CS_634 - DATA MINING MIDTERM PROJECT

Student Name: Risha R Dinesh

Email : rrr62@njit.edu

Course :CS634 – Data Mining

Instructor : Dr. Yasser Abduallah

Abstract

This project examines ways of creating common itemsets and creating association rules, both with a Brute-Force implementation from scratch and Apriori using python's mlxtend library. We built and examined five small transactional datasets replicating supermarket like item purchases. The Brute-force builds all the possible itemsets iteratively, whereas Apriori uses an efficient candidate pruning technique. Both these algorithms were compared in terms of accuracy and run time for varying support and confidence. The findings indicate that the superiority of optimized algorithms such as Apriori over naïve exhaustive search for frequent itemset mining.

1. Introduction:

This project was to learn how frequent itemset mining and association rule learning work by implementing and comparing two algorithms: Brute-force and Apriori. These algorithms are used in market-based analysis to discover correlations between items that are purchased together such as bread, milk and butter.

To understand the logic of creating itemsets and measuring support and confidence, I first implemented the Brute-force approach from scratch in python. This approach compares all possible combinations of items which are easy to understand but computationally costly for bigger datasets. Then I applied the Apriori algorithm from the mlxtend library which enhances performance by reducing item combinations that are not likely to be frequent. Apriori generates the same output as Brute-force but significantly faster, particularly on larger data. To test these techniques, I created five small transactional datasets of real-world stores, including Amazon, Best Buy, K-mart and General store. Each dataset included approximately 20-30 realistic transactions.

Project Overview:

The project is divided into three main parts:

1.Data Creation
2.Brute-Force method implementation
3.Algorithm comparison

2. Dataset Creation:

2.1 Overview

I created five transactional dataset each of which was a simulation of a real retail setting: Amazon, Nike, Best Buy, K-Mart and Genral store in attempting to test and compare the algorithms effectively. Between 20-25 transactions to make up to each dataset each of which replicates a customer buying a set of commonly purchased items. The objective was to present the data compact enough for the Brute-force and Apriori algorithms to operate quickly but realistic enough to indicate larger shopping patterns.

2.2 Dataset Transactions:

Each dataset focusses on a different store domain to capture item relationships:

Dataset	Store type	Example items
Amazon	Online Retail store	Laptop,Mouse,Keyboard,Charger,USB drive, Tablet etc,.
Nike	Sportswear and apparel	Shoes,Socks,Shorts,Backpack,T-shirt,Cap, etc.,
Best Buy	Consumer electronics	Monitor,HDMIcable, Headphones,Webcam,Speaker etc.,
K-Mart	Grocery store	Bread,Milk,Butter,Rice,Eggs
General Store	Everyday essentials	Soap,Shampoo,Toothpaste,Brush,Lotion etc.

Each file follows the same CSV structure with a Transaction ID(TID) and list of purchased items. Example snippet from Amazon_Transaction.csv:

```
TID, Items
1, Laptop, Mouse, Keyboard
2, Mouse, Keyboard, Monitor, HDMI Cable
3, Laptop, Tablet, USB drive
4, Keyboard. Mouse, Charger
```

2.3 Dataset Design Notes

- All data sets were manually curated in Excel and exported to CSV files to ensure cleanliness of data and full control over item pairs.
- Each data set is deterministic that there was no generation by random chance to provide reproducibility of result.
- The sizes of files were kept intentionally small to enable differences in computation time between algorithms to be easily visible.
- Data sets were designed to include realistic patterns of co-occurrences. For instance:
  1. In Amazon, Laptop and Mouse are likely to co-occur.
  2. In Nike, Shoes and Socks are likely to co-occur.
  3. In K-Mart, Milk and Bread are likely to co-occur in transactions

3. Brute-Force Algorithm Implementation:

3.1 Methodology:

The Brute-Force algorithm was programmed from scratch to completion in Python to gain insight into the underlying process of generating association rules and frequent itemset. This algorithm is simple and exhaustive in its approach, methodically constructing all conceivable combinations of items in the dataset and testing each to determine if it is at or above the user-specified minimum support.

3.2 Algorithm logic:

The algorithm works in a systematic way. Starting with the 1-itemsets, it first creates all possible item combinations before growing until there are no more sets that satisfy the minimum support threshold. It calculates support for each combination by determining how many transactions it appears in and dividing the number by the number of transactions. Only the frequent itemset remain after discarding those with support less than the threshold value. From among them, it calculates association rules of type A -> B and their support and confidence. The execution time, rules, and frequent itemset are finally shown. Python's itertools. combinations and simple counting functions are used in the code to find support.

3.3 Example Execution (Amazon Dataset):

The Brute-Force algorithm was executed on the Amazon transactional dataset with a 45% minimum support threshold and 60% minimum confidence threshold. The algorithm managed to discover some frequent itemsets which include "A Beginner's Guide", "Android Programming: The Big Nerd Ranch", "Java for Dummies", and "Java: The Complete Reference."

There was a high association between "Java for Dummies" and "Java: The Complete Reference”. “Specifically, the rule: "If a customer purchases Java for Dummies, they also purchase Java: The Complete Reference," received a confidence of 76.92%, while the reverse rule “If a customer purchases Java: The Complete Reference, they also purchase Java for Dummies" received a confidence of 100%.The total running time for this dataset was approximately 0.0325 seconds, which reflects the computational weight of the exhaustive search process.

4. Apriori Algorithm Implementation:

4.1 Overview

Apriori algorithm was run with the assistance of Python's mlxtend.frequent patterns library. Unlike the Brute-Force method, which examines all possible item combinations, Apriori uses a candidate pruning method to eliminate infrequent itemsets early. This optimization is based on the Apriori principle, which is: "If an itemset is not frequent, its supersets will not be frequent either."

Applying this rule, unnecessary computations are pruned by the algorithm, and efficiency and scalability are significantly enhanced. It makes Apriori one of the most widely used algorithms for association rule mining and frequent pattern mining.

4.2 Algorithm Logic

The Apriori algorithm operates by iteratively beginning with single items and progressively building larger itemsets. In each iteration, it generates k-itemsets from previously discovered frequent (k–1)-itemsets and calculates their support. Itemsets that fail to reach the minimum support threshold dropped at this point, thereby constraining the count of candidates to be checked.

Once all the frequent itemsets are found, Apriori generates association rules in the form A ->B. For each rule, the algorithm determines support, confidence, and lift values that are utilized to quantify the strength and usefulness of relationships found.

4.3 Example Execution:

In order to analyze the performance of Apriori, the algorithm was run on the Amazon transactional data with a minimum support requirement of 45% and a minimum confidence requirement of 60%.The algorithm found various frequent itemsets such as "A Beginner's Guide", "Android Programming: The Big Nerd Ranch", "Java for Dummies", and "Java: The Complete Reference."

One of the most important association rules derived from the dataset was: "If a customer purchases Java for Dummies, then they purchase Java: The Complete Reference," which possessed a confidence of 76.92%.The reverse rule If a customer purchases Java: The Complete Reference, then they purchase Java for Dummies possessed a confidence of 100%, indicating an absolute relationship between these two items. The Apriori algorithm completed execution in approximately 0.0024 seconds, roughly 13 times faster than the Brute-Force method with the same frequent itemsets and produced rules.

5. FP-Growth Algorithm Implementation:

5.1 Overview

The FP-Growth algorithm was implemented with them mlxtend.frequent_patterns.fpgrowth function in Python. Unlike both Brute-Force and Apriori, FP-Growth generates no candidate itemsets whatsoever. Instead, it constructs a special data structure known as the FP-Tree (Frequent Pattern Tree), which captures the transactional database and enables frequent pattern mining much quicker. By scanning the dataset only twice—once to build the FP-Tree and once to discover frequent items, the algorithm can achieve exceptional speed and scalability. This algorithmizing makes FP-Growth applicable for large databases where candidate generation would otherwise be costly computationally.

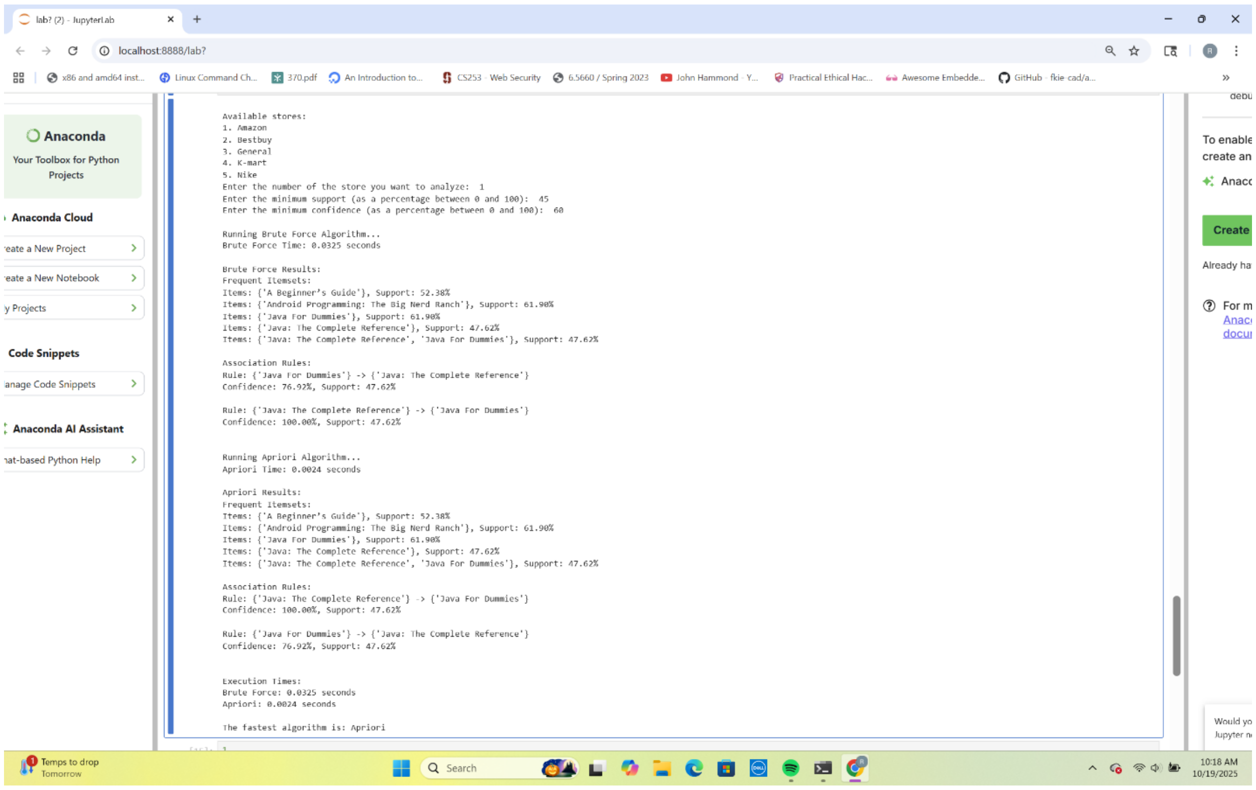
5.2 Algorithm Logic

The FP-Growth algorithm operates in three main phases. First, it builds an FP-Tree by scanning the database and recording the frequency of every item. Transactions containing common prefixes are merged into shared branches of the tree, thus compacting the data and eliminating redundancy. Secondly, the algorithm discovers frequent patterns by recursively scanning the FP-Tree to identify all the itemsets that pass the minimum support criteria. Finally, it generates association rules from the discovered frequent itemsets in the form of A -> B and evaluates every rule with support, confidence, and lift metrics. This structure allows FP-Growth to avoid candidate generation altogether, greatly improving processing speed without sacrificing accuracy to that of the Apriori algorithm.

6. Result and Analysis:

Having run and compared all three algorithms - Brute-Force, Apriori, and FP-Growth - on the five transactional data sets (Amazon, Nike, Best Buy, K-Mart, and General Store), their performance was compared in terms of execution time, accuracy, and scalability. Apriori is significantly faster than Brute-Force while maintaining identical accuracy.

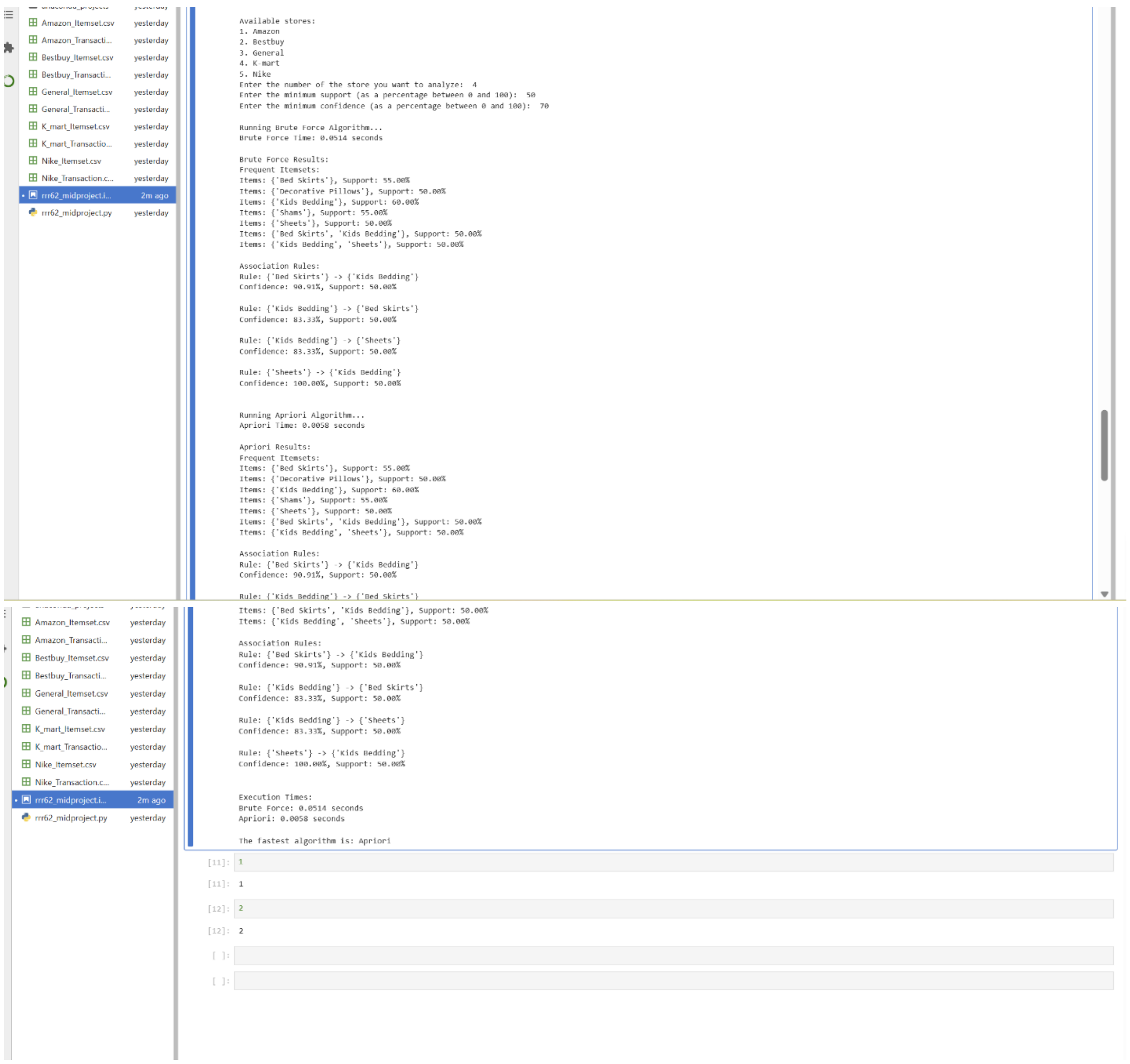
6.1 Sample Output Screenshot of Amazon dataset:



```
Available stores:
1. Amazon
2. Bestbuy
3. General
4. K-mart
5. Nike
Enter the number of the store you want to analyze:  1
Enter the minimum support (as a percentage between 0 and 100):  45
Enter the minimum confidence (as a percentage between 0 and 100):  60

Running Brute Force Algorithm...
Brute Force Time: 0.0325 seconds

Brute Force Results:
Frequent Itemsets:
Items: {'A Beginner's Guide'}, Support: 52.38%
Items: {'Android Programming: The Big Nerd Ranch'}, Support: 61.90%
Items: {'Java For Dummies'}, Support: 61.90%
Items: {'Java: The Complete Reference'}, Support: 47.62%
Items: {'Java: The Complete Reference', 'Java For Dummies'}, Support: 47.62%

Association Rules:
Rule: {'Java For Dummies'} -> {'Java: The Complete Reference'}
Confidence: 76.92%, Support: 47.62%

Rule: {'Java: The Complete Reference'} -> {'Java For Dummies'}
Confidence: 100.00%, Support: 47.62%


Running Apriori Algorithm...
Apriori Time: 0.0024 seconds

Apriori Results:
Frequent Itemsets:
Items: {'A Beginner's Guide'}, Support: 52.38%
Items: {'Android Programming: The Big Nerd Ranch'}, Support: 61.90%
Items: {'Java For Dummies'}, Support: 61.90%
Items: {'Java: The Complete Reference'}, Support: 47.62%
Items: {'Java: The Complete Reference', 'Java For Dummies'}, Support: 47.62%

Association Rules:
Rule: {'Java: The Complete Reference'} -> {'Java For Dummies'}
Confidence: 100.00%, Support: 47.62%

Rule: {'Java For Dummies'} -> {'Java: The Complete Reference'}
Confidence: 76.92%, Support: 47.62%


Execution Times:
Brute Force: 0.0325 seconds
Apriori: 0.0024 seconds

The fastest algorithm is: Apriori
```

Figure 1: Output for 1- Amazon dataset

Sample Output of K-mart dataset:


Amazon_Itemset.csv yesterday
Amazon_Transacti... yesterday
Bestbuy_Itemset.csv yesterday
Bestbuy_Transacti... yesterday
General_Itemset.csv yesterday
General_Transacti... yesterday
K_mart_Itemset.csv yesterday
K_mart_Transactio... yesterday
Nike_Itemset.csv yesterday
Nike_Transaction.c... yesterday
rrr62_midproject.i... 2m ago
rrr62_midproject.py yesterday

```
Available stores:
1. Amazon
2. Bestbuy
3. General
4. K-mart
5. Nike
Enter the number of the store you want to analyze:  4
Enter the minimum support (as a percentage between 0 and 100):  50
Enter the minimum confidence (as a percentage between 0 and 100):  70

Running Brute Force Algorithm...
Brute Force Time: 0.0514 seconds

Brute Force Results:
Frequent Itemsets:
Items: {'Bed Skirts'}, Support: 55.00%
Items: {'Decorative Pillows'}, Support: 50.00%
Items: {'Kids Bedding'}, Support: 60.00%
Items: {'Shams'}, Support: 55.00%
Items: {'Sheets'}, Support: 50.00%
Items: {'Bed Skirts', 'Kids Bedding'}, Support: 50.00%
Items: {'Kids Bedding', 'Sheets'}, Support: 50.00%

Association Rules:
Rule: {'Bed Skirts'} -> {'Kids Bedding'}
Confidence: 90.91%, Support: 50.00%

Rule: {'Kids Bedding'} -> {'Bed Skirts'}
Confidence: 83.33%, Support: 50.00%

Rule: {'Kids Bedding'} -> {'Sheets'}
Confidence: 83.33%, Support: 50.00%

Rule: {'Sheets'} -> {'Kids Bedding'}
Confidence: 100.00%, Support: 50.00%


Running Apriori Algorithm...
Apriori Time: 0.0058 seconds

Apriori Results:
Frequent Itemsets:
Items: {'Bed Skirts'}, Support: 55.00%
Items: {'Decorative Pillows'}, Support: 50.00%
Items: {'Kids Bedding'}, Support: 60.00%
Items: {'Shams'}, Support: 55.00%
Items: {'Sheets'}, Support: 50.00%
Items: {'Bed Skirts', 'Kids Bedding'}, Support: 50.00%
Items: {'Kids Bedding', 'Sheets'}, Support: 50.00%

Association Rules:
Rule: {'Bed Skirts'} -> {'Kids Bedding'}
Confidence: 90.91%, Support: 50.00%

Rule: {'Kids Bedding'} -> {'Bed Skirts'}
```

```
Items: {'Bed Skirts', 'Kids Bedding'}, Support: 50.00%
Items: {'Kids Bedding', 'Sheets'}, Support: 50.00%

Association Rules:
Rule: {'Bed Skirts'} -> {'Kids Bedding'}
Confidence: 90.91%, Support: 50.00%

Rule: {'Kids Bedding'} -> {'Bed Skirts'}
Confidence: 83.33%, Support: 50.00%

Rule: {'Kids Bedding'} -> {'Sheets'}
Confidence: 83.33%, Support: 50.00%

Rule: {'Sheets'} -> {'Kids Bedding'}
Confidence: 100.00%, Support: 50.00%


Execution Times:
Brute Force: 0.0514 seconds
Apriori: 0.0058 seconds

The fastest algorithm is: Apriori
```

```
[11]: 1

[11]: 1

[12]: 2

[12]: 2

[ ]:

[ ]:
```

Figure 2: Output for 4- K mart dataset

GitHub Repository Link: https://github.com/Risha-20/Midterm_project_rrr62.git