association_rules

November 25, 2024

1 Imports

[1]:
```
### IMPORTS ###

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from tqdm import tqdm
```

2 Data Preprocessing

[2]:
```
# load movie data
movies = pd.read_pickle('data/imdb/ml_movies_description.pkl')

movies.head()
```

[2]:
```
   movieId                title                                       genres  \
0        1     Toy Story (1995)  Adventure|Animation|Children|Comedy|Fantasy
1        2       Jumanji (1995)                   Adventure|Children|Fantasy
2        6          Heat (1995)                        Action|Crime|Thriller
3        8  Tom and Huck (1995)                           Adventure|Children
4        9  Sudden Death (1995)                                       Action

   imdbId  tmdbId      id                                        description
0  114709    862.0  114709  A cowboy doll is profoundly threatened and jea…
1  113497   8844.0  113497  When two kids find and play a magical board ga…
2  113277    949.0  113277  A group of high-end professional thieves start…
3  112302  45325.0  112302  Two best friends witness a murder and embark o…
4  114576   9091.0  114576  A former fireman takes on a group of terrorist…
```

[3]:
```
# get a list of unique movie ids
movie_ids_imdb = movies['movieId'].unique()

# print the number of unique movie ids
print('Number of unique movie ids: ', len(movie_ids_imdb))
```

```
Number of unique movie ids:  51860
```

[4]:
```python
# load ratings
ratings = pd.read_csv('data/ml-32m/ratings.csv')

ratings.head()
```

[4]:

	userId	movieId	rating	timestamp
0	1	17	4.0	944249077
1	1	25	1.0	944250228
2	1	29	2.0	943230976
3	1	30	5.0	944249077
4	1	32	5.0	943228858

[5]:
```python
# get a list of unique user ids
movie_ids_ml = ratings['movieId'].unique()

# print the number of unique user ids
print('Number of unique user ids: ', len(movie_ids_ml))
```

```
Number of unique user ids:  84432
```

[6]:
```python
### DATA PREPROCESSING ###

# get the intersection of movie ids
movie_ids = np.intersect1d(movie_ids_imdb, movie_ids_ml)
print('Number of common movie ids: ', len(movie_ids))

# filter the ratings data to contain only the common movie ids
ratings = ratings[ratings['movieId'].isin(movie_ids)]

# drop timestamp column
ratings = ratings.drop('timestamp', axis=1)

# print the number of ratings
print('Number of ratings: ', len(ratings))

# get the number of unique users
user_ids = ratings['userId'].unique()

# print the number of unique users
print('Number of unique users: ', len(user_ids))
```

```
Number of common movie ids:  49892
Number of ratings:  25723135
Number of unique users:  200947
```

[7]:
```python
# remove alle ratings for user 304, 6741 and 147001 for testing purposes
ratings = ratings[~ratings['userId'].isin([304, 6741, 147001])]
```

[8]:
```
### CREATE LIKED MOVIES DATAFRAME ###
# create a dataframe to only store ratings of 4 or 5
liked_movies = ratings[ratings['rating'] >= 4]

# print the number of liked movies
print('Number of ratings 4 or 5: ', len(liked_movies))

# print the number of unique users who liked movies
print('Number of unique users who liked movies: ', len(liked_movies['userId'].
 ↪unique()))

# print the number of unique movies that were liked
print('Number of unique movies that were liked: ', len(liked_movies['movieId'].
 ↪unique()))

liked_movies.head()
```

```
Number of ratings 4 or 5:  12781764
Number of unique users who liked movies:  200609
Number of unique movies that were liked:  33491
```

[8]:

	userId	movieId	rating
3	1	30	5.0
4	1	32	5.0
9	1	111	5.0
11	1	166	5.0
15	1	260	5.0

3 Exhaustive approach (not used)

[9]:
```
### Create basket data ###
# create a basket data
baskets = liked_movies.groupby('userId')['movieId'].apply(list).
 ↪reset_index(name='basket')
items = np.unique(np.concatenate(baskets['basket']))
```

[10]:
```
len(baskets)
```

[10]: 200609

[11]:
```
# create a a DataFrame to hash items to integers
#df_item_hash = pd.DataFrame({'item': items, 'hashcode': range(len(items))}).
 ↪set_index('item')
df_item_hash = pd.DataFrame(range(len(items)), index=items,␣
 ↪columns=['hashcode'])

df_item_hash.head()
```

[11]:
```
   hashcode
1         0
2         1
6         2
8         3
9         4
```

[12]:
```
# Count the occurrences of each item and store in an array
### SLOW ###
#item_count_arr = np.zeros((len(items), 1))
#for basket in tqdm(baskets['basket']):
#    for item in basket:
#        idx = df_item_hash.loc[item, 'hashcode']
#        item_count_arr[idx] += 1

# Count the occurrences of each item and store in an array
item_count_arr = np.zeros((len(items), 1))

# Flatten the list of baskets and get the corresponding hashcodes
flattened_baskets = np.concatenate(baskets['basket'].values)
hashcodes = df_item_hash.loc[flattened_baskets, 'hashcode'].values

# Use np.bincount to count occurrences of each hashcode
item_count_arr[:len(np.bincount(hashcodes))] = np.bincount(hashcodes).
 ↪reshape(-1, 1)
```

[13]:
```
# find frequent items (items that appear in more than 0.5% of the baskets)
freq_items = np.array([df_item_hash[df_item_hash['hashcode'] == x].index[0] for␣
 ↪x in tqdm(np.where(item_count_arr > 0.1 * len(baskets))[0])])
freq_items
```

```
100%|      | 100/100 [00:00<00:00, 14229.07it/s]
```

[13]:
```
array([     1,     32,     47,     50,    110,    111,    150,    260,
          293,    296,    364,    377,    380,    457,    480,    527,
          541,    588,    589,    590,    593,    595,    608,    733,
          750,    780,    858,    904,    912,    924,   1036,   1089,
         1097,   1136,   1196,   1197,   1198,   1200,   1206,   1208,
         1210,   1213,   1214,   1221,   1222,   1240,   1258,   1265,
         1270,   1291,   1527,   1580,   1617,   1732,   2028,   2324,
         2329,   2571,   2716,   2762,   2997,   3114,   3147,   3578,
         3996,   4011,   4226,   4306,   4878,   4886,   4963,   4993,
         4995,   5418,   5445,   5618,   5952,   5989,   6016,   6377,
         6539,   6874,   7153,   7361,   7438,   8961,  33794,  44191,
        48516,  48780,  58559,  59315,  60069,  68157,  68954,  74458,
        79132,  91529,  99114, 109487])
```

[14]:
```
### THIS IS SLOW ###

### hash the frequent items (starting from 1)
#df_freq_item_hash = pd.DataFrame(range(1,len(freq_items)+1), index=freq_items,␣
↪columns=['hashcode'])

# create a matrix to store the pair counts
#pair_mat_hashed = np.zeros((len(freq_items)+1, len(freq_items)+1))

#for b in tqdm(baskets['basket']):
#    cand_list = [item for item in b if item in freq_items]
#    if len(cand_list)<2:
#        continue
#    for idx, item1 in enumerate(cand_list):
#        for item2 in cand_list[idx+1:]:
#            i = df_freq_item_hash.loc[item1,'hashcode']
#            j = df_freq_item_hash.loc[item2,'hashcode']
#            pair_mat_hashed[max(i,j),min(i,j)]+=1

# pair_mat
#pair_mat_hashed
```

[15]:
```
# Create a matrix to store the pair counts
pair_mat_hashed = np.zeros((len(freq_items)+1, len(freq_items)+1))

# Create a dictionary for quick lookup of hashcodes
df_freq_item_hash = pd.DataFrame(range(1,len(freq_items)+1), index=freq_items,␣
↪columns=['hashcode'])

hashcode_dict = df_freq_item_hash['hashcode'].to_dict()

for b in tqdm(baskets['basket']):
    # Filter items in the basket to only those in hashcode_dict
    cand_list = [hashcode_dict[item] for item in b if item in hashcode_dict]
    if len(cand_list) < 2:
        continue

    # Convert cand_list to a numpy array
    cand_list = np.array(cand_list)

    # Get unique pairs of indices
    i_indices = np.maximum.outer(cand_list, cand_list)
    j_indices = np.minimum.outer(cand_list, cand_list)

    # Increment only unique pairs (i, j) where i > j
    unique_pairs = np.triu_indices_from(i_indices, k=1)
    pair_mat_hashed[i_indices[unique_pairs], j_indices[unique_pairs]] += 1
```

```
# Display the pair matrix
pair_mat_hashed
```

100%| | 200609/200609 [00:03<00:00, 55619.26it/s]

[15]:
```
array([[    0.,     0.,     0., …,     0.,     0.,     0.],
       [    0.,     0.,     0., …,     0.,     0.,     0.],
       [    0., 13780.,     0., …,     0.,     0.,     0.],
       …,
       [    0.,  6572.,  4285., …,     0.,     0.,     0.],
       [    0.,  6391.,  5096., …, 10369.,     0.,     0.],
       [    0.,  6995.,  5332., …, 11635., 12167.,     0.]])
```

[16]:
```
# Get frequent pairs (pairs that appear in more than 0.05% of the baskets -␣
↪support = 0.0005)
# Extract frequent pairs that exceed support s2 (assume s2 = 0.02), and hash␣
↪back.

# Find indices where pair counts exceed the threshold
pair_indices = np.where(pair_mat_hashed > 0.1 * len(baskets))

# Extract frequent pairs and hash back
freq_pairs = np.array([
    [df_freq_item_hash.index[x-1], df_freq_item_hash.index[y-1]]
    for x, y in zip(pair_indices[0], pair_indices[1])
])

print('Number of frequent pairs: ', len(freq_pairs))

# make this into a list of tuples
freq_pairs = [tuple(x) for x in freq_pairs]
```

Number of frequent pairs: 216

[17]:
```
# calculate the support of each frequent pair
pair_support = pair_mat_hashed[pair_indices] / len(baskets)

# also calculate the confidence of each frequent pair (confidence =␣
↪support(pair) / support(item1))
```

[18]:
```
# create a DataFrame to store the frequent pairs and their support
df_freq_pairs = pd.DataFrame(freq_pairs, columns=['item1', 'item2'])
df_freq_pairs['support'] = pair_support.flatten()

# make one column called itemsets that contains the frequent pairs as tuples␣
↪(frozen sets)
```

```python
df_freq_pairs['itemsets'] = df_freq_pairs[['item1', 'item2']].apply(lambda x:␣
↪frozenset(x), axis=1)

# drop the item1 and item2 columns
df_freq_pairs = df_freq_pairs.drop(['item1', 'item2'], axis=1)

df_freq_pairs.head()
```

[18]:

	support	itemsets
0	0.135672	(50, 47)
1	0.103016	(110, 47)
2	0.104726	(50, 110)
3	0.120329	(1, 260)
4	0.101685	(260, 47)

4 Association Rule Mining w. A-priori Algorithm

[19]:
```python
### ASSOCIATION RULE MINING WITH APRIORI ###
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules
```

4.1 Find Transactions

[20]:
```python
df_subset = liked_movies
transactions = df_subset.groupby('userId')['movieId'].apply(list).values

# Use TransactionEncoder to convert the data into a binary matrix
te = TransactionEncoder()
te_ary = te.fit(transactions).transform(transactions)
binary_df = pd.DataFrame(te_ary, columns=te.columns_)
```

4.2 Find Frequent Itemsets

4.2.1 Using Minimum Support Threshold of 0.2

[21]:
```python
# frequent itemsets
min_support = 0.2
frequent_itemsets_2 = apriori(binary_df, min_support=min_support,␣
↪use_colnames=True)
print("Frequent Itemsets (with more than one element):")
print(frequent_itemsets_2)
```

```
Frequent Itemsets (with more than one element):
    support   itemsets
0   0.226894       (1)
1   0.237975      (47)
2   0.276025      (50)
```

```
3    0.239411         (110)
4    0.318321         (260)
5    0.385706         (296)
6    0.208525         (480)
7    0.295635         (527)
8    0.235533         (589)
9    0.351694         (593)
10   0.220598         (608)
11   0.272894         (858)
12   0.275681        (1196)
13   0.253284        (1198)
14   0.238334        (1210)
15   0.211356        (1270)
16   0.211561        (2028)
17   0.358648        (2571)
18   0.205679        (4226)
19   0.272126        (4993)
20   0.248533        (5952)
21   0.250183        (7153)
22   0.230089       (58559)
23   0.220204       (79132)
24   0.229815   (1196, 260)
25   0.200156   (1210, 260)
26   0.222119    (296, 593)
27   0.219970  (5952, 4993)
28   0.218071  (4993, 7153)
29   0.212493  (5952, 7153)
```

[22]:
```
frequent_itemsets_2['length'] = frequent_itemsets_2['itemsets'].apply(lambda x:␣
↪len(x))
len(frequent_itemsets_2)
```

[22]: 30

Generating Association Rules

[23]:
```
# Generate association rules
rules_apriori_2 = association_rules(frequent_itemsets_2, metric="confidence",␣
↪min_threshold=0.5, num_itemsets=len(transactions))
```

[24]:
```
rules_apriori_2
```

[24]:
```
  antecedents consequents  antecedent support  consequent support   support  \
0      (1196)       (260)            0.275681            0.318321  0.229815
1       (260)      (1196)            0.318321            0.275681  0.229815
2      (1210)       (260)            0.238334            0.318321  0.200156
3       (260)      (1210)            0.318321            0.238334  0.200156
4       (296)       (593)            0.385706            0.351694  0.222119
```

```
5        (593)       (296)            0.351694            0.385706  0.222119
6       (5952)      (4993)            0.248533            0.272126  0.219970
7       (4993)      (5952)            0.272126            0.248533  0.219970
8       (4993)      (7153)            0.272126            0.250183  0.218071
9       (7153)      (4993)            0.250183            0.272126  0.218071
10      (5952)      (7153)            0.248533            0.250183  0.212493
11      (7153)      (5952)            0.250183            0.248533  0.212493

    confidence      lift  representativity  leverage  conviction  \
0     0.833629  2.618833               1.0  0.142060    4.097336
1     0.721961  2.618833               1.0  0.142060    2.605102
2     0.839810  2.638251               1.0  0.124289    4.255445
3     0.628786  2.638251               1.0  0.124289    2.051822
4     0.575876  1.637435               1.0  0.086468    1.528577
5     0.631568  1.637435               1.0  0.086468    1.667320
6     0.885074  3.252436               1.0  0.152338    6.333390
7     0.808338  3.252436               1.0  0.152338    3.920799
8     0.801359  3.203090               1.0  0.149990    3.774737
9     0.871645  3.203090               1.0  0.149990    5.670793
10    0.854988  3.417448               1.0  0.150314    5.170728
11    0.849349  3.417448               1.0  0.150314    4.988145

    zhangs_metric   jaccard  certainty  kulczynski
0        0.853422  0.631038   0.755939    0.777795
1        0.906805  0.631038   0.616138    0.777795
2        0.815267  0.561447   0.765007    0.734298
3        0.910928  0.561447   0.512628    0.734298
4        0.633717  0.431063   0.345797    0.603722
5        0.600471  0.431063   0.400235    0.603722
6        0.921582  0.731553   0.842107    0.846706
7        0.951454  0.731553   0.744950    0.846706
8        0.944946  0.716776   0.735081    0.836502
9        0.917293  0.716776   0.823658    0.836502
10       0.941338  0.742402   0.806604    0.852169
11       0.943409  0.742402   0.799525    0.852169
```

[25]:
```python
# find all unique movie ids in antecedents and consequents
unique_ante = np.unique(np.concatenate(rules_apriori_2['antecedents'].
 ↪apply(list)))
unique_cons = np.unique(np.concatenate(rules_apriori_2['consequents'].
 ↪apply(list)))

#print(len(unique_ante), len(unique_cons))

# find the union of unique movie ids in antecedents and consequents
unique_movies = np.unique(np.concatenate([unique_ante, unique_cons]))
```

```
# print the unique movie ids and names
print(unique_movies)
print(movies[movies['movieId'].isin(unique_movies)][['movieId', 'title']])
```

```
[ 260  296  593 1196 1210 4993 5952 7153]
      movieId                                              title
163       260          Star Wars: Episode IV - A New Hope (1977)
182       296                                Pulp Fiction (1994)
373       593                   Silence of the Lambs, The (1991)
743      1196  Star Wars: Episode V - The Empire Strikes Back…
755      1210  Star Wars: Episode VI - Return of the Jedi (1983)
3366     4993  Lord of the Rings: The Fellowship of the Ring,…
4031     5952      Lord of the Rings: The Two Towers, The (2002)
4831     7153  Lord of the Rings: The Return of the King, The…
```

4.2.2 Using Minimum Support Threshold of 0.15

[26]:
```
# frequent itemsets
min_support = 0.15
frequent_itemsets_15 = apriori(binary_df, min_support=min_support,␣
 ↪use_colnames=True)
print("Frequent Itemsets (with more than one element):")
print(frequent_itemsets_15)
```

```
Frequent Itemsets (with more than one element):
     support             itemsets
0   0.226894                  (1)
1   0.184767                 (32)
2   0.237975                 (47)
3   0.276025                 (50)
4   0.239411                (110)
..       …                    …
73  0.172988    (1210, 260, 1196)
74  0.155427    (1196, 2571, 260)
75  0.153612   (5952, 4993, 2571)
76  0.152595   (7153, 4993, 2571)
77  0.198810   (5952, 4993, 7153)

[78 rows x 2 columns]
```

[27]:
```
frequent_itemsets_15['length'] = frequent_itemsets_15['itemsets'].apply(lambda␣
 ↪x: len(x))
len(frequent_itemsets_15)
```

[27]: 78

Generating association rules

[28]:
```
# Generate association rules
rules_apriori_15 = association_rules(frequent_itemsets_15, metric="confidence",␣
↪min_threshold=0.5, num_itemsets=len(transactions))
```

[29]:
```
rules_apriori_15
```

[29]:
```
      antecedents    consequents  antecedent support  consequent support  \
0            (47)          (296)            0.237975            0.385706
1            (47)          (593)            0.237975            0.351694
2            (50)          (296)            0.276025            0.385706
3            (50)          (593)            0.276025            0.351694
4          (1196)          (260)            0.275681            0.318321
..            …              …                 …                   …
63  (5952, 7153)         (4993)            0.212493            0.272126
64  (4993, 7153)         (5952)            0.218071            0.248533
65        (5952)   (4993, 7153)            0.248533            0.218071
66        (4993)   (5952, 7153)            0.272126            0.212493
67        (7153)   (5952, 4993)            0.250183            0.219970

     support  confidence      lift  representativity  leverage  conviction  \
0   0.169589    0.712631  1.847604               1.0  0.077800    2.137650
1   0.157814    0.663155  1.885601               1.0  0.074120    1.924639
2   0.188102    0.681469  1.766812               1.0  0.081638    1.928525
3   0.165182    0.598432  1.701571               1.0  0.068106    1.614438
4   0.229815    0.833629  2.618833               1.0  0.142060    4.097336
..       …           …         …                 …         …           …
63  0.198810    0.935606  3.438129               1.0  0.140985   11.303387
64  0.198810    0.911674  3.668218               1.0  0.144612    8.507872
65  0.198810    0.799932  3.668218               1.0  0.144612    3.908313
66  0.198810    0.730578  3.438129               1.0  0.140985    2.922953
67  0.198810    0.794656  3.612563               1.0  0.143777    3.798653

    zhangs_metric   jaccard  certainty  kulczynski
0        0.602026  0.373467   0.532197    0.576158
1        0.616338  0.365434   0.480422    0.555941
2        0.599480  0.397152   0.481469    0.584576
3        0.569505  0.357122   0.380589    0.534054
4        0.853422  0.631038   0.755939    0.777795
..            …         …          …           …
63       0.900492  0.695601   0.911531    0.833092
64       0.930248  0.742396   0.882462    0.855803
65       0.967958  0.742396   0.744135    0.855803
66       0.974268  0.695601   0.657880    0.833092
67       0.964487  0.732685   0.736749    0.849229

[68 rows x 14 columns]
```

[30]:
```python
# find all unique movie ids in antecedents and consequents
unique_ante = np.unique(np.concatenate(rules_apriori_15['antecedents'].
    ↪apply(list)))
unique_cons = np.unique(np.concatenate(rules_apriori_15['consequents'].
    ↪apply(list)))

#print(len(unique_ante), len(unique_cons))

# find the union of unique movie ids in antecedents and consequents
unique_movies = np.unique(np.concatenate([unique_ante, unique_cons]))

# print the unique movie ids and names
print(unique_movies, len(unique_movies))
print(movies[movies['movieId'].isin(unique_movies)][['movieId', 'title']])
```

```
[  47   50  260  296  527  593  608  858 1196 1198 1210 1221 2571 4993
 5952 7153] 16
      movieId                                              title
31         47                           Seven (a.k.a. Se7en) (1995)
33         50                             Usual Suspects, The (1995)
163       260              Star Wars: Episode IV - A New Hope (1977)
182       296                                    Pulp Fiction (1994)
334       527                                Schindler's List (1993)
373       593                       Silence of the Lambs, The (1991)
382       608                                          Fargo (1996)
528       858                                  Godfather, The (1972)
743      1196  Star Wars: Episode V - The Empire Strikes Back…
745      1198  Raiders of the Lost Ark (Indiana Jones and the…
755      1210  Star Wars: Episode VI - Return of the Jedi (1983)
766      1221                         Godfather: Part II, The (1974)
1656     2571                                     Matrix, The (1999)
3366     4993  Lord of the Rings: The Fellowship of the Ring,…
4031     5952      Lord of the Rings: The Two Towers, The (2002)
4831     7153  Lord of the Rings: The Return of the King, The…
```

4.2.3 Using Minimum Support Threshold of 0.1

[31]:
```python
# frequent itemsets
min_support = 0.1
frequent_itemsets_1 = apriori(binary_df, min_support=min_support,␣
    ↪use_colnames=True)
print("Frequent Itemsets (with more than one element):")
print(frequent_itemsets_1)
```

```
Frequent Itemsets (with more than one element):
     support                     itemsets
0   0.226894                          (1)
1   0.184767                         (32)
```

```
2    0.237975                      (47)
3    0.276025                      (50)
4    0.239411                     (110)
..        ...                       ...
394  0.114147    (5952, 4993, 260, 7153)
395  0.103804    (296, 4993, 5952, 7153)
396  0.110778   (5952, 4993, 1196, 7153)
397  0.140617   (5952, 7153, 4993, 2571)
398  0.105962  (5952, 4993, 7153, 58559)

[399 rows x 2 columns]
```

[32]:
```python
frequent_itemsets_1['length'] = frequent_itemsets_1['itemsets'].apply(lambda x:
    len(x))
len(frequent_itemsets_1)
```

[32]: 399

Generating association rules

[33]:
```python
# Generate association rules
rules_apriori_1 = association_rules(frequent_itemsets_1, metric="confidence",
    min_threshold=0.5, num_itemsets=len(transactions))
```

[56]:
```python
print(len(rules_apriori_1))

rules_apriori_1.head()
```

507

[56]:
```
  antecedents consequents  antecedent support  consequent support   support  \
0         (1)       (260)            0.226894            0.318321  0.120329
1        (32)       (296)            0.184767            0.385706  0.122726
2        (32)       (593)            0.184767            0.351694  0.104985
3        (47)        (50)            0.237975            0.276025  0.135672
4        (47)       (296)            0.237975            0.385706  0.169589

   confidence      lift  representativity  leverage  conviction  \
0    0.530329  1.666022               1.0  0.048104    1.451398
1    0.664221  1.722092               1.0  0.051460    1.829458
2    0.568203  1.615616               1.0  0.040004    1.501412
3    0.570109  2.065429               1.0  0.069985    1.684091
4    0.712631  1.847604               1.0  0.077800    2.137650

   zhangs_metric   jaccard  certainty  kulczynski
0       0.517093  0.283202   0.311009    0.454170
1       0.514345  0.274098   0.453390    0.491204
2       0.467402  0.243317   0.333961    0.433358
3       0.676932  0.358609   0.406208    0.530815
```

```
4       0.602026  0.373467   0.532197    0.576158
```

[35]:
```
# find all unique movie ids in antecedents and consequents
unique_ante = np.unique(np.concatenate(rules_apriori_1['antecedents'].
 ↪apply(list)))
unique_cons = np.unique(np.concatenate(rules_apriori_1['consequents'].
 ↪apply(list)))

#print(len(unique_ante), len(unique_cons))

# find the union of unique movie ids in antecedents and consequents
unique_movies = np.unique(np.concatenate([unique_ante, unique_cons]))

# print the unique movie ids and names
print(len(unique_movies))
print(movies[movies['movieId'].isin(unique_movies)][['movieId', 'title']])
```

```
49
      movieId                                              title
0           1                                   Toy Story (1995)
20         32          Twelve Monkeys (a.k.a. 12 Monkeys) (1995)
31         47                         Seven (a.k.a. Se7en) (1995)
33         50                          Usual Suspects, The (1995)
71        110                                   Braveheart (1995)
72        111                                  Taxi Driver (1976)
91        150                                    Apollo 13 (1995)
163       260          Star Wars: Episode IV - A New Hope (1977)
181       293  Léon: The Professional (a.k.a. The Professiona…
182       296                                 Pulp Fiction (1994)
289       457                                Fugitive, The (1993)
305       480                                Jurassic Park (1993)
334       527                             Schindler's List (1993)
343       541                                 Blade Runner (1982)
369       589                  Terminator 2: Judgment Day (1991)
373       593                    Silence of the Lambs, The (1991)
382       608                                        Fargo (1996)
528       858                               Godfather, The (1972)
648      1036                                     Die Hard (1988)
681      1089                               Reservoir Dogs (1992)
709      1136              Monty Python and the Holy Grail (1975)
743      1196  Star Wars: Episode V - The Empire Strikes Back…
744      1197                            Princess Bride, The (1987)
745      1198  Raiders of the Lost Ark (Indiana Jones and the…
747      1200                                       Aliens (1986)
755      1210  Star Wars: Episode VI - Return of the Jedi (1983)
758      1213                                   Goodfellas (1990)
759      1214                                        Alien (1979)
766      1221                      Godfather: Part II, The (1974)
```

```
778     1240                                     Terminator, The (1984)
802     1270                                Back to the Future (1985)
819     1291         Indiana Jones and the Last Crusade (1989)
1262    2028                               Saving Private Ryan (1998)
1477    2329                              American History X (1998)
1656    2571                                      Matrix, The (1999)
1783    2762                                 Sixth Sense, The (1999)
2338    3578                                        Gladiator (2000)
2817    4226                                          Memento (2000)
2860    4306                                            Shrek (2001)
3366    4993  Lord of the Rings: The Fellowship of the Ring,…
4031    5952      Lord of the Rings: The Two Towers, The (2002)
4649    6874                                Kill Bill: Vol. 1 (2003)
4831    7153  Lord of the Rings: The Return of the King, The…
4961    7361       Eternal Sunshine of the Spotless Mind (2004)
5006    7438                                Kill Bill: Vol. 2 (2004)
6858   33794                                    Batman Begins (2005)
8368   58559                                  Dark Knight, The (2008)
10097  79132                                        Inception (2010)
13780 109487                                     Interstellar (2014)
```

4.2.4 Using Minimum Support Threshold of 0.05

THIS DID NOT WORK

[36]:
```
# frequent itemsets
#min_support = 0.05
#frequent_itemsets_05 = apriori(binary_df, min_support=min_support,␣
 ↪use_colnames=True)
#print("Frequent Itemsets (with more than one element):")
#print(frequent_itemsets_05)
```

4.3 Association Rule Evaluation

[37]:
```
# load ratings
ratings = pd.read_csv('data/ml-32m/ratings.csv')

ratings.head()
```

[37]:

	userId	movieId	rating	timestamp
0	1	17	4.0	944249077
1	1	25	1.0	944250228
2	1	29	2.0	943230976
3	1	30	5.0	944249077
4	1	32	5.0	943228858

[38]:
```
# get a list of unique user ids
movie_ids_ml = ratings['movieId'].unique()
```

```python
# print the number of unique user ids
print('Number of unique user ids: ', len(movie_ids_ml))
```

```
Number of unique user ids:  84432
```

[39]:
```python
### DATA PREPROCESSING ###

# get the intersection of movie ids
movie_ids = np.intersect1d(movie_ids_imdb, movie_ids_ml)
print('Number of common movie ids: ', len(movie_ids))

# filter the ratings data to contain only the common movie ids
ratings = ratings[ratings['movieId'].isin(movie_ids)]

# drop timestamp column
ratings = ratings.drop('timestamp', axis=1)

# print the number of ratings
print('Number of ratings: ', len(ratings))

# get the number of unique users
user_ids = ratings['userId'].unique()

# print the number of unique users
print('Number of unique users: ', len(user_ids))
```

```
Number of common movie ids:  49892
Number of ratings:  25723135
Number of unique users:  200947
```

[40]:
```python
# get only the ratings for user 304, 6741 and 147001
test_ratings = ratings[ratings['userId'].isin([304, 6741, 147001])]
```

[41]:
```python
def eval_rules(user_id, rules):
    # get the test ratings for the user
    test_user = test_ratings[test_ratings['userId'] == user_id]

    # get num ratings
    num_ratings = len(test_user)
    print('Number of ratings for user', user_id, ':', num_ratings)

    # using the rules, check if any of the antecedents are in the test ratings
    # if they are, check if the consequents are also in the test ratings
    # if both are and both are rated higher than 4, increment perfect␣
↪classification
    perfect_recommendation = 0
    perfect_rule = []
    # if both are rated higher than 3.5, increment good classification
```

```
    good_recommendation = 0
    good_rule = []
    # if both are and only one is rated higher than 4, increment bad␣
↪classification
    bad_recommendation = 0
    bad_rule = []

    for idx, row in rules.iterrows():
        if row['antecedents'].issubset(test_user['movieId']) and␣
↪row['consequents'].issubset(test_user['movieId']):
            if test_user[test_user['movieId'].
↪isin(row['antecedents'])]['rating'].values[0] >= 4 and␣
↪test_user[test_user['movieId'].isin(row['consequents'])]['rating'].values[0]␣
↪>= 4:
                if [row['consequents'], row['antecedents']] in perfect_rule:
                    continue
                perfect_recommendation += 1
                # append string "item1 -> item2" to good_rule
                perfect_rule.append([row['antecedents'], row['consequents']])

            elif test_user[test_user['movieId'].
↪isin(row['antecedents'])]['rating'].values[0] >= 3.5 and␣
↪test_user[test_user['movieId'].isin(row['consequents'])]['rating'].values[0]␣
↪>= 3.5:
                if [row['consequents'], row['antecedents']] in good_rule:
                    continue
                good_recommendation += 1
                # append string "item1 -> item2" to good_rule
                good_rule.append([row['antecedents'], row['consequents']])
            elif test_user[test_user['movieId'].
↪isin(row['antecedents'])]['rating'].values[0] >= 3.5 and␣
↪test_user[test_user['movieId'].isin(row['consequents'])]['rating'].values[0]␣
↪< 3.5:
                if [row['consequents'], row['antecedents']] in bad_rule:
                    continue
                bad_recommendation += 1
                # append string "item1 -> item2" to bad_rule
                bad_rule.append([row['antecedents'], row['consequents']])
            else:
                pass

    # it might be the case that rules go both ways, so we need to check for␣
↪that as well
```

```
    return perfect_recommendation, good_recommendation, bad_recommendation,
 ↪perfect_rule, good_rule, bad_rule
```

Rule Evaluation for Minimum Support = 0.2

[46]:
```
# evaluate the rules for user 304 with a support of 0.2
perfect_recommendation, good_recommendation, bad_recommendation, perfect_rule,
 ↪good_rule, bad_rule = eval_rules(304, rules_apriori_2)

print('Perfect recommendation:', perfect_recommendation)
print('Good recommendation:', good_recommendation)
print('Bad recommendation:', bad_recommendation)

# evaluate the rules for user 6741 with a support of 0.2
perfect_recommendation, good_recommendation, bad_recommendation, perfect_rule,
 ↪good_rule, bad_rule = eval_rules(6741, rules_apriori_2)

print('Perfect recommendation:', perfect_recommendation)
print('Good recommendation:', good_recommendation)
print('Bad recommendation:', bad_recommendation)

# evaluate the rules for user 147001 with a support of 0.2
perfect_recommendation, good_recommendation, bad_recommendation, perfect_rule,
 ↪good_rule, bad_rule = eval_rules(147001, rules_apriori_2)

print('Perfect recommendation:', perfect_recommendation)
print('Good recommendation:', good_recommendation)
print('Bad recommendation:', bad_recommendation)
```

```
Number of ratings for user 304 : 168
Perfect recommendation: 1
Good recommendation: 0
Bad recommendation: 0
Number of ratings for user 6741 : 336
Perfect recommendation: 5
Good recommendation: 0
Bad recommendation: 0
Number of ratings for user 147001 : 223
Perfect recommendation: 3
Good recommendation: 1
Bad recommendation: 0
```

Rule Evaluation for Minimum Support = 0.15

[49]:
```
# evaluate the rules for user 304 with a support of 0.15
perfect_recommendation, good_recommendation, bad_recommendation, perfect_rule,
 ↪good_rule, bad_rule = eval_rules(304, rules_apriori_15)

print('Perfect recommendation:', perfect_recommendation)
```

```python
print('Good recommendation:', good_recommendation)
print('Bad recommendation:', bad_recommendation)


# evaluate the rules for user 6741 with a support of 0.15
perfect_recommendation, good_recommendation, bad_recommendation, perfect_rule, good_rule, bad_rule = eval_rules(6741, rules_apriori_15)

print('Perfect recommendation:', perfect_recommendation)
print('Good recommendation:', good_recommendation)
print('Bad recommendation:', bad_recommendation)


# evaluate the rules for user 147001 with a support of 0.15
perfect_recommendation, good_recommendation, bad_recommendation, perfect_rule, good_rule, bad_rule = eval_rules(147001, rules_apriori_15)

print('Perfect recommendation:', perfect_recommendation)
print('Good recommendation:', good_recommendation)
print('Bad recommendation:', bad_recommendation)
```

```
Number of ratings for user 304 : 168
Perfect recommendation: 6
Good recommendation: 7
Bad recommendation: 0
Number of ratings for user 6741 : 336
Perfect recommendation: 23
Good recommendation: 1
Bad recommendation: 13
Number of ratings for user 147001 : 223
Perfect recommendation: 18
Good recommendation: 2
Bad recommendation: 1
```

Rule Evaluation for Minimum Support = 0.1

[50]:
```python
# evaluate the rules for user 304 with a support of 0.1
perfect_recommendation, good_recommendation, bad_recommendation, perfect_rule, good_rule, bad_rule = eval_rules(304, rules_apriori_1)

print('Perfect recommendation:', perfect_recommendation)
print('Good recommendation:', good_recommendation)
print('Bad recommendation:', bad_recommendation)

# evaluate the rules for user 6741 with a support of 0.1
perfect_recommendation, good_recommendation, bad_recommendation, perfect_rule, good_rule, bad_rule = eval_rules(6741, rules_apriori_1)
```

```python
print('Perfect recommendation:', perfect_recommendation)
print('Good recommendation:', good_recommendation)
print('Bad recommendation:', bad_recommendation)

# evaluate the rules for user 147001 with a support of 0.1
perfect_recommendation, good_recommendation, bad_recommendation, perfect_rule,␣
↪good_rule, bad_rule = eval_rules(147001, rules_apriori_1)

print('Perfect recommendation:', perfect_recommendation)
print('Good recommendation:', good_recommendation)
print('Bad recommendation:', bad_recommendation)
```

```
Number of ratings for user 304 : 168
Perfect recommendation: 29
Good recommendation: 76
Bad recommendation: 0
Number of ratings for user 6741 : 336
Perfect recommendation: 240
Good recommendation: 16
Bad recommendation: 55
Number of ratings for user 147001 : 223
Perfect recommendation: 105
Good recommendation: 24
Bad recommendation: 5
```

Further Tests for User 6741

[53]:
```python
# evaluate the rules for user 6741 with a support of 0.1
perfect_recommendation, good_recommendation, bad_recommendation, perfect_rule,␣
↪good_rule, bad_rule = eval_rules(6741, rules_apriori_1)

print('Perfect recommendation:', perfect_recommendation)
print('Good recommendation:', good_recommendation)
print('Bad recommendation:', bad_recommendation)

# go through the bad rules and print movie names
for rule in bad_rule[:5]:
    print(f"Antecedent :{movies[movies['movieId'].isin(rule[0])][['movieId',␣
↪'title']]}, consequent: {movies[movies['movieId'].isin(rule[1])][['movieId',␣
↪'title']]}")
```

```
Number of ratings for user 6741 : 336
Perfect recommendation: 240
Good recommendation: 16
Bad recommendation: 55
Antecedent :    movieId                              title
31        47  Seven (a.k.a. Se7en) (1995), consequent:       movieId
title
1656      2571  Matrix, The (1999)
```

```
Antecedent :    movieId                       title
33       50  Usual Suspects, The (1995), consequent:       movieId
title
1656     2571  Matrix, The (1999)
Antecedent :     movieId                                     title
163      260  Star Wars: Episode IV - A New Hope (1977), consequent:
movieId               title
1656     2571  Matrix, The (1999)
Antecedent :     movieId                              title
369      589  Terminator 2: Judgment Day (1991), consequent:       movieId
title
1656     2571  Matrix, The (1999)
Antecedent :     movieId                  title
528      858  Godfather, The (1972), consequent:       movieId
title
1656     2571  Matrix, The (1999)
```

[ ]:

```

```