

# **Differential Virtual Support Algorithm** **for Association Rule Mining**

by

**Akash Saxena**

Dr B R Ambedkar National Institute of Technology

Under the guidance of

**Prof B Chandra**

Indian Institute of Technology-Delhi

## **Abstract:**

A major problem faced by existing algorithms for Association Rule mining is to discover frequent itemsets efficiently w.r.t time and space. Typical algorithms evaluate support for all itemsets by scanning through the database, transaction by transaction, which is very time consuming as I/O speed using normal storage devices is very less, and then predicting whether the itemset checked is frequent or not. Also it is a very cumbersome approach as you have to first store the itemset, then scan for its support, reading transaction by transaction doing pattern matching, and finally predict the frequent nature of the itemset, repeating the process for every possible itemset. When dealing with large databases having thousands of records and hundreds of items then such an approach can be rather very expensive. I, in this paper, present a new algorithm which scans the database just twice, irrespective of size of database and number of items, and predicts all frequent itemsets with a new measures, namely, *Differential Support* and *Virtual Support*, which are calculated using just the actual support counts of 1-itemsets. In the start, it scans database once for calculating support of 1-itemsets and then goes on to calculate virtual support for rest of the 2-itemsets, 3-itemsets and so on, using the actual support count of 1-itemsets evaluated initially. It predicts all frequent itemsets in its way of calculating virtual support counts and in the end re-scans database once more to calculate Actual support of each predicted frequent itemset. Therefore, just by doing 2 database scans, it predicts all possible frequent itemsets. I have evaluated the performance of this algorithm using well-known databases. The results have shown striking improvements as compared to other existing algorithms.

## **1. INTRODUCTION**

Today there are several algorithms dealing with popular and computationally expensive task of Association rule mining. A key methodology employed in all of these is, to scan a given large database of set of items, discover all frequent itemsets (set of items), where a frequent itemset is one that appears more frequently than a user-specified minimum frequency threshold (also called Minimum *Support* threshold), by repeated scans of database. Then on basis of results obtained, i.e. the set of all possible frequent itemsets, association rules are prepared.

Typical algorithms for finding the frequent itemsets, for each possible itemset, first save the itemset, evaluate its *support*, by scanning database, doing pattern matching at every transaction, and if its *support* is greater than Minimum *Support* threshold, then declare it as frequent else infrequent itemset. This approach proves reasonably efficient if size of database is small, size of memory available is large, and I/O speed with the storage device is very high. However, if any of these idealistic conditions do not exist, the performance of the algorithms decreases drastically.

However, if we have a technique to scan the database just once to evaluate actual support of 1-itemsets and then use this knowledge to predict frequent 2-itemsets, 3-

itemsets, etc., it can prove to be very time and space efficient. In this way, we rely more on CPU computations rather than cumbersome I/O computations, which are drastically faster. Also, by directly predicting frequent itemsets without storing them, but by just checking the features of their subsets, we save a large amount of memory space and time.

In this paper, I present a novel *Differential Virtual Support* Algorithm, which implements the above technique. The algorithm has a very high performance irrespective of size of database or number of items.

I will explain all concepts, methods and algorithm step by step. So I have organized the paper in an easily interpretable and sequential format. Section 2 explains the basic concepts of Association Rule Mining and various concepts related to my algorithm. Section 3 explains my algorithm and the idea behind it. Section 4 presents the results of experiments and tests I have put this algorithm through. Finally, section 5 concludes this paper. In the end I have included all the references.

## 2. ASSOCIATION RULE MINING

This section briefly discusses the Association Rule mining concept.

Since its introduction in 1993 [1] the task of association rule mining has received a great deal of attention. Today the mining of such rules is still one of the most popular pattern-discovery methods in KDD.

Let  $I = (i_1, i_2, \dots, i_m)$  be a set of  $m$  distinct items.

**Transaction**: A transaction  $T$  is defined as any subset of items in  $I$ .

**Database**: A database  $D$  is a set of transactions.

**Itemset**: A set of items is called an Itemset.

**Length of Itemset**: is the number of items in the itemset. Itemsets of length 'k' are referred to as k-itemsets.

**Actual Support Count**: It is the total frequency of appearance of a particular pattern in a database.

In other terms: If  $T$  is a transaction in database  $D$  and  $X$  is an itemset then  $T$  is said to *support*  $X$ , if  $X \subseteq T$ , hence support of  $X$  is the fraction of transactions that support  $X$ .

**Minimum Support threshold**: it is the minimum support threshold provided by the user, which is used as a standard to decide if an itemset is sufficiently frequent to be determined as 'useful for the purpose of association rule mining'.

**Frequent Itemset**: An itemset whose support count is greater than or equal to the minimum support threshold specified by the user.

**Infrequent Itemset**: An itemset, which is not frequent, is infrequent.

**Association Rule**: is the rule of the form  $R : X \rightarrow Y$ , where  $X$  and  $Y$  are two non-empty and non-intersecting itemsets. *Support* for rule  $R$  is defined as support of  $X \cup Y$ . *Confidence* is defined as  $\text{Support of } X \cup Y / \text{Support of } X$ .

**Interesting Association Rule**: An association rule is said to be interesting if its support and confidence measures are equal to or greater than minimum support and confidence thresholds (specified by user) respectively.

**Candidate Itemsets**: It is a set of itemsets, which are to be tested to determine whether they are frequent or infrequent.

**\*Virtual Support Count**: it is the support count of an itemset, computed by using the actual support count of its subsets. It may or may not be equal to the actual support of the itemset (which is the frequency of appearance of itemset pattern).

**\*Minimum Virtual Support count**: it is the minimum virtual support an itemset (formed due to union of two subsets) used as a standard to determine if the itemset is frequent or not.

**\*Differential Support Count**: it is the difference of actual (or virtual) support count and minimum support threshold provided by user.

**Problem of Mining Association Rules can be decomposed into two stages:**

# Stage 1: Finding all frequent itemsets

# Stage 2: Generating association rules using the discovered frequent itemsets.

**Common Approach to Stage 1**: Initially all candidate 1-itemsets are generated. Then by scanning the database, the support of each candidate 1-itemset is found. Then all, which are frequent (having support greater than minimum support threshold), are saved in a set. Then this set of frequent 1-itemsets is used to generate all possible candidate 2-itemsets. Again support of each is calculated and frequent nature is determined and process goes on, till no more candidate sets can be generated.

The cost of frequent itemset-discovery process comes from scanning of database (I/O time) and the generation of new candidate itemsets (CPU time). Thus if we reduce I/O time and CPU time needed by reducing the need of multiple scanning of database and number of candidates tested, we can further improve the performance of the mining process.

This is what is done in my algorithm by introducing new concepts to improve the overall performance.

### 3. DIFFERENTIAL-VIRTUAL SUPPORT ALGORITHM

#### 3.1 The Basic Idea

Here, I first find the actual support of 1-itemsets and use the support of frequent 1-itemsets to calculate virtual support of 2-itemsets formed due to union of two frequent 1-itemsets, then virtual support of frequent 2-itemsets is used to calculate frequent 3-itemsets formed due to union of two 2-itemsets and so on. All itemsets are declared frequent if their virtual support (actual support incase of 1-itemsets) is greater than Minimum Virtual support threshold (incase of 1-itemsets Minimum actual support threshold). Then in the end, I scan the database once more to calculate the actual support of all frequent itemsets discovered. Then to generate the *interesting* rules, I just use the Minimum *Confidence* threshold to find out that, of all the rules generated, which are interesting.

Thus, by just two scans on database, irrespective of its size; I generate all interesting rules very quickly by just relying on CPU speed. Also, since I don't use any new big data structures so I save considerable space too.

Thus DVS Algorithm turns out to be very space and time economic!!!

#### 3.2 Method

Step 1: Generate all 1-itemsets

Step 2: Measure their Actual support count by scanning the database

Step 3: If any 1-itemset has Actual support count < Minimum Actual Support Threshold  
Then prune it else save it in L (set of all frequent itemsets, initialized to NULL).

Step 4: Generate Differential Virtual Support Count Table, where differential virtual support count (DVS (X)) of each itemset X = difference of Actual Support count of each 1-itemset and Minimum Actual Support Threshold (user-defined).

Step 5: If DVS (X) = 0, prune the itemset from set of frequent 1-itemsets formed.

Step 6: Now generate 2-itemsets by union of only those two frequent 1-itemsets which satisfy the following conditions:  
for  $(X \cup Y)$  where X and Y are frequent 1-itemsets:

*Condition 1:* DVS (X) > 0

*Condition 2:* DVS (Y) > 0

*Condition 3:* DVS (X) + DVS (Y) > Minimum Virtual Support Threshold  
where Minimum Virtual Support threshold equals Minimum Actual Support threshold (user-defined) for maintaining symmetry in testing of any type of itemsets because 1-itemsets are tested using Minimum Actual support threshold and rest are tested using Minimum Virtual Support threshold.

*Condition 4:* if (  $DVS(X) + DVS(Y) > \text{Minimum of } DIV(X) \& \text{ } DIV(Y)$  )  
then  
 $V_{sup}(X \cup Y)$  (Virtual Support of  $X \cup Y$ ) = Minimum of  $DIV(X)$  &  $DIV(Y)$   
(because  $X \cup Y$  cannot occur more frequently than either  $X$  or  $Y$ )  
Else  
 $V_{sup}(X \cup Y) = DIV(X) + DIV(Y)$

Step 7: if all *Conditions* in step 6 are satisfied then save ( $X \cup Y$ ) in L.

Step 8: initialize  $k=3$

Step 9: Generate Differential Virtual Support Count Table, where differential virtual support count ( $DVS(P)$ ) of each  $k$ -itemset  $P$  = difference of Virtual Support count of each  $k$ -itemset and Minimum Virtual Support Threshold.

Step 10: If  $DVS(P) = 0$ , prune the itemset from set of frequent  $k$ -itemsets formed.

Step 11: Now generate  $(k+1)$ -itemsets by union of only those two frequent  $k$ -itemsets which satisfy the following conditions:  
for ( $X \cup Y$ ) where  $X$  and  $Y$  are frequent  $k$ -itemsets:

*Condition 1:*  $DVS(X) > 0$

*Condition 2:*  $DVS(Y) > 0$

*Condition 3:*  $DVS(X) + DVS(Y) > \text{Minimum Virtual Support Threshold}$   
where Minimum Virtual Support threshold equals Minimum Actual Support threshold (user-defined) for maintaining symmetry in testing of any type of itemsets because 1-itemsets are tested using Minimum Actual support threshold and rest are tested using Minimum Virtual Support threshold.

*Condition 4:* if (  $DVS(X) + DVS(Y) > \text{Minimum of } DIV(X) \& \text{ } DIV(Y)$  )  
then

$V_{sup}(X \cup Y)$  (Virtual Support of  $X \cup Y$ ) = Minimum of  $DIV(X)$  &  $DIV(Y)$   
(because  $X \cup Y$  cannot occur more frequently than either  $X$  or  $Y$ )

Else

$V_{sup}(X \cup Y) = DIV(X) + DIV(Y)$

Step 12: if all *Conditions* in step 11 are satisfied then save ( $X \cup Y$ ) in L.

Step 13:  $k = k+1$

Step 14: Repeat steps 9, 10, 11 and 12, till no more virtual frequent itemsets can be found.

Step 15: Scan Database, to find, Actual support count of each virtually frequent itemset in L.

Step 16: Prune any virtual itemset in L, if its Actual support count  $<$  Minimum Actual Support threshold.

Step 17: Generate rules for all frequent itemsets in L.

Step 18: if *Confidence* measure of any rule generated in step 17 is less than Minimum Confidence threshold (user-defined) then remove it.

Step 19: Display all interesting rules generated in step 18.

Step 20: Stop

**Doubt 1:** “Why do we subtract Minimum Actual (Virtual) Support Threshold from Actual (Virtual) Support count for an itemset?”

**Reason:** Well subtracting  $\sigma$  from  $\text{Sup}(X)$  to get  $\text{DVS}(X)$  and then checking  $\text{DVS}(X) > 0$  is same as writing  $\text{Sup}(X) > \sigma$ .

Proof:  $\text{DVS}(X) = \text{Sup}(X) - \sigma$   
Condition 1:  $\text{DVS}(X) > 0$   
 $\Rightarrow \text{Sup}(X) - \sigma > 0$   
 $\Rightarrow \text{Sup}(X) > \sigma$ , which is what we normally check to

declare an itemset frequent.

Similarly for itemset Y,  $\text{DVS}(Y) > 0 \Rightarrow \text{Sup}(Y) > \sigma$

Then combining two of them for  $X \cup Y$ , we say  $\text{VSup}(X \cup Y) = \text{DVS}(X) + \text{DVS}(Y)$   
And ideally like normal test for determining if a set is frequent, we check if  
 $\text{VSup}(X \cup Y) > \text{Minimum Virtual Support threshold}$   
to predict whether  $X \cup Y$  is frequent or not.

### 3.3 Algorithm

Nomenclature:

|           |   |
|-----------|---|
| DVS (X)   | = Differential Virtual Support of itemset X   |
| VSup (X)  | = Virtual Support count of itemset X  |
| Sup (X)   | = Actual Support count of itemset X   |
| L         | = Set of all frequent itemsets  |
| $C_k$     | = Set of Candidate k-itemsets   |
| n         | = Number of items in database provided  |
| $\sigma$  | = Minimum Actual Support threshold (user-defined)   |
| $\phi$    | = Minimum Virtual Support threshold (= $\sigma$ )   |
| D         | = Database provided by user on which mining has to be done  |
| k         | = A Variable storing only integer values depicts number of levels (for 1-itemset level is 1, for 2-itemsets it is 2; so on) |
| $\lambda$ | = Minimum Confidence Threshold (user-defined)   |

Input: A Transaction Database  $D$ , Minimum Actual Support Threshold  $\sigma$ , Minimum Confidence Threshold  $\lambda$ , Minimum Virtual Support Threshold  $\phi (= \sigma)$

```
// Function to find Minimum of DVS of two itemsets
```

```
Min ( DVS (X), DVS (Y) )  
{  
  if ( DVS (X) < DVS (Y) )  
    return DVS (X)  
  else  
    return DVS (Y)  
}
```

```
// Function to calculate virtual support of a set resulting from union of X  
// and Y
```

```
VSUP_calc ( DVS (X) , DVS (Y), k )  
{  
  if (length (X  $\cup$  Y) = k)  
  {  
    if (DVS (X) > 0 && DVS (Y) > 0 )  
    {  
      if (DVS (X) + DVS (Y) >  $\phi$  )  
      {  
        if (DVS (X) + DVS (Y) > Min ( DVS (X) , DVS (Y) ))  
        {  
          VSUP (X  $\cup$  Y) = Min (DVS (X), DVS (Y))  
        }  
        else  
        {  
          VSUP (X  $\cup$  Y) = DVS (X) + DVS (Y)  
        }  
      }  
      return VSUP ( X  $\cup$  Y )  
    }  
  }  
}  
}  
return 0  
}
```

```
// DVS Algorithm function
```

```
DVS_Algo (D,  $\sigma$  )  
{  
  initialize k=1, L =  $\emptyset$ , C1 =  $\emptyset$ , n=1  
  C1 = all 1-itemsets
```

```

for each transaction 't' in D
{
  for each element  $c_t$  in 't'
  Sup [ $c_t$ ] ++
  n++
}

for (I=0; I<n; I++)
{
  if (Sup [ $e_i \in C_1$ ] >  $\sigma$ )
  {
    L = L  $\cup$  {  $e_i$  }
    DVS ( $e_i$ ) = Sup [ $e_i$ ] -  $\sigma$ 
  }
  else  $C_1 = C_1 / \{ e_i \}$ 
}

ABC:
for all ( $f \in C_k \ \&\& \ g \in C_k \mid e \neq d$ )
{
  integer p = VSup_calc (DVS(f), DVS (g), k)
  if (p != 0)
  {
     $C_{k+1} = C_{k+1} \cup (f \cup g)$ 
    L = L  $\cup$   $C_{k+1}$ 
    VSup ( $f \cup g$ ) = p
    DVS ( $f \cup g$ ) = VSup ( $f \cup g$ ) -  $\phi$ 
  }
}

k = k + 1
if (length ( $C_{k+1}$ ) = 1)
  goto LAST
else
  goto ABC

LAST:
for each transaction 't' in D
{
  if any itemset h in L is found in 't' using pattern matching
  Sup (I) ++;
}

for all h  $\in$  L
{
  if (Sup (h) <  $\sigma$ )
  L = L / {h}
}

```

```

    gen_rules (L,  $\lambda$ )
  }

  gen_rules (L,  $\lambda$ )
  {
    for all h  $\in$  L
    {
      for every k  $\subseteq$  h
      if (Sup (h) / Sup (k) >  $\lambda$  )
      {
        print ("If customer takes combination", k, "then implies he will take", h-k)
      }
    }
  }
}

```

Output: *Interesting Rules*

**Theorem 1:** *The DVS algorithm generates all possible frequent itemsets and all interesting rules.*

**Proof:** “ Yes, since the Virtual Support count is nearly equal to Actual support count, the DVS algorithm takes care of not pruning out any important itemset which is frequent. Also since not all rules generated by any algorithm are *interesting*, so some rules generated by other algorithms are not generated by DVS because it doesn’t find them *interesting* as according to the Minimum Confidence threshold provided by user.”

#### 4. Performance Evaluation

The key question in any reader’s mind would come that “*Can this approach of Virtual Support generate all required frequent itemsets and all interesting rules?*”

Well, like every statement of a lawyer in court of Law needs a proof of validity of statement, I have tested the algorithm, with all kinds of transaction databases of various sizes, to certify its high performance, the details and summary of which has been described below.

## 5. Conclusion

An approach, which decreases the time and space factors related to the complex problem of Association Rule Mining, can lead to very fast conclusions and decisions in favor of various enterprises. Since time is of essence to all corporate companies, such a time-efficient approach can be of great use.

Also as company databases are constantly growing, fast and space-economic approach can help in analyzing greater amounts of data in less time!

In this paper, I present DVS algorithm based on this kind of approach. It has had relatively the best performance even with largest of databases as compared to any other Association Rule mining algorithm. It has proven to be excessively time and space efficient.

### ***References:-***

- [1] R. Agrawal, T. Imielinski, and A. Swami. Mining Association rules between sets of items in large databases. In *proc of the ACM SIGMOD Int'l Conf, on Management of Data (ACM SIGMOD '93)*, Washington, USA May 1993.