Finding frequent items

Groupe 71 : Alga Nour Elimane , Karim Haque

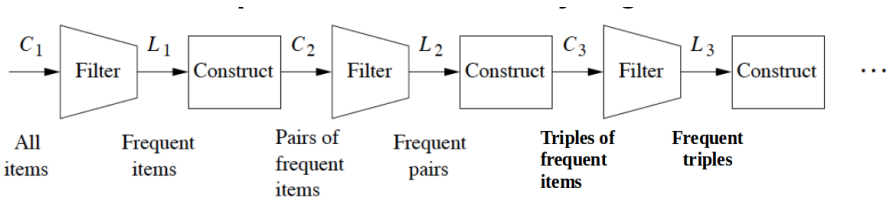
November 2025

1 Introduction :

The **Apriori** algorithm is a classic data mining method used to discover *frequent itemsets* in transaction data. An itemset is considered frequent if its *support* exceeds a user-defined threshold s. Apriori is widely used in market basket analysis, product recommendation, and association rule mining.

Algorithm Pipeline

Apriori works iteratively and relies on the **monotonicity property of support**: *any subset of a frequent itemset must also be frequent.*



For each itemset size k, the algorithm constructs two sets:

- C_k: the set of *candidate k-tuples*, i.e., itemsets that *might* be frequent (support $> s$) based on the frequent itemsets from the previous pass ($k-1$).
- L_k: the set of *truly frequent k-tuples*, obtained by filtering only those candidates in C_k whose support is at least s.

The process continues by incrementing k until no new frequent itemsets are found. This approach significantly reduces the number of candidates to check and optimizes memory usage.

2 How to run :

There is one file with all the classes and the main test : Aprior.py you just have to execute it : **python3 Aprior.py**
As you can see there is also a jupyternotebook where some Experiments and plots have been shown

3 A prior Algorithm implementation

We create A prior class that is structured around three main methods:

3.1 First pass : pass1

The method `pass1()` corresponds to the first step of the Apriori algorithm .Its goal is to identify all frequent 1-itemsets.

The method proceeds as follows:

1. It initializes a dictionary `item_count` that maps each item to the number of times it appears in the dataset. This dictionary is implemented using `defaultdict(int)`, which automatically starts all counts at 0.

2. It then iterates through each "basket" in the dataset (transaction). For every basket, the method loops over each item it contains and increments its count in the dictionary:

$$item\_count[item] \leftarrow item\_count[item] + 1.$$

3. After scanning all baskets, the method filters the dictionary and keeps only the items whose support is greater than or equal to the minimum support threshold. Each such item is stored as a 1-itemset in the form of a tuple:

$$L_1 = \{(i) \mid item\_count[i] \geq min\_support\}.$$

4. The method stores the resulting frequent itemsets in `self.L[1]`, using a dictionary that maps each itemset to its support value.

5. Finally, the list of frequent 1-itemsets is returned.

3.2 Getting candidates : Optimazation

This function produces the candidate itemsets of size k from the frequent itemsets of size $k-1$. To make the algorithm faster, we use optimization methode stating that two $(k-1)$-itemsets can be joined only if their first $(k-2)$ elements are identical.

The method works in three main steps:

- All $(k-1)$-itemsets are sorted to allow lexicographic comparison.

- Two itemsets are joined only if they share their first $(k-2)$ items. This avoids generating useless candidates.
- Before keeping a candidate, all its $(k-1)$ subsets are checked to ensure they are frequent .

The function returns the final list of valid candidate k-itemsets.

3.3 Generalization of the A prior Algorithme :

This method generalizes the first pass of Apriori to any $k \geq 2$. It takes the frequent $(k-1)$-itemsets and produces the frequent k-itemsets. **Key steps:**

1. Generate candidate k-itemsets C_k from L_{k-1}.
2. Count support of each candidate in all baskets.
3. Keep only candidates whose support $\geq$ threshold to form L_k.

we stock all L_k in self.L

4 Some results :

```
======================================================================
Number of transactions: 100000

======================================================================
EXECUTING A-PRIORI (all passes)
======================================================================
Running A-Priori with support threshold = 1000
Pass 1: Finding frequent 1-itemsets...
  Found 375 frequent 1-itemsets
Pass 2: Finding frequent 2-itemsets...
  Found 9 frequent 2-itemsets
Pass 3: Finding frequent 3-itemsets...
  Found 1 frequent 3-itemsets
Pass 4: Finding frequent 4-itemsets...
  No frequent 4-itemsets found. Stopping.
```

As we can see here we choose our support threshold equals 1000 and after each pass we show how many frequent items we find we run the algorithm until there is no frequent itemsets found In the Jupyter notebook we ran more experiments to see how the threshold affect the number of items found and the execution time please take a look at it

5 Bonus part : Association Rules

5.1 Association Rules Generation

After finding all frequent itemsets with the Apriori algorithm, we generate **association rules** of the form $X \Rightarrow Y$, meaning "if X occurs, then Y also occurs". Rules are only kept if their **confidence** is above a threshold c:

$$confidence(X \Rightarrow Y) = \frac{support(X \cup Y)}{support(X)}$$

Algorithm steps:

1. Combine all frequent itemsets from all passes $(L_1, L_2, ...)$ into a single dictionary for fast support lookup.
2. Consider only itemsets of size ≥ 2, since smaller sets cannot form rules.
3. For each itemset, generate all possible **antecedents** X and corresponding **consequents** $Y = itemset - X$.
4. Compute confidence for each rule $X \Rightarrow Y$ and keep only those with $confidence \geq c$.
5. Store valid rules in a dictionary mapping antecedents to consequents.

Output: A dictionary containing all valid association rules.

```
{704} => {39} | conf=0.6171
{704} => {825} | conf=0.6143
{704, 825} => {39} | conf=0.9392
{704, 39} => {825} | conf=0.9350
{825, 39} => {704} | conf=0.8719
```

6 Expériments: Look at JupyterNotebook