Association Rule Mining Project Report

Supermarket Transactions – Apriori & Eclat Implementation

1. Introduction and System Design

This project implements an interactive system for association rule mining on supermarket transaction data. The application lets users upload their own CSV files or use existing, create manual transactions, preprocess items, and run Apriori or Eclat to uncover frequent itemsets and association rules. The system is designed around a modular structure with clear separation between UI, preprocessing, and algorithm logic.

System Architecture:

```
project-root/
├── .devcontainer/
│   └── devcontainer.json
├── data/
│   ├── products.csv
│   └── sample_transactions.csv
├── src/
│   ├── algorithms/
│   │   ├── apriori.py
│   │   └── eclat.py
│   ├── preprocessing/
│   │   └── cleaner.py
│   ├── ui/
│   │   └── app.py
│   └── main.py
├── LICENSE
├── README.md
└── requirements.txt
```

2. Data Preprocessing Approach

Approach:

1. **Column selection**
   If multiple columns exist, the last column is treated as the list of items.
2. **Normalization**
   a. Convert to lowercase
   b. Strip extra spaces
   c. Split on commas or whitespace
3. **Remove empty rows**
   Discard transactions with no valid items.

4. **Remove duplicates**
   Each transaction is treated as a set.
5. **Single-item removal**
   Transactions with only one product are removed since they cannot form associations.
6. **Product validation**
   Only items found in products.csv (from product_name column) are kept.

Output

- Cleaned transaction list
- Report with counts of removed empties, duplicates, invalid items, etc.

3. Algorithm Implementation Details

3.1 Apriori Algorithm

Apriori uses a level-wise, breadth-first approach. It generates candidate itemsets by joining frequent itemsets from the previous level, pruning anything that contains an infrequent subset.

Data Structure

- Dictionary of itemsets by size
  L[k] = {frozenset(itemset): support}

Pseudocode

```
APRIORI(transactions, min_support):
   L1 = frequent 1-itemsets
   L = {1: L1}
   k = 2
   while L[k-1] is not empty:
      Ck = generate candidates from L[k-1]
      remove candidates with infrequent subsets
      count support for each candidate
      Lk = candidates with support >= min_support
      store Lk
      k = k + 1
   return L
```

Rule Generation

```
confidence = support(A ∪ B) / support(A)
lift = confidence / support(B)
```

3.2 Eclat Algorithm

Eclat uses a depth-first search strategy and stores data in vertical format. Each item maps to a TID set (transactions where it appears). Support is computed by intersecting TID sets.

Data Structure

- item –> set(transaction_ids)

Pseudocode

```
ECLAT(prefix, items_tidsets):
   for each (item, tidset):
      new_itemset = prefix ∪ item
      support = |tidset| / total_transactions
      if support >= min_support:
         output new_itemset
         new_items = intersect tidsets of remaining items
         ECLAT(new_itemset, new_items)
```

4. Performance Analysis

After multiple tests using the default dataset and custom uploads Apriori turned out to be faster. The dataset was relatively small, so Apriori benefited from simple support counting and fewer candidate layers.

5. User Interface Design

- **Sidebar**
    - Upload a different CSV file
    - Adjust minimum support & confidence
- **Manual Transaction Builder**
    - Multiselect from product list
- **Data Preview**
    - Raw transaction preview
    - Cleaned transaction preview
- **Mining Dashboard**
    - Runtime comparison
    - Association rule lists
- **Recommendation System**
    - Select an item to see associated predictions

6. Testing and Results

Test Scenario 1: Default Dataset

- Verified correct preprocessing
- Apriori and Eclat produced matching rule sets
- Apriori runtime same or less than Eclat runtime

Test Scenario 2: Default Dataset + Adding Transactions

- Users successfully built and added custom transactions
- They appeared correctly after preprocessing
- Rules adjusted accordingly when re-run

Test Scenario 3: Imported Dataset

- Confirmed successful file parsing
- It appeared correctly onto table
- Rules adjusted accordingly when re-run

7. Conclusion and Reflection

The project provided hands-on experience with association rule mining, UI design, and preprocessing pipelines. The Streamlit UI made it easier to see how parameter changes affect results, and integrating custom uploads taught me how to handle real world messy data. Overall, the project strengthened my skills in algorithms, data cleaning, user interface design, and end-to-end system development.