Why we need support:

Cause if we use confidence only some of the rules might ge produced by chance. So support helps us to find itemsets that people seldom buy together so that we can generate association rules out of them.

Confidence:

Reliability of the inference made by the rule. Higher thr confidence, the more likely it is for Y to be present in the transactions that contain X.

Total possible rules: 3^d - 2 ^ (d + 1) + 1

X -> Y only depends upon support of (xUy)

if support if (x U y) is less than all the 2 (|x| + |y| - 1) rules generated will waste computing power.

So problem is divided into two parts:

1. Frequent itemset generation

2. Rule generation

Frequent Item set generation:

O(N\*M\*w) N -> transactions, M -> itemsets w -> max width of itemset.

So two ways:

1. Reduce M

2. Reduce # of comparisons for finding support.

The Apriori principle:

if an itemset is frequent them all of its subsets must be frequent.

Conversely if item set is infrequent then all of its supersets are infrequent.

Support based pruning: Trimming exponential search space based on support measure.

Candidate generation and pruning:

1. Candidates -> Ck is set of all possible candidates.

2. Fk is set of frequent candidates:

Methods for candidate generation:

**1. Brute force method**:

O( sum(k\*dCk)) here K is size of itemset as this is the amount of computation required for each K size itemset.

**2.Fk-1 \* Fk  Method**: First sort each itemset in Fk-1 and pair it with only those entries in F1 which are lexicographically greater than Fk-1 to avoid duplicate candidate generation. This still generates unnecessary candidates cause the Fk-1  paired with F1 , it might happen that one of the item from Fk-1  when paired with F1  which generated F2  itemset has not required support.

3. **Fk-1 \* Fk-1  Method:**

Find pair of itemsets in Fk-1 (lexicographically sorted) whose first (k-2) items are equal i.e. they’ve got same prefix. If so merge them together. This ensures completeness and also avoids generating duplicate candidates (due to sorting). As we are just merging 2 of the all k possible (k-1) subsets of newly generated (k)-itemset we need next step to make sure that other K – 2 subsets of candidate are frequent

**Support counting using Hash Trees.**

Candidate itemsets are partitioned into different buckets and stored in hash tree. During support counting, itemsets contained in each transaction are also hashed into appropriate buckets. That way instead of comparing each transaction with every candidate itemset, it is matched only against candidate itemsets that belong to the same bucket.

**Computational complexity depends upon**:

1. **Threshold Support**: Size of C increases.

**2**. **Number of items**: Size of both C, F may increase, requires more space and IO cost will increase

3. **Number of transactions:** Since Apriori makes use number of passes on database

4. **Average width of transactions:** Increases hash tree traversals during support count phase.

**5. Generation of frequent 1 itemsets: O(N\*w) where w is average width**

**6. Candidate generation:**

**7. Support counting: O(N \* sum(k \* wCk \* alpha))**

Each transaction generates tCk itemsets of size K and each of which requires K steps to go down the hash tree and alpha is the cost associated with updating count of candidate inside bucket.