CS634 – Data Mining Midterm Project Report

Student Name: Taymar Walters
Email: tw237@njit.edu
Course: CS634 - Data Mining
Instructor: Dr. Yasser Abduallah

1. Introduction

The purpose of this project is to explore how frequent itemsets and association rules can be discovered from transactional data using three different methods:

1. A Brute Force algorithm built entirely from scratch in Python.
2. The Apriori algorithm being implemented via the ***mlxtend*** library.
3. The FP-Growth algorithm, also from ***mlxtend***.

The python script being demonstrated runs all algorithms with user-specified parameters and displays them on the console. This project compares each of the three approaches that are applied to five different transactional datasets, each representing a real-world scenario such as retail shopping or product associations. The report follows a tutorial style, providing step-by-step explanations so that readers can easily reproduce the results.

2. How to run the code

Tools Used:

Python: 3.11 or higher
Environment: VS Code / PowerShell
Libraries: pandas, mlxtend, and time

IMPORTANT:
Ensure that the python file is saved in the same folder as the dataset files (.csv) or it won't work.

Run in VS Code
Open the python file in Visual Studio Code and run the code through the powershell terminal.

Run through Command Prompt
1) Open Command Prompt (press Win + R, type cmd, hit Enter), then to verify that you have python installed, type: *python --version.*
2) If you see something like *Python 3.11* or higher, you're good. Otherwise, install Python from https://www.python.org/downloads/ and during installation, check the box that says:
Add Python to PATH

3) Once you verify that you have python installed, use the **cd** command to go to the folder where you saved the file. It the command should look something like this:

cd C:\Users\YourName\Documents\lastname_firstname_midtermproj

4) Once you're inside the correct folder, just run: *python lastname_firstname_midterm.py*

3. Project Layout (Actual Setup)

```
walters_taymar_midtermproj/
 |
 ├── .git/
 ├── walters_taymar_midterm.py
 ├── walters_taymar_midterm_notebook.ipynb
 ├── generic_items.csv
 ├── generic_transactions. csv
 ├── nike_products. csv
 ├── nike_product_transactions. csv
 ├── bestbuy_products. csv
 ├── bestbuy_transactions. csv
 ├── coffee_items. csv
 ├── coffee_transactions. Csv
 ├── walters_taymar_midterm_report.pdf
 ├── k-mart_items. csv
 ├── k-mart_transactions.csv
```

4. Dataset Creation

There are five pairs of datasets. Each dataset pair represents product listings (**_items.csv* or **_products.csv*) and transactions (**_transactions*.csv). Each product listing has at least 5 unique items while each transaction dataset contains exactly 20 transactions that are deterministic which means no random generation.

These coinciding datasets include:
- Generic: Letters A-F
- Nike Products: athletic wear and accessories
- BestBuy: consumer electronics
- Coffee Items: café menu products
- K-Mart: mixed retail inventory

<u>Note:</u> The Coffee Items dataset was founded on Kaggle.com and every other dataset was provided through the file: *Midterm_Project_Items_Datasets_Examples.pdf*

5. Algorithm Explanations

Brute Force: Enumerates all combinations and counts occurrences to determine support. It's slow but guarantees correctness.
Apriori: Improves efficiency by pruning infrequent itemsets. Uses a bottom-up search to discover frequent sets.
FP-Growth: Avoids candidate generation by compressing data into an FP-tree and directly mining frequent patterns.

6. Results

Example outputs from ***generic_transactions.csv*** with *minimum_support* = 0.3 and *minimum_confidence* = 0.6:

```
==================================================
🔹 FREQUENT ITEMS FOUND BY BRUTE FORCE:
==================================================
        ('A',) | support: 1.00
        ('B',) | support: 0.40
        ('C',) | support: 0.60
        ('D',) | support: 0.45
        ('E',) | support: 0.70
        ('A', 'B') | support: 0.40
        ('A', 'C') | support: 0.60
        ('A', 'D') | support: 0.45
        ('A', 'E') | support: 0.70
        ('C', 'D') | support: 0.30
        ('C', 'E') | support: 0.35
        ('A', 'C', 'D') | support: 0.30
        ('A', 'C', 'E') | support: 0.35

==================================================
🔸 ASSOCIATION RULES — BRUTE FORCE
==================================================

        ('B',) → ('A',) (support: 0.40, confidence: 1.00)
        ('A',) → ('C',) (support: 0.60, confidence: 0.60)
        ('C',) → ('A',) (support: 0.60, confidence: 1.00)
        ('D',) → ('A',) (support: 0.45, confidence: 1.00)
        ('A',) → ('E',) (support: 0.70, confidence: 0.70)
        ('E',) → ('A',) (support: 0.70, confidence: 1.00)
        ('D',) → ('C',) (support: 0.30, confidence: 0.67)
        ('D',) → ('C', 'A') (support: 0.30, confidence: 0.67)
        ('A', 'D') → ('C',) (support: 0.30, confidence: 0.67)
        ('C', 'D') → ('A',) (support: 0.30, confidence: 1.00)
        ('C', 'E') → ('A',) (support: 0.35, confidence: 1.00)
```

7. Reproducibility

All datasets are deterministic and identical values yield identical results across all algorithms.

8. Key Takeaways

- Building Brute Force helps understand the fundamentals.
- Apriori introduces pruning efficiency.
- FP-Growth is the most efficient for larger datasets.

9. Links

- ➔ **Github Repository:**
  https://github.com/tw237njit/walters_taymar_midtermproj.git

- ➔ **Coffee Dataset from Kaggle:**
  https://www.kaggle.com/datasets/ayeshasiddiqa123/coffee-dataset

10. Screenshots

User selecting the Nike Products dataset w/ a min_support = 0.5 and min_confidence = 0.8

```
Here are the following transactional databases
 1) Generic
 2) Nike
 3) Best Buy
 4) Coffee Shop
 5) K-mart

Enter number to select a database:
2
============================================================
Here are the unique items corresponding to the transactions:
============================================================
 Item #      Item Name
      1   Running Shoe
      2    Soccer Shoe
      3          Socks
      4 Swimming Shirt
      5 Dry Fit V-Nick
      6     Rash Guard
      7    Sweatshirts
      8        Hoodies
      9     Tech Pants
     10   Modern Pants
================================================
Enter minimum support (e.g., 0.3 for 30%): 0.5
Enter minimum confidence (e.g., 0.6 for 60%): 0.8

Using min_support = 0.5 and min_confidence = 0.8


Running Brute-Force Algorithm...
Found 6 frequent 1-itemsets
Found 5 frequent 2-itemsets
Found 1 frequent 3-itemsets

Running Apriori Algorithm...
Running FP-Growth Algorithm...
```

```
================================================
🔹 FREQUENT ITEMS FOUND BY BRUTE FORCE:
================================================
('ModernPants',) | support: 0.50
('RashGuard',) | support: 0.60
('RunningShoe',) | support: 0.70
('Socks',) | support: 0.65
('Sweatshirts',) | support: 0.65
('SwimmingShirt',) | support: 0.55
('ModernPants', 'Sweatshirts') | support: 0.50
('RashGuard', 'SwimmingShirt') | support: 0.50
('RunningShoe', 'Socks') | support: 0.55
('RunningShoe', 'Sweatshirts') | support: 0.55
('Socks', 'Sweatshirts') | support: 0.60
('RunningShoe', 'Socks', 'Sweatshirts') | support: 0.50


================================================
🔸 ASSOCIATION RULES – BRUTE FORCE
================================================
('ModernPants',) → ('Sweatshirts',) (support: 0.50, confidence: 1.00)
('RashGuard',) → ('SwimmingShirt',) (support: 0.50, confidence: 0.83)
('SwimmingShirt',) → ('RashGuard',) (support: 0.50, confidence: 0.91)
('Socks',) → ('RunningShoe',) (support: 0.55, confidence: 0.85)
('Sweatshirts',) → ('RunningShoe',) (support: 0.55, confidence: 0.85)
('Socks',) → ('Sweatshirts',) (support: 0.60, confidence: 0.92)
('Sweatshirts',) → ('Socks',) (support: 0.60, confidence: 0.92)
('RunningShoe', 'Socks') → ('Sweatshirts',) (support: 0.50, confidence: 0.91)
('RunningShoe', 'Sweatshirts') → ('Socks',) (support: 0.50, confidence: 0.91)
('Socks', 'Sweatshirts') → ('RunningShoe',) (support: 0.50, confidence: 0.83)
```

```
================================================
🔸 ASSOCIATION RULES – APRIORI
================================================
('ModernPants',) → ('Sweatshirts',) (support: 0.50, confidence: 1.00)
('SwimmingShirt',) → ('RashGuard',) (support: 0.50, confidence: 0.91)
('RashGuard',) → ('SwimmingShirt',) (support: 0.50, confidence: 0.83)
('Socks',) → ('RunningShoe',) (support: 0.55, confidence: 0.85)
('Sweatshirts',) → ('RunningShoe',) (support: 0.55, confidence: 0.85)
('Socks',) → ('Sweatshirts',) (support: 0.60, confidence: 0.92)
('Sweatshirts',) → ('Socks',) (support: 0.60, confidence: 0.92)
('RunningShoe', 'Socks') → ('Sweatshirts',) (support: 0.50, confidence: 0.91)
('RunningShoe', 'Sweatshirts') → ('Socks',) (support: 0.50, confidence: 0.91)
('Socks', 'Sweatshirts') → ('RunningShoe',) (support: 0.50, confidence: 0.83)


================================================
🔸 ASSOCIATION RULES – FP-GROWTH
================================================
('Sweatshirts',) → ('RunningShoe',) (support: 0.55, confidence: 0.85)
('Socks',) → ('Sweatshirts',) (support: 0.60, confidence: 0.92)
('Sweatshirts',) → ('Socks',) (support: 0.60, confidence: 0.92)
('Socks',) → ('RunningShoe',) (support: 0.55, confidence: 0.85)
('RunningShoe', 'Socks') → ('Sweatshirts',) (support: 0.50, confidence: 0.91)
('RunningShoe', 'Sweatshirts') → ('Socks',) (support: 0.50, confidence: 0.91)
('Socks', 'Sweatshirts') → ('RunningShoe',) (support: 0.50, confidence: 0.83)
('ModernPants',) → ('Sweatshirts',) (support: 0.50, confidence: 1.00)
('SwimmingShirt',) → ('RashGuard',) (support: 0.50, confidence: 0.91)
('RashGuard',) → ('SwimmingShirt',) (support: 0.50, confidence: 0.83)


================================================
⏱EXECUTION TIME SUMMARY (seconds)
================================================
Brute-Force Algorithm: 0.0169 sec
Apriori Algorithm:     0.0068 sec
FP-Growth Algorithm:   0.0080 sec
================================================
```

Jupyter Screenshots

Running Brute Force for frequent itemset mining

[ ]

```
def get_support(itemset, transactions):
    """Compute support count for a given itemset."""
    return sum(1 for t in transactions if set(itemset).issubset(set(t)))

def brute_force_mining(transactions, min_support):
    num_transactions = len(transactions)
    frequent_itemsets = []
    k = 1

    while True:
        candidates = [list(i) for i in itertools.combinations(all_items, k)]
        level_frequent = []
        for c in candidates:
            support = get_support(c, transactions) / num_transactions
            if support >= min_support:
                level_frequent.append((tuple(c), support))

        if not level_frequent:
            break

        frequent_itemsets.extend(level_frequent)
        print(f"Found {len(level_frequent)} frequent {k}-itemsets")
        k += 1

    return frequent_itemsets
```

Apriori and FP-Growth Execution

```
import warnings
warnings.filterwarnings("ignore", category=RuntimeWarning)

print("\nRunning Apriori Algorithm...")
start_apriori = time.time()
frequent_itemsets_ap = apriori(one_hot, min_support=min_support, use_colnames=True)
rules_ap = association_rules(frequent_itemsets_ap, metric="confidence", min_threshold=min_confidence)
rules_ap = rules_ap.dropna()
rules_ap = rules_ap[(rules_ap['support'] > 0) & (rules_ap['confidence'] > 0)]
end_apriori = time.time()
apriori_time = end_apriori - start_apriori

print("Running FP-Growth Algorithm...")
start_fp = time.time()
frequent_itemsets_fp = fpgrowth(one_hot, min_support=min_support, use_colnames=True)
rules_fp = association_rules(frequent_itemsets_fp, metric="confidence", min_threshold=min_confidence)
rules_fp = rules_fp.dropna()
rules_fp = rules_fp[(rules_fp['support'] > 0) & (rules_fp['confidence'] > 0)]
end_fp = time.time()
fp_growth_time = end_fp - start_fp
```