_ui=nan, est=10, details={'actual_k': 0, 'was_impossible': False}),
       Prediction(uid='A2IGWLX4I6AANY', iid='Movie75', r_ui=nan, est=10, details={'actual_k': 0, 'was_impossible': False}),
       Prediction(uid='A18F33LA4U6DUE', iid='Movie115', r_ui=nan, est=10, details={'actual_k': 0, 'was_impossible': False}),
       Prediction(uid='A2KZYM943W5FSY', iid='Movie189', r_ui=nan, est=10, details={'actual_k': 0, 'was_impossible': False}),
       Prediction(uid='AE62TIY3899YL', iid='Movie169', r_ui=nan, est=10, details={'actual_k': 0, 'was_impossible': False}),
       Prediction(uid='A2IB9Y3Q28V62G', iid='Movie17', r_ui=nan, est=10, details={'actual_k': 0, 'was_impossible': False}),
       Prediction(uid='A3MEHHL9LNKW4Z', iid='Movie111', r_ui=nan, est=10, details={'actual_k': 0, 'was_impossible': False}),
       Prediction(uid='AV4F7AERY36IC', iid='Movie32', r_ui=5.0, est=10, details={'actual_k': 0, 'was_impossible': False}),
       Prediction(uid='A202EENCKN1198', iid='Movie64', r_ui=nan, est=10, details={'actual_k': 0, 'was_impossible': False}),
       Prediction(uid='A3B3TO1YTPOUQT', iid='Movie202', r_ui=nan, est=10, details={'actual_k': 0, 'was_impossible': False}),
       Prediction(uid='A1CA4IP0DCPF7W', iid='Movie12', r_ui=nan, est=10, details={'actual_k': 0, 'was_impossible': False}),
       Prediction(uid='A2UP24TFJJLCKT', iid='Movie31', r_ui=nan, est=10, details={'actual_k': 0, 'was_impossible': False}),
       Prediction(uid='A17IIHVVWJYZCQ', iid='Movie53', r_ui=nan, est=10, details={'actual_k': 0, 'was_impossible': False}),
       Prediction(uid='A3LI15T4YR5WD6', iid='Movie144', r_ui=nan, est=10, details={'actual_k': 0, 'was_impossible': False}),
       Prediction(uid='AB4F6UHL20U95', iid='Movie206', r_ui=nan, est=10, details={'actual_k': 0, 'was_impossible': False}),
       Prediction(uid='ANYDEWJQN2GBX', iid='Movie19', r_ui=nan, est=10, details={'actual_k': 0, 'was_impossible': False}),
       Prediction(uid='AHEBU54IYSSW9', iid='Movie48', r_ui=nan, est=10, details={'actual_k': 0, 'was_impossible': False}),
```

Input Code:

finalResult = [(x.uid, x.iid) for x in results]

finalResult

Output: The following image gives an idea of movies recommended for each user:

```
[33]: finalResult

[33]: [('A22GZ2B2A12RZP', 'Movie104'),
       ('A2MHSG6GPW910M', 'Movie46'),
       ('A5KLP39OKXD5P', 'Movie114'),
       ('ARNOT3G5WS1ML', 'Movie22'),
       ('A29IT4Q0CBLDHK', 'Movie19'),
       ('A1OUGM8CVX8CEI', 'Movie203'),
       ('A23RA8T2NSUN4F', 'Movie19'),
       ('AVS3ODLPWSTN5', 'Movie144'),
       ('A2IGWLX4I6AANY', 'Movie75'),
       ('A18F33LA4U6DUE', 'Movie115'),
       ('A2KZYM943W5FSY', 'Movie189'),
       ('AE62TIY3899YL', 'Movie169'),
       ('A2IB9Y3Q28V62G', 'Movie17'),
       ('A3MEHHL9LNKW4Z', 'Movie111'),
       ('AV4F7AERY36IC', 'Movie32'),
       ('A202EENCKN1198', 'Movie64'),
       ('A3B3TO1YTPOUQT', 'Movie202'),
       ('A1CA4IP0DCPF7W', 'Movie12'),
       ('A2UP24TFJJLCKT', 'Movie31'),
       ('A17IIHVVWJYZCQ', 'Movie53'),
       ('A3LI15T4YR5WD6', 'Movie144'),
       ('AB4F6UHL20U95', 'Movie206'),
       ('ANYDEWJQN2GBX', 'Movie19'),
       ('AHEBU54IYSSW9', 'Movie48'),
       ('ACE89I7YDRLK7', 'Movie117'),
```