CHAPTER 6

HYBRID SCHEMES COMBINING ONTOLOGY AND FREQUENT ITEM CLUSTERING

6.1 INTRODUCTION

A vast amount of data are available in internet, social media network and news collection database in the past few years. All documents are mostly used by the user, but it is difficult to retrieve the appropriate document from the database collection. The document clustering methods with respect to traditional clustering algorithms have challenges against high dimensionality, low clustering accuracy in high volume of data. To fulfill the above circumstance, this chapter suggests, applying the frequent itemsets with association rule mining (Agrawal R. and Srikant R. 1994), (Sarawagi S. et al. 1998), (Hipp J. et al. 2000), for document clustering. The frequent itemsets form a task-oriented Ontology tree structure which is useful to make efficient clustering, based on the itemset occurrence in the input documents. The frequent itemsets work on the basis of keyword or term analysis (Jiang D. et al. 2004). The keyword based methodology alone is not efficient to document clustering. Hence, new hybrid techniques are presented to perform document clustering using Ontology framework. The hybrid technique which combines Apriori algorithm, FP-Growth (Borgelt C. 2005) and FP-Bonsai approach is used to (Bonchi F. et al. 2004) discover frequent itemsets using Apriori algorithm. FP-Growth approach is used to form lexicographic order for ensuring quality of document clustering by constructing FP-Tree for different
logically equivalent databases. FP-Bonsai tree is used to improve document clustering in an efficient manner (Sathiyakumari K. et al. 2011). FP-Bonsai is applied to the input dataset to reduce the un-uniqueness of FP-Tree. This tree is called as pruned tree or FP-bonsai tree. The performance of an Ontology framework based frequent itemset clustering of Apriori algorithm, FP-Growth and FP-Bonsai results are more efficient than existing traditional approaches such as Apriori, FP-Growth, and FP-Bonsai.

The commonly used method for frequent itemsets has been developed based on Apriori, FP-Growth and FP-Bonsai algorithm. The accompaniment of frequent itemsets alone does not assure efficient document clustering (Baghel R. and Dhir R. 2010). The use of Ontology based frequent itemsets produce high quality itemsets as well as frequent itemsets. The consideration of Ontology based frequent itemsets, produce an efficient document clustering using frequent term-based document clustering.

6.2 DOCUMENT CLUSTERING AND FREQUENT ITEMSET-BASED METHOD

This chapter discusses document clustering, which consists of frequent itemset-based method. The Frequent itemset-based method works with the contribution of Apriori algorithm, FP-Growth, and FP-Bonsai approach. Wherever, the document clustering is used, the following three processing steps are essential to make document clustering. (i) Frequent itemset selection with respect to itemset input data. (ii) Using Apriori algorithm, to create and mine FP-Tree which is useful for sub-graph discovery. iii) Finally, the sub-graph discovery is used to cluster the documents by using FP-Bonsai. FP-Bonsai is used to improve pruning (Bonchi F et al. 2004) for document clustering. In a consecutive manner, apply these three steps to the dataset corpus, the frequent
term-based document clustering (Baghel R. and Dhir R. 2010), formed according to the user need. The frequent term-based document clustering has three techniques such as Apriori, FP-Growth, and FP-Bonsai algorithm which is applied to Ontology framework. Document clustering is done in an efficient manner with respect to the document term and document frequency.

6.3 FREQUENT TERM-BASED DOCUMENT CLUSTERING

Frequent itemsets is formed based on association rule mining (Kumar A.K. et al. 2013) to exploit monotonicity property of frequent itemsets (Baghel R. and Dhir R. 2010). (i.e), subset of a frequent itemset is also frequent. In large databases, the frequent itemsets can be efficiently determined by support count. Many algorithms have been developed for the frequent itemsets task, including Apriori. A frequent itemset based approach of clustering is promising for classification task because it provides a natural way to reduce large dimensionality of document vector space. Frequent term-based document clustering (Baghel R. and Dhir R. 2010), is dealt with documents instead of transactions. For this reason, the notion of term sets is used instead of itemsets. A term is any preprocessed word within a document, and a term occurring in that document at least once is called subset. The cluster candidates are selected based on the low-dimensional frequent term sets. In the process of cluster formation, a frequent termset is not a cluster (candidate) but only the description of a cluster (candidate). A well-selected subset is considered as a cluster. Unlike incase of classification, there are no class labels to guide the selection of subset to form a set of all frequent termsets.

From these circumstances, this chapter is proposed to use mutual overlap of the frequent term sets with respect to their sets of supporting documents (clusters) for clustering. The principle behind this approach is, a
small overlap of clusters will result in a small classification error. In this section, three algorithms for frequent term-based text clustering are discussed.

6.4 FREQUENT ITEMSET GENERATION BY APRIORI ALGORITHM

This chapter makes effective document clustering by frequent itemset-based document clustering with Ontology framework. Before that, input terms are applied to preprocessing techniques such as BOW model, StopWord elimination, and stemming. The preprocessed documents are applied to the frequent itemset generation process which makes itemset based clustering with the help of Apriori Algorithm.

The steps for document clustering are clearly expressed as follows:

- Preprocessing is based on BOW model, StopWord elimination, Stemming.
- Finding the frequent words (extracted words) by removing redundant words from each document; the extracted words are taken as a top level item.
- Forming the binary mapped database by using extracted words.
- Finding the support and confidence of extracted words.
- Drawing frequent itemsets form Ontology based tree by applying Apriori algorithm.
- Sorting out mined frequent itemsets in descending order.
- Partitioning the text documents based on frequent Itemsets.
- Clustering the text documents based on frequent Itemsets.
6.4.1 Preprocessing: Bag of Words model, StopWord elimination, Stemming

The sample text documents are preprocessed by the bag-of-words model. Without changing the desired meaning, frequently repeated words is cumulated in a single database and fixed to get the common words from the database. The BOW model is based on the input terms and word occurrence count. The above outcomes of the BOW model are applied to StopWord elimination process for removing the list of StopWord. Moreover, the porter stemming process is applied to the outcome of StopWord elimination. The Porter stemmer is used to reduce the document size by removing unwanted words. These preprocessed documents are applied to the frequent pattern tree construction.

6.4.2 Apriori Algorithm

Apriori Algorithm in figure 6.1. is used to determine the relation between terms in large-scale transaction data. With respect to the support and confidence, the association rule pattern (Aggarwal S. and Kaur R. 2013), (Goswami D.N. et al. 2010) is used to make term 1, 0 term 2 and both are compared with term 3. This comparison is used in finding association rules.

Let \( I = \{i_1, i_2, \ldots, i_m\} \) be the set of the document items. Let \( D = \{t_1, t_2, \ldots, t_m\} \) be the set of document transactions, where each transaction “t” is an itemset such that \( D \subseteq I \). An association rule is an implication of the form term 1 \( \Rightarrow \) term 2, where term 1 \( \subseteq I \), Term 2 \( \subseteq I \), term \( \cap \) term 2 = D. The rule term 1 \( \Rightarrow \) term 2 has support ‘s’ in the transaction set D. if s% of the transactions in D contain term 1 \( \cup \) term 2. Given a set of transactions D, the problem of mining association rules is to generate all association rules that have certain
user-specified minimum support (min_support) and confidence (min_confidence).

An association rule satisfies both minimum support threshold and minimum confidence threshold that is determined by the user.

**Figure 6.1 Apriori Algorithm**

Candidate itemset Generation: Given an itemset, the set of all frequent k itemsets is to generate a superset of k-itemsets. Do calculation of itemset, term 1 has minimum support, so do all subsets of term1. From this
consideration, two levels of steps are available to make an efficient Candidate itemset Generation. First, in the support count, verify Support (itemset) > Support (threshold) and verify Confidence (subset) > Min_Confidence (subset). The second step is pruning, delete all itemsets by multiple scan (k itemset). The Apriori algorithm working rule is explained in the following seven sample dataset in Table 6.1.

<table>
<thead>
<tr>
<th>Actual document (D)</th>
<th>Preprocessed document</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Document 1: (D1)</strong>&lt;br&gt;As an example of clustering, early in Childhood, we learn to differentiate between cats and dogs.</td>
<td><strong>Document 1: (D1)</strong>&lt;br&gt;Clustering, early Childhood, learn differentiate cats dogs.</td>
</tr>
<tr>
<td><strong>Document 2: (D2)</strong>&lt;br&gt;Similarly, we learn to differentiate between animals and plants by continuously improving subconscious clustering schemes.</td>
<td><strong>Document 2: (D2)</strong>&lt;br&gt;Learn differentiate animals plants continuously improving subconscious clustering schemes.</td>
</tr>
<tr>
<td><strong>Document 3: (D3)</strong>&lt;br&gt;The requirements of clustering in data mining include scalability, ability to deal with different attributes and the ability to deal with noise data.</td>
<td><strong>Document 3: (D3)</strong>&lt;br&gt;Requirements clustering data mining scalability, ability deal attributes ability deal noise data.</td>
</tr>
<tr>
<td><strong>Document 4: (D4)</strong>&lt;br&gt;Some other requirements are ability to deal with High dimensionality and constraint based clustering.</td>
<td><strong>Document 4: (D4)</strong>&lt;br&gt;Requirements ability deal High dimensionality constraint clustering.</td>
</tr>
<tr>
<td><strong>Document 5: (D5)</strong>&lt;br&gt;And a few more requirements are Interoperability and usability.</td>
<td><strong>Document 5: (D5)</strong>&lt;br&gt;Requirements Interoperability usability.</td>
</tr>
<tr>
<td><strong>Document 6: (D6)</strong>&lt;br&gt;To form different clustering discussed above, different clustering algorithms have to be applied to the data objects.</td>
<td><strong>Document 6: (D6)</strong>&lt;br&gt;Form clustering discussed clustering algorithms applied data objects.</td>
</tr>
<tr>
<td><strong>Document 7: (D7)</strong>&lt;br&gt;Different clustering algorithms include exclusive and hierarchical.</td>
<td><strong>Document 7: (D7)</strong>&lt;br&gt;Clustering algorithms exclusive hierarchical.</td>
</tr>
</tbody>
</table>

Table 6.1 Sample data documents and preprocessed documents
The working procedures of Apriori algorithm are explained using the sample dataset (Table 6.1). The above document set is applied to Apriori, for finding the frequent itemset based clustering in terms of document modified TF/IDF term frequency. This TF/IDF term frequency is represented in Table 6.2.

The modified TF/IDF is evaluated based on the equation (6.1). The modified TF/IDF algorithm includes TSW and TDW with the traditional TF/IDF algorithm.

\[
a_{ik} = f_{ik} \times \log \left( \frac{N}{d_{fi}} \right) \ \text{TSW} \times \text{TDW}
\]  
(6.1)

Here the TF/IDF of the word distinguishes the text that appears frequently in the text but reversely in other texts. TSW is the estimation of relevancy or importance of the terms of the document. TDW is the estimation of distinguishing the ability among the terms. N is the number of documents present in the dataset corpus. \( f_{ik} \) is the frequency of term i in document j. \( d_{fi} \) is the document frequency.

This new weighting scheme considers both positive and negative occurrences of a term and are given as in equation (6.2)

\[
\text{TSW} = \text{TF} \times \log_2 \left( \frac{\max(1, DP1)}{\max(1, DN1)} \right)
\]  
(6.2)
Here P1 represents all terms that have TF/IDF > T (positive Terms), N1 represents the negative terms, DP1 is the number of documents with P1 and DN1 is the number of documents with N1. The Discriminating Weight is given in equation

$$TDW = \log \left( \frac{N}{n} \right) \times \left( \frac{1}{(n - m + 1)} \right)$$

(6.3)

Here “N” is the total number of documents; n represents the number of documents containing the term “t”; “m” is the maximum number of documents containing the term “t” in a certain category. “n-m” is the difference between the number of all documents containing the term “t” and the maximum number of documents containing the term “t” in a certain category. When the number of documents in a category containing term “t” is large, the number of the documents in the other categories containing term t, i.e., (n-m) is small.
In Apriori algorithm, seven different types of data samples are taken to document clustering. To find the TF/IDF of individual texts, the modified TF/IDF is used.

The modified TF/IDF algorithm is applied to different datasets for calculating frequent itemset. In the case of seven document dataset corpus, the minimum support count required is 4. With respect to the minimum_support_count, (SC) the new support count is calculated by scanning the term id (TID).

Table 6.3 Sample Dataset with support count (SC) of each term
Scan candidate-1 itemsets

Scanning of candidate -1 itemsets, consists of the following process; candidate-1 is a representation of $C_1 = \{D_1, D_2, D_3, D_4, D_5, D_6, D_7\}$. At the time of Scan candidate-1 itemsets, the $C_1$ itemset are compared in the dataset and their counts are determined. These outputs are given in Table 6.4. If the user assumed, support count as 35%, large itemset ($L_{itemset}^1$) initiates itemsets with support count greater than 35% of term as 2. $L_{itemset}^1 = \{D_1, D_2, D_3, D_4, D_5, D_6\}$.
The candidate-1 itemsets \((C_2)\) is calculated by \(L_{item\_set1} \times L_{item\_set1}\). 
\[C_2 = \{D1 D2, D1 D3, D1 D4, D1 D5, D1 D6, D2 D3, D2 D4, D2 D5\}.

- **Scan candidate-2 itemsets**

The scan candidate-2 itemsets are compared with previous results; the corresponding support counts are calculated and tabulated in Table 6.5. The new itemset of \((L_{item\_set2})\) is calculated by using previous \(L_{item\_set1}\) itemsets. If the support count is greater than 35% of terms (i.e. 2), the \(L_{item\_set2}\) is found by 
\[L_{item\_set2} = \{D1 D2, D1 D3, D1 D4, D1 D5, D1 D6, D2 D3, D2 D4, D2 D5\}.
The candidate itemsets \(C_3\) is computed as \(L_{item\_set2} \times L_{item\_set2}\). Finally \[C_3 = \{D1 D2 D6, D1 D2 D3 D4, D2 D3 D4 D6\}. The pruning step of Apriori eliminates itemsets which are less than 35%, after pruning \(C_3 = \{D1 D2 D6, D1 D2 D3 D4, D2 D3 D4 D6\}.

<table>
<thead>
<tr>
<th>TID</th>
<th>Support Count</th>
<th>TID</th>
<th>Support Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1, D2</td>
<td>3</td>
<td>D2, D6</td>
<td>3</td>
</tr>
<tr>
<td>D1, D3</td>
<td>2</td>
<td>D3, D4</td>
<td>5</td>
</tr>
<tr>
<td>D1, D4</td>
<td>1</td>
<td>D3, D5</td>
<td>1</td>
</tr>
<tr>
<td>D1, D5</td>
<td>1</td>
<td>D3, D6</td>
<td>2</td>
</tr>
<tr>
<td>D1, D6</td>
<td>2</td>
<td>D4, D5</td>
<td>1</td>
</tr>
<tr>
<td>D2, D3</td>
<td>2</td>
<td>D4, D6</td>
<td>1</td>
</tr>
<tr>
<td>D2, D4</td>
<td>1</td>
<td>D5, D6</td>
<td>0</td>
</tr>
<tr>
<td>D2, D5</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 6.5 Scan candidate-2 itemsets**
• **Scan candidate-3 itemsets**

In scan candidate 3-itemsets, the previous result of support count is compared with the dataset count which is tabulated in Table 6.6. The output of the scan is \(L_{\text{item_set}3}\). \(L_{\text{item_set}3} = \{D1, D2, D6\}\) and \(C4 = \{D1, D2, D6\}\). The candidate itemsets \(C4\) is calculated based on the \(L_{\text{item_set}3} \times L_{\text{item_set}3}\). \(C4 = \{D1, D2, D6\}\). In the next scan the count of \(\{D1, D2, D6\}\) is computed and it is found to be 2, which is equal to the minimum support count required. So \(L_{\text{item_set}4} = \{D1, D2, D6\}\). The algorithm stops here as no candidate itemset of size 2 is found (\(L_{\text{item_set}4}\) is contains only 1 item).

<table>
<thead>
<tr>
<th>TID</th>
<th>Support Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1, D2, D6</td>
<td>2</td>
</tr>
<tr>
<td>D1, D2, D3, D4</td>
<td>1</td>
</tr>
<tr>
<td>D2, D3, D4, D6</td>
<td>1</td>
</tr>
</tbody>
</table>

**Table 6.6 Scan candidate-3 itemsets**

In this example, the number of candidate itemsets, scans done is 4, and in every scan, the entire dataset of 31 transactions is read. Thus the total number of transactions read in the entire process is 124. The effort of optimization is to reduce the number of transactions to be read to compute large itemsets. The overall performance of mining association rules is determined by finding the frequent itemsets (Kaur C. 2013).
6.4.2.1. Apriori Algorithm with Ontology framework

The Apriori algorithm is used to find frequent itemsets with the help of candidate generation scan. The candidate generation scan is calculated based on the dataset samples. To improve efficiency of the Apriori algorithm, the following conditions are used:

- If an itemset does not satisfy the minimal support, itemset will not be considered as a frequent itemset.
- If an itemset is not a frequent itemset, all supersets of itemset will also fail to form the frequent itemset.

To fulfill the above failure condition, Apriori algorithm uses optimizing search time. This will happen based on the two step process such as (i). Join, Prune step (ii). Ontology based Apriori algorithm.

In this case, the minimum support count required is 2. With respect to the minimum support count, the new support count is 35% and the minimum confidence required is 65% for occurrence in the document dataset. Ontology based Apriori algorithm finds out the frequent itemset with the presence of association rules (Sarawagi S. et al. 1998). It is used to find the minimum support and minimum confidence.

From figure 6.2, it is noted that there are ‘k’ iterations available to find the set of candidate itemsets $C_n$. At first, the algorithm scans the entire item from the data base. In the next iteration, the algorithm attempts to generate the set of all itemset.
Apriori Algorithm
Input initialization:
  Ck: Candidate itemset of size k
  Lk: frequent itemset of size k
  L1= {frequent items};
for(k= 1; Lk!=∅; k++)
  do begin
    Ck+1= candidates generated from Lk;
    for each transaction t in database do
      Increment the count of all candidates in
      Ck+1 that are contained in t
    end
    Lk+1= candidates in Ck+1 with min_support
  end
return Ck Lk;
create data package;
getConstruct Ontology(OWL model);
end

Apriori Algorithm with Ontology
Construct OWL from Apriori data package count;
setpkg_counter←Zero
startpkg_count= pkg_counter;

  construct OWL model(pkg(pkg_counter));
  pkg_counter++; continue;

start set class_counter←Zero
  class_count= class_counter;
  construct OWL model(class(class_counter));
  class_counter++; continue;

start set param_counter←Zero
  param_count= param_counter;
  start set method_counter←Zero
    method_count= method_counter;
    construct OWL model(method(method_counter));
    method_count++; continue;
    construct OWL model(param(param_counter));
pkg_counter++; break;

Result: Write the OWL model
Until Apriori data package count data File reach

Figure 6.2 Apriori Algorithm with Ontology
The above figure 6.3. illustrates the demonstration of Apriori algorithm. To find the first itemset by, assigning $k=2$, with respect to the
Apriori algorithm, the initial itemset is formed based on the minimum support, which is less than the support values which gives the pruning values for each iteration (Bonchi F. et al. 2004). For example, to derive the minimum item constructed using the support count calculation, the manual minimum support count is taken as 65%. Based on this, all the less support count is eliminated for each iteration.

The frequent sets generated are greater than the minimum support supplied by the user. The sorting of itemset which is useful to gather all itemsets are checked by the minimum support count. During initialization, the top level of Ontology tree is set to null value which is used to construct all frequent itemsets in the tree. Finally, after tracing the input item, the entire document set is involved to form Ontology tree structure.

6.5 FP-GROWTH ALGORITHM

The FP-Growth algorithm is a traversing, searching and multi-branched tree recursion algorithm (Borgelt C. et al. 2005). FP-Growth algorithm has the searching tree structure called FP-Tree It is used to find an optimized pattern of the dataset corpus (Han J. et al. 2007). The FP-Tree is constructed based on the subset of frequent itemset and the header table 6.3. It contains the nodes for each frequent item which is arranged in decreasing order of frequency in the frequent itemset. These decreasing orders of tree structure are used to optimize the pattern of dataset corpus and build the FP-Tree. This optimized FP-Tree contains complete information about the frequent patterns (Bonchi F. et al. 2003). In addition to this, the header table contains fields like (i) itemset-name, (ii) support count, and (iii) node-link. From this field, the frequent itemsets can be found from FP-Tree quickly without having to scan the
database on disk frequently. Each itemset points to an occurrence in the tree via a head of node-link. Nodes with the same item are linked via node-links.

**Figure 6.4 FP-Growth Algorithm**

The above figure 6.4 describes the FP-Growth algorithm which consists of the following processes for scanning the database to the construction of FP-Tree.

- **Header table is created based on the frequent items**

  In this process, the frequent items are identified and inserted in to the header table in a decreasing order based on their support counts.
• **FP-Tree root creation**

In this process, initially, the root of FP-Tree is created with null. Each frequent itemset is inserted into this root as a branch. In some cases, if an itemset has the same prefix with another itemset already in the tree, this part of the branch will be shared. Finally, the FP-tree is completely constructed.

• **Frequent itemsets finding**

After constructing the FP-Tree, frequent itemsets findings start from the bottom of header table. Paths of the FP-Tree are created based on the header table support count. This path creation is repeated for constructing FP-Tree until a single path is found. The single path is generated based on the similar prefix frequent itemsets.

Finally, the FP-Growth algorithm in figure 6.4 finds all frequent items in the dataset corpus and constructs a new header table for new FP-Tree. The following example shows that the example of frequent pattern called FP-Tree is used in this method.

In this manner, the above algorithm work is based on the following steps as follows,

Step 1: Scanning the transaction database and the support for each item.
Step 2: Comparing support to the minimum support value.
  Step 2.1: Adding frequent itemset.
  Step 2.2: Repeating the process until last itemset and generate all itemset.
Step 3: Finding support for all itemset.
Step 4: Doing step 2 again and to check 2.1 and 2.2 and to add the result to the frequent itemset.

Step 5: If resulted frequent itemset is empty.

Step 6: Find the confidence else do step 2 and 3 until finding the confidence value of all itemset.

Step 7: If confidence is compared to the minimum confidence value.

Step 8: Add to the tree else finish the tree formation.

```
Input: F_items, support (counts),
L_freq : list of frequent items
E is an initial element
Elist the remaining list
Transaction (scan):
  Sort (F_items);
  New support (counts) < min support (counts);
  Transaction (scan) +1;
  Create top of the tree;
  Sort F_items descending support (counts) as 
  L_freq is apply to Itemset++;
  assign root of an FP-tree “null”
sort the F_items in Transaction (scan)L_freq
  sort result F_items[e | E_list],
  insert_tree([e | E_list],T),
Form the FP tree till ends of the all frequent itemset
```

**Figure 6.5. Frequent Pattern- Tree Algorithm**

The above figure 6.5 shows the FP-Tree Construction based on term frequency and inverse document frequency as well as the number of occurrences in the same documents. With respect to the dataset in Table 6.1, the term frequency in Table 6.2 the FP-Tree is constructed using minimized mined frequent itemsets.
During FP-Tree root creation, the dataset document shown in figure 6.5 constructs the FP-Tree as shown in Figure 6.6. The mining of “frequent itemsets finding” is applied to the FP-Tree as shown in Figure 6.6. The header table is used to store frequent itemsets efficiently and to do frequent itemset optimization.

From the above example of tree structure formation, the following algorithm is used to construct the Ontology tree structure. The structural techniques are relatively common to FP-Tree construction.

<table>
<thead>
<tr>
<th>TID</th>
<th>TID in list of document</th>
<th>Support Count</th>
<th>Node Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>D1, D2, D3, D6, D7</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>D1, D2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>D1, D2, D3, D6, D7</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>D2, D6</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>D3, D4</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>D1, D4</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>D3, D4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>D3, D4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>D6, D7</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 6.6 The FP-Tree construction based on subset of frequent itemset**

From the above figure 6.6, the FP-Tree scans the itemset two times; in the first scanning, all the itemsets are scanned and stored in FP-Tree construction based on the collected frequent itemset. This collected itemset
forms two levels of FP-Tree structure and this itemset is an input for the next tree level. When the tree reaches the above two levels, the itemset share the parent node to the child node. Now the top level node will act as the root node, the main root node is assigned as the null node. Depending on this null node all tree levels are constructed based on the stored itemset of the database.

The second scanning is used to form the root node of the FP-Tree; the top of the tree is constructed with null value based on initial frequent itemset. This initial root construction gives information about the original dataset item values. Finally, the scanning of each process performs FP-Tree structure by using the bottom-up approach. The frequent itemset mining (Badal S. and Shruti T. 2010), (Garg. R. and Gulia P. 2015) produced efficient frequent itemset retrieval with the lowest degree of time interval and memory utilization.

6.5.1 ONTOLOGY BASED FP-GROWTH

The generated frequent itemset candidates are applied to Ontology framework to improve frequent itemsets for finding frequently closed itemsets. Ontology based FP-Growth needs to form the FP-Tree pattern to reduce the time for frequent itemset mining (Dandu S. et al. 2013). The proposed hybrid algorithm for FP-Growth based frequent itemsets mining and Ontology framework reduces the execution time and memory usage.

6.5.1.1. Formation of Ontology based FP-Growth Frequent Itemsets Mining
The Frequent-pattern growth (FP-Growth) algorithm with Ontology framework is used for frequent itemset extraction from the vast number of candidate itemsets. Compared to the Apriori algorithm, the FP-Growth is an efficient technique (Borgelt C. et al. 2005), (Zeng Y. et al. 2015) because FP-Growth has divide-and-conquer method. Divide-and-conquer method is used to make the candidates by dividing itemset into several smaller subsets, and conquer the sub-problems by solving them recursively till all the input of the subset in the data base incomplete. Finally, it compresses the database into a compact tree form, known as the FP-Tree, and extracts the frequent patterns by tree traversing. This is also called as ontological tree structure of the particular input itemset.

**FP-Growth Algorithm with Ontology**

Construct Ontology OWL from FP-Growth data package count;

- set pkg_counter ← Zero
- start pkg_count = pkg_counter;
  - construct OWL model(pkg(pkg_counter));
  - pkg_counter++; continue;
- start class_counter ← Zero
  - class_count = class_counter;
  - construct OWL model(class(class_counter));
  - class_counter++; continue;
- start param_counter ← Zero
  - param_count = param_counter;
  - start set method_counter ← Zero
    - method_count = method_counter;
    - construct OWL model(method(method_counter));
    - method_counter++; continue;
  - construct OWL model(param(param_counter));
  - param_counter++; break;

**Result:** Write the OWL model

Until FP-Growth cluster data File reach

---

**Figure 6.7 FP-Growth Algorithm with Ontology Framework**
Ontological tree structure is used to minimize candidate generation process, to only those most likely to be frequent and employs a compact prefix-tree data structure, FP-Tree, avoids repetitive scanning of the database. This Ontology based tree structure is very efficient because it easily puts into practice to construct an input data item based FP-Tree construction. Ontology based framework in figure 6.7 works even when the database is large. This method is very common and efficient in document clustering because frequent patterns are discovered based on frequent subgraphs discovery. Formation of Ontology based FP-Growth frequent itemsets mining (Zeng Y. et al. 2015) is applied to Reuters-21578, 20 NG and TDT-2 document dataset, the resultant FP-Growth graph mines the efficient frequent itemset.

6.6 FP-BONSAI TREE

Apriori Algorithm and FP-Growth cause minimization of Frequent Pattern-Tree, it takes more time delay and memory consumption. To resolve these drawbacks, this chapter introduces the FP-Bonsai tree and Ontology based FP-Bonsai Tree with the indexed data value. FP-Bonsai tree is used to form the minimized tree structure and applied this tree structure to the vast amount of dataset to produce the performance optimization of large databases with respect to dynamic indexes. The previous chapter discusses the FP-Growth algorithm along with FP-Tree construction, to improve FP-Growth performance, FP-Bonsai Tree is applied to the FP-Growth, which is used to prune (Bonchi F. et al. 2004) the FP-Tree with respect to Ontology framework. The pruned FP-Tree is called as FP-Bonsai tree.

From the description of FP-Bonsai, the minimum number of transactions that occurs in any frequent itemset is called as the least frequent item. Depending upon the FP-Bonsai algorithm, least frequent item is deleted
from the FP-Tree. After removing, the minimum number of transactions of frequent itemset, next nearby transactions are combined together to form the new FP-tree with the minimized pruning.

FP-Bonsai tree is used to enhance frequent pattern item with the Ontology structure. This chapter introduces a hybrid model of Ontology based FP-Bonsai algorithm which includes tree structure with Apriori and FP-Growth algorithm. The data reduction for Apriori and FP-Growth tree structure is done using FP-Bonsai. The recursive process is executed to each itemset by scanning the itemset. Finally, the constructed FP-Trees (Grahne. G. et al. 2005) produce a FP-Growth computation (Borgelt C. et al. 2005) to give the pruned tree comprehensively with lesser number of smaller trees. These simplified FP-tree, is acquired by growing and pruning FP-Bonsai (Bonchi F. et al. 2004).

![Diagram of FP-Bonsai Algorithm](image-url)

**Figure 6.8 FP-Bonsai Algorithm**
processes, which include the Term Search Order (TSO), Conditional Database Representation (CDR), Conditional Database Construction Strategy (CDCS) and Tree Traversal Strategy (TTS). The steps involved in the construction of FP-Bonsai Tree is shown in figure 6.8.

- **Term Search Order (TSO):**
  In this step, all frequent itemset are sorted in the order of support count. TSO is used to construct the number of conditional databases by using different item search orders.

- **Conditional Database Representation (CDR):**
  The tree traversal search and construction of a conditional database depend on its representation. If the constructed conditional database is adequate to the other dataset item, the represented database easily forms the frequent itemset. Otherwise, the frequent itemset formation is difficult.

- **Conditional Database Construction (CDC):**
  In each individual conditional database representation, the database construction is done in a physical manner. In this procedure, the mining cost is affected because the conditional database construction depends on the tree traversal search and term search order.

- **Tree Traversal Strategy (TTS):**
  In the Tree Traversal Strategy, the traversal search order cost of a tree is minimal using top-down traversal strategy.
A. Initial FP-Bonsai

Consider the News Ontology database, particularly the seven document dataset items in Table 6.2 and Table 6.3, the minimum support threshold is 2. All infrequent items are removed from the news Ontology database such as Reuters-21578, 20 NG, and TDT-2 individually. With respect to Term Search Order (TSO), the documents are reordered to support descending order and are inserted into the FP-tree, resulting as the tree shown in Figure 6.9.

![Figure 6.9 Initial FP-Bonsai](image)

Figure 6.9 Initial FP-Bonsai

Figure 6.9 shows, the construction of a minimized level of a tree structure, namely initial FP-Bonsai tree structure (Bonchi F. et al. 2004). This tree structure is minimizes the number of term occurrence in a particular document. It produces the concise tree level by improving the purity and the consumed time. It is used to optimize document clustering (Jain A.K. et al. 1999), retrieval time.
The initial process is executed until no more items exist. In this era, all trees can be pruned using the frequent itemset reduction technique. This pruned FP-Tree is called as FP-bonsai tree.

B. FP-Bonsai after the removal of Item based on minimal support threshold (σ)

The pruned FP-Bonsai tree is shown in Figure 6.10. The pruned FP-Bonsai tree is constructed based on minimal support threshold (σ). The tree is minimized based on the threshold value. The static threshold is calculated based on the unit value, ceil and the respective number of datasets and its size. The threshold value is calculated based on the equation (6.4).

\[
\text{Threshold}(\sigma) = \text{unit} \times \left(1 + \text{ceil}\left[\frac{\text{Dataset} - \text{size}}{p}\right]\right)
\]

(6.4)

The threshold (σ) is calculated and it produces the pruned FP-Bonsai tree. The level of a frequent itemset is fixed with the minimum threshold level 2. The FP-Bonsai removes itemset term lower than the threshold level. This is illustrated in figure 6.10.

Figure 6.10 FP-Bonsai after the removal of item D1 for lower threshold (threshold,\(<=7)
In order to find the complete algorithm, that finds all itemsets satisfying the given constraints, the pruning should be called at the time of FP-Growth. The FP-Tree is the most effective data structure for FP-Growth and FP-Bonsai algorithm to use. The following steps describe how pruning mechanisms can be effectively used in the FP-Tree structure formation (Zhichun L. and Fengxin Y. 2008).

C. The final minimal FP-bonsai

After reaching the fixed point, the final FP-Bonsai has been created based on the given problem, which is shown in figure 6.11. The final size of the FP-Bonsai tree has 3 nodes with support = 2.

![Figure 6.11 The Final Minimal FP-Bonsai](image)

When compared to FP-Growth algorithm and Apriori algorithm, the final minimal FP-Bonsai makes an effective f-measure, purity, execution time and memory utilization. Moreover, to improve the pruning result, the final
minimal FP-Bonsai is applied to Ontology framework to construct FP-Bonsai. It is used to prune the entire frequent itemsets.

6.6.1 FP-Bonsai Algorithm with Ontology

An Ontology framework based on FP-Bonsai algorithm in figure 6.12. works well with Ontology based levels such as top levels, middle levels and bottom levels of FP-Tree (Zhichun L. and Fengxin Y. 2008). The below constructed FP-Bonsai has been pruning the entire process of frequent itemsets. So the FP-Growth (Zeng Y. et al. 2015) can be effectively mined, to calculate the time taken for construction of the tree. FP-Bonsai tree construction is based on FP-Growth frequent itemset data process, executes till all the itemset scan and form the tree.

Bonchi F. et al. (2004) derivation, constructs the pure frequent itemset mining called pruning. This is shown in Figure 6.11 (final minimal FP-Bonsai). Finally, Ontology FP-Tree starts with the top level that has a non-visible meaning of the document database itemset because the support is greater than the minimum support, so it is ignored. Next, the middle level has the parent pointer so it gives the meaning of traversal without bypassing the nodes corresponding to infrequent items. Finally, the merged bottom level fulfills the entire document itemset with the minimal tree structure, even different original parents were found to make the tree. Whenever the threshold level becomes less than the minimal support, the itemset traversal will be removed from FP-Bonsai and then the remaining itemsets are merged to form the new traversal of the tree.
Ontology framework consists of Ontological itemset tree with three levels such as top, middle and bottom level which is described as shown in the figure 6.13.

**FP-Bonsai Tree**

*Initialize* data index value  
*Read* data from FP-Growth  
*Form*: minimum tree structure  
*start* FP-Bonsai tree construction  
  - Remove: infrequent due to the frequent itemset;  
  - Result: pruned FP-trees;  
*Find* least frequent item by:  
  - if number of itemset travel  
    - *continue* through any frequent itemset;  
  - *Find* minimum number of transactions:*end*  
*do* Find least frequent item until last  
  - *continue*min_pruning become zero  
    - construct the FP-itemset tree; *continue*;  
    - new tree itemset=next nearby transaction++;  
    - Find FP-itemset tree; *end*  
*do* process till complete data; *break*;  
create data package; get Construct Ontology(OWL model);*end end*  

**FP-Bonsai Tree with Ontology**

get input from FP-bonsai tree package data count  
setpkg_counter←Zero  
*start* pkg_counter  
  - construct OWL model(pkg(pkg_counter));  
    pkg_counter++; *continue*;  
*start* set class_counter←Zero  
  - construct OWL model(class(class_counter));  
    class_counter++; *continue*;  
*start* set param_counter←Zero  
  - construct OWL model(param(param_counter));  
    param_counter++; *continue*;  
*start* set method_counter←Zero  
  - construct OWL model(method(method_counter));  
    method_counter++; *continue*;  
  - constructOWL model(param(param_counter));  
    param_counter++; *break*;  
*Result*: Write the OWL model  
Until FP-bonsai tree data File reach

**Figure 6.12 FP-Bonsai Algorithm with Ontology Framework**
Figure 6.13 Ontological Itemset-Tree with three levels

Figure 6.13 shows the Ontological itemset-Tree with three ontological levels, such as top, middle and bottom level with respect to the hybrid Ontology combines upper level and domain level Ontology together to form the hybrid Ontology. Here top level acts as the upper level, middle and bottom level act as the domain, bottom level as task and application Ontology. Finally, it forms the Ontological Itemset-Tree with three levels of Ontology.

After constructing FP-Tree (Zhichun L. and Fengxin Y. 2008) with a final minimal FP-Bonsai based on the database, the time utilization, and memory reduction are the crucial parts in FP-Bonsai Algorithm. To fulfill the reduction, Ontology framework is applied to FP-Bonsai based algorithm on the database. In addition to this, pruning step is applied to FP-Bonsai tree for simplification and reduces utilization time and memory consumption.
In this chapter, Ontologies have evolved throughout the research to concentrate the definition of Ontology elements. Finally, outlines of Apriori, FP-Growth and FP-Bonsai tree Ontology are brought together (Fatima P. 2012), to describe an itemset data scheme and provides a controlled vocabulary of concepts, with respect to four types of ontologies such as upper ontologies, domain ontologies, task ontologies, and application ontologies.

When Ontology based document clustering is used (Jain A.K. et al. 1999), (Berry. M.W. and Castellanos M. 2007), the entire process reduces the number of irrelevant or infrequent itemsets. It converts the maximal itemsets into the minimum optimal itemsets, by using Apriori, FP-Growth and FP-Bonsai tree Ontology framework. All these steps work based on frequent itemset formation and pruning techniques (Bonchi F. et al. 2004). In addition to this, domain Ontology improves the integration of user domain knowledge in a database field and to improve the computational time and pruning efficiency.

6.7 PERFORMANCE EVALUATION

Ontology tree structure has four levels such as Top-Level Ontology, Domain Ontology, Task Ontology and Application Ontology. The Top-Level Ontology is also called as concept Ontology. Other three ontologies are called as general Ontology. To combine concepts and general Ontology, the mining task becomes easy and sufficient to document clustering to analyze the efficiency in pruning. In order to describe the performance of document clustering (Julian S. et al. 2004), frequent pattern mining (Goswami D.N. et al. 2010), (Han J. et al.2007) is applied to the documents based on dataset corpus.
There are three datasets taken in this research, such as Reuters-21578, 20 NG, and TDT-2. Apriori algorithm is applied to these datasets for mining frequent itemsets. Then, the initial partition is constructed using these frequent itemsets by FP-Growth. Finally, FP-Bonsai tree is used to cluster the partition frequent itemsets data.

The performance measures of Precision, Recall, f-measure and Purity are calculated based on the following equation (6.5. to 6.8.). The equation 6.5 & 6.6 are used to calculate Precision (P) and Recall (R) as follows

\[
P(i, j) = \frac{N_{ij}}{N_j} \tag{6.5}
\]

\[
R(i, j) = \frac{N_{ij}}{N_i} \tag{6.6}
\]

\(N_{ij}\) – is the Number of objects in class ‘i’ in cluster ‘j’, \(N_j\) is the number of objects in cluster ‘j’, \(N_i\) is the number of objects of class ‘i’. The f-measure is calculated using the following equation (6.7).

\[
F(i, j) = 2 \times \frac{P(i, j) \times R(i, j)}{P(i, j) + R(i, j)} \tag{6.7}
\]

It is always desired to obtain a large f-measure, which indicates better clustering performance. The average purity is calculated using the following equation (6.8).

\[
Purity(k) = \frac{1}{N_k} \text{Max}N_{mk} \tag{6.8}
\]
Where \( N_k \) denotes the number of elements lying in cluster \( C_k \) and let \( N_{mk} \) be the number of elements of the class, \( I_m \) in the cluster \( C_k \). Then, the purity \( \text{pur}(k) \) of the cluster \( C_k \).

To validate the clustering measure, purity calculation is one of the best ways to determine the cluster quality. Purity is widely used measure for external information class labels. In this case, purity of the clusters is measured by refering to the class labels.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Clustering Algorithm</th>
<th>Dimensionality reduction and feature selection</th>
<th>Precision</th>
<th>Recall</th>
<th>F measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reuters-21578</td>
<td>Apriori Algorithm</td>
<td>TF/IDF</td>
<td>0.751</td>
<td>0.785</td>
<td>0.768</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modified TF/IDF</td>
<td>0.762</td>
<td>0.8</td>
<td>0.781</td>
</tr>
<tr>
<td></td>
<td>Ontology-based Apriori Algorithm</td>
<td>TF/IDF</td>
<td>0.781</td>
<td>0.789</td>
<td>0.785</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modified TF/IDF</td>
<td>0.794</td>
<td>0.81</td>
<td>0.802</td>
</tr>
<tr>
<td></td>
<td>FP Growth Algorithm</td>
<td>TF/IDF</td>
<td>0.786</td>
<td>0.816</td>
<td>0.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modified TF/IDF</td>
<td>0.8</td>
<td>0.821</td>
<td>0.81</td>
</tr>
<tr>
<td></td>
<td>Ontology-based FP Growth Algorithm</td>
<td>TF/IDF</td>
<td>0.809</td>
<td>0.839</td>
<td>0.824</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modified TF/IDF</td>
<td>0.822</td>
<td>0.844</td>
<td>0.833</td>
</tr>
<tr>
<td></td>
<td>FP Bonsai Algorithm</td>
<td>TF/IDF</td>
<td>0.814</td>
<td>0.834</td>
<td>0.824</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modified TF/IDF</td>
<td>0.82</td>
<td>0.849</td>
<td>0.834</td>
</tr>
<tr>
<td></td>
<td>Ontology-based FP Bonsai Algorithm</td>
<td>TF/IDF</td>
<td>0.812</td>
<td>0.832</td>
<td>0.822</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modified TF/IDF</td>
<td>0.856</td>
<td>0.898</td>
<td>0.84</td>
</tr>
</tbody>
</table>

Table 6.7 Performance measure Precision, Recall and f-measure for Reuters – 21578 using Ontology and frequent itemset clustering techniques
From the above Table 6.7, Performance measure such as Precision, Recall and f-measure for Reuters –21578 using Ontology and frequent itemset clustering techniques are discussed

- Ontology based Apriori Clustering on Reuters-21578 dataset, f-measure is 97.38% efficient than traditional Apriori Clustering on the Modified TF/IDF.

- Comparison of Modified TF/IDF on Ontology based Apriori Clustering, f-measure is 97.88% efficient than traditional TF/IDF.

- FP-Growth with Ontology on Reuters-21578 dataset, the f-measure is 97.24% efficient than traditional FP Growth with Ontology on the Modified TF/IDF.

- Comparison of Modified TF/IDF on Ontology based FP-Growth Clustering, f-measure is 98.91% efficient than traditional TF/IDF.

- FP Bonsai tree with Ontology on Reuters-21578 dataset, the f-measure is 99.2% efficient than traditional FP Bonsai with Ontology on the Modified TF/IDF.

- Comparison of Modified TF/IDF on Ontology based FP- Bonsai Clustering, f-measure is 97.85% efficient than traditional TF/IDF.
From the above Table 6.8, Performance measure in terms of Purity for Reuters – 21578 using Ontology and frequent itemset clustering techniques are discussed.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Clustering Algorithm</th>
<th>Dimensionality reduction and feature selection</th>
<th>Purity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reuters-21578</td>
<td>Apriori Algorithm</td>
<td>TF/IDF</td>
<td>0.659</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modified TF/IDF</td>
<td>0.79</td>
</tr>
<tr>
<td></td>
<td>Ontology-based Apriori Algorithm</td>
<td>TF/IDF</td>
<td>0.812</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modified TF/IDF</td>
<td><strong>0.827</strong></td>
</tr>
<tr>
<td></td>
<td>FP Growth Algorithm</td>
<td>TF/IDF</td>
<td>0.821</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modified TF/IDF</td>
<td>0.836</td>
</tr>
<tr>
<td></td>
<td>Ontology-based FP Growth Algorithm</td>
<td>TF/IDF</td>
<td>0.835</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modified TF/IDF</td>
<td><strong>0.842</strong></td>
</tr>
<tr>
<td></td>
<td>FP Bonsai Algorithm</td>
<td>TF/IDF</td>
<td>0.876</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modified TF/IDF</td>
<td>0.883</td>
</tr>
<tr>
<td></td>
<td>Ontology-based FP Bonsai Algorithm</td>
<td>TF/IDF</td>
<td>0.912</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modified TF/IDF</td>
<td><strong>0.956</strong></td>
</tr>
</tbody>
</table>

**Table 6.8 Performance measure purity for Reuters – 21578 using Ontology and frequent itemset clustering techniques**

From the above Table 6.8, Performance measure in terms of Purity for Reuters – 21578 using Ontology and frequent itemset clustering techniques are discussed.
• Ontology based Apriori Clustering on Reuters-21578 dataset, the purity is 95.53% higher than traditional Apriori Clustering on the Modified TF/IDF.
• Comparison of Modified TF/IDF on Ontology based Apriori Clustering, f-measure is 98.81% efficient than traditional TF/IDF.
• FP-Growth with Ontology on Reuters-21578 dataset, the purity is 99.29% higher than traditional FP Growth with Ontology on the Modified TF/IDF.
• Comparison of Modified TF/IDF on Ontology based FP-Growth Clustering, f-measure is 99.16% efficient than traditional TF/IDF.
• FP-Bonsai tree with Ontology on Reuters-21578 dataset, the purity is 92.36% higher than traditional FP Bonsai with Ontology on the Modified TF/IDF.
• Comparison of Modified TF/IDF on Ontology based FP- Bonsai Clustering, f-measure is 95.40% efficient than traditional TF/IDF.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Clustering Algorithm</th>
<th>Memory Utilization (MB)</th>
<th>Execution Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reuters - 21578</td>
<td>Apriori Algorithm</td>
<td>31</td>
<td>386000</td>
</tr>
<tr>
<td></td>
<td>Ontology-based Apriori Algorithm</td>
<td>16</td>
<td>252000</td>
</tr>
<tr>
<td></td>
<td>FP-Growth Algorithm</td>
<td>26</td>
<td>354000</td>
</tr>
<tr>
<td></td>
<td>Ontology-based FP-Growth Algorithm</td>
<td>12</td>
<td>212000</td>
</tr>
<tr>
<td></td>
<td>FP-Bonsai Algorithm</td>
<td>22</td>
<td>337000</td>
</tr>
<tr>
<td></td>
<td>Ontology-based FP-Bonsai Algorithm</td>
<td>10</td>
<td>188000</td>
</tr>
</tbody>
</table>

Table 6.9 Memory Utilization and Execution time of Reuters – 21578 using Ontology and frequent itemset clustering techniques
From the above Table 6.9, Memory Utilization of Reuters – 21578 using Ontology and frequent itemset clustering techniques are discussed as below,

- Ontology based Apriori algorithm takes 15 MB less memory utilization when compared to traditional Apriori algorithm on Reuters-21578 data set.
- Ontology based FP-Growth tree takes 14 MB less memory utilization when compared to traditional FP-Growth tree on Reuters-21578 data set.
- Ontology based FP–Bonsai algorithm consumes 12 MB less memory utilization when compared to traditional FP–Bonsai algorithm on Reuters-21578 data set.

From the Table 6.9, Execution time of Reuters – 21578 using Ontology and frequent itemset clustering techniques are given

- Ontology based Apriori algorithm provides 65.28% higher execution time when compared to traditional Apriori algorithm on Reuters-21578.
- Ontology based FP-Growth algorithm provides 59.89% higher execution time when compared to the traditional FP-Growth algorithm on Reuters-21578.
- Ontology based FP-Bonsai algorithm provides 55.79% higher execution time when compared to the traditional FP-Bonsai algorithm on Reuters-21578.
From the above Table 6.10, Performance measure such as Precision, Recall and f-measure for 20 NG using Ontology and frequent itemset clustering techniques

- Ontology based Apriori Clustering on 20NG dataset, the f-measure is 97.73% higher than traditional Apriori Clustering on the Modified TF/IDF, f-measure is 97.73% is efficient than traditional TF/IDF.
- FP-Growth with Ontology on 20NG dataset, the Modified TF/IDF has efficient f-measure of 97.65% higher than traditional FP Growth with Ontology, the f-measure is 98.76% is efficient than traditional TF/IDF.
- FP Bonsai tree with Ontology on 20NG dataset, f-measure is 99.51% efficient than traditional FP Bonsai with Ontology on the Modified TF/IDF, f-measure is 99.14% efficient than traditional TF/IDF.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Clustering Algorithm</th>
<th>Dimensionality reduction and feature selection</th>
<th>Purity</th>
</tr>
</thead>
<tbody>
<tr>
<td>20NG</td>
<td>Apriori Algorithm</td>
<td>TF/IDF</td>
<td>0.741</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modified TF/IDF</td>
<td>0.752</td>
</tr>
<tr>
<td></td>
<td>Ontology-based Apriori Algorithm</td>
<td>TF/IDF</td>
<td>0.796</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modified TF/IDF</td>
<td>0.805</td>
</tr>
<tr>
<td></td>
<td>FP Growth Algorithm</td>
<td>TF/IDF</td>
<td>0.796</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modified TF/IDF</td>
<td>0.802</td>
</tr>
<tr>
<td></td>
<td>Ontology-based FP Growth Algorithm</td>
<td>TF/IDF</td>
<td>0.802</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modified TF/IDF</td>
<td>0.845</td>
</tr>
<tr>
<td></td>
<td>FP Bonsai Algorithm</td>
<td>TF/IDF</td>
<td>0.85</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modified TF/IDF</td>
<td>0.861</td>
</tr>
<tr>
<td></td>
<td>Ontology-based FP Bonsai Algorithm</td>
<td>TF/IDF</td>
<td>0.865</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modified TF/IDF</td>
<td>0.876</td>
</tr>
</tbody>
</table>

Table 6.11 Performance measure purity for 20 NG using Ontology and frequent itemset clustering techniques

From the Table 6.11, Performance measure in terms of purity for 20 NG using Ontology and frequent itemset clustering techniques

- Ontology based Apriori Clustering on 20NG dataset, the purity is 93.42% higher than traditional Apriori Clustering on the Modified TF/IDF.
- Comparison of Modified TF/IDF on Ontology based Apriori Clustering, the purity is 98.88% is efficient than traditional TF/IDF.
• FP Growth with Ontology on 20NG dataset, the Modified TF/IDF has higher purity by 94.91% than traditional FP Growth with Ontology.
• Comparison of Modified TF/IDF on FP Growth with Ontology, the purity is 94.91% is efficient than traditional TF/IDF.
• FP Bonsai tree with Ontology on 20NG dataset, the purity is 98.29% higher than traditional FP Bonsai with Ontology on the Modified TF/IDF.
• Comparison of Modified TF/IDF on FP Bonsai tree with Ontology, the purity is 98.74% efficient than traditional TF/IDF.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Clustering Algorithm</th>
<th>Memory Utilization (MB)</th>
<th>Execution Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20NG</td>
<td>Apriori Algorithm</td>
<td>26</td>
<td>265000</td>
</tr>
<tr>
<td></td>
<td>Ontology-based Apriori Algorithm</td>
<td>17</td>
<td>204000</td>
</tr>
<tr>
<td></td>
<td>FP-Growth Algorithm</td>
<td>22</td>
<td>240000</td>
</tr>
<tr>
<td></td>
<td>Ontology-based FP-Growth Algorithm</td>
<td>15</td>
<td>212000</td>
</tr>
<tr>
<td></td>
<td>FP-Bonsai Algorithm</td>
<td>16</td>
<td>213000</td>
</tr>
<tr>
<td></td>
<td>Ontology-based FP-Bonsai Algorithm</td>
<td>9</td>
<td>162000</td>
</tr>
</tbody>
</table>

Table 6.12 Memory Utilization and Execution time of 20 NG using Ontology and frequent itemset clustering techniques

From the Table 6.12, Memory Utilization of 20 NG using Ontology and frequent itemset clustering techniques are discussed as below,

• Ontology based Apriori algorithm takes 10 MB less memory utilization when compared to traditional Apriori algorithm 20 NG and consume 14 MB less memory utilization than the traditional Apriori algorithm.
• Ontology based FP-Growth tree takes 7 MB less memory utilization when compared to traditional FP-Growth tree on 20 NG and consume 12 MB less memory utilization than the traditional FP-Growth tree algorithm.

• Ontology based FP-Bonsai algorithm consumes 7 MB less memory utilization when compared to traditional FP-Bonsai algorithm on 20 NG and consume 11 MB less memory utilization than the traditional FP-Bonsai algorithm.

From the above Table 6.12, Execution time of 20 NG using Ontology and frequent itemset clustering techniques are given

• Ontology based Apriori algorithm provides 76.98% faster execution time when compared to traditional Apriori algorithm on 20 NG dataset.

• Ontology based FP-Growth algorithm provides 88.33% faster execution time when compared to the traditional FP-Growth algorithm on 20 NG dataset.

• Ontology based FP-Bonsai algorithm provides 76.05% faster execution time when compared to the traditional FP-Bonsai algorithm on 20 NG dataset.
From the Table 6.13, Performance measure such as precision, recall and f-measure for TDT-2 using Ontology and frequent itemset clustering techniques are discussed

- Ontology based Apriori Clustering on TDT-2 dataset, the f-measure is 97.70% higher than traditional Apriori Clustering on the Modified TF/IDF.
• Comparison of Modified TF/IDF on Ontology based Apriori Clustering, the f-measure is 97.70% is efficient than traditional TF/IDF.
• FP-Growth with Ontology on TDT-2 dataset, the Modified TF/IDF has efficient f-measure of 98.21% efficient than traditional FP-Growth with Ontology.
• Comparison of Modified TF/IDF on FP-Growth with Ontology, the f-measure is 99.52% is efficient than traditional TF/IDF.
• FP-Bonsai tree with Ontology on TDT-2 dataset, the f-measure is 99.76% higher than traditional FP-Bonsai with Ontology on the Modified TF/IDF.
• Comparison of Modified TF/IDF on FP-Bonsai tree with Ontology, the f-measure is 98.81% is efficient than traditional TF/IDF.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Clustering Algorithm</th>
<th>Dimensionality reduction and feature selection</th>
<th>Purity</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDT-2</td>
<td>Apriori Algorithm</td>
<td>TF/IDF</td>
<td>0.779</td>
</tr>
<tr>
<td></td>
<td>Modified TF/IDF</td>
<td></td>
<td>0.787</td>
</tr>
<tr>
<td></td>
<td>Ontology-based Apriori Algorithm</td>
<td>TF/IDF</td>
<td>0.81</td>
</tr>
<tr>
<td></td>
<td>Modified TF/IDF</td>
<td></td>
<td>0.839</td>
</tr>
<tr>
<td></td>
<td>FP Growth Algorithm</td>
<td>TF/IDF</td>
<td>0.802</td>
</tr>
<tr>
<td></td>
<td>Modified TF/IDF</td>
<td></td>
<td>0.826</td>
</tr>
<tr>
<td></td>
<td>Ontology-based FP Growth Algorithm</td>
<td>TF/IDF</td>
<td>0.829</td>
</tr>
<tr>
<td></td>
<td>Modified TF/IDF</td>
<td></td>
<td>0.868</td>
</tr>
<tr>
<td></td>
<td>FP Bonsai Algorithm</td>
<td>TF/IDF</td>
<td>0.883</td>
</tr>
<tr>
<td></td>
<td>Modified TF/IDF</td>
<td></td>
<td>0.888</td>
</tr>
<tr>
<td></td>
<td>Ontology-based FP Bonsai Algorithm</td>
<td>TF/IDF</td>
<td>0.876</td>
</tr>
<tr>
<td></td>
<td>Modified TF/IDF</td>
<td></td>
<td>0.913</td>
</tr>
</tbody>
</table>

Table 6.14 Performance measure purity for TDT-2 using Ontology and frequent itemset clustering techniques
From the Table 6.14, Performance measure in terms of Purity for TDT-2 using Ontology and frequent itemset clustering techniques is given

- Ontology based Apriori Clustering on TDT-2 dataset, the purity is 93.8% higher than traditional Apriori Clustering on the Modified TF/IDF.
- Comparison of Modified TF/IDF on Ontology based Apriori Clustering, the purity is 96.54% is efficient than traditional TF/IDF.
- FP-Growth with Ontology on TDT-2 dataset, the Modified TF/IDF has higher purity by 95.16% than traditional FP-Growth with Ontology.
- Comparison of Modified TF/IDF on FP-Growth with Ontology, the purity is 95.51% efficient than traditional TF/IDF.
- FP-Bonsai tree with Ontology on TDT-2 dataset, the purity is 97.26% higher than traditional FP-Bonsai with Ontology on the Modified TF/IDF.
- Comparison of Modified TF/IDF on FP-Bonsai tree with Ontology, the purity is 95.95% efficient than traditional TF/IDF.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Clustering Algorithm</th>
<th>Memory Utilization (MB)</th>
<th>Execution Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDT-2</td>
<td>Apriori Algorithm</td>
<td>38</td>
<td>355000</td>
</tr>
<tr>
<td></td>
<td>Ontology-based Apriori Algorithm</td>
<td>24</td>
<td>281000</td>
</tr>
<tr>
<td></td>
<td>FP-Growth Algorithm</td>
<td>32</td>
<td>337000</td>
</tr>
<tr>
<td></td>
<td>Ontology-based FP-Growth Algorithm</td>
<td>20</td>
<td>269000</td>
</tr>
<tr>
<td></td>
<td>FP-Bonsai Algorithm</td>
<td>30</td>
<td>303000</td>
</tr>
<tr>
<td></td>
<td>Ontology-based FP-Bonsai Algorithm</td>
<td>19</td>
<td>221000</td>
</tr>
</tbody>
</table>

Table 6.15 Memory Utilization and Execution time of TDT-2 using Ontology and frequent itemset clustering techniques
From the Table 6.15, Memory Utilization of TDT-2 using Ontology and frequent itemset clustering techniques are compared and listed below

- Ontology based Apriori algorithm takes 10 MB less memory utilization when compared to traditional Apriori algorithm on the TDT-2 data set and consume 14 MB less memory utilization than the traditional Apriori algorithm.
- Ontology based FP-Growth tree takes 7 MB less memory utilization when compared to traditional FP-Growth tree on TDT-2 data set consume 12 MB less memory utilization than the traditional FP-Growth tree algorithm.
- Ontology based FP–Bonsai algorithm consumes 7 MB less memory utilization when compared to traditional FP–Bonsai algorithm on TDT-2 data set, consumes 11 MB less memory utilization than the traditional FP –Bonsai algorithm.

From the Table 6.15, Execution time of TDT-2 using Ontology and frequent itemset clustering techniques

- Ontology based Apriori algorithm provides 79.15% faster execution time when compared to traditional Apriori algorithm on the TDT-2 dataset.
- Ontology based FP-Growth algorithm provides 79.82% faster execution time when compared to the traditional FP-Growth algorithm on the TDT-2 dataset.
- Ontology based FP-Bonsai algorithm provides 72.93% faster execution time when compared to the traditional FP-Bonsai algorithm on the TDT-2 dataset.
6.8. SUMMARY

In this chapter, variety of frequent itemset based clustering process is explained. The state-of-the-art approach deals with the three different datasets such as Reuters-21578, 20 NG and TDT-2 based on Apriori, FP-Growth, and FP-Bonsai. Hence, Apriori algorithm is used to cluster the document by mining single dimensional Boolean association with respect to the minimum support and minimum confidence. The Apriori in document clustering suffers from trivial limitations such as a number of iterations for scanning input. The important issues resolved by making Ontology framework to Apriori algorithm, produce pruning and time consumption of document clustering. The process can also get the input itemset of larger datasets. To improve pruning and reduce the time in document clustering, various approaches are taken on to clustering processes such as FP-Growth and FP-Bonsai. The scanning of input data item is very significant to clustering. To address this problem, this chapter ensemble clustering approach such as FP-tree construction and it is applied to the Ontology pattern. This creates the iteration, based on frequent itemsets generation. Mining the Frequent Patterns without candidate generation is called an FP-Tree method. It is used to extend a large amount of data with less time to improve the speed of the process than Apriori, and also faster than tree projection. It removes the repeated database scan and also FP-Tree building based on the input dataset item. Finally, depending upon Apriori algorithm, FP-Growth method, and FP-Bonsai, the frequent itemsets are mined and applied to Ontology framework. The combined results of Ontology based Frequent-Pattern mining performance results show that the Ontology based frequent pattern mining are best ever with respect to purity and time management.