

Joseph Marquez

CS634

Github Link: <https://github.com/jvm2NJIT/CS634-Midterm-Project>

File Overview

screenshots folder: Screenshots of code that are used in this report

Five Database#.csv files: The databases that were created for this project

part1.ipynb: My code for the creation of the 5 databases

part2.ipynb: My code for brute force

part3.ipynb: My code comparing

MidtermProject.py: The actual executable project for you to run

MidtermReport.pdf: Report of the project

How to Run

1. Extract files
2. Run MidtermProject.py

Part 1

I decided to create my five databases myself. Every transaction in my five databases consists of ten common grocery store items. The generation of transactions was entirely random to ensure there were no biases in future parts of the project.

Part 2

I then created my brute force method, which loops through every possible transaction that could be made with the ten specific items and crosschecked them with the inputted database and inputted support to determine if it was a frequent itemset. It then extracted every possible association rule that could be created from the frequent itemsets, and checked if they were association rules with the inputted confidence level.

Part 3

I simply used this part to compare my results when using my brute force method with the existing implementation of the Apriori and FP Tree methods from the mlxtend package. Using Database1.csv, support=0.35, and confidence=0.4, I found my results to be the same for all three methods.

MidtermProject.py

Run this file in the console and input your desired database, support, and confidence. The program will output every frequent itemset and association rule according to your specifications, and then output the runtime of all three methods.

Here are some screenshots showing that the output for the three methods is the same:

<pre>----- Brute Force Time Taken: 0.5288600921630859 seconds ('Apple',): Support 0.5 ('Banana',): Support 0.55 ('Eggs',): Support 0.65 ('Soap',): Support 0.6 ('Water',): Support 0.65 ('Yogurt',): Support 0.55 ('Soap', 'Water'): Support 0.5 ('Soap',) --> ('Water',): Confidence 0.8333333333333334 ('Water',) --> ('Soap',): Confidence 0.7692307692307692 ----- Apriori Time Taken: 0.018670082092285156 seconds ('Apple',): Support 0.5 ('Banana',): Support 0.55 ('Eggs',): Support 0.65 ('Soap',): Support 0.6 ('Water',): Support 0.65 ('Yogurt',): Support 0.55 ('Soap', 'Water'): Support 0.5 ('Soap',) --> ('Water',): Confidence 0.8333333333333334 ('Water',) --> ('Soap',): Confidence 0.7692307692307692 ----- FP Tree Time Taken: 0.017527103424072266 seconds ('Water',): Support 0.65 ('Eggs',): Support 0.65 ('Soap',): Support 0.6 ('Apple',): Support 0.5 ('Yogurt',): Support 0.55 ('Banana',): Support 0.55 ('Soap', 'Water'): Support 0.5 ('Soap',) --> ('Water',): Confidence 0.8333333333333334 ('Water',) --> ('Soap',): Confidence 0.7692307692307692 -----</pre>	<pre>----- Brute Force Time Taken: 0.37562131881713867 seconds ('Eggs',): Support 0.7 ('Milk',): Support 0.75 ('Shampoo',): Support 0.65 ('Soap',): Support 0.8 ('Sugar',): Support 0.65 ('Milk', 'Soap'): Support 0.65 ('Milk',) --> ('Soap',): Confidence 0.8666666666666667 ('Soap',) --> ('Milk',): Confidence 0.8125 ----- Apriori Time Taken: 0.014174222946166992 seconds ('Eggs',): Support 0.7 ('Milk',): Support 0.75 ('Shampoo',): Support 0.65 ('Soap',): Support 0.8 ('Sugar',): Support 0.65 ('Milk', 'Soap'): Support 0.65 ('Milk',) --> ('Soap',): Confidence 0.8666666666666667 ('Soap',) --> ('Milk',): Confidence 0.8125 ----- FP Tree Time Taken: 0.008939266204833984 seconds ('Soap',): Support 0.8 ('Eggs',): Support 0.7 ('Sugar',): Support 0.65 ('Milk',): Support 0.75 ('Shampoo',): Support 0.65 ('Milk', 'Soap'): Support 0.65 ('Milk',) --> ('Soap',): Confidence 0.8666666666666667 ('Soap',) --> ('Milk',): Confidence 0.8125 -----</pre>
--	---

After doing many tests using different databases, support values, and confidence values,

here are my results for the runtime of the three methods.

By far, the slowest method was Brute Force. The efficiency for Apriori and FP Tree

depended on the inputted support and confidence values.

In this case, FP Tree was actually slower than Apriori when both support and confidence were 0.5. When the confidence was raised, the performance for FP Tree significantly decreased.

```
Database1.csv --- Support: 0.5 --- Confidence: 0.5
Runtime of the 3 methods:
Brute Force: 0.4671168327331543 seconds
Apriori: 0.0153350830078125 seconds
FP Tree: 0.07620930671691895 seconds
```

```
Database1.csv --- Support: 0.5 --- Confidence: 0.8
Runtime of the 3 methods:
Brute Force: 0.48352766036987305 seconds
Apriori: 0.01503443717956543 seconds
FP Tree: 0.00919795036315918 seconds
```

In most cases, however, FP Tree was consistently faster than Apriori. Here are some screenshots showing the runtime comparison using various databases, support values, and confidence values.

```
-----
Database2.csv --- Support: 0.4 --- Confidence: 0.75
Runtime of the 3 methods:
Brute Force: 0.462252140045166 seconds
Apriori: 0.02149343490600586 seconds
FP Tree: 0.014642477035522461 seconds
```

Brute Force

Time Taken: 0.5023200511932373 seconds

Apriori

Time Taken: 0.00942373275756836 seconds

FP Tree

Time Taken: 0.0010001659393310547 seconds

Database1.csv --- Support: 0.9 --- Confidence: 0.3

Runtime of the 3 methods:

Brute Force: 0.5023200511932373 seconds

Apriori: 0.00942373275756836 seconds

FP Tree: 0.0010001659393310547 seconds

Database3.csv --- Support: 0.5 --- Confidence: 0.8

Runtime of the 3 methods:

Brute Force: 0.502882719039917 seconds

Apriori: 0.020616769790649414 seconds

FP Tree: 0.003720998764038086 seconds

```
-----  
Database4.csv --- Support: 0.65 --- Confidence: 0.8  
Runtime of the 3 methods:  
Brute Force: 0.37562131881713867 seconds  
Apriori: 0.014174222946166992 seconds  
FP Tree: 0.008939266204833984 seconds
```

```
-----  
Database5.csv --- Support: 0.4 --- Confidence: 0.6  
Runtime of the 3 methods:  
Brute Force: 0.41196417808532715 seconds  
Apriori: 0.01277017593383789 seconds  
FP Tree: 0.005632162094116211 seconds
```