Data Mining Assignment 2

Udit Kumar MT21148

Mahvash Fatima MT21126

Techniques and Assumptions for solving each question :

Ques 1.

We were asked to perform EDA on the given datasets:

Preparation of the dataset to perform EDA on.

We've been given 4 datasets:

1) **Links.csv:** it has three attributes [movieId,imdbId,tmdbId].

Exploration:

There are 8 null values in tmdbId column, no other null values present.

2) **Movies.csv:** it has [movieId , title, genre]

Both have one attribute common and using pandas merge function, we first merge these two datasets on the basis of common attribute movieId.

Exploration:

There are no NaN values in the movies dataset.

We explored the data by checking for the Nan values before and after merging both the datasets.

link_and_Movie=pd.merge(links_Data,movies_Data,on='movieId')

Link and movie combined have only 8 null values in the tmdbId column

Then moving on to the next two datasets:

3) Rating.csv:

Attributes are [userId,movieId,rating,timestamp]

Exploration:

Rating data doesn’t have any null values in any columns.

4)Tags.csv:

Attributes are [userId, movieId ,tag ,timestamp]

Exploration:

Tags data does not have any null values in any column.

To merge these two, since there are multiple common attributes, we experimented with various joins (inner, outer, right, left) and finalized outer join based on the most relevant outcome covering all the attributes and merging the common ones.

rating_and_tag=pd.merge(rating_Data,tags_Data,how="outer",on=["userId", "movieId","timestamp"])

NaN values in merged dataset

```
********outer join ******
userId            0
movieId           0
rating         3683
timestamp         0
tag          100836
dtype: int64
```

further, we merged all four (2 + 2) to get the final_Merge data.

P.S: We explored all the datasets individually to know the count of null values in each attribute before and after the merge.

So now we’ve prepared our final dataset to perform Exploratory data analysis on:

1.) **Null values in the final merged dataset (NaN values in data):**

```
********final merge ******
movieId           0
imdbId            0
tmdbId           13
title             0
genres            0
userId            0
rating         3683
```

```
timestamp          0
tag           100836
```

To fix the NaN values for rating , we've populated them with mean(of that particular movie's ratings) if movie has already been rated by another user.

And we've eliminated the tmdbId in our final dataframe, so now we do not have any null values in our data.

2.) Categorical attributes and numerical attributes

```
eda_data=final_Merge.copy()
categorical_Features=[feature for feature in final_Merge.columns if eda_data[feature].dtypes=='O']
print("categorical features : ",categorical_Features)
```

```
categorical features :  ['title', 'genres', 'tag']
```

Note: We created a copy of the data in the first place so that any accidental/intentional changes do not reflect on our original data.

```
numerical_Features=[feature for feature in final_Merge.columns if eda_data[feature].dtypes!='O']
print("numerical features : ",numerical_Features)
```

```
numerical features :  ['movieId', 'imdbId', 'tmdbId', 'userId', 'rating', 'timestamp']
```

3.) Count of unique values for each categorical attribute.

```
In [33]: #counting the unique features of categorical data
for feature in categorical_Features:
    print('The feature is {} and number of categories are {}'.format(feature,len(eda_data[feature].unique())))
```

```
The feature is title and number of categories are 9737
The feature is genres and number of categories are 951
The feature is tag and number of categories are 1590
```

4.) Counting unique values for each attribute

```
[40]: # counting unique values in each column
unique_Values=final_Merge.nunique()
print(unique_Values)
```

```
movieId        9742
imdbId         9742
tmdbId         9733
title          9737
genres          951
userId          610
rating           10
timestamp     88453
tag            1589
dtype: int64
```

5.) Basic statistical details of our new merged data using describe function in pandas.

```
#describing the data
final_Merge.describe(include='all')
```

	movieId	imdbId	tmdbId	title	genres	userId	rating	timestamp	tag
count	104519.000000	1.045190e+05	104506.000000	104519	104519	104519.000000	100836.000000	1.045190e+05	3683
unique	NaN	NaN	NaN	9737	951	NaN	NaN	NaN	1589
top	NaN	NaN	NaN	Pulp Fiction (1994)	Comedy	NaN	NaN	NaN	In Netflix queue
freq	NaN	NaN	NaN	488	7359	NaN	NaN	NaN	131
mean	19710.738191	3.559796e+05	20543.697395	NaN	NaN	329.828280	3.501557	1.209966e+09	NaN
std	35870.238985	6.297058e+05	54157.108637	NaN	NaN	182.849716	1.042529	2.158859e+08	NaN
min	1.000000	4.170000e+02	2.000000	NaN	NaN	1.000000	0.500000	8.281246e+08	NaN
25%	1200.000000	9.965300e+04	710.000000	NaN	NaN	177.000000	3.000000	1.026225e+09	NaN
50%	3022.000000	1.187990e+05	6964.000000	NaN	NaN	333.000000	3.500000	1.186163e+09	NaN
75%	8361.000000	3.172190e+05	11704.000000	NaN	NaN	477.000000	4.000000	1.439317e+09	NaN
max	193609.000000	8.391976e+06	525662.000000	NaN	NaN	610.000000	5.000000	1.537799e+09	NaN

6.) Most Frequently occurring values in Categorical attributes.

For all 3 categorical attributes:

For these, we have ordered the value count of all movies in descending order and printed the top 10 for each categorical attribute.

Top 10 most frequent in title category

```
Pulp Fiction (1994)                           488
Forrest Gump (1994)                           338
Shawshank Redemption, The (1994)              321
Silence of the Lambs, The (1991)              285
Matrix, The (1999)                            283
Star Wars: Episode IV - A New Hope (1977)     277
Fight Club (1999)                             272
Braveheart (1995)                             247
Jurassic Park (1993)                          239
Terminator 2: Judgment Day (1991)             232
Name: title, dtype: int64
```

Top 10 most frequent in genres category

```
Comedy                                  7359
Drama                                   6649
Comedy|Romance                          4073
Comedy|Drama|Romance                    3105
Drama|Romance                           2968
Comedy|Drama                            2931
Action|Adventure|Sci-Fi                 2467
Crime|Drama                             2386
Action|Crime|Thriller                   1585
Action|Adventure|Sci-Fi|Thriller        1473
Name: genres, dtype: int64
```

Top 10 most frequent in tag category

```
In Netflix queue     131
atmospheric           36
thought-provoking     24
superhero             24
funny                 23
Disney                23
surreal               23
religion              22
psychology            21
sci-fi                21
```

Counting of NaN values per column

```
movieId          0
imdbId           0
tmdbId          13
title            0
genres           0
userId           0
rating        3683
timestamp        0
tag         100836
```

Unique values in each column

```
unique_Values=final_Merge.nunique()
print(unique_Values)
```

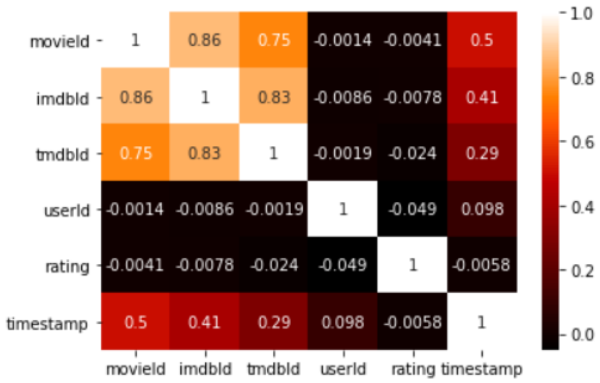
```
movieId       9742
imdbId        9742
tmdbId        9733
title         9737
genres         951
```

```
userId          610
rating           10
timestamp     88453
tag            1589
dtype: int64
```

Plotting the Correlation between different features

We’ve plotted the correlation matrix and beautifully presented it through a heatmap to show the correlation between all the features.

	movieId	imdbId	tmdbId	userId	rating	timestamp
movieId	1.000000	0.860234	0.746341	-0.001438	-0.004061	0.500822
imdbId	0.860234	1.000000	0.834988	-0.008595	-0.007806	0.410893
tmdbId	0.746341	0.834988	1.000000	-0.001941	-0.023532	0.293208
userId	-0.001438	-0.008595	-0.001941	1.000000	-0.049348	0.097758
rating	-0.004061	-0.007806	-0.023532	-0.049348	1.000000	-0.005802
timestamp	0.500822	0.410893	0.293208	0.097758	-0.005802	1.000000



From the given correlation matrix, we can infer that :

a) MovieId and ImdbId have a strong relation (0.86).
(Therefore, we decided to go with one of these only i.e. MovieId)

b) MovieId and tmdbId have a strong relation (0.75).
(Therefore, we decided to go with one of these only i.e. MovieId)

Thus, we have removed these 2 columns in our final dataframe.

Ques 2.

For this question , we were supposed to apply association rule mining to generate 4 recommendations for a given customer profile. (list of movies taken as input from th e user)

We have implemented this using the apriori algorithm, this is possible through the mlxtend.frequent_patterns module, by importing apriori, association_rules.

To prepare the data for this, we created a pivot table with userId, title and rating.

And further used a binarizing function(encoded) to populate the movie columns with rating >3.5 as 1 and below that with 0. **(assumption as preferred for precision@K**)

The recomm_rules variables generates the ruleset for the prepared dataset using the function association_rules in the mlxtend module.

Further, we sorted these rules in the descending order of lift, confidence .

Now our dataframe containing the rules is ready,

We just need to extract the right rules by matching the antecedents with the user input which we've implemented with a function movieinput() for user input for n movies, where n can be 0,1,2…n.

The results are returned by matching the input with the antecedents first one by one and finally giving 4 recommendations as consequents(based on the movies that cover all the genres liked by the user) .

Ques 3 :

For Question 3 we have used the TransactionEncoder and fpgrowth

With the logic we have made an array called movie_wrt_id , which cantains the all movies of user 1 at index 1 , user 2 movies at index 2 … and so on.

All final movies w.r.t is stored in final_Movies

We did the transaction encoding on the final_Movies and then made prep_data which is used as the input for our fpgrowth function, Frequent Itemset has the frequent itemset which has a minimum support of 0.068 and the support column for corresponding support value.

Then we store the all the values of itemsets of <= to a particular support in less than or equal ,

Then we are evaluating the max_Freq_set holds our frequent itemset wrt to a N number in the code, If you give N=35 , this code will give you only 35 max frequent itemset.

For visualization we have used the matplot lib bar graph..

Visualization :

Screenshot 1 :

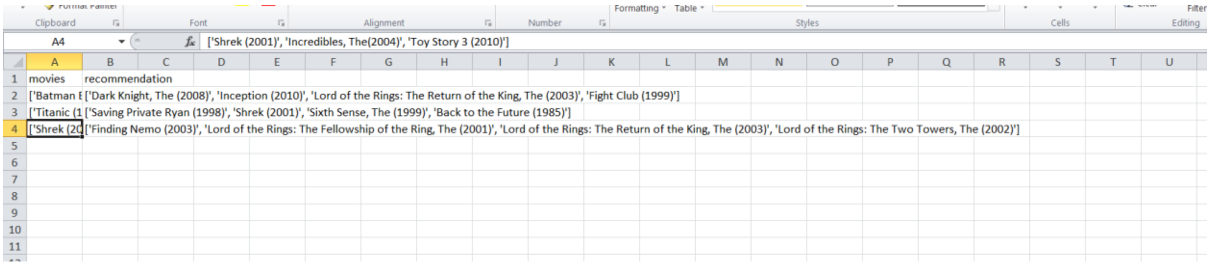
Screenshot of the 4 movies recommended for 3 user

```
User 1 input is :
Batman Begins (2005)
Dark Knight Rises, The (2012)

User 1  recommended movies are :
Dark Knight, The (2008)
Inception (2010)
Lord of the Rings: The Return of the King, The (2003)
Fight Club (1999)

User 2 input is :
Titanic (1997)

User 2  recommended movies are :
Saving Private Ryan (1998)
Shrek (2001)
Sixth Sense, The (1999)
Back to the Future (1985)

User 3 input is :
Shrek (2001)
Incredibles, The(2004)
Toy Story 3 (2010)

User 3  recommended movies are :
Finding Nemo (2003)
Lord of the Rings: The Fellowship of the Ring, The (2001)
Lord of the Rings: The Return of the King, The (2003)
Lord of the Rings: The Two Towers, The (2002)
```

Screenshot 2:

Final csv of result generated in Q2

Output.csv

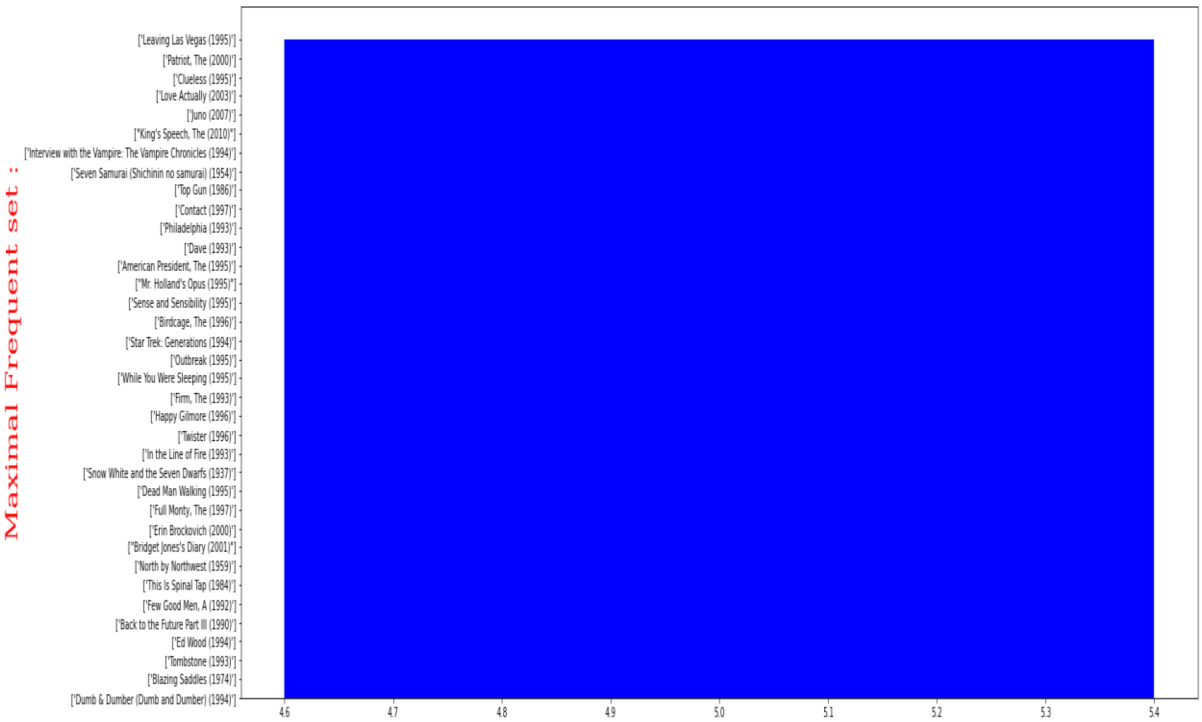


	A	B
1	movies	recommendation
2	['Batman E	['Dark Knight, The (2008)', 'Inception (2010)', 'Lord of the Rings: The Return of the King, The (2003)', 'Fight Club (1999)']
3	['Titanic (1	['Saving Private Ryan (1998)', 'Shrek (2001)', 'Sixth Sense, The (1999)', 'Back to the Future (1985)']
4	['Shrek (20	['Finding Nemo (2003)', 'Lord of the Rings: The Fellowship of the Ring, The (2001)', 'Lord of the Rings: The Return of the King, The (2003)', 'Lord of the Rings: The Two Towers, The (2002)']

Screenshot 3:

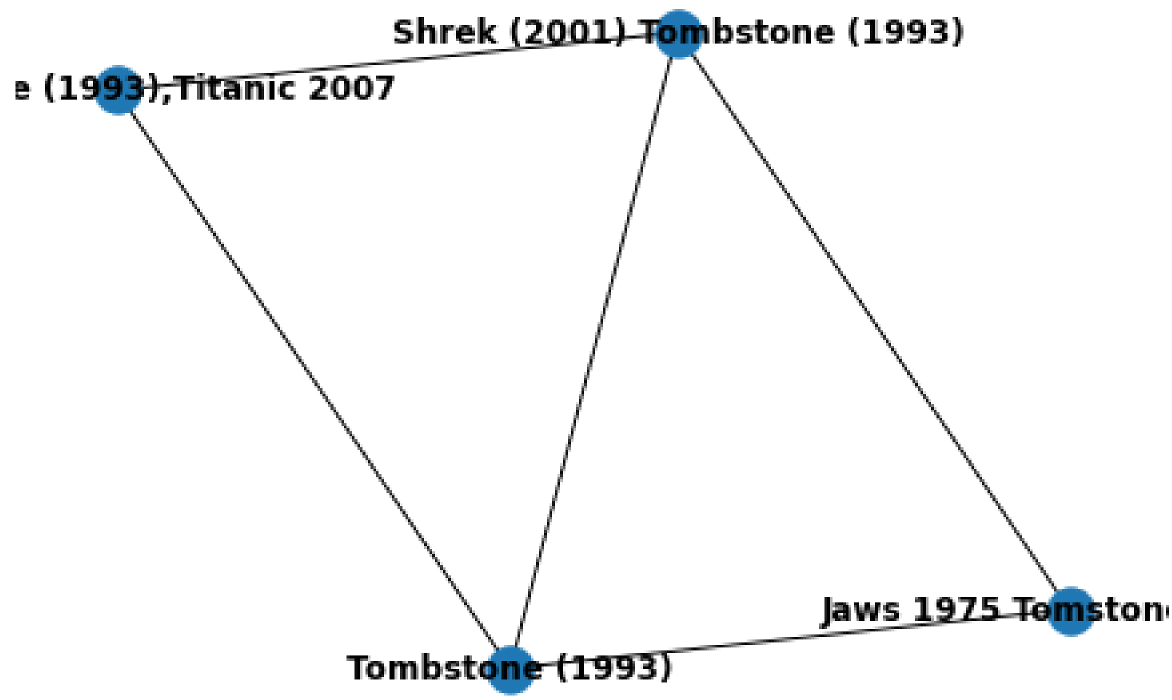
Maximal frequent set of N length generated

Below N=35



Screenshot 4 : small example of Maximal frequent set ,

Using networkx python library :



Here is Tombstone is the Maximal frequent Itemset among these as there are other nodes which are superset of Tombstone (1993) but they are not Maximal frequent itemset their support are less than Tombstone (1993).

Learnings:

1.How to perform data analysis on the given data,

Learn how to use different kind of joins method and merge two data frames.

2.Learn about the .describe () and .info() method in pandas.

3.How to find the most frequent values by value_counts(),

How to count the NAN and unique values in a data frame.

4.How to do the preprocessing of the data if Nan values are present In the data frame.

5.How to plot the correlation between different columns of a data frame.

6.How to use the .pivot function and how to do make your own self encoder

7.We learned about the Apriori algorithm and how to generate the frequent Itemset, How to make your own rules from the given data,

How to make a Movie Recommendation System using Apriori algorithm.

8.How to make use of Fpgrowth function to generate the frequent Itemset ,

How to find the Maximal Frequent Itemset ,

Logically, How to do the visualization part .

9.For visualization we have used the networkx python library , we have learned various approaches to make a complex graph.

References :

1.Recommended System :

https://www.youtube.com/watch?v=EjOlN6uVBOg

2.Apriori Algorithm for recommendation system :

https://medium.datadriveninvestor.com/recommendation-system-using-association-rule-mining-for-implicit-data-6fba0f6c5012

3.Most frequent values :

https://datascientyst.com/get-most-frequent-values-pandas-dataframe/

4.merge concatenate :

https://realpython.com/pandas-merge-join-and-concat/

5.merge join 2 :

https://blockgeni.com/guide-to-merge-and-join-dataframes-with-pandas/

6.Recommended System using python :

https://www.youtube.com/watch?v=R64Lh1Qwl_0&list=LL&index=2

7.Recommended System using Knn :

https://www.youtube.com/watch?v=kccT0FVK6OY

8.Heat map :

https://matplotlib.org/stable/tutorials/colors/colormaps.html

9.Apririo algo :

https://harshavardhan198.medium.com/market-basket-analysis-in-python-using-apriori-algorithm-to-recommend-movies-bb575019aa57

10.you tube karan examples :

https://www.youtube.com/watch?v=EFXeiD-jZrQ

https://www.youtube.com/watch?v=Bo8auDKeujM

11.For ques 3 :

https://towardsdatascience.com/how-to-find-closed-and-maximal-frequent-itemsets-from-fp-growth-861a1ef13e21