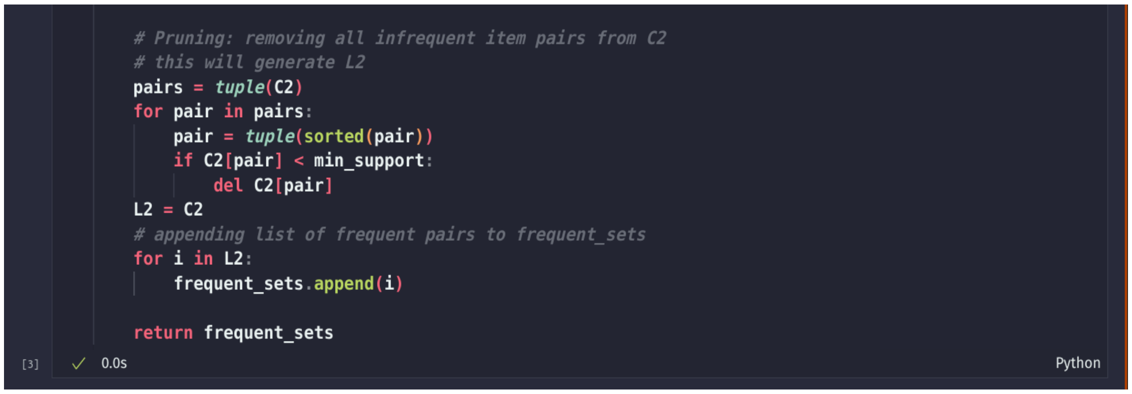
CSCI 4030U: Big Data Analytics
Labs 4 and 5

Syed Naqvi
100590852

February 24, 2024

Lab 4

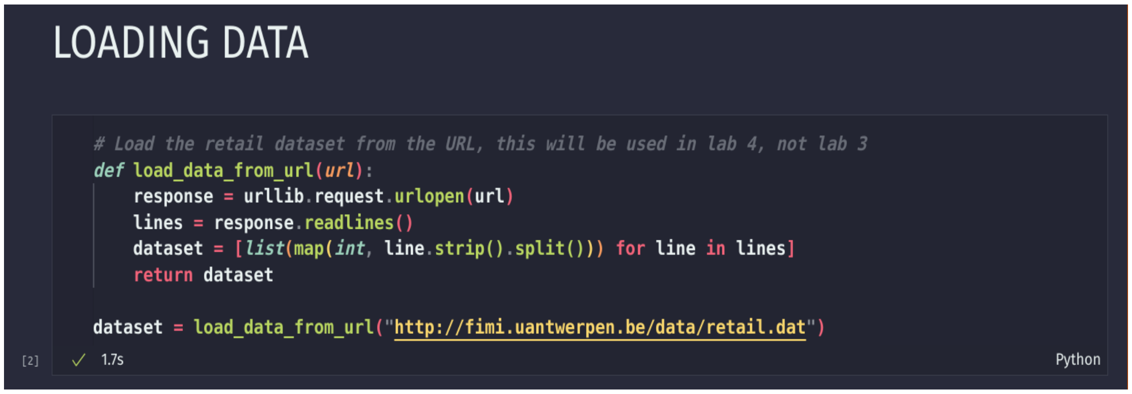
a. Limiting the Apriori algorithm to only find up to frequent pairs:



```python
    # Pruning: removing all infrequent item pairs from C2
    # this will generate L2
    pairs = tuple(C2)
    for pair in pairs:
        pair = tuple(sorted(pair))
        if C2[pair] < min_support:
            del C2[pair]
    L2 = C2
    # appending list of frequent pairs to frequent_sets
    for i in L2:
        frequent_sets.append(i)

    return frequent_sets
```

[3] ✓ 0.0s Python

b. Downloading retail dataset for the PCY and Apriori algorithms:

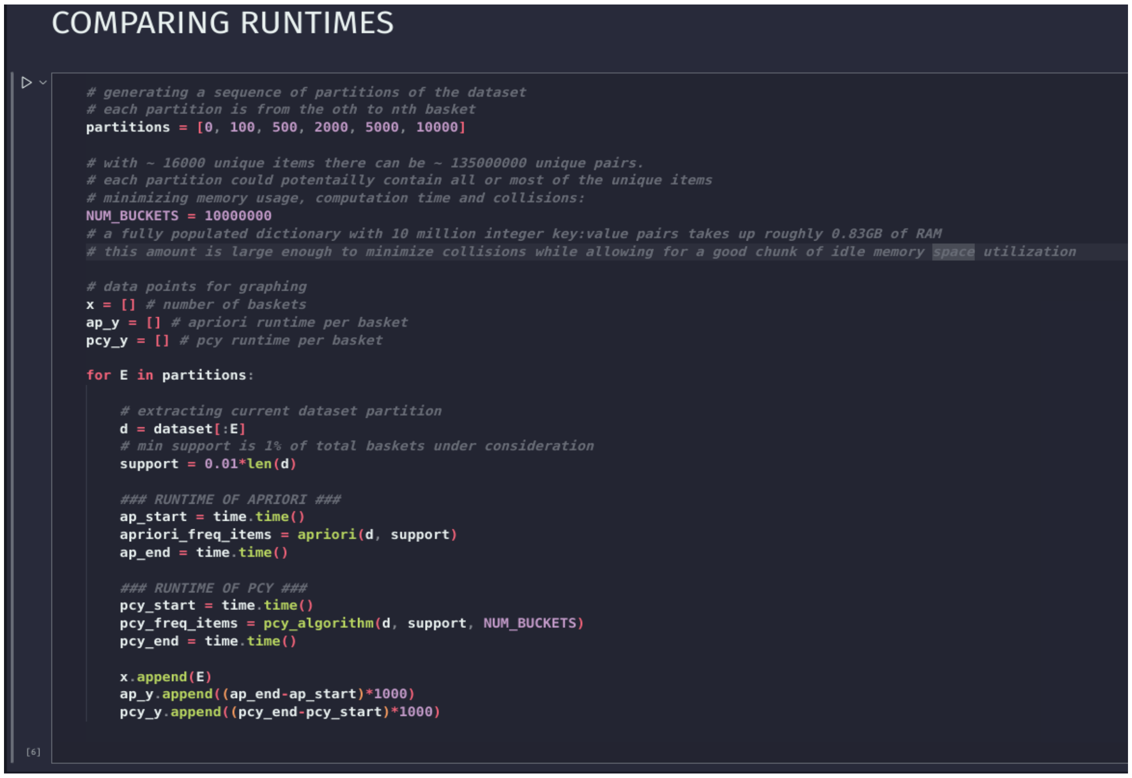


LOADING DATA

```python
# Load the retail dataset from the URL, this will be used in lab 4, not lab 3
def load_data_from_url(url):
    response = urllib.request.urlopen(url)
    lines = response.readlines()
    dataset = [list(map(int, line.strip().split())) for line in lines]
    return dataset

dataset = load_data_from_url("http://fimi.uantwerpen.be/data/retail.dat")
```

[2] ✓ 1.7s Python

c. Comparing Apriori and PCY algorithms using the provided partitions of the dataset:

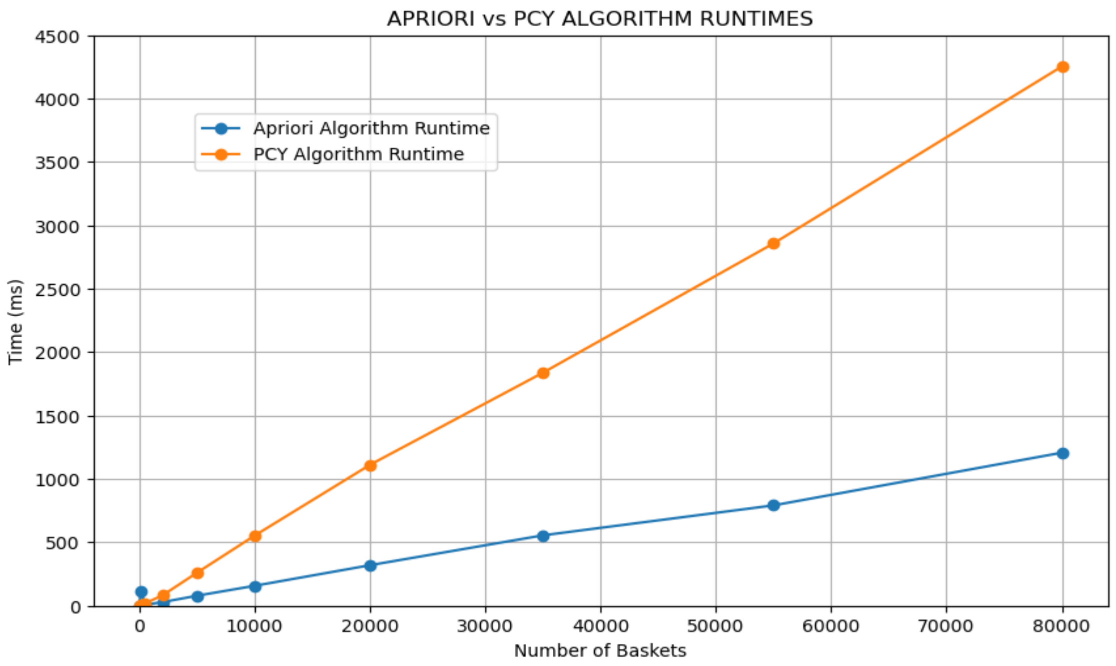


COMPARING RUNTIMES

```
# generating a sequence of partitions of the dataset
# each partition is from the oth to nth basket
partitions = [0, 100, 500, 2000, 5000, 10000]

# with ~ 16000 unique items there can be ~ 135000000 unique pairs.
# each partition could potentailly contain all or most of the unique items
# minimizing memory usage, computation time and collisions:
NUM_BUCKETS = 10000000
# a fully populated dictionary with 10 million integer key:value pairs takes up roughly 0.83GB of RAM
# this amount is large enough to minimize collisions while allowing for a good chunk of idle memory space utilization

# data points for graphing
x = [] # number of baskets
ap_y = [] # apriori runtime per basket
pcy_y = [] # pcy runtime per basket

for E in partitions:

    # extracting current dataset partition
    d = dataset[:E]
    # min support is 1% of total baskets under consideration
    support = 0.01*len(d)

    ### RUNTIME OF APRIORI ###
    ap_start = time.time()
    apriori_freq_items = apriori(d, support)
    ap_end = time.time()

    ### RUNTIME OF PCY ###
    pcy_start = time.time()
    pcy_freq_items = pcy_algorithm(d, support, NUM_BUCKETS)
    pcy_end = time.time()

    x.append(E)
    ap_y.append((ap_end-ap_start)*1000)
    pcy_y.append((pcy_end-pcy_start)*1000)
```

[6]

d. Graphing results. Dataset size is on the x-axis while runtime (ms) is on y-axis:



e. All screenshots have been provided.

f. The runtimes of both algorithms seem to increase roughly linearly with increasing dataset size. The PCY algorithm is more computationaly expensive as evident by the steeper line. This faster increase in runtime is due to the additional hashing step that is not part of the Apriori algorithm. Although more computationaly expensive, the PCY algorithm is less spatially expensive thanks to the stringent candidacy requirements it imposes on potential frequent item pairs.

Lab 5

a. The Jaccard similarity of two sets is the size of their intersection divided by the union. We have the following sets:

$$\begin{aligned} C_1 &= \{2,3,4,5\} \\ C_2 &= \{3,4,6,8\} \\ C_3 &= \{2,3,6\} \end{aligned}$$

We can now calculate the Jaccard similarity of each pair of the above sets:

$$\begin{aligned} sim(C_1,C_2) &= \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|} \\ &= \frac{|\{2,3,4,5\} \cap \{3,4,6,8\}|}{|\{2,3,4,5\} \cup \{3,4,6,8\}|} \\ &= \frac{|\{3,4\}|}{|\{2,3,4,5,6,8\}|} \\ &= \frac{2}{6} \\ &= \frac{1}{3} = 0.3\overline{3} \end{aligned}$$

$$\begin{aligned} sim(C_1,C_3) &= \frac{|C_1 \cap C_3|}{|C_1 \cup C_3|} \\ &= \frac{|\{2,3,4,5\} \cap \{2,3,6\}|}{|\{2,3,4,5\} \cup \{2,3,6\}|} \\ &= \frac{|\{2,3\}|}{|\{2,3,4,5,6\}|} \\ &= \frac{2}{5} = 0.4 \end{aligned}$$

$$\begin{aligned} sim(C_2,C_3) &= \frac{|C_2 \cap C_3|}{|C_2 \cup C_3|} \\ &= \frac{|\{3,4,6,8\} \cap \{2,3,6\}|}{|\{3,4,6,8\} \cup \{2,3,6\}|} \\ &= \frac{|\{3,6\}|}{|\{2,3,4,6,8\}|} \\ &= \frac{2}{5} = 0.4 \end{aligned}$$

b. **Answer:** The expected value of the jaccard similarity would be $frac_m n$.

Reasoning: