Modelling Techniques

Association Rule Mining

Load Data

In [1]:
```
import json

def to_json(filename, collection):
    with open(filename, 'w') as file:
        json.dump(collection, file)

    print(f"Collection successfully saved to {filename}")

def read_json(filename):
    with open(filename, 'r') as f:
        data = json.load(f)
    print(f"Collection successfully loaded from {filename}")
    return data
```

V1 - Entire rating dataset

In [2]:
```
filename = "E:\\applied data science capstone\\data\\etl\\extract\\transactions\
```

Load data from json file

In [3]:
```
transactions = read_json(filename)
```

```
Collection successfully loaded from E:\applied data science capstone\data\etl\ext
ract\transactions\all_transactions_8_Jul.json
```

Create sparse matrix from the ratings data

In [ ]:
```
from mlxtend.preprocessing import TransactionEncoder
import pandas as pd
```

In [5]:
```
te = TransactionEncoder()
te_ary = te.fit(transactions).transform(transactions, sparse=True)
sparse_df = pd.DataFrame.sparse.from_spmatrix(te_ary, columns=te.columns_)

sparse_df.head()
```

```
C:\Users\Asus-Home\AppData\Local\Temp\ipykernel_10008\3604965926.py:3: FutureWarn
ing: Allowing arbitrary scalar fill_value in SparseDtype is deprecated. In a futu
re version, the fill_value must be a valid value for the SparseDtype.subtype.
  sparse_df = pd.DataFrame.sparse.from_spmatrix(te_ary, columns=te.columns_)
```

Out[5]:

	, Revbahaf Kingdom Rebuilding Story, The Legend of Prince Van, Revbahaf: The Story of Rebuilding the Kingdom	Gundam	Gundam I: Earth Light	Gundam II: Moonlight Butterfly	.Koni-chan	.hack//Beyond the World	.hack//G.U. Trilogy	.hack
0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	

In [6]:
```
from sklearn.model_selection import train_test_split
train_df, test_df = train_test_split(sparse_df, test_size=0.2, random_state=42)
```

Support - Train

Support is the fraction of transactions that contain the item set

In [7]:
```
from mlxtend.frequent_patterns import apriori, association_rules
```

In [8]:
```python
frequent_itemsets = apriori(train_df, min_support=0.5, use_colnames=True)
print("\nFrequent Itemsets:")
print(frequent_itemsets)
```

```
e:\organized-anime-recommendation\organized-anime-recommendation\.venv\Lib\site-packages\mlxtend\frequent_patterns\fpcommon.py:161: DeprecationWarning: DataFrames with non-bool types result in worse computationalperformance and their support might be discontinued in the future.Please use a DataFrame with bool type
  warnings.warn(
Frequent Itemsets:
    support              itemsets
0  0.529988      (Attack on Titan)
1  0.580365           (Death Note)
2  0.522273  (Fullmetal Alchemist)
3  0.511223     (Sword Art Online)
```

With only 3 frequent itemsets and the itemsets containing only 1 item, the minimum support has to be adjusted to generate more frequent item sets

In [9]:
```python
frequent_itemsets = apriori(train_df, min_support=0.4, use_colnames=True)
print("\nFrequent Itemsets:")
print(frequent_itemsets)
```

```
e:\organized-anime-recommendation\organized-anime-recommendation\.venv\Lib\site-packages\mlxtend\frequent_patterns\fpcommon.py:161: DeprecationWarning: DataFrames with non-bool types result in worse computationalperformance and their support might be discontinued in the future.Please use a DataFrame with bool type
  warnings.warn(
Frequent Itemsets:
    support                                  itemsets
0  0.440351                             (Angel Beats!)
1  0.529988                          (Attack on Titan)
2  0.440609  (Code Geass: Lelouch of the Rebellion)
3  0.580365                               (Death Note)
4  0.522273                      (Fullmetal Alchemist)
5  0.434062                                   (Naruto)
6  0.401767                            (One Punch Man)
7  0.511223                         (Sword Art Online)
8  0.418556                                (Toradora!)
```

In [10]:
```python
frequent_itemsets = apriori(train_df, min_support=0.3, use_colnames=True)
print("\nFrequent Itemsets:")
print(frequent_itemsets)
```

```
e:\organized-anime-recommendation\organized-anime-recommendation\.venv\Lib\site-packages\mlxtend\frequent_patterns\fpcommon.py:161: DeprecationWarning: DataFrames with non-bool types result in worse computationalperformance and their support might be discontinued in the future.Please use a DataFrame with bool type
  warnings.warn(
```

```
Frequent Itemsets:
     support                                          itemsets
0   0.440351                                    (Angel Beats!)
1   0.320264                (Anohana: The Flower We Saw That Day)
2   0.336218                                         (Another)
3   0.529988                                  (Attack on Titan)
4   0.333453                                    (Blue Exorcist)
5   0.343413                                         (Clannad)
6   0.440609             (Code Geass: Lelouch of the Rebellion)
7   0.580365                                      (Death Note)
8   0.300641                                          (ERASED)
9   0.375970                                      (Elfen Lied)
10  0.394974                                 (Fate/stay night)
11  0.522273                              (Fullmetal Alchemist)
12  0.345668                          (High School of the Dead)
13  0.344454                                (My Hero Academia)
14  0.434062                                          (Naruto)
15  0.301456                          (Neon Genesis Evangelion)
16  0.385629                                (No Game, No Life)
17  0.338669                                        (Noragami)
18  0.401767                                   (One Punch Man)
19  0.393665                                   (Spirited Away)
20  0.374337                                     (Steins;Gate)
21  0.511223                                (Sword Art Online)
22  0.368543                                (The Future Diary)
23  0.390305                                     (Tokyo Ghoul)
24  0.418556                                       (Toradora!)
25  0.327492                                      (Your Name.)
26  0.313334                   (Attack on Titan, Angel Beats!)
27  0.311806                        (Death Note, Angel Beats!)
28  0.324560                  (Sword Art Online, Angel Beats!)
29  0.385286                     (Death Note, Attack on Titan)
30  0.351296            (Attack on Titan, Fullmetal Alchemist)
31  0.314911               (Attack on Titan, No Game, No Life)
32  0.335417                  (One Punch Man, Attack on Titan)
33  0.394431               (Sword Art Online, Attack on Titan)
34  0.335999                    (Tokyo Ghoul, Attack on Titan)
35  0.343584 (Death Note, Code Geass: Lelouch of the Rebellion)
36  0.328539 (Code Geass: Lelouch of the Rebellion, Fullmetal Alchemist)
37  0.390871                 (Death Note, Fullmetal Alchemist)
38  0.322243                              (Death Note, Naruto)
39  0.353334                    (Sword Art Online, Death Note)
40  0.306675                     (Naruto, Fullmetal Alchemist)
41  0.329821           (Sword Art Online, Fullmetal Alchemist)
42  0.322826              (Sword Art Online, No Game, No Life)
43  0.304646                 (One Punch Man, Sword Art Online)
44  0.307431                   (Sword Art Online, Tokyo Ghoul)
```

We finally have item sets with more than 1 items

In [11]:

```
# Confidence - fraction of time items in Y appear in transactions that contain X
# Lift - Used to determine the likelihood of 2 anime being enjoyed together.
# > 1 means the occurrence of 1 has a positive effect on the other

rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=

# filter for rules that have a positive association between itemsets
rules = rules[rules['lift'] > 1]
```

```python
print("\nAssociation Rules:")
rules.head()
```

```
Association Rules:
```

Out[11]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift
0	(Attack on Titan)	(Angel Beats!)	0.529988	0.440351	0.313334	0.591209	1.342585
1	(Angel Beats!)	(Attack on Titan)	0.440351	0.529988	0.313334	0.711554	1.342585
2	(Death Note)	(Angel Beats!)	0.580365	0.440351	0.311806	0.537258	1.220068
3	(Angel Beats!)	(Death Note)	0.440351	0.580365	0.311806	0.708084	1.220068
4	(Sword Art Online)	(Angel Beats!)	0.511223	0.440351	0.324560	0.634869	1.441736

Support - Assess

In [12]:
```python
from collections import defaultdict
```

In [13]:
```python
# combine all consequents for a given antecedent
rule_dict = defaultdict(list)
for _, row in rules.iterrows():
    antecedent = tuple(sorted(row['antecedents']))
    consequent = tuple(sorted(row['consequents']))
    rule_dict[antecedent].append((consequent, row['confidence']))

# order the consequents by confidence in descending order
for ant in rule_dict:
    rule_dict[ant] = sorted(rule_dict[ant], key=lambda x: x[1], reverse=True)
```

In [14]:
```python
# recommend new anime based on anime list
def recommend(anime_list, k=5):
    anime_list = set(anime_list)
    candidates = {}
    for ant, conseq_list in rule_dict.items():
        if set(ant).issubset(anime_list):
            for conseq, conf in conseq_list:
                for item in conseq:
                    if item not in anime_list:  # don’t recommend anime already
                        candidates[item] = max(candidates.get(item, 0), conf)
    # sort by confidence
    return [item for item, _ in sorted(candidates.items(), key=lambda x: x[1], r
```

In [15]:
```python
test_df
```

Out[15]:

	, Revbahaf Kingdom Rebuilding Story, The Legend of Prince Van, Revbahaf: The Story of Rebuilding the Kingdom	Gundam	Gundam I: Earth Light	Gundam II: Moonlight Butterfly	.Koni-chan	.hack//Beyond the World	.hack//G.U. Trilogy
265703	0	0	0	0	0	0	0
17623	0	0	0	0	0	0	0
342714	0	0	0	0	0	0	0
373791	0	0	0	0	0	0	0
17282	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...
253804	0	0	0	0	0	0	0
313953	0	0	0	0	0	0	0
34659	0	0	0	0	0	0	0
169427	0	0	0	0	0	0	0
376804	0	0	0	0	0	0	0

76406 rows × 5279 columns

In [16]:
```
# Load anime df
anime_df = pd.read_csv("E:\\applied data science capstone\\data\\combined\\anime
total_anime_catalogue = anime_df.shape[0]
```

In [17]:
```
def evaluate(test_df, k=5):
    hits, total_recommendations, total_relevant = 0, 0, 0
    recommended_items = set()

    for _, row in test_df.iterrows():
        user_anime = [anime for anime, val in row.items() if val == 1]
        if len(user_anime) < 2:
            continue
        # simulate known/unknown split
        split_point = len(user_anime) // 2
        known_anime = user_anime[:split_point]
        actual_items = set(user_anime[split_point:])

        recommendations = recommend(known_anime, k)
        recommended_items.update(recommendations)

        hits += len(actual_items & set(recommendations))
        total_recommendations += len(recommendations)
        total_relevant += len(actual_items)

    precision = hits / total_recommendations if total_recommendations > 0 else 0
    recall = hits / total_relevant if total_relevant > 0 else 0
    coverage = len(recommended_items) / total_anime_catalogue

    return precision, recall, coverage
```

In [18]:
```
precision_scores = []
recall_scores = []
coverage_scores = []
```

In [19]:
```
precision, recall, coverage = evaluate(test_df, k=5)

precision_scores.append(precision)
recall_scores.append(recall)
coverage_scores.append(coverage)

print(f"Precision@5: {precision:.6f}")
print(f"Recall@5:    {recall:.6f}")
print(f"Coverage@5:    {coverage:.6f}")
```

```
Precision@5: 0.420699
Recall@5:    0.026650
Coverage@5:    0.001651
```

Support - fraction of transactions that contain the item set

Confidence - fraction of times item in Y appear in transactions that contain X

Lift - Used to determine the likelihood of 2 anime being enjoyed together. > 1 means the occurrence of 1 has a positive effect on the other

Lift = 1 means no association

Lift < 1 presence of one means another won't be present

Support@0.2 - Train

In [20]:
```
frequent_itemsets = apriori(train_df, min_support=0.2, use_colnames=True)
print("\nFrequent Itemsets:")
print(frequent_itemsets)
```

```
e:\organized-anime-recommendation\organized-anime-recommendation\.venv\Lib\site-p
ackages\mlxtend\frequent_patterns\fpcommon.py:161: DeprecationWarning: DataFrames
with non-bool types result in worse computationalperformance and their support mi
ght be discontinued in the future.Please use a DataFrame with bool type
  warnings.warn(
Frequent Itemsets:
      support                                           itemsets
0    0.265042                          (5 Centimeters per Second)
1    0.261763                                    (A Silent Voice)
2    0.283116                                    (Akame ga Kill!)
3    0.440351                                     (Angel Beats!)
4    0.320264               (Anohana: The Flower We Saw That Day)
..        ...                                                ...
590  0.202886  (Sword Art Online, Tokyo Ghoul, Attack on Titan, Fullmet...
591  0.207686  (One Punch Man, My Hero Academia, Attack on Titan, Sword...
592  0.212499  (One Punch Man, Sword Art Online, Attack on Titan, No Ga...
593  0.215235  (Sword Art Online, Tokyo Ghoul, Attack on Titan, No Game...
594  0.213255  (One Punch Man, Sword Art Online, Tokyo Ghoul, Attack on...

[595 rows x 2 columns]
```

In [21]:
```
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=
rules = rules[rules['lift'] > 1]
print("\nAssociation Rules:")
rules.head()
```

Association Rules:

Out[21]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift
0	(Attack on Titan)	(A Silent Voice)	0.529988	0.261763	0.214593	0.404902	1.546827
1	(A Silent Voice)	(Attack on Titan)	0.261763	0.529988	0.214593	0.819800	1.546827
2	(Your Name.)	(A Silent Voice)	0.327492	0.261763	0.226281	0.690952	2.639609
3	(A Silent Voice)	(Your Name.)	0.261763	0.327492	0.226281	0.864450	2.639609
4	(Akame ga Kill!)	(Attack on Titan)	0.283116	0.529988	0.245462	0.867000	1.635884

Support@0.2 - Assess

In [22]:
```
rule_dict = defaultdict(list)
for _, row in rules.iterrows():
    antecedent = tuple(sorted(row['antecedents']))
    consequent = tuple(sorted(row['consequents']))
    rule_dict[antecedent].append((consequent, row['confidence']))
```

```python
for ant in rule_dict:
    rule_dict[ant] = sorted(rule_dict[ant], key=lambda x: x[1], reverse=True)
```

In [23]:
```python
precision, recall, coverage = evaluate(test_df, k=5)

precision_scores.append(precision)
recall_scores.append(recall)
coverage_scores.append(coverage)

print(f"Precision@5: {precision:.6f}")
print(f"Recall@5:    {recall:.6f}")
print(f"Coverage@5:    {coverage:.6f}")
```

```
Precision@5: 0.450136
Recall@5:    0.033110
Coverage@5:    0.003303
```

In [24]:
```python
models = ["support@0.3", "support@0.2"]
rules_recommendation_metrics_df = pd.DataFrame({"model": models, "precision": pr
                                                "recall": recall_scores, "covera
rules_recommendation_metrics_df
```

Out[24]:

	model	precision	recall	coverage
0	support@0.3	0.420699	0.02665	0.001651
1	support@0.2	0.450136	0.03311	0.003303

It is evident that the model with support@0.2 is the better of the 2.

In [25]:
```python
import pickle

def to_pickle(data, filename):
    with open(filename, 'wb') as f:
        pickle.dump(data, f)
```

In [ ]:
```python
# save model
to_pickle(data=rule_dict, filename="E:\\applied data science capstone\\rules\\ru
```