Apriori Algorithm Report

Index

Modeling Technique:

After assessing the data we had available to us, we determined that a recommender system could be built on the user rating data we had. We followed the logic that if a large number of people like a similar set of games as you, and they also largely liked another game, that would be a game you'd be likely to enjoy as well. This sort of thinking is very similar to what is used with transaction lists and apriori algorithms: if a subset of items are frequently bought together, they can be recommended together. Thus, we decided to process the data into transaction lists, and use apriori algorithms to extract knowledge from these lists.

Apriori Algorithm:

Apriori is an algorithm for frequent item set mining and association rule learning over relational databases. It proceeds by identifying the frequent individual items in the database and extending them to larger and larger item sets as long as those item sets appear sufficiently often in the database. The frequent item sets determined by Apriori can be used to determine association rules which highlight general trends in the database: this has applications in domains such as market basket analysis.
Hence, this fits the best with what the team wants to achieve with the project.

To create the lists, we explored several ideas. Users in the database have ranked games on a discrete ordinal scale from 1 to 10, so the first idea was to create a "transaction" list for each pair of user and ranking, where each user was associated with 10 lists, each of which included all the games they gave the same rank. However, we found that categorizing the games into 10 lists per user was too granular, and that the data from the lower-ranked lists was not very useful. We tried many combinations of lists, but found the most subjectively interesting and useful results with the following technique: create a transaction list for each user including all the games that they have ranked at 9 or 10 (their favorite games). With this list, we can generate rules using the apriori algorithm, and then filter those rules to make recommendations.

Assessment Criteria

A recommender system's performance is largely subjective, insofar as it can only truly be evaluated by making recommendations and receiving feedback on them. Thus, Andrew, the member of our team who (by far) plays the most board games and whose Boardgamegeek user account is included in our data, will make a subjective evaluation of the accuracy of the model. In particular, if the model recommends games that Andrew has played and enjoyed, but does not own, then we will consider it certainly successful. If it recommends games he has played and not enjoyed, we will consider it unsuccessful. If it recommends games that he has not played, he will either get them and play it, or we will use other criteria to attempt to determine the success or failure of the model. Some of those other criteria will be: examining the recommendations made by the model to see if they are subjectively interesting and give real recommendations (i.e. they don't just recommend expansions to games the user already owns, and they don't recommend solely re-implementations of games the user already owns); and, testing the model with well-known board game reviewers to see if its recommendations align with their publicly-stated preferences.

Parameters

Selection of dataset

The following parameters were available to us to modify: support, confidence, and the selected data from the dataset. Selecting data from the dataset was largely an exercise in selecting data that would be useful, but still be in small enough quantity for it to be processed in a reasonable amount of time. We chose to only include users who shared at least 4 games with rank 9 or 10 with the test user, so that the apriori algorithm would not even have to spend any time checking users whose preferences do not strongly align with the test user. This choice does bring a limitation, however, which is that the model can only be used if the test user has sufficiently many games that they've rated at 9 or 10; in reality, though, this limitation is not problematic, because if they have too few games that they've rated highly, any recommendations that we make would be based on so little data that they'd be useless.

Support Count

Support represents the popularity of that item of all the item transactions. Support of the item is calculated as the ratio of the number of transactions includes that item and the total number of transactions. We chose to use very low support, because unlike a transaction list of items at a store, board game preferences are extremely varied. With thousands of board games, there is a relatively low percentage of people who will own any particular game. Thus, low support is sensible. We experimented with several different values for support, and ended up with 0.0057143, which was the lowest support that we found could still be processed in a

reasonable amount of time given any user as the test user (for users with the least data, it runs in a few seconds, for users with the most data, it runs in a few minutes). This support generally comes out to a support count of around 150 - 300 for the vast majority of users; for the few exceptional users with a very large number of games in their collection (and thus more users in the filtered dataset), we set the highest possible support count at 400 so that the less-supported rules wouldn't be lost.

Confidence

Confidence can be interpreted as the likelihood of purchasing both the products A and B. Confidence is calculated as the number of transactions that include both A and B divided by the number of transactions includes only product A. Finally, we had to set the confidence threshold. This was the hardest decision -- for high confidence values, there were almost no rules generated, and those that were generated were fairly trivial and obvious (for instance, owning Pandemic Legacy: Season 1 would recommend Pandemic Legacy: Season 2). Thus, we had to set the confidence a bit lower, and settled on 0.5. With a minimum of 50% confidence for each rule generated, the rules would not necessarily be extremely accurate. To make up for this lack of accuracy, we did not directly use the rules that were generated. Instead, we counted the number of unique rules that recommended a particular game, and assigned that score to the game. Thus, if an association rule does not have high confidence and is inaccurate, it will not have a large impact on the final suggestion unless there are many other rules like it (and if there are many rules, it is far less likely that all of them are inaccurate). This approach has the drawback that very niche suggestions will not likely be produced, since they'd only have a small number of recommendations; this side effect is unfortunate, but realistically, if the user's preferences fall into a very particular niche, they will likely already be aware of that niche and other games that fit into it.

Running the model

- Creating a review_lists of users that have rated the games and grouping each user.

```
In [5]:  1  reviews_lists = reviews.groupby('user').name.apply(list)
```

```
In [6]:  1  reviews_lists
```

```
Out[6]: user
        Fu_Koios            [Shadowrift, Approaching Dawn: The Witching Hour]
        beastvol            [Caylus, Puerto Rico, Catan, Ticket to Ride, M...
        mycroft             [Betrayal at House on the Hill, Catan, Arkham ...
        woh                                  [Chess, Go, Bridge, Arimaa, Exxit]
        (mostly) harmless                                        [Red November]
                                                  ...
        zzzuzu              [Bohnanza, BANG!, Battlestar Galactica: The Bo...
        zzzvone             [Alhambra, Puerto Rico, Ticket to Ride: Europe...
        zzzxxxyyy           [Great Western Trail, The Resistance: Avalon, ...
        zzzzzane            [Five Tribes, Scrabble, Agricola, Codenames, L...
        Æleksandr Þræð      [UNO, Mille Bornes, Mahjong, Chez Geek, Battle...
        Name: name, Length: 351048, dtype: object
```

- Selecting the ratings which are been rated 9 and 10 only

```
1  test_user = 'ColorfulPockets'
2  great = []
3  good = []
4  bad = []
5  games = []
6  user_games = reviews.loc[reviews['user'] == test_user]
7
8  for i in range(len(user_games)):
9      game = user_games.iloc[i]['name']
10     rating = user_games.iloc[i]['rating']
11     games.append(game)
12     if rating > 8:
13         great.append(game)
14     elif rating > 6:
15         good.append(game)
16     else:
17         bad.append(game)
18
19
20 print(great)
21 print(good)
22 print(bad)
```

```
['Great Western Trail', 'Dominion', 'The Resistance: Avalon', 'Azul', 'The Mind', 'Welcome To...', 'Root', 'Codenam
es: Duet', 'Decrypto', 'Agricola (Revised Edition)', 'The Crew: The Quest for Planet Nine', 'Dominion (Second Editi
on)', 'The Shipwreck Arcana', 'Electronic Catch Phrase', 'Blood on the Clocktower']
['Mysterium', 'Machi Koro', 'Sheriff of Nottingham', 'Love Letter', 'Sushi Go!', 'Lords of Waterdeep', 'Hanabi', 'T
he Resistance', 'Dixit', 'Ticket to Ride: Europe', 'Pandemic', 'Codenames', 'Carcassonne', 'One Night Ultimate Were
wolf', 'Sushi Go Party!', 'Mr. Jack', 'Welcome to the Dungeon', 'Diplomacy', 'Wits & Wagers', 'Taboo', 'Scattergori
es', 'Cockroach Poker', 'Cribbage', 'Disney Villainous', 'Cryptid', 'Railroad Ink: Blazing Red Edition', 'Whitehall
Mystery', 'One Night Ultimate Vampire', 'Ticket to Ride: London', 'Fast Forward: FORTRESS', 'Sonar', 'Rod Hockey']
['Blokus', 'Chess', 'Scrabble', 'BANG!', 'Monopoly', 'Risk', 'Munchkin', 'Forbidden Island', 'Betrayal at House on
the Hill', 'Dead of Winter: A Crossroads Game', 'Catan', 'Qwirkle', 'Diamant', 'Clue', 'Yahtzee', 'Telestrations',
'Stratego', 'Bananagrams', 'The Game', 'Shadow Hunters', 'The Game of Life', 'Quoridor', 'Candy Land', 'Quiddler',
'Quelf', 'The Captain Is Dead', 'Fast Forward: FEAR', 'Trash Pandas', "Rubik's Race", 'Jenga Xtreme', 'Bananagrams
WildTiles']
```

- Selecting users who have rated at least 3 games from the above list of good rating

```
In [143]:
users = reviews_lists.index.tolist()
users_with_great = []
for user in users:
    if len(set(great) & set(reviews_lists.loc[user])) > 3:
        users_with_great.append(user)

users_with_great
```

```
'210pasadena',
'216stitches',
'28green',
'295min',
'2Yellow',
'2bit',
'2d20',
'2dTones',
'2fofofofo',
'2la_fr',
'2lip',
'2ombie',
'33cb',
'38thDoE',
'3Beez',
'3N1GM4',
'3ldfilms',
'3rn3st',
'3urogamer',
```

- Grouping the user and the lists obtained from above to be used in the model

```
In [144]:
great_df = reviews.loc[reviews['rating'] >= 9.0]
print(great_df.shape)
great_df = great_df.loc[great_df['user'].isin(users_with_great)]
# nines = reviews.loc[reviews['rating'] == 9.0]
# eights = reviews.loc[reviews['rating'] == 8.0]
# sevens = reviews.loc[reviews['rating'] == 7.0]
# sixes = reviews.loc[reviews['rating'] == 6.0]
# fives = reviews.loc[reviews['rating'] == 5.0]
# fours = reviews.loc[reviews['rating'] == 4.0]
# threes = reviews.loc[reviews['rating'] == 3.0]
# twos = reviews.loc[reviews['rating'] == 2.0]
# ones = reviews.loc[reviews['rating'] == 1.0]
```

```
(2335908, 4)
```

```
In [145]:
great_df.shape
```

```
Out[145]: (503989, 4)
```

```
In [146]:
great_lists = great_df.groupby('user').name.apply(list)
```

```
In [147]:
len(great_lists)
```

```
Out[147]: 19723
```

- Running the actual model

In [148]:

```
# to use support count instead, use: 1 - (len(great_lists) - 150)/len(great_lists)
# (in this case, support count of 150)

# experimentally I found that 0.0057143 is a good balance of efficiency and utility, and 150-400 is always fast

support = 0.0057143
if 1 - (len(great_lists) - 400)/len(great_lists) < support:
    support = 1 - (len(great_lists) - 400)/len(great_lists)
elif 1 - (len(great_lists) - 150)/len(great_lists) > support:
    support = 1 - (len(great_lists) - 150)/len(great_lists)

association_rules = apriori(great_lists, min_support = support, min_confidence = 0.5)
association_results = list(association_rules)
association_results
```

```
confidence=0.7864864864864864, lift=19.03297297297297)]),
 RelationRecord(items=frozenset({"Aeon's End: Legacy", 'Gloomhaven'}), support=0.009988338488059625, ordered_statis
tics=[OrderedStatistic(items_base=frozenset({"Aeon's End: Legacy"}), items_add=frozenset({'Gloomhaven'}), confidenc
e=0.5660919540229884, lift=2.415627782171225)]),
 RelationRecord(items=frozenset({"Aeon's End: Legacy", 'Pandemic Legacy: Season 1'}), support=0.009532018455610202,
ordered_statistics=[OrderedStatistic(items_base=frozenset({"Aeon's End: Legacy"}), items_add=frozenset({'Pandemic L
egacy: Season 1'}), confidence=0.5402298850574713, lift=1.9687646014391176)]),
 RelationRecord(items=frozenset({"Aeon's End: War Eternal", 'Gloomhaven'}), support=0.009582720681437916, ordered_s
tatistics=[OrderedStatistic(items_base=frozenset({"Aeon's End: War Eternal"}), items_add=frozenset({'Gloomhaven'}),
confidence=0.5108108108108108, lift=2.1797320687195203)]),
 RelationRecord(items=frozenset({'Brass: Lancashire', 'Age of Steam'}), support=0.021852659331744663, ordered_stati
stics=[OrderedStatistic(items_base=frozenset({'Age of Steam'}), items_add=frozenset({'Brass: Lancashire'}), confide
nce=0.5005807200929152, lift=3.9794250473166333)]),
 RelationRecord(items=frozenset({'Agricola (Revised Edition)', 'Agricola'}), support=0.036961922628403386, ordered_
statistics=[OrderedStatistic(items_base=frozenset({'Agricola (Revised Edition)'}), items_add=frozenset({'Agricola'}
), confidence=0.5424107142857143, lift=3.164142714539232)]),
 RelationRecord(items=frozenset({'Agricola: All Creatures Big and Small', 'Agricola'}), support=0.009024996197333063
, ordered_statistics=[OrderedStatistic(items_base=frozenset({'Agricola: All Creatures Big and Small'}), items_add=
frozenset({'Agricola'}), confidence=0.5129682997118156, lift=2.992390942092913)]),
```

We tried to run this model for high confidence values but there were almost no rules generated, and those that were generated were fairly trivial and obvious (for instance, owning Pandemic Legacy: Season 1 would recommend Pandemic Legacy: Season 2)

- Generating association rules

In [149]:

```
relevant_rules = []
for result in association_results:
    if len(set(great) & set(result[0])) > 1 and not all(game in games for game in result[0]):
        relevant_rules.append(result)

relevant_rules
```

```
188, ordered_statistics=[OrderedStatistic(items_base=frozenset({'Dominion', 'A Feast for Odin'}), items_add=frozens
et({'Great Western Trail'}), confidence=0.5130641330166271, lift=2.0890098875902017)]),
 RelationRecord(items=frozenset({'Great Western Trail', 'Root', 'A Feast for Odin'}), support=0.011154489682097044,
ordered_statistics=[OrderedStatistic(items_base=frozenset({'Root', 'A Feast for Odin'}), items_add=frozenset({'Grea
t Western Trail'}), confidence=0.5080831408775982, lift=2.0687291056005095)]),
 RelationRecord(items=frozenset({'Great Western Trail', 'Agricola (Revised Edition)', 'Agricola'}), support=0.01363
8898747655021, ordered_statistics=[OrderedStatistic(items_base=frozenset({'Great Western Trail', 'Agricola (Revised
Edition)'}), items_add=frozenset({'Agricola'}), confidence=0.5557851239669421, lift=3.242162082224194)]),
 RelationRecord(items=frozenset({'Terraforming Mars', 'Agricola (Revised Edition)', 'Azul'}), support=0.00780814277
7467931, ordered_statistics=[OrderedStatistic(items_base=frozenset({'Agricola (Revised Edition)', 'Azul'}), items_a
dd=frozenset({'Terraforming Mars'}), confidence=0.555956678700361, lift=1.6334177825126204)]),
 RelationRecord(items=frozenset({'Great Western Trail', 'Brass: Birmingham', 'Agricola (Revised Edition)'}), suppor
t=0.0100390407138873 39, ordered_statistics=[OrderedStatistic(items_base=frozenset({'Brass: Birmingham', 'Agricola (
Revised Edition)'}), items_add=frozenset({'Great Western Trail'}), confidence=0.5089974293059126, lift=2.0724517543
766545)]),
 RelationRecord(items=frozenset({'Concordia', 'Great Western Trail', 'Agricola (Revised Edition)'}), support=0.0096
84125133093344, ordered_statistics=[OrderedStatistic(items_base=frozenset({'Concordia', 'Agricola (Revised Edition)
'}), items_add=frozenset({'Great Western Trail'}), confidence=0.5365168539325843, lift=2.184500807207341)]),
 RelationRecord(items=frozenset({'Great Western Trail', 'Terraforming Mars', 'Agricola (Revised Edition)'}), suppor
t=0.013638898747655021, ordered_statistics=[OrderedStatistic(items_base=frozenset({'Great Western Trail', 'Agricola
```

- Final output

```
points = {}
recommenders = {}

for rule in relevant_rules:
    for game in rule[0]:
        points[game] = 0
        recommenders[game] = []

# these restrictions prevent double counting C for [A, B, C, D] and [A, B, C, E]
for rule in relevant_rules:
    for game in rule[0]:
        shared = set(rule[0]) & set(great)
        if not len(set(recommenders[game]) & shared) == len(shared):
            if not game in games:
                recommenders[game].extend(shared.difference(shared & set(recommenders[game])))
                points[game] = points[game] + 1

dict(sorted(points.items(), key=lambda item: item[1]))

```

```
'Brass: Lancashire': 4,
'Clans of Caledonia': 4,
'Gaia Project': 4,
'Mage Knight Board Game': 4,
'The 7th Continent': 4,
'Le Havre': 4,
'Teotihuacan: City of Gods': 4,
'Terra Mystica': 4,
'Through the Ages: A New Story of Civilization': 4,
'Pandemic Legacy: Season 2': 4,
'7 Wonders Duel': 4,
'Viticulture Essential Edition': 4,
'Wingspan': 4,
'A Feast for Odin': 5,
'Orléans': 5,
'The Voyages of Marco Polo': 5,
"Tzolk'in: The Mayan Calendar": 5,
'Dominion: Intrigue': 5,
'The Castles of Burgundy': 6,
'Brass: Birmingham': 7,
'Concordia': 7,
'Pandemic Legacy: Season 1': 7,
'Terraforming Mars': 9}
```

Based on the results, we can observe the game with the highest values is recommended to the most to the test users

The Resulting Model and Evaluation

While we created several models by changing the parameters discussed above, we will be discussing only the most successful. The final model produced the following results when testing with Andrew's Boardgamegeek username:

```
'Brass: Lancashire': 4,
'Clans of Caledonia': 4,
'Gaia Project': 4,
'Mage Knight Board Game': 4,
'The 7th Continent': 4,
'Le Havre': 4,
'Teotihuacan: City of Gods': 4,
'Terra Mystica': 4,
'Through the Ages: A New Story of Civilization': 4,
'Pandemic Legacy: Season 2': 4,
'7 Wonders Duel': 4,
'Viticulture Essential Edition': 4,
'Wingspan': 4,
'A Feast for Odin': 5,
'Orléans': 5,
'The Voyages of Marco Polo': 5,
"Tzolk'in: The Mayan Calendar": 5,
'Dominion: Intrigue': 5,
'The Castles of Burgundy': 6,
'Brass: Birmingham': 7,
'Concordia': 7,
'Pandemic Legacy: Season 1': 7,
'Terraforming Mars': 9}
```

The higher numbers following the games mean that the game is more highly recommended. The model recommends the game Terraforming Mars most highly, followed by Pandemic Legacy: Season 1, Concordia, and Brass: Birmingham. We consider that these are all very good predictions -- Andrew has played and greatly enjoyed Terraforming Mars, and before seeing these results, he independently researched and ordered Concordia for his family. The results listed here are all relatively popular games, but the ordering of the games is not exactly in-line with their overall popularity. For instance, Concordia (which, as discussed, is an excellent recommendation) is ranked well below Castles of Burgundy and 7 Wonders Duel, which are both recommended below it. Thus, for Andrew, the model produces interesting and useful results.

We also tested the model on Quintin Smith, a popular board game reviewer at Shut Up and Sit Down. His Boardgamegeek profile includes far more games than Andrew's does, and his preferences are well-known, so we can interpret the results based on that.

```
'Anachrony': 1,
'Le Havre': 1,
'Maracaibo': 1,
'Caylus': 1,
'Through the Ages: A Story of Civilization': 1,
'Mansions of Madness: Second Edition': 1,
'Mechs vs. Minions': 1,
'Pandemic: Iberia': 1,
'The 7th Continent': 1,
'Twilight Struggle': 1,
'Viticulture Essential Edition': 2,
'Brass: Lancashire': 3,
'Gloomhaven': 5,
'Scythe': 5,
'Puerto Rico': 7}
```

This test reveals a limitation of our system, but also does still give a useful result. Fewer games are recommended because Quinns already has so many popular games that he has rated and

reviewed. However, the model found that he has not rated Puerto Rico, and recommends it! Shut Up and Sit Down has covered the game San Juan in the past, and given it a glowing review; San Juan is a game that is heavily based on Puerto Rico, so it makes sense that if Quinns hasn't played Puerto Rico yet, it would be a great recommendation for him.

We tested the model on several other users and found similar results. Thus, we believe that the model is quite successful and works for a wide variety of users. The biggest weakness of the model is not particularly relevant to the business goal of recommending games to users -- its weakness is that it primarily recommends only very popular games, but that is totally fine when the goal is just to make sure that the user finds a game that is new to them that they will enjoy. Overall, it meets the business objectives by recommending games that users are likely to enjoy.