Mid-Term Project

Submitted By:

Pendyala Hemanth Sai

Table of Contents

Introduction

Two methods are used in this project to mine frequently occurring itemsets and produce association rules from transactional data:

An iterative process known as the "apriori algorithm" creates candidate itemsets level by level and ends early if no new, frequently occurring itemsets are discovered.

To incorporate the early stop mechanism as needed, the Brute Force technique counts all candidate itemsets (starting with 1-itemsets upward) and stops producing higher-level itemsets when none are found.

Description of the Project

Processing CSV file with items data and generating the transaction data—each transaction being a list of items—is the project's main objective. The undertaking:

- Loads CSV file with items data.
- Generate the 5 transactions datasets using python with atleast 20 transactions.
- Determines the number of candidate itemsets that have support.
- Depends on a minimal support threshold to generate frequent itemsets.
- Uses the frequently occurring itemsets that satisfy a minimum confidence threshold to generate association rules.
- Applies an early ending condition to the brute force approach, meaning that no (k+1)-itemsets are produced if no candidate itemset of a specified size is frequent.

Implementation Summary

This project is composed of three major modules that work together to generate transaction data, mine frequent itemsets, and derive association rules using multiple methods.

1. Transaction Database Generation

- **Product Reading & Transaction Generation:**
  - **read_products(file_path):** Reads product names from an external CSV file, ensuring each row provides a valid product name.
  - **generate_transaction(product_list, txn_index, items_per_txn):** Creates a single transaction by cyclically selecting products from the list.
  - **create_database(product_list, offset, total_txns, items_per_txn):** Constructs a database (list) of transactions using a specified offset to ensure variation between databases.
  - **export_database_csv(db, file_name):** Exports the transaction database to a CSV file with a header (TransactionID and Items), where items are concatenated with semicolons.
  - **main():** Reads the product list, validates that there are enough products (at least 10), and generates multiple CSV databases (5 databases of 20 transactions each) using different offsets.

2. Apriori Frequent Itemset Mining & Association Rule Generation

- **Transaction Loading & Support Counting:**
  - **load_transactions(filename):** Loads transactions from a CSV file by reading each line after the header and splitting items based on semicolons.

    - **get_support(itemset, transactions):** Calculates the support (frequency) of a given itemset by counting how many transactions contain it.
- **Apriori Algorithm Implementation:**
    - **generate_candidates(prev_frequents, k):** Forms candidate k-itemsets by taking unions of pairs of frequent (k-1)-itemsets and retaining only those with the correct size.
    - **apriori_frequent_itemsets(transactions, min_support):**
        - Starts with frequent 1-itemsets, then iteratively generates candidates for higher sizes.
        - Uses an **early stop mechanism**—if no candidate itemset at a given level is frequent, it terminates without generating further (k+1)-itemsets.
- **Association Rule Generation:**
    - **generate_association_rules(frequent_itemsets, transactions, min_confidence):** For each frequent itemset (of size 2 or more), this function generates all possible rules by considering every non-empty proper subset as the antecedent. It calculates the confidence for each rule and selects those that meet the minimum confidence threshold.
- **File Processing:**
    - **process_file(filename, min_support, min_confidence):** Combines the steps above by loading transactions, finding frequent itemsets, and generating association rules from a given CSV file. Results are printed in a structured format.
    - **main():** Iterates through a list of transaction files and processes each one using the Apriori algorithm.

3. Comparative Analysis Using Multiple Methods (Brute Force, MLXtend Apriori, MLXtend FP-Growth)

- **Transaction Data Handling:**
    - **TransactionData Class:**
        - **load_csv(file_path):** Reads transactions from a CSV file with error handling, ensuring items are correctly split by semicolons.
        - **to_one_hot(transactions):** Converts transactions into a one-hot encoded pandas DataFrame using the TransactionEncoder from mlxtend, a format required by mlxtend's algorithms.
- **Brute Force Frequent Itemset Mining:**
    - **compute_support(candidate, transactions):** Counts the occurrences of a candidate itemset.
    - **compute_frequent_itemsets_brute(transactions, min_support_count):** Enumerates candidate itemsets for increasing sizes and stops early when no candidate of the current size is frequent.
    - **derive_rules_brute(frequent_sets, transactions, min_conf):** Derives association rules from the frequent itemsets generated by the brute force method, by iterating over every possible non-empty subset.
    - **display_rules(rules, method_label):** A helper function that prints the association rules with a method label.
- **MLXtend Implementations:**
    - **execute_mlxtend_apriori(onehot_df, support_frac, min_conf):** Uses the mlxtend implementation of the Apriori algorithm to generate frequent itemsets and association rules.
    - **execute_mlxtend_fpgrowth(onehot_df, support_frac, min_conf):** Uses mlxtend's FP-Growth algorithm for the same purpose.
- **Performance Comparison:**
    - **execute_brute_force(...):** Times and executes the brute force method, then prints out the generated rules and the time taken.
    - **main():**
        - Prompts the user to enter CSV file paths and thresholds (support and confidence).
        - For each file, loads the transactions, converts the data to one-hot encoding, runs all three methods (brute force, MLXtend Apriori, and MLXtend FP-Growth), and summarizes the execution time and number of rules generated by each method.

Code Annotations

Transaction Database Generation Module

- **read_products(file_path)**
  - **Purpose:** Reads a CSV file and extracts product names.
  - **Key Points:**
    - Skips any blank rows.
    - Trims whitespace from each product name.
- **generate_transaction(product_list, txn_index, items_per_txn=3)**
  - **Purpose:** Generates a transaction deterministically by selecting items in a cyclic manner.
  - **Key Points:**
    - Uses modulo arithmetic (txn_index + i) % n to wrap around the product list.
    - Ensures every transaction has a consistent number of items (items_per_txn).
- **create_database(product_list, offset, total_txns=20, items_per_txn=3)**
  - **Purpose:** Creates a database (list) of transactions.
  - **Key Points:**
    - Varies transactions by adding an offset to the transaction index.
    - Iterates for a specified total number of transactions.
- **export_database_csv(db, file_name)**
  - **Purpose:** Writes the generated transactions to a CSV file.
  - **Key Points:**
    - Includes a header (TransactionID, Items).
    - Joins transaction items with semicolons for CSV formatting.
- **main()** in this module
  - **Purpose:** Orchestrates the reading of products, database creation, and export.
  - **Key Points:**
    - Validates that there are at least 10 products.
    - Generates multiple (5) databases using different offsets to ensure diversity.
    - Provides user feedback via print statements.

Apriori Frequent Itemset Mining & Association Rule Generation Module

- **load_transactions(filename)**
  - **Purpose:** Loads transaction data from a CSV file.
  - **Key Points:**
    - Skips the header.
    - Splits items using a semicolon and stores them as a set (to avoid duplicates).
- **get_support(itemset, transactions)**
  - **Purpose:** Counts how many transactions contain a given itemset.
  - **Key Points:**
    - Uses Python's set method issubset to check inclusion.
- **generate_candidates(prev_frequents, k)**
  - **Purpose:** Generates candidate itemsets of size *k* by joining pairs of frequent (k-1)-itemsets.
  - **Key Points:**
    - Ensures that the union of two frequent itemsets results in a set of the desired size *k*.
- **apriori_frequent_itemsets(transactions, min_support)**
  - **Purpose:** Implements the Apriori algorithm.
  - **Key Points:**
    - Starts with 1-itemsets, then iteratively generates candidates.
    - **Early Stop Mechanism:** If no candidates at a given level are frequent, the loop terminates, avoiding unnecessary computation.
- **generate_association_rules(frequent_itemsets, transactions, min_confidence)**
  - **Purpose:** Derives association rules from the frequent itemsets.

    - **Key Points:**
        - For each frequent itemset with at least 2 items, considers every non-empty subset as an antecedent.
        - Calculates the rule's confidence and only includes rules meeting the minimum confidence threshold.
- **process_file(filename, min_support, min_confidence)**
    - **Purpose:** Integrates the transaction loading, frequent itemset mining, and rule generation.
    - **Key Points:**
        - Outputs frequent itemsets and corresponding association rules for a given file.
- **main()** in this module
    - **Purpose:** Processes multiple CSV transaction files.
    - **Key Points:**
        - Iterates over a predefined list of transaction files.
        - Invokes the processing function for each file and prints results.

Comparative Analysis Module (Brute Force, MLXtend Apriori, MLXtend FP-Growth)

- **TransactionData Class**
    - **load_csv(file_path)**
        - **Purpose:** Reads transactions robustly from a CSV file using a dictionary reader.
        - **Key Points:**
            - Handles errors gracefully with a try/except block.
            - Splits items based on semicolons.
    - **to_one_hot(transactions)**
        - **Purpose:** Converts transaction sets into a one-hot encoded DataFrame.
        - **Key Points:**
            - Uses mlxtend's TransactionEncoder to format the data appropriately for further processing by mlxtend algorithms.
- **Brute Force Frequent Itemset Mining**
    - **compute_support(candidate, transactions)**
        - **Purpose:** Counts the support for a candidate itemset.
        - **Key Points:**
            - Iterates over all transactions and uses subset checking.
    - **compute_frequent_itemsets_brute(transactions, min_support_count)**
        - **Purpose:** Enumerates candidate itemsets for increasing sizes.
        - **Key Points:**
            - Uses itertools.combinations to generate candidates.
            - Implements an early stopping condition—if no candidate at a given size is frequent, it halts further generation.
    - **derive_rules_brute(frequent_sets, transactions, min_conf)**
        - **Purpose:** Derives association rules from the frequent itemsets obtained via brute force.
        - **Key Points:**
            - Considers all non-empty proper subsets as potential antecedents.
            - Computes confidence for each candidate rule and selects those that pass the threshold.
- **MLXtend Methods**
    - **execute_mlxtend_apriori(onehot_df, support_frac, min_conf)**
        - **Purpose:** Applies mlxtend's apriori function to compute frequent itemsets and generates association rules.
        - **Key Points:**
            - Times the execution.
            - Iterates through the results to print the rules.
    - **execute_mlxtend_fpgrowth(onehot_df, support_frac, min_conf)**
        - **Purpose:** Uses mlxtend's fpgrowth function in a similar way to compute frequent itemsets and rules.

        - **Key Points:**
            - Provides a performance comparison with the Apriori method.
- **Performance Comparison and Execution**
    - **execute_brute_force(transactions, support_frac, min_conf)**
        - **Purpose:** Executes the brute force frequent itemset mining and rule derivation.
        - **Key Points:**
            - Measures execution time.
            - Displays the derived rules.
    - **main()** in this module
        - **Purpose:**
            - Prompts the user to input file paths and threshold values (support and confidence).
            - Processes each file: loads transactions, converts to one-hot encoding, and executes all three methods (Brute Force, MLXtend Apriori, MLXtend FP-Growth).
        - **Key Points:**
            - Summarizes and prints the performance metrics (execution time and rule count) for each method.

Running Code

Please open the attached link in google collab upload the dataset file into the environment and run the cells cell by cell.

Source Code Link: Data Mining Mid Term Project

Screenshots

1. Transaction Datasets Creation:

```
Database 1 created and saved as 'db_transactions_1.csv' with 20 transactions.
Database 2 created and saved as 'db_transactions_2.csv' with 20 transactions.
Database 3 created and saved as 'db_transactions_3.csv' with 20 transactions.
Database 4 created and saved as 'db_transactions_4.csv' with 20 transactions.
Database 5 created and saved as 'db_transactions_5.csv' with 20 transactions.
```

2. Brute Force Method:

```
Processing file: db_transactions_1.csv
Frequent Itemsets:
  {'Pasta'}: support = 5
  {'Olive Oil'}: support = 6
  {'Rice'}: support = 4
  {'Coffee'}: support = 6
  {'Tea'}: support = 6
  {'Sugar'}: support = 6
  {'Salt'}: support = 6
  {'Flour'}: support = 6
  {'Butter'}: support = 6
  {'Cheese'}: support = 5
  {'\ufeffProduct'}: support = 4
  {'Flour', 'Sugar'}: support = 2
  {'Sugar', 'Tea'}: support = 4
  {'Cheese', 'Flour'}: support = 2
  {'Pasta', 'Coffee'}: support = 2
  {'Tea', 'Coffee'}: support = 4
  {'Pasta', 'Rice'}: support = 3
  {'Salt', 'Tea'}: support = 2
  {'Olive Oil', 'Rice'}: support = 2
  {'Butter', '\ufeffProduct'}: support = 2
  {'Salt', 'Flour'}: support = 4
  {'Cheese', 'Butter'}: support = 4
  {'Olive Oil', 'Tea'}: support = 2
  {'Pasta', 'Olive Oil'}: support = 4
  {'Salt', 'Sugar'}: support = 4
  {'Sugar', 'Coffee'}: support = 2
  {'\ufeffProduct', 'Rice'}: support = 2
  {'Cheese', '\ufeffProduct'}: support = 3
  {'Salt', 'Butter'}: support = 2
  {'Flour', 'Butter'}: support = 4
  {'Olive Oil', 'Coffee'}: support = 4
  {'Pasta', 'Olive Oil', 'Rice'}: support = 2
  {'Cheese', 'Butter', '\ufeffProduct'}: support = 2
  {'Salt', 'Sugar', 'Tea'}: support = 2
  {'Cheese', 'Flour', 'Butter'}: support = 2
  {'Coffee', 'Sugar', 'Tea'}: support = 2
  {'Pasta', 'Olive Oil', 'Coffee'}: support = 2
  {'Flour', 'Salt', 'Sugar'}: support = 2
  {'Tea', 'Olive Oil', 'Coffee'}: support = 2
  {'Salt', 'Flour', 'Butter'}: support = 2
```

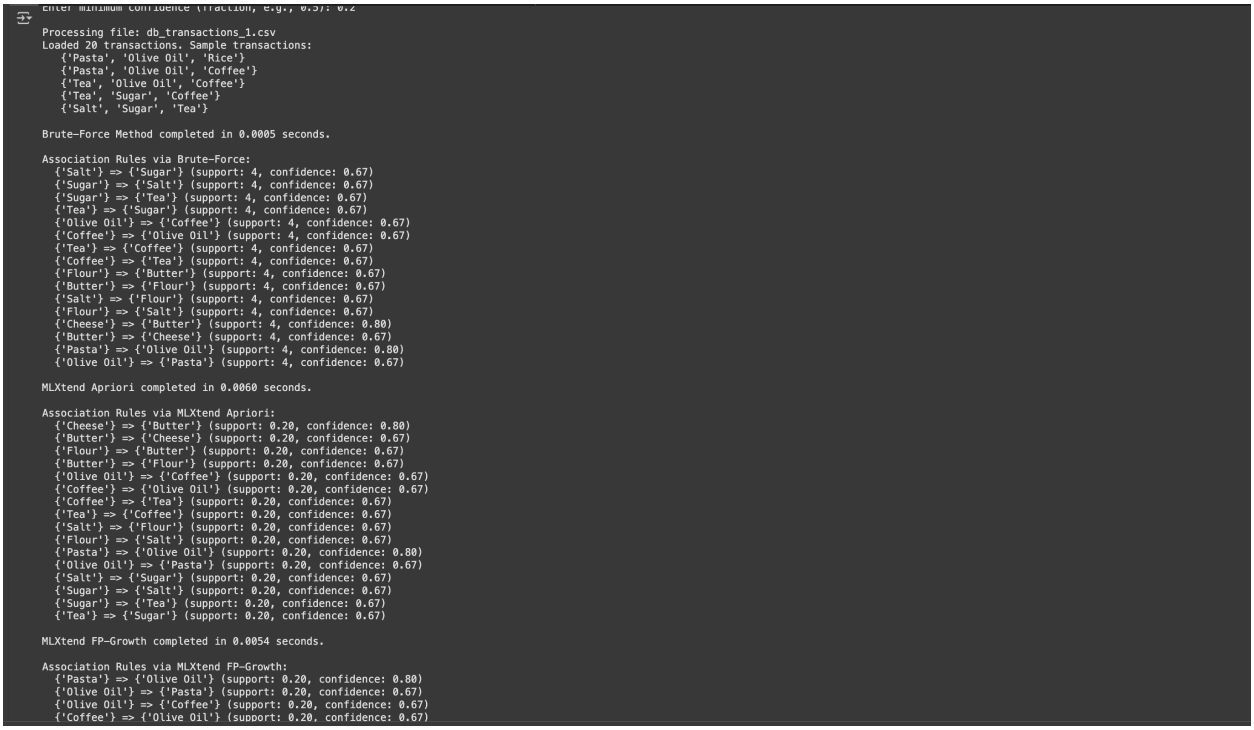
```
Association Rules:
  {'Sugar'} -> {'Tea'} (support: 4, confidence: 0.67)
  {'Tea'} -> {'Sugar'} (support: 4, confidence: 0.67)
  {'Tea'} -> {'Coffee'} (support: 4, confidence: 0.67)
  {'Coffee'} -> {'Tea'} (support: 4, confidence: 0.67)
  {'Pasta'} -> {'Rice'} (support: 3, confidence: 0.60)
  {'Rice'} -> {'Pasta'} (support: 3, confidence: 0.75)
  {'Rice'} -> {'Olive Oil'} (support: 2, confidence: 0.50)
  {'\ufeffProduct'} -> {'Butter'} (support: 2, confidence: 0.50)
  {'Salt'} -> {'Flour'} (support: 4, confidence: 0.67)
  {'Flour'} -> {'Salt'} (support: 4, confidence: 0.67)
  {'Cheese'} -> {'Butter'} (support: 4, confidence: 0.80)
  {'Butter'} -> {'Cheese'} (support: 4, confidence: 0.67)
  {'Pasta'} -> {'Olive Oil'} (support: 4, confidence: 0.80)
  {'Olive Oil'} -> {'Pasta'} (support: 4, confidence: 0.67)
  {'Salt'} -> {'Sugar'} (support: 4, confidence: 0.67)
  {'Sugar'} -> {'Salt'} (support: 4, confidence: 0.67)
  {'\ufeffProduct'} -> {'Rice'} (support: 2, confidence: 0.50)
  {'Rice'} -> {'\ufeffProduct'} (support: 2, confidence: 0.50)
  {'Cheese'} -> {'\ufeffProduct'} (support: 3, confidence: 0.60)
  {'\ufeffProduct'} -> {'Cheese'} (support: 3, confidence: 0.75)
  {'Flour'} -> {'Butter'} (support: 4, confidence: 0.67)
  {'Butter'} -> {'Flour'} (support: 4, confidence: 0.67)
  {'Olive Oil'} -> {'Coffee'} (support: 4, confidence: 0.67)
  {'Coffee'} -> {'Olive Oil'} (support: 4, confidence: 0.67)
  {'Rice'} -> {'Pasta', 'Olive Oil'} (support: 2, confidence: 0.50)
  {'Pasta', 'Olive Oil'} -> {'Rice'} (support: 2, confidence: 0.50)
  {'Pasta', 'Rice'} -> {'Olive Oil'} (support: 2, confidence: 0.67)
  {'Olive Oil', 'Rice'} -> {'Pasta'} (support: 2, confidence: 1.00)
  {'\ufeffProduct'} -> {'Cheese', 'Butter'} (support: 2, confidence: 0.50)
  {'Cheese', 'Butter'} -> {'\ufeffProduct'} (support: 2, confidence: 0.50)
  {'Cheese', '\ufeffProduct'} -> {'Butter'} (support: 2, confidence: 0.67)
  {'Butter', '\ufeffProduct'} -> {'Cheese'} (support: 2, confidence: 1.00)
  {'Salt', 'Sugar'} -> {'Tea'} (support: 2, confidence: 0.50)
  {'Salt', 'Tea'} -> {'Sugar'} (support: 2, confidence: 1.00)
  {'Sugar', 'Tea'} -> {'Salt'} (support: 2, confidence: 0.50)
  {'Cheese', 'Flour'} -> {'Butter'} (support: 2, confidence: 1.00)
  {'Cheese', 'Butter'} -> {'Flour'} (support: 2, confidence: 0.50)
  {'Flour', 'Butter'} -> {'Cheese'} (support: 2, confidence: 0.50)
  {'Sugar', 'Coffee'} -> {'Tea'} (support: 2, confidence: 1.00)
  {'Tea', 'Coffee'} -> {'Sugar'} (support: 2, confidence: 0.50)
  {'Sugar', 'Tea'} -> {'Coffee'} (support: 2, confidence: 0.50)
  {'Pasta', 'Olive Oil'} -> {'Coffee'} (support: 2, confidence: 0.50)
  {'Pasta', 'Coffee'} -> {'Olive Oil'} (support: 2, confidence: 1.00)
  {'Olive Oil', 'Coffee'} -> {'Pasta'} (support: 2, confidence: 0.50)
  {'Salt', 'Flour'} -> {'Sugar'} (support: 2, confidence: 0.50)
  {'Sugar', 'Flour'} -> {'Salt'} (support: 2, confidence: 1.00)
  {'Salt', 'Sugar'} -> {'Flour'} (support: 2, confidence: 0.50)
  {'Olive Oil', 'Tea'} -> {'Coffee'} (support: 2, confidence: 1.00)
  {'Coffee', 'Tea'} -> {'Olive Oil'} (support: 2, confidence: 0.50)
  {'Olive Oil', 'Coffee'} -> {'Tea'} (support: 2, confidence: 0.50)
  {'Salt', 'Flour'} -> {'Butter'} (support: 2, confidence: 0.50)
  {'Salt', 'Butter'} -> {'Flour'} (support: 2, confidence: 1.00)
  {'Flour', 'Butter'} -> {'Salt'} (support: 2, confidence: 0.50)
```

3. Comparative Analysis:

```
Enter CSV file paths (comma-separated): db_transactions_1,db_transactions_2
Enter minimum support (fraction, e.g., 0.05): 0.2
Enter minimum confidence (fraction, e.g., 0.5): 0.2
```

Enter the CSV Transaction files names in the box you want to process.

Enter Minimum Support value.
Enter Minimum confidence value.



Conclusion:

In summary, this project successfully demonstrates multiple approaches for mining association rules from transactional data. The implemented methodologies—including a deterministic database generator, a brute-force frequent itemset miner with an early stopping mechanism, and advanced implementations using MLXtend's Apriori and FP-Growth algorithms—collectively illustrate both the theoretical and practical aspects of frequent pattern mining.

Key outcomes of the project include:

- **Efficient Data Handling:**
  The project effectively reads and processes raw product data to generate transaction databases, ensuring data integrity and consistency throughout the analysis.
- **Robust Frequent Itemset Mining:**
  By integrating an early stop mechanism in the brute-force method, the project minimizes unnecessary computations and optimizes the candidate generation process. This approach is critical when handling large datasets, as it prevents the generation of (k+1)-itemsets when no frequent k-itemsets exist.
- **Diverse Methodological Comparison:**
  The use of both classical (Apriori, brute-force) and modern (MLXtend's Apriori and FP-Growth) techniques provided a practical comparison of performance and efficiency. The analysis of execution times and the number of derived rules offers insights into the trade-offs between algorithmic complexity and real-world performance.
- **Comprehensive Association Rule Generation:**
  The extraction of association rules with a focus on support and confidence thresholds allowed for

meaningful insights into the data relationships, emphasizing the importance of parameter tuning in association rule mining.

Overall, the project not only validates the effectiveness of incorporating an early stopping condition in a brute-force approach but also highlights how modern library implementations can streamline the process of frequent itemset mining and rule derivation. These findings pave the way for future enhancements, including scalability to larger datasets, integration of additional rule metrics, and further exploration of hybrid mining techniques.