Apriori Algorithm

The Apriori Algorithm is an influential algorithm for mining frequent itemsets for boolean association rules.

Key Concepts :

Frequent Itemsets: The sets of item which has minimum support (denoted by Li for ith-Itemset).

Apriori Property: Any subset of frequent itemset must be frequent.

Join Operation: To find Lk , a set of candidate k-itemsets is generated by joining Lk-1 with itself.

Pseudo code

- Join Step: C_k is generated by joining L_{k-1} with itself
- Prune Step: Any (k-1)-itemset that is not frequent cannot be a subset of a frequent k-itemset
- Pseudo-code:

```
$C_k$: Candidate itemset of size k
$L_k$ : frequent itemset of size k

$L_1$ = {frequent items};
for ($k$ = 1; $L_k$ !=∅; $k$++) do begin
    $C_{k+1}$ = candidates generated from $L_k$;
    for each transaction $t$ in database do
        increment the count of all candidates in $C_{k+1}$
        that are contained in $t$
    $L_{k+1}$ = candidates in $C_{k+1}$ with min_support
end
return $\cup_k L_k$;
```

Advantages

1. Easy to understand algorithm
2. Join and Prune steps are easy to implement on large itemsets in large databases

Disadvantages

1. It requires high computation if the itemsets are very large and the minimum support is kept very low.
2. The entire database needs to be scanned.

Methods to Improve Apriori’s Efficiency

- **Hash-based itemset counting**: A *k*-itemset whose corresponding hashing bucket count is below the threshold cannot be frequent.
- **Transaction reduction**: A transaction that does not contain any frequent k-itemset is useless in subsequent scans.
- **Partitioning**: Any itemset that is potentially frequent in DB must be frequent in at least one of the partitions of DB.
- **Sampling**: mining on a subset of given data, lower support threshold + a method to determine the completeness.
- **Dynamic itemset counting**: add new candidate itemsets only when all of their subsets are estimated to be frequent.

Example of Apriori: **Support threshold=50%, Confidence= 60%**

Transaction	List of items
T1	I1,I2,I3
T2	I2,I3,I4
T3	I4,I5
T4	I1,I2,I4
T5	I1,I2,I3,I5
T6	I1,I2,I3,I4

Solution:

Support threshold=50% => 0.5*6= 3 => min_sup=3

1. Count Of Each Item

Item	Count
I1	4
I2	5
I3	4
I4	4
I5	2

2. Prune Step: TABLE -2 shows that I5 item does not meet min_sup=3, thus it is deleted, only I1, I2, I3, I4 meet min_sup count.

Item	Count
I1	4
I2	5
I3	4
I4	4

3. Join Step: Form 2-itemset. From **TABLE-1** find out the occurrences of 2-itemset.

Item	Count
I1,I2	4
I1,I3	3
I1,I4	2
I2,I3	4
I2,I4	3
I3,I4	2

4. Prune Step: TABLE -4 shows that item set {I1, I4} and {I3, I4} does not meet min_sup, thus it is deleted.

Item	Count
I1,I2	4
I1,I3	3
I2,I3	4
I2,I4	3

5. Join and Prune Step: Form 3-itemset. From the **TABLE- 1** find out occurrences of 3-itemset. From **TABLE-5**, find out the 2-itemset subsets which support min_sup.

We can see for itemset {I1, I2, I3} subsets, {I1, I2}, {I1, I3}, {I2, I3} are occurring in **TABLE-5** thus {I1, I2, I3} is frequent.

We can see for itemset {I1, I2, I4} subsets, {I1, I2}, {I1, I4}, {I2, I4}, {I1, I4} is not frequent, as it is not occurring in **TABLE-5** thus {I1, I2, I4} is not frequent, hence it is deleted.

Item
I1,I2,I3
I1,I2,I4
I1,I3,I4
I2,I3,I4

Only {I1, I2, I3} is frequent.

6. Generate Association Rules: From the frequent itemset discovered above the association could be:

{I1, I2} => {I3}

Confidence = support {I1, I2, I3} / support {I1, I2} = (3/ 4)* 100 = 75%

{I1, I3} => {I2}

Confidence = support {I1, I2, I3} / support {I1, I3} = (3/ 3)* 100 = 100%

{I2, I3} => {I1}

Confidence = support {I1, I2, I3} / support {I2, I3} = (3/ 4)* 100 = 75%

{I1} => {I2, I3}

Confidence = support {I1, I2, I3} / support {I1} = (3/ 4)* 100 = 75%

{I2} => {I1, I3}

Confidence = support {I1, I2, I3} / support {I2 = (3/ 5)* 100 = 60%

{I3} => {I1, I2}

Confidence = support {I1, I2, I3} / support {I3} = (3/ 4)* 100 = 75%

This shows that all the above association rules are strong if minimum confidence threshold is 60%.

Hash-based technique

(hashing itemsets into corresponding buckets)

A hash-based technique can be used to reduce the size of the candidate k-itemsets, Ck, for k > 1. For example, when scanning each transaction in the database to generate the frequent 1-itemsets, L1, we can generate all the 2-itemsets for each transaction, hash (i.e., map) them into the different buckets of a hash table structure, and increase the corresponding bucket counts (Figure 6.5). A 2-itemset with a corresponding bucket count in the hash table that is below the support threshold cannot be frequent and thus should be removed from the candidate set. Such a hash-based technique may substantially reduce the number of candidate k-itemsets examined (especially when k = 2).

Dynamic itemset counting

(adding candidate itemsets at different points during a scan)

A dynamic itemset counting technique was proposed in which the database is partitioned into blocks marked by start points. In this variation, new candidate itemsets can be added at any start point, unlike in Apriori, which determines new candidate itemsets only immediately before each complete database scan. The technique uses the count-so-far as the lower bound of the actual count. If the count-so-far passes the minimum support, the itemset is added into the frequent itemset collection and can be used to generate longer candidates. This leads to fewer database scans than with Apriori for finding all the frequent itemsets.

Reference Reading:

Book: Data Mining Concepts and Techniques - Morgan
Chapter 6 Mining Frequent Patterns, Associations, and Correlations