Assignment 3 Report

1. Abstract

This report explores the use of Association Rule Mining (ARM) on the MovieLens dataset to extract patterns and generate movie recommendations. The dataset includes 100,836 ratings and 3,683 tag applications for 9,742 movies. We generated association rules using the FP-Growth algorithm and evaluated them based on support and confidence metrics. The rules were then used to recommend movies, and their effectiveness was measured using precision and recall metrics. The analysis highlights the potential of ARM to enhance recommendation systems by capturing user preferences through frequent itemsets and association rules.

2. Introduction

The goal of this assignment is to explore the effectiveness of Association Rule Mining (ARM) for generating movie recommendations using the MovieLens dataset. Association Rule Mining is a method for discovering relationships between variables in large datasets. In the context of movie recommendations, ARM can identify patterns in user preferences and suggest relevant movies based on these patterns.

The dataset used in this project consists of user ratings and tags applied to movies. Each user has rated movies on a 5-star scale, and only users who have rated more than 10 movies are considered for analysis. The transactional dataset is constructed from these ratings, with a focus on movies rated above a certain threshold.

We applied the FP-Growth algorithm to generate frequent itemsets and extract association rules from the training data. These rules are of the form $X \rightarrow Y$, where X is a movie or a set of movies and Y is a set of movies that are likely to be watched together. The extracted rules are evaluated and used for recommendations, with the performance of these recommendations assessed using precision and recall.

The assignment aims to demonstrate how ARM can be leveraged to create effective recommendation systems by understanding user preferences and providing targeted suggestions. By varying the number of rules considered, we analyze how recommendation accuracy changes and identify the optimal number of rules for precise recommendations.

---

3. Data Preprocessing and Preparation

To build an effective recommendation system, we began by preprocessing the MovieLens dataset. Our goal was to filter and transform the data to create transactional entries suitable for Association Rule Mining.

1. **Loading and Filtering Data:**
   - The `ratings.csv` and `movies.csv` files were loaded into separate dataframes.
   - We selected active users who rated more than 10 movies and retained only ratings greater than 2:

     ```
     active_users = ratings_df['userId'].value_counts()[ratings_df['userId'].value_counts() > 10].index
     filtered_ratings = ratings_df[(ratings_df['userId'].isin(active_users)) & (ratings_df['rating'] > 2)]
     ```

2. **Grouping Data:**
   - The dataset was grouped by `userId` to create a list of movies rated above 2 for each user:

     ```
     transactional_data = filtered_ratings.groupby('userId')['movieId'].apply(list).reset_index()
     ```

3. **Data Splitting:**
   - Each user’s movie list was split into an 80% training set and a 20% test set:

```
def split_user_movies(movies_list):
    shuffled_movies = np.random.permutation(movies_lis
t)
    split_index = int(0.8 * len(shuffled_movies))
    return shuffled_movies[:split_index], shuffled_movi
es[split_index:]
```

- We then iterated through each user to generate `training_df` and `test_df` containing the movie sets for training and testing, respectively.

4. **Output Datasets:**

   - The final preprocessed datasets were the `training_df` (80% of movies per user) and `test_df` (20% of movies per user), which served as inputs for Association Rule Mining and evaluation.

---

4. Frequent Itemset Mining

In this section, we utilize the FP-Growth algorithm to extract frequent itemsets from the training data. The FP-Growth algorithm constructs a compact data structure called the FP-Tree, which allows for efficient mining of frequent itemsets.

FP-Tree Node Class

We define a Node class that serves as the basic structure for an FP-Tree node, holding the item name, frequency count, parent node, and child nodes.

```
class Node:
    def __init__(self, itemName, frequency, parentNode):
        self.itemName = itemName
        self.count = frequency
        self.parent = parentNode
        self.children = {}
        self.next = None

    def increment(self, frequency):
        self.count += frequency
    def display(self, ind=1):
        print(' ' * ind, self.itemName, ' ', self.count)
```

Tree Construction Functions

We implement functions to construct the FP-Tree, update the header table, and mine the frequent itemsets.

1. **Constructing the FP-Tree**

   The `constructTree` function counts the frequency of each item in the dataset and builds the FP-Tree by linking nodes with the same item name.

   ```
   def constructTree(itemSetList, frequency, minSup):
       # Frequency counting and FP-Tree construction logic
   ```

2. **Mining Frequent Itemsets**

   The `mineTree` function recursively mines the FP-Tree to extract frequent itemsets based on a given minimum support threshold.

   ```
   def mineTree(headerTable, minSup, preFix, freqItemList):
       # Logic to mine frequent itemsets from the FP-Tree
   ```

After executing these functions, we obtain the frequent itemsets necessary for generating association rules.

---

5. Association Rule Mining

In this section, we generate association rules from the frequent itemsets mined in the previous step. Association rules help in identifying relationships between items in the dataset.

Generating Association Rules

We calculate association rules based on the frequent itemsets and a specified minimum confidence threshold.

1. **Support Calculation**

   The `getSupport` function counts how many times a given set of items appears in the dataset.

   ```
   def getSupport(testSet, itemSetList):
       # Logic to calculate support
   ```

2. **Creating Rules**

   The `associationRule` function generates rules of the form X → Y for each frequent itemset. For each subset of the itemset, we compute the confidence and include the rule if it meets the minimum confidence requirement.

   ```
   def associationRule(freqItemSet, itemSetList, minConf):
       # Logic to generate association rules
   ```

Finally, we execute the complete FP-Growth algorithm using the `fp_algo` function, which combines the above steps to return both frequent itemsets and their corresponding association rules.

```
def fp_algo(itemSetList, frequency, minSup, minConf):
    # Complete FP-Growth algorithm implementation
```

Why FP Growth Algorithm?

We preferred FP-Growth algorithm over the Apriori algorithm for frequent itemset mining due to several advantages:

1. Efficiency in Data Scanning

- **FP-Growth:** Scans the dataset only twice—once to build the FP-Tree and once to mine the frequent itemsets from the tree.
- **Apriori:** Requires multiple passes over the dataset to generate candidate itemsets and check their support, which can be computationally expensive, especially with large datasets.

2. Handling Large Datasets

- **FP-Growth:** Efficiently compresses the dataset into a compact FP-Tree structure, which helps in handling large datasets more effectively. This reduces the memory overhead and speeds up the mining process.
- **Apriori:** As the number of itemsets increases, the number of candidates generated grows exponentially, leading to high computational and memory costs.

3. No Candidate Generation

- **FP-Growth:** Avoids candidate generation altogether by using the FP-Tree structure, which directly represents frequent patterns.
- **Apriori:** Generates a large number of candidate itemsets that need to be tested for support, resulting in significant overhead.

4. Better Performance with Sparse Data

- **FP-Growth:** Works well with sparse datasets (where most items are not frequently purchased together), as it constructs a smaller tree that focuses only on frequent itemsets.
- **Apriori:** May struggle with sparse datasets due to the large number of potential candidate itemsets that do not meet the minimum support threshold.

5. Flexibility with Minimum Support

- **FP-Growth:** Allows for the extraction of frequent itemsets at different support levels by simply changing the minimum support threshold without needing to redo previous steps.
- **Apriori:** Changing the support level may require reprocessing the entire dataset, leading to inefficiencies.

6. Selecting and Arranging Top Association Rules

In accordance with the recommendation, we identified the association rules that appeared in both the top 100 lists based on support and confidence. These rules were organized according to their confidence scores.

Top 100 Rules by Confidence

```
Rule 1: {500, 318} => {356},Confidence: 0.8548387096774194, Sup
Rule 2: {5952, 296} => {7153},Confidence: 0.8548387096774194, Su
Rule 3: {7153, 1210} => {260},Confidence: 0.8548387096774194, Su
Rule 4: {1196, 1221} => {858},Confidence: 0.85, Support: 51
Rule 5: {2571, 1221} => {858},Confidence: 0.847457627118644, Sup
Rule 6: {364, 318} => {356},Confidence: 0.8472222222222222, Sup
Rule 7: {4993, 260} => {7153},Confidence: 0.8428571428571429, Su
Rule 8: {356, 1527} => {2571},Confidence: 0.8333333333333334, Su
Rule 9: {2762, 3578} => {2571},Confidence: 0.8333333333333334, S
Rule 10: {780, 589} => {480},Confidence: 0.8333333333333334, Sup
....
```

Top 100 Rules by Support

```
Rule 1: {296} => {318},Confidence: 0.6186440677966102, Support:
Rule 2: {318} => {296},Confidence: 0.5816733067729084, Support:
Rule 3: {356} => {318},Confidence: 0.5538461538461539, Support:
Rule 4: {318} => {356},Confidence: 0.5737051792828686, Support:
Rule 5: {296} => {356},Confidence: 0.597457627118644, Support:
Rule 6: {356} => {296},Confidence: 0.5423076923076923, Support:
Rule 7: {593} => {356},Confidence: 0.5833333333333334, Support:
Rule 8: {356} => {593},Confidence: 0.4846153846153846, Support:
Rule 9: {593} => {318},Confidence: 0.5787037037037037, Support:
Rule 10: {318} => {593},Confidence: 0.49800796812749004, Support
```

7. Recommendation and Evaluation

For each user in the test set, we selected association rules \(X \rightarrow Y \), where \(X \) corresponds to a movie from the training set, and the set \(Y \) represents the recommended movies. To evaluate the recommendation system, we computed average precision and recall metrics, varying the number of rules considered from 1 to 10.

Precision and Recall Analysis

1. **Precision**:

   Precision measures the accuracy of the recommendations. It answers the question: "Of all the items we recommended, how many are relevant?" Precision is calculated as:

   $$\text{Precision} = \frac{\text{Number of relevant items recommended}}{\text{Total number of items recommended}}$$

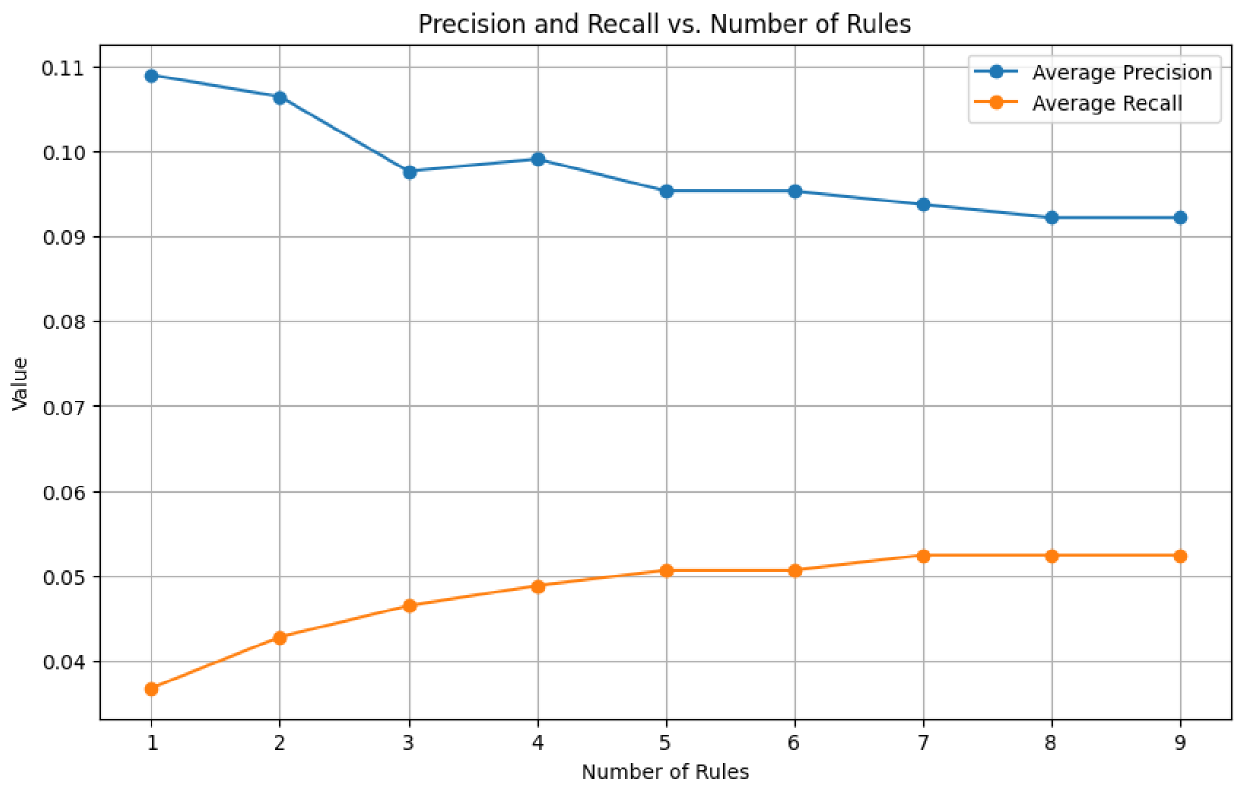
   Precision decreases as we increase the number of false positives, meaning more items that are recommended but are not actually relevant. In other words, if the recommendations include more irrelevant items, precision diminishes.

2. **Recall**:

   Recall measures the coverage of relevant items. It answers the question: "Of all the relevant items for us, how many were successfully recommended?" Recall is calculated as:

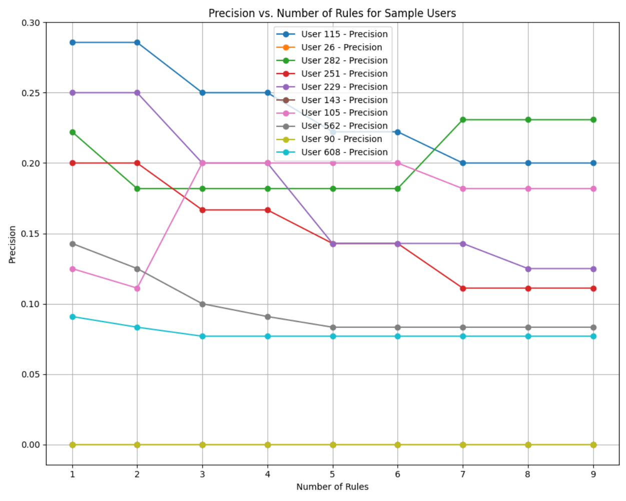
   $$\text{Recall} = \frac{\text{Number of relevant items recommended}}{\text{Total number of relevant items}}$$

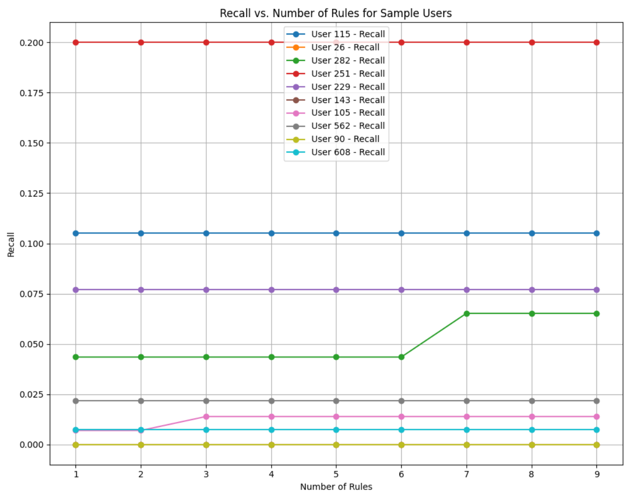
   Recall increases as we receive more relevant recommendations. As the number of recommended items (K) increases, the system captures a larger portion of the items that are potentially relevant.



Sample User Analysis

To gain a deeper understanding of the system's performance, we selected a sample of users from the test set and examined their precision and recall values. The results were plotted as precision and recall graphs for these sample users.





Observations

1. **Precision**:
   - Initially, precision is relatively high, especially if the top rules are strong.
   - As we increase the number of rules, precision slightly decreases due to the inclusion of less accurate recommendations.
   - The decline in precision is often gradual, as we still leverage rules with high confidence.
2. **Recall**:
   - Recall typically increases with the inclusion of more rules.
   - A larger number of rules increases the likelihood of recommending a wider range of items, enhancing the chances of matching items in the test set.
   - The increase in recall often exhibits a diminishing returns pattern, showing rapid initial growth that slows as more rules are added.

Overall Trend

- A slight trade-off is observed between precision and recall.
- The ideal number of rules often lies where a balance between precision and recall is achieved.
- After a certain threshold, adding more rules may not significantly enhance recall while potentially reducing precision.

---

8. Conclusion

In this report, we outlined the process of building a movie recommendation system using association rules. We began by extracting frequent itemsets and generating association rules, which served as the foundation for our recommendation engine. The evaluation of the system's performance was conducted through precision and recall metrics, providing a clear picture of its effectiveness.

The careful selection of association rules enables the system to recommend movies that align closely with user preferences, enhancing the overall user experience. Our analysis of precision and recall offers valuable insights into the system's accuracy and completeness, demonstrating its robustness as a personalized movie recommendation tool. By striking a balance between these metrics, we can confidently assert that our recommendation system is well-equipped to deliver relevant movie suggestions tailored to individual user