Deriving Frequent Itemsets from Lossless Condensed Representation

A. Subashini, M. Karthikeyan

Abstract: In data mining, major research topic is frequent itemset mining (FIM). Frequent Itemsets (FIs) usually generating a large amount of itemsets from database it causing from high memory and long execution time usage. Frequent Closed Itemsets (FCI) and Frequent Maximal Itemsets (FMI) are a reduced lossless representation of frequent itemsets. The FCI allows to decreasing the memory usage and execution time while comparing to FMIs. The whole data of frequent itemsets (FIs) may be derived from FCIs and FMIs with correct methods. While various study has presented several efficient approach for FCIs and FMIs mining. In sight of this, that we proposed an algorithm called DCFI-Mine for capably derive FIs from Closed FIs and RFMI algorithm derive FMIs to FIs. The advantages of DCFI-Mine algorithm has two features: First, efficiency, different existing algorithm that tends to develop an enormous quantity of itemsets all through process, DCFI-Mine process the itemsets straight without candidate generation. But in proposed RFMI multiple scan occurs due to search of item support so efficiency is less than proposed algorithm DCFI-Mine. Second, in terms of losslessness DCFI-Mine and RFMI can discover complete frequent itemset without lapse. Experimental result shows That DCFI-Mine is best deriving FIs in term of memory usage and executions time.

Keywords: Deriving algorithm, Frequent itemset mining, maximal itemset, closed, itemset mining, Lossless condensed representation.

I. INTRODUCTION

The purpose of FIs (Frequent Itemsets) mining [1]-[3] is to determine the sets of item in database looking frequently. Those skills have been generally applicable to many real-life application are association analysis [4], text mining [6], bioinformatics [7] and web mining [5]. Numerous effective procedures has been established for mining FIs are Apriori [1], Eclat [8] and FP-Growth [7]. When FIs mining from databases type is sparse, these procedures typically have performance good because the correlation of items between in sparse database are comparatively weak, then the FIs length in the database are too comparatively short. The traditional FIs mining techniques be able to make use of the downward closure property [1], [11] to prune effectively the space of search. A typical data type Sparse transaction dataset examples are used in real-life situations, several types of dense dataset are as features of plant (e.g., Mushrooms datasets [10]), census statistical data (e.g., the Pumsb datasets [10]), and records in games steps (e.g., the Connect and Chess datasets [10]). To resolve the overhead difficulties, various study has enthusiastic to develop condensed representation of Frequent Itemsets. The experiment result of prior study [11]-[14] has present the condensed representation which can be greatly decrease the memory usage and execution times from dense dataset. Several type of condensed representation of FIs has been propose, like maximal itemsets [12], free itemsets [11], maximal itemsets [12], closed itemsets [14], and generator itemsets [13]. Between those representations, (maximal itemsets) FMIs and FCIs is one of the popular one. The frequent maximal itemsets (FMI) and FCI is a lossless reduced representation [13] of Frequent Itemsets. Through the exact algorithms, entire FIs with their support be able to derive completely from all the FMIs and FClIs. So, the whole set of FMIs, FCIs are preserves the whole data without loss of Frequent Itemsets. The various studies [13], [15]-[17] had efficiently proposed for mining FMIs, some of them think through to develop an algorithm for deriving efficiently FIs from FMIs and FCIs (derive an algorithm used for easiness). But evolve deriving efficient algorithm is a significant works for both FCIs and FMIs. When every FIs cannot stay positively mined from dense data type, an another solution, to mine whole FCIs and FMIs from dataset, next apply deriving effective algorithm to derive the set of complete Frequent Itemsets with their supports also from FMIs and FCIs.

A maximal frequent itemsets has no super subsets is called frequent. Though FIs be able to condense for mines a “border” in an itemsets lattice, as found as in [9]. The whole itemset below the borders are all frequent and above the border are infrequent. Bayardo [19] intro MaxMiner, that extend Apriori for mining only “lengthy” pattern (FMI). To decrease, the space of search. Then we able to decide that some of its subsets was also a frequent so supersets of frequency prune decreases the time of search radically, MaxMiner need several pass to acquire whole lengthy pattern. Though, the LevelWise traditional deriving algorithm [14] adopts topdown strategy and breadth-first search to derives the FIs. This one derives the FIs of (k-1) length from closed FIs and FIs of k lengths. Though, traditional derive algorithms [12] implements topdown and breadth-first search approach to derive FIs is called levelwise algorithm.
A weakness of a LevelWise algorithm that need to preserve the whole sets of the FIs of k length and (k-1) length in memory for duration of the derive process. Hence, when there was numerous FIs of k length and (k-1) length, algorithm levelwise will takes a large memory and also running out of memory due to deriving task. Likewise, the deriving procedure of the LevelWise consist of a huge number of search tasks, results in lengthy executions time problematic. In sight of above, the proposed algorithm, named DCFI-Mine (Discovering Compact Frequent Itemset is Pattern Growth based), for effective derive FIs and with the support from FCIs. An ideas design of DCFI-Mine construct the FP-Tree first to preserve the data of FCIs, then from FP-tree the FIs is generated using FP-Growth. However, traditional algorithm FP-Tree and FP-Growth is not considered for derive task then applying straight will results improper result. So, we proposed two approaches then alter the algorithm process of FP-growth and FP-tree, constructing those to derive the correct data of FIs from the FCIs. The proposed approaches are respectively call the selecting maximum supports and replace maximum supports. Then previous method is applying through building of FP-Tree and pattern generate latter. The experiments that we performs on Dense and sparse real-life dataset to estimate the presentation on the propose algorithm called DCFI-Mine. Extensive experiment shows that DCFI-Mine is fairly efficient and derive all FIs with supports. RMFI (Recover all Frequent Itemsets from maximal Items) to recover entire FIs from set of maximal FIs. Additionally, DCFI-Mine comparing the performance is considerably better than RFMI algorithms in term of execution time and memory usages.

Table I. An example of a transaction database

<table>
<thead>
<tr>
<th>TID</th>
<th>TRANSACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A E O U</td>
</tr>
<tr>
<td>2</td>
<td>E I U</td>
</tr>
<tr>
<td>3</td>
<td>A E O U</td>
</tr>
<tr>
<td>4</td>
<td>A E I U</td>
</tr>
<tr>
<td>5</td>
<td>A E I O U</td>
</tr>
<tr>
<td>6</td>
<td>A E I O</td>
</tr>
</tbody>
</table>

II. BASIC CONCEPT AND DEFINITIONS

Let assume P= \{I_1, I_2, ..., I_k\} be a finite set of distinct items. A transaction database \(D = \{i_1, i_2, ..., i_k\}\) is a transaction sets, where every transactions \(i_r \in D (1 \leq r \leq K)\) is a subsets of \(P\) and it has sole identifier transactions \(\epsilon\). An itemsets is a sets of item. If entire item of an itemsets \(Z\) can contain in a transactions \(T\), \(Z\) is supposed to contain in \(T\), which was denote as \(Z \subseteq T\).

**Definition 1 (Tidset).** Transaction identifiers set of an itemsets \(Z\) is denote as \(r(Z)\) and the set of transactions ID(identifiers) defined as entire transaction contains \(Z\) in \(D\).

**Definition 2 (Itemset Length).** Length of an itemset of \(Z = \{i_1, i_2, ..., i_k\}\) is define as \(L\), where \(L\) was the complete distinct item in \(Z\).

**Definition 3 (Itemset Support of items).** Support count of an itemset of \(X\) was denoted as SCI(\(Z\)) and defined as \(|r(Z)|\). Moreover, the support of \(Z\) is defined as SCI(\(Z\)) \| D \|, where \(|D|\) is the total number of transactions in \(D\).

**Definition 4 (Frequent itemset).** An itemsets \(Z\) is called as frequent itemsets if and only if the supp(\(Z\)) isn’t less than a user-specified minimum support threshold \(\theta (0 < \theta \leq 1)\). Otherwise, \(Z\) is low support.

**Definition 5 (Closed itemset).** An itemsets \(Z\) is called as closed itemsets if it has no proper superset \(Y\) in \(D\) such that SCI(\(Z\)) = SCI(\(Y\)). Else, \(Z\) is not a closed itemset. The complete set of closed itemsets is denoted as \(C\).

**Definition 6 (Frequent closed itemset).** A closed itemset \(Z\) is called frequent closed itemsets if and only the support(\(X\)) was not less than the user-specified minimum support threshold \(\theta (0 < \theta \leq 1)\). The complete set of Frequent closed itemset is denote as \(FCl\).

**Definition 7 (Maximal itemset).** An itemsets \(Z\) is called Maximal itemsets if none of its immediate superset is frequent. Else, \(Z\) is not a maximal itemset.

**Definition 8.** (Frequent Maximal itemset). An itemsets \(Z\) is called maximal itemsets iff the support of \(Z\) was not less than the user-specified minimum support threshold \(\theta (0 < \theta \leq 1)\). The complete set of Frequent Maximal Itemset is denoted as \(FMI\).

Problem Statement for Maximal Itemset Mining: Given a dataset \(D\) with itemset support, and a user-specified \(\min\_sup\) threshold, the problematic is to determine the whole set of FMs in \(D\). Then compared frequent closed itemset with frequent maximal Itemset from altered viewpoints. In terms of compression, the sets FMI is lesser than FCI because FCI is a subsets of FMI. A FMs only says the user, certain of its subset can be FIs. But FCIs are additional significant as this are lossless and first eliminate redundancy [20, 21]. In terms of recovering, representation of both can be used to recover entire FIs, then it’s extra costly with FMs. This is a reason that FMs aren’t lossless. So, asking the support of a FIs using FMs needs to make a surplus databases scan, though it’s not essential for closed FIs. Both FMI and FCI are represent because both of them can stay used to find entire FIs.

III. THE PROPOSED METHOD

3.1 Derive Frequent Itemsets from Closed Itemsets:

In this division the proposed algorithm DCFI-Mine (Deriving Frequent Itemsets from closed itemsets). In first approach of function InputFCI was developed based on the modification of FP-Growth and FP-Tree refer to [2]. And second approach RFMI function deriving FIs from Maximal frequent itemset. In first approach: The FP-Tree is a tree based structure adopted in DCFI-Mine. The DCFI-Mine, instead of transactions of the FP-Tree are used to retain the data of FCIs. The FP-Tree of every node is denotes as ‘T’ and an items denotes as I \& I^*, it consists of fields five: T.Name, T.Counts, T.Parents, T.Childs and T.Links. In T.Name and T.Counts field store respectively the names of item ‘I’ and value of count ‘I’. The T.Parents and T.Childs field store respective nodes of parent ‘T’ and a nodes of children ‘T’. The T.Links fields store a nodes link, that which point to next nodes have the items name as same ‘T’. Every FP-Tree was associates with the header tables.

The information of items records in a header table and those items is every sub-itemset of FCIs (Frequent closed itemsets). The HT header table, every items I \& I^* in HT consist of three field: HT[I].Names, HT[I].Counts and HT[I].Links, that which correspondingly registers the names of items ‘I’, a values of counts ‘I’, and a link nodes point a FP-Tree.
nodes have the name of item same as I. refer[2].

A. Construction of a FP-Tree for Deriving Task

This subdivision intro the building a FP-Tree to process with following four stages:

Stage 1. Let a C is the entire itemset of FCI (Frequent Closed Itemset) exposed from a database D. In DFCI-Mine ‘C’ scans one time and derive their support count of every items looking in C rendering towards the Selecting Maximum Support of Item (SMS-I) approach.

\[
\begin{array}{|c|c|}
\hline
\text{Frequent Closed Itemset} & \text{Support Count} \\
\hline
\{EUA\} & 4 \\
\{EU\} & 5 \\
\{EI\} & 4 \\
\{E\} & 6 \\
\{EO\} & 4 \\
\{EU\} & 3 \\
\{EUAO\} & 3 \\
\hline
\end{array}
\]

Fig. 1(a). FCM with Support count

Stage 2. Every arranged FCI Inserting into T FP-Tree by calling the function InputFCI. The pseudo code of function InputFCI was presented in Fig. 2. Every time when a FCI, X in Lines 7. In Fig.3 show an example inserting closed itemset Figure 3(a), 3(b)-interchange maximum support count for item E:5 and U:5, same as in 3(c) and 3(d) also.

Method 1 (SMS-I). Let an item be I and \( S(I) = \{c_1, c_2, \ldots, c_z\} \) be the set of entire FCI’s contain I. The support count of I was equivalent to \( \text{max}\{\text{SCI}(c_1), \text{SCI}(c_2), \ldots, \text{SCI}(c_z)\} \). Fig.1 shows that left table is closed itemset with minimum support count is 3 and in right table explains the selection of maximum support item approach.

Stage 3. Create an H header table and items placed into header table H by a stable sort in order f. next, create a root node R of T FP-Tree. Then, scans C one time and sorts item in every FCI’s by the sorted order f.

Stage 4. All through FP-Tree structure process, entire nodes having the item name same will be link together via the link nodes of HT header tables and associated node in T FP-Tree. Over those link nodes.

\[
\text{Method 2 (RMS)}.
\]

If \( \text{SCI}(X) > T'.\text{Counts} \), then swapping \( T'.\text{Counts} \) by \( \text{SCI}(X) \). Else, if \( \text{SCI}(X) \) isn’t greater than \( T'.\text{Counts} \), then it creates a replacement node \( T'' \) and it sets T child node. Besides, the \( T'.\text{Counts} \) and \( T'.\text{Links} \) set toward support counts of X and finally nodes was created which have items in same name as \( I_w \), respectively in lines 3 and 6. Following, increments the variable w by 1. then, if w isn’t greater than K, then call the repetitive function InputFCI (\( T', I_l \)) for process the \( w-th \) items of X in Lines 7.

Fig.2. The Pseudo code of the function

Function : InputFCI(T, I_w)

01 If(T has a child T’ such that T’.names == I_w)
02 then T’.counts max{T’.counts, SCI(X)};
03 else
04 Create a new child node T’ and
05 Set T’.names I_w, T’.counts SCI(X), T’.parent T;
06 Set T’.links to the lastly created node having the same item name as I_w;
07 w ← w +1, If(w < K), call InputFCI(T, I_w);

Published By:
Blue Eyes Intelligence Engineering & Sciences Publication

ISSN: 2249 – 8958, Volume-9 Issue-3, February, 2020
Deriving Frequent Itemsets from Lossless Condensed Representation

DCFI-Mine allows towards nodes travel have items name as same powerfully.

B. Deriving Frequent Itemsets:
Afterward constructing T FP-Tree, HT header tables then called function DFCI-Mine (HT, F, β) towards deriving FIs from FP-Tree T, the variables α is initialized to φ. This process of the function DFCI-Mine (HT, F, α) working as following.

Stage 1. This algorithm visit every item of I in HT header tables. For every visited items, the algorithms output {β U I} then HT[I].Counts as FIs and with their support counts. Over the link nodes in HT[I].Links, DFCI-Mine pass through entire node having the name of same item as in T FP-Tree.

Stage 2. Let α = {α₁, α₂, ..., αₙ} a set of node I have its name of item is same. For every node αᵢ ∈ α (1 ≤ i ≤ n), this algorithm visit from parent nodes of αᵢ to R root node of T. Then, this algorithms collect the item of each visited nodes. Those item form a sequences Cᵢ (1 ≤ i ≤ n) and that sequences is called conditional itemsets. Each conditional itemset Cᵢ was associated with a counts called condition of itemsets and was denote as C(αᵢ). This count of conditional itemsets Cᵢ is primarily set to αᵢ.Counts. Then handling entire nodes in α in same manner, conditional databases of the itemsets {α U I} was acquired. Let as D_{α U I} = {C₁; C(α₁), C₂; C(α₂) ……, Cₙ; C(αₙ)} denoted the conditional databases of {α U I}.

Stage 3. Scan the conditional databases once D_{α U I} to discover entire distinct item in the conditional itemset. For every items X, this algorithms evaluates the support counts of frequent itemsets {β U I X} through the proposed Selecting Maximum Support for itemsets(SMS-IS) approach.

Method 3. (SMS-IS) Let X be an items appeared in D_{β U I} then a set of entire conditional itemsets be C'' = {C₁; C(α₁), C₂; C(α₂) ……, Cₙ; C(αₙ)} contain X in D_{α U I}. Then sup_count of a frequent itemsets {α U I} was equivalent to max{C(α₁), C(α₂) ……, Cₙ; C(αₙ)} which was conditional itemset count of Cᵢ (1 ≤ i ≤ n). Fig.4, shows a conditional databases of itemsets {O} are {EUA}:3 and {EU}:4. These are the two conditional itemsets {EUA}:3 and {C}:4. In the steps of algorithm, the worth of the variables α is currently φ. The conditional itemsets covering the items {E} is {(EUA):3, {E}:4}. The support count of φ{JOUE}{OE} is Max{CCI({EUA})}, CIC({E}) = max{3, 4} = 4. Therefore, this algorithms output {OE} as a FIs and its support count is 4.

Fig.4(a) Conditional Database of {O}

<table>
<thead>
<tr>
<th>Conditional Database of {O}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conditional Itemset</td>
</tr>
<tr>
<td>{EUA}</td>
</tr>
<tr>
<td>{E}</td>
</tr>
</tbody>
</table>

Fig.4. An example of the Selecting Maximum Support for ItemSet (SMS-IS)

<table>
<thead>
<tr>
<th>FIs</th>
<th>Selecting Maximum Support</th>
<th>Sup_count</th>
</tr>
</thead>
<tbody>
<tr>
<td>{OE}</td>
<td>Max(CCI({EUA})C(α₁))</td>
<td>4</td>
</tr>
<tr>
<td>{OU}</td>
<td>Max(CCI({EUA}))</td>
<td>3</td>
</tr>
<tr>
<td>{OA}</td>
<td>Max(CCI({EUA}))</td>
<td>3</td>
</tr>
</tbody>
</table>

Max{5,4}

Stage 4. This algorithm locates item in D_{α U I} into a novel HT’ header tables in the sort order f. Then, algorithms scan conditional databases D_{α U I} for a next time and sort item in every conditional itemsets in D_{α U I} given to f.

Stage 5. Create a novel R’ nodes as the root of a novel T’ FP-Tree for the itemsets {α U I}. Insert each sort conditional itemsets in D_{α U I} into T’ to build the conditional FP-Tree of itemsets {α U I}. All time after a sort conditional itemsets J is insert then sets the variable N and Iₑ into R and I₁, next calling a function InputFCI( Tₑ, Iₑ). When construct T’, entire node in T’ have the name of items same that was link organised by the link nodes.

Stage 6. When constructing T’ FP-Tree has node other than the R’ root nodes, next to derive FIs call the function DFCI-Mine (H’, T’, α U I).

3.2 Recover all Frequent Itemset from Maximal Items:
In this section, the proposed algorithm named as RMFI (Recover all Frequent Itemset from Maximal Items) is efficient to derive entire FIs from set of maximal FIs. The min_sup threshold and the set of FMIs are taken as input. RMFI recover entire FIs is outputs respecting to min_sup. RMFI process as follows. The set of maximal frequent itemset (FMI) allow recovering all FIs. FMIs defines, that a maximal frequent itemset not have proper superb subsets that’s a frequent itemset. Thus, if an Itemsets is frequent, it is also a proper subsets of maximal itemsets. Figure 5 represents simple algorithms for recovering entire FIs from the set of FMIs. It produce entire subset of all the FMIs. Furthermore, it makes a check to identify if a FIs has previously been outputs in line 3 because a FIs can be a subset of other than one maximal Itemset. Afterward FIs had recovered and an additional scan in database can be executed to calculate their support, if required.

Fig. 5 Algorithm to derive all FIs from FMIs.

Function: RMFI (J/P, a set of Maximal Itemset FMI)
01 For each FIs j ∈ FMI
02 For each frequent k(j)
03 If k has not been output
04 Then output k.

IV. RESULT AND DISCUSSION
In this division, we estimate the presentation of the proposed algorithm DFCI-Mine, RMFI algorithm and compared the both algorithms. This determination of two algorithms are same, which derives FIs with their supports from FMIs and FCI. This experiment is directed on a computer equipped with Intel i5 with 8 GB of RAM, compile on Windows 10 OS. This algorithm was executed in Java programming language. Two real-life dataset that is Mushrooms (Dense) and BMS_WebView_1 (sparse) are used for the performance estimate. All the dataset are download from SPMF [10].
Table II: Characteristics of Datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Number of distinct Items</th>
<th>Number of transactions</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mushrooms</td>
<td>119</td>
<td>8124</td>
<td>Dense</td>
</tr>
<tr>
<td>BMS_WebView_1</td>
<td>59,602</td>
<td>497</td>
<td>Sparse</td>
</tr>
</tbody>
</table>

In Table II show the dataset characteristics. The executions time and its memory usage of algorithms was estimated. We extent the algorithm maximum memory usage by Java API. Figures 6 shows the executions time and the memory usage of compare algorithm. The executions time of DFCI-Mine and RFMI on the dense dataset that is Mushrooms are shown in Table III and in Fig. 6(a), 6(b) and on Sparse datasets BMS_WebView_1 are presented in Table IV and in Fig. 7(a) and 7(b), respectively. As exposed in those figure, the performance of DFCI-Mine is best compared to RFMI. For example, using Mushroom datasets, when the min_sup threshold is lesser than 1%, RFMI even can’t complete the entire derive tasks due to running out of memory. However, our proposed DFCI-Mine takes minimum time to derive all FIs and with support from FCIIs because of RFMI take on many scans for searching and used top-down methods to deriving all FIs from FMIIs. RFMI occupy much times for search the similar item. Furthermore, it keep two level of itemset in the memory, that which occupies great memory space during the derive task. However, DFCI-Mine implements pattern growth with divide-and-conquer system to deriving all FIs, which divides the entire derive task into minor separate from one and so, normally it take less times than RFMI. In figures, we can witness that presentation between the RFMI and DFCI-Mine is lesser when the threshold was higher. It happens because that when the threshold was higher, the number of FCIIs is lesser and the length of FCIIs are relatively short. As shown in the experiment, no matter about the executions time or memory usages, the propose DFCI-Mine and RFMI generally has better performance of deriving all FIs.

Table III: Mushrooms (dense) Datasets on both proposed algorithms

<table>
<thead>
<tr>
<th>Min_Supp (%)</th>
<th>Execution Time(s)</th>
<th>Memory Usage(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DFCI-Mine</td>
<td>RMFI</td>
</tr>
<tr>
<td>0.1</td>
<td>1415</td>
<td>12063</td>
</tr>
<tr>
<td>0.2</td>
<td>92</td>
<td>969</td>
</tr>
<tr>
<td>0.3</td>
<td>4</td>
<td>170</td>
</tr>
<tr>
<td>0.4</td>
<td>1</td>
<td>183</td>
</tr>
<tr>
<td>0.5</td>
<td>1</td>
<td>161</td>
</tr>
<tr>
<td>0.6</td>
<td>1</td>
<td>200</td>
</tr>
<tr>
<td>0.7</td>
<td>1</td>
<td>137</td>
</tr>
<tr>
<td>0.8</td>
<td>0</td>
<td>172</td>
</tr>
<tr>
<td>0.9</td>
<td>0</td>
<td>146</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>146</td>
</tr>
</tbody>
</table>
Deriving Frequent Itemsets from Lossless Condensed Representation

V. CONCLUSION

We propose an algorithm, named DFCI-Mine (Deriving Frequent Itemsets from closed Itemsets) and RFMI (Recover all FIs from maximal Itemsets) for the FIs deriving task. Likewise, we proposed two Methods in DFCI-Mine, named selecting maximum support of Itemsets (SMS-I) and replace maximum support (RMS), and join them into DFCI-Mine for efficient derive the whole FIs with its supports. The SMS-IS approach is proposed for computing the exact support with FIs by data from FCIs. In RMS approach is implemented through building FP-Tree, then its proposed set of the accurate info of FP-tree node. RMFI recover entire FIs from the set of maximal FIs. This one takes input as min_sup threshold with the set of FMIs. RMFI gives the entire FIs with respect to min_sup. Experiment result shows that the execution time and memory usage of DFCI-Mine are significantly better than that of the RFMI.

REFERENCES


AUTHORS PROFILE

A. Subashini completed her M.Phil degree in the year 2008 at Annamalai University, at present doing her PhD degree in Annamalai University. She has fourteen years of teaching experience. Presently she is working as Assistant Professor in the Department of Computer Application at Government Arts College, C.Muthur, Chidambaram. Her research interests are Neural Networks and Data Mining.

Dr. M. Karthikeyan received the PhD degree from Annamalai University. Presently he is working as Assistant Professor in the Division of Computer & Information Science, Faculty of Science, Annamalai University. He published ten research papers in International journals and eight research papers in national journals. He has nineteen years of teaching experience and five years of research experience. His area of specialization includes Neural networks & Fuzzy systems, Data Mining and Digital Image processing.