LECTURE 25

Dr. Vani V

Recap

- Item set
- Support Count
- Support
- Frequent Itemset
- ARM – Brute Force approach
- Apriori Principle – Illustration
- Apriori Algorithm - Example

Algorithm 6.1 Frequent itemset generation of the *Apriori* algorithm.

1: $k = 1$.
2: $F_k = \{ i \mid i \in I \wedge \sigma(\{i\}) \geq N \times minsup \}$. {Find all frequent 1-itemsets}
3: **repeat**
4: $k = k + 1$.
5: $C_k = \text{apriori-gen}(F_{k-1})$. {Generate candidate itemsets}
6: **for** each transaction $t \in T$ **do**
7: $C_t = \text{subset}(C_k, t)$. {Identify all candidates that belong to t}
8: **for** each candidate itemset $c \in C_t$ **do**
9: $\sigma(c) = \sigma(c) + 1$. {Increment support count}
10: **end for**
11: **end for**
12: $F_k = \{ c \mid c \in C_k \wedge \sigma(c) \geq N \times minsup \}$. {Extract the frequent k-itemsets}
13: **until** $F_k = \emptyset$
14: Result $= \bigcup F_k$.

Frequent itemset Generation

- The pseudocode for the frequent itemset generation part of the Apriori algorithm is shown in Algorithm 6.1.
- Let C_k denote the set of candidate k-itemsets and F_k denote the set of frequent k-itemsets: The algorithm initially makes a single pass over the data set to determine the support of each item. Upon completion of this step, the set of all frequent 1-itemsets, F_1, will be known (steps 1 and 2).
- Next, the algorithm will iteratively generate new candidate k-itemsets using the frequent $(k - 1)$-itemsets found in the previous iteration (step 5). Candidate generation is implemented using a function called apriorigen
- To count the support of the candidates, the algorithm needs to make an additional pass over the data set (steps 6–10). The subset function is used to determine all the candidate itemsets in C_k that are contained in each transaction t.

Candidate Generation and Pruning

The **apriori-gen function** shown in Step 5 of Algorithm 6.1 **generates candidate itemsets by performing the following two operations**

1. **Candidate Generation.** This operation generates new candidate k itemsets based on the frequent (k − 1)-itemsets found in the previous iteration.

2. **Candidate Pruning.** This operation eliminates some of the candidate k-itemsets using the support-based pruning strategy.

Candidate Pruning

To illustrate the **candidate pruning operation**,

1. Consider a candidate k-itemset, $X = \{i_1, i_2, \ldots, i_k\}$.
2. The algorithm must determine whether all its proper subsets, $X - \{i_j\}$ ($\forall_j = 1, 2, \ldots, k$), are frequent.
3. If one of them is infrequent, then X is immediately pruned. This approach can effectively reduce the number of candidate itemsets considered during support counting.
4. The complexity of this operation is **O(k)** for each candidate k-itemset.
5. **Note :** we do not have to examine all k subsets of a given candidate itemset. If m of the k subsets were used to generate a candidate, we only need to check the remaining k −m subsets during candidate pruning

Candidate Generation…

In principle, there are many ways to generate candidate itemsets. The following is a list of requirements for an effective candidate generation procedure:

1. It **should avoid generating too many unnecessary candidates.**
2. A candidate itemset is unnecessary if at least one of its subsets is infrequent. Such a candidate is guaranteed to be infrequent according to the antimonotone property of support.
3. It must ensure that the candidate set is complete, i.e., **no frequent itemsets are left out by the candidate generation procedure.**

Candidate Generation…

1. **To ensure completeness, the set of candidate itemsets must subsume the set of all frequent itemsets, i.e., $\forall_k : F_k \subseteq C_k$.**
2. It should not generate the same candidate itemset more than once. For example, the candidate itemset {a, b, c, d} can be generated in many ways—by merging {a, b, c} with {d}, {b, d} with {a, c}, {c} with {a, b, d}, etc. Generation of duplicate candidates leads to wasted computations and thus should be avoided for efficiency reasons.

Candidate Generation Procedure 1…

Brute-Force Method

The brute-force method considers every k-itemset as a potential candidate and then applies the candidate pruning step to remove any unnecessary candidates.

The number of candidate itemsets generated at level k is equal to dC_k, where d is the total number of items.

Although candidate generation is rather trivial, candidate pruning becomes extremely expensive because many itemsets must be examined.

Given that the number of computations needed for each candidate is O(k),the overall complexity of this method is

$$O\left(\sum_{k=1}^{d} k \times \binom{d}{k}\right) = O\left(d \cdot 2^{d-1}\right).$$

Brute-Force Method

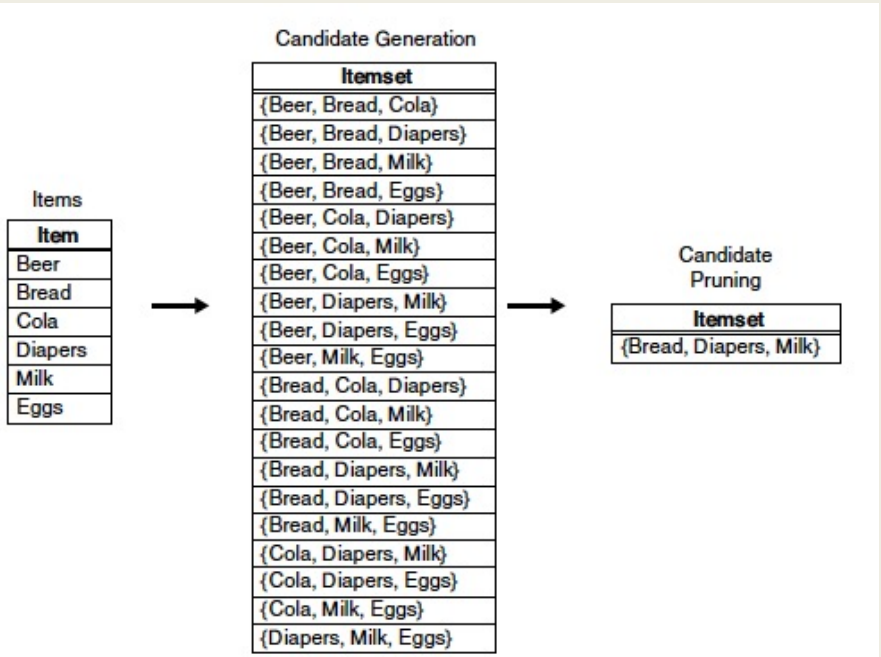

Figure 6.6. A brute-force method for generating candidate 3-itemsets.

Candidate Generation Procedure 2…

- An alternative method for candidate generation is to extend each frequent (k − 1)-itemset with other frequent items.

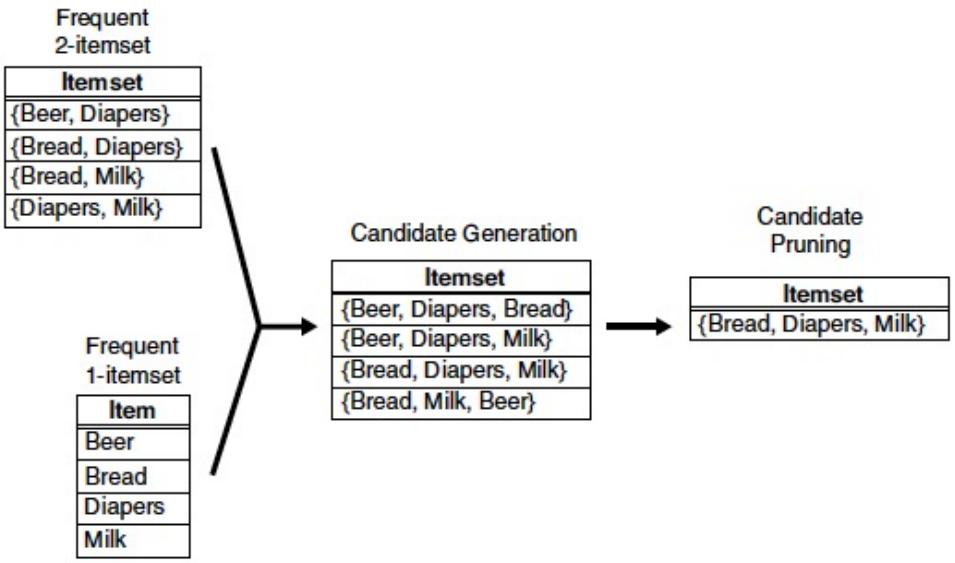

Figure 6.7. Generating and pruning candidate k-itemsets by merging a frequent $(k-1)$-itemset with a frequent item. Note that some of the candidates are unnecessary because their subsets are infrequent.

F_{k-1} x F_1 Method…

Figure illustrates how a frequent 2-itemset such as {Beer, Diapers} can be augmented with a frequent item such as Bread to produce a candidate 3-itemset {Beer, Diapers, Bread}. This method will produce $O(|F_{k-1}| x |F_1|)$ candidate k-itemsets, where $|F_j|$ is the number of frequent j-itemsets.

The overall complexity of this step is $O(\sum_k k|F_{k-1}||F_1|)$.

The procedure is complete because every frequent k-itemset is composed of a frequent (k − 1)-itemset and a frequent 1-itemset. Therefore, all frequent k-itemsets are part of the candidate k-itemsets generated by this procedure.

F_{k-1} x F_1 Method…

- This approach, however, does not prevent the same candidate itemset from being generated more than once.
- For instance, {Bread, Diapers, Milk} can be generated by merging {Bread, Diapers} with {Milk}, {Bread, Milk} with {Diapers}, or {Diapers, Milk} with {Bread}.
- One way to avoid generating duplicate candidates is by ensuring that the items in **each frequent itemset are kept sorted in their lexicographic order.**

F_{k-1} x F_1 Method…

- Each frequent (k−1)-itemset X is then extended with frequent items that are lexicographically larger than the items in X. For example, the itemset {Bread, Diapers} can be augmented with {Milk} since Milk is lexicographically larger than Bread and Diapers.

- However, we should not augment {Diapers, Milk} with {Bread} nor {Bread, Milk} with {Diapers} because they violate the lexicographic ordering condition.

- While **this procedure is a substantial improvement over the brute-force method, it can still produce a large number of unnecessary candidates.**

F_{k-1} x F_1 Method

For example, the candidate itemset obtained by merging {Beer, Diapers} with {Milk} is unnecessary because one of its subsets, {Beer, Milk}, is infrequent.

There are several heuristics available to reduce the number of unnecessary candidates. For example, note that, for every candidate k-itemset that survives the pruning step, every item in the candidate must be contained in at least $k - 1$ of the frequent $(k - 1)$-itemsets.

Otherwise, the candidate is guaranteed to be infrequent. For example, {Beer, Diapers, Milk} is a viable candidate 3-itemset only if every item in the candidate, including Beer, is contained in at least two frequent 2-itemsets. Since there is only one frequent 2-itemset containing Beer, all candidate itemsets involving Beer must be infrequent.

Candidate Generation Procedure 3…

- F_{k-1} x F_{k-1} Method

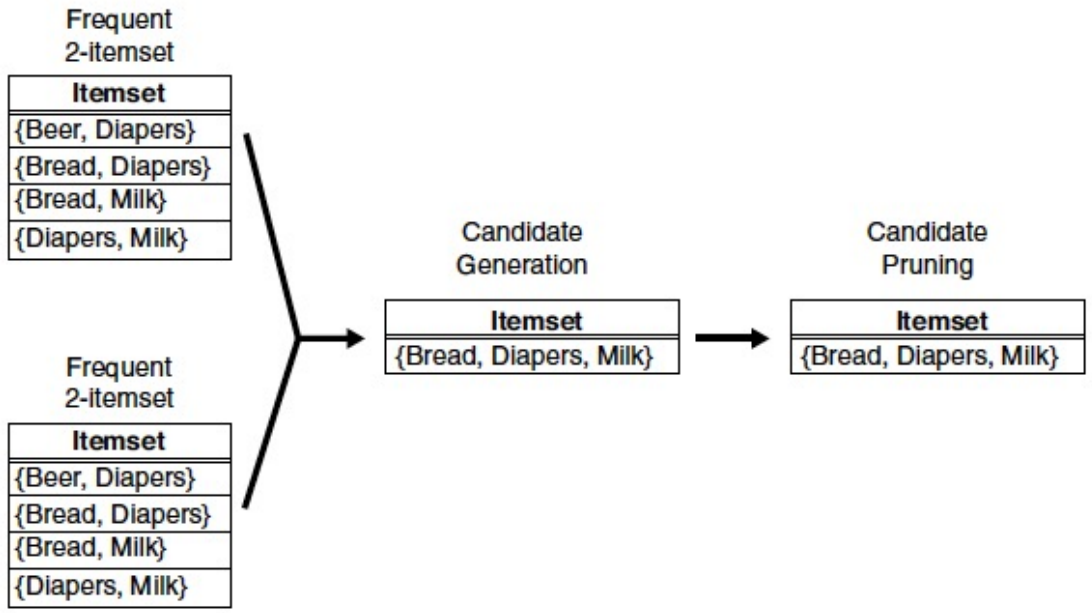

Figure 6.8. Generating and pruning candidate k-itemsets by merging pairs of frequent $(k-1)$-itemsets.

F_{k-1} x F_{k-1} Method...

- The candidate generation procedure in the apriori-gen function merges a pair of frequent (k–1)-itemsets only if their first k–2 items are identical.
- Let A = $\{a_1, a_2, \ldots, a_{k-1}\}$ and B = $\{b_1, b_2, \ldots, b_{k-1}\}$ be a pair of frequent (k – 1)-itemsets. A and B are merged if they satisfy the following conditions:

$$a_i = b_i \text{ (for } i = 1, 2, \ldots, k-2) \text{ and } a_{k-1} \neq b_{k-1}.$$

F_{k-1} x F_{k-1} Method…

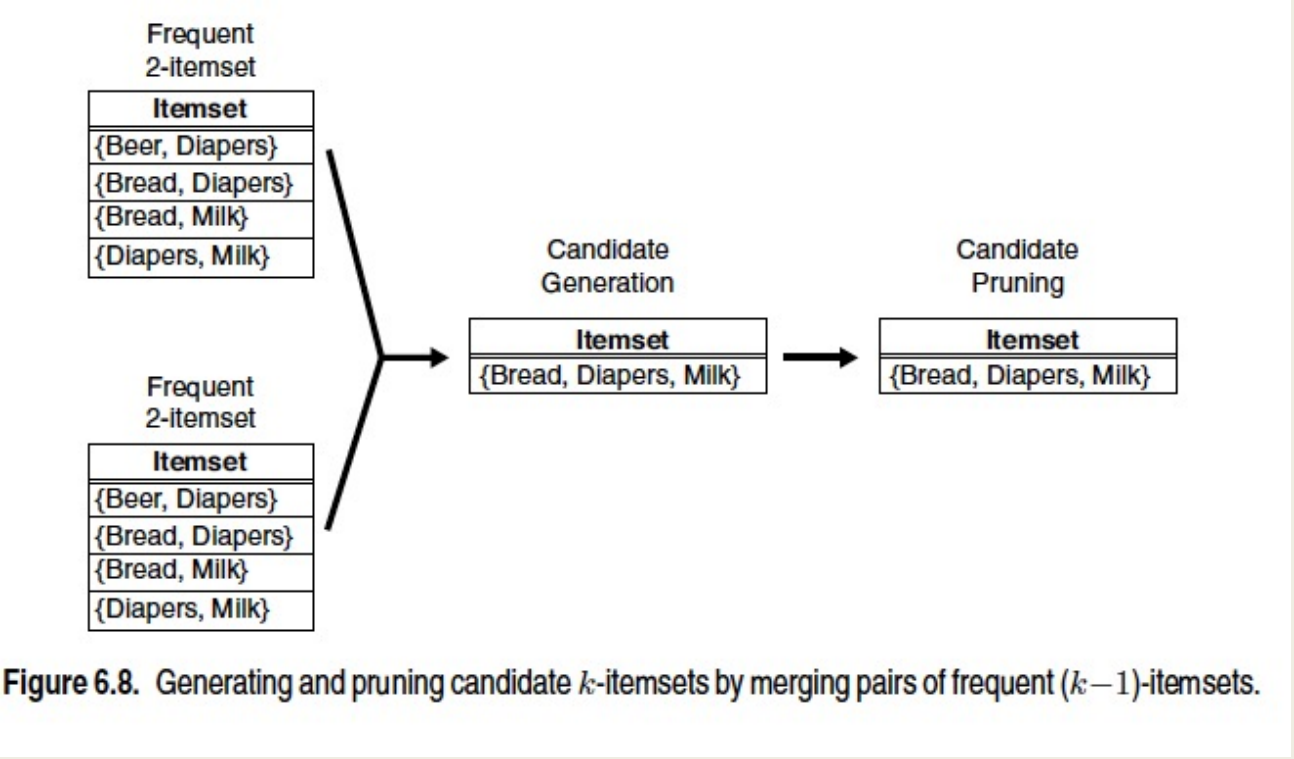

Figure 6.8. Generating and pruning candidate k-itemsets by merging pairs of frequent $(k-1)$-itemsets.

F_{k-1} x F_{k-1} Method

- In Figure 6.8, the frequent itemsets {Bread, Diapers} and {Bread, Milk} are merged to form a candidate 3-itemset {Bread, Diapers, Milk}.
- The algorithm does not have to merge {Beer, Diapers} with {Diapers, Milk} because the first item in both itemsets is different.
- If {Beer, Diapers, Milk} is a viable candidate, it would have been obtained by merging {Beer, Diapers} with {Beer, Milk} instead.
- This example **illustrates both the completeness of the candidate generation procedure and the advantages of using lexicographic** ordering to prevent duplicate candidates.
- Because each candidate is obtained by merging a pair of frequent (k−1)-itemsets, an additional candidate pruning step is needed to ensure that the remaining k − 2 subsets of the candidate are frequent.

References

- TEXTBOOKS :

1. Pang-Ning Tan, Vipin Kumar, Michael Steinbach: **Chapter 6, Introduction to Data Mining**, Pearson, 2012.
2. Jiawei Han and Micheline Kamber: **Chapter 6, Data Mining - Concepts and Techniques**, 3rd Edition, MorganKaufmann Publisher, 2014