Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.18576/amis/13S101

Sequential Pattern Mining using RadixTreeMiner Algorithm and Neural Network-Based Classification

K. Poongodi[*] *and A. K. Sheik Manzoor*

Department of Management Studies, Anna University, CEG Campus, Chennai, Tamil Nadu, India

Received: 2 Mar. 2019, Revised: 2 May 2019, Accepted: 11 May 2019
Published online: 1 Aug. 2019

Abstract: Handling large amount of data arriving from internet-based applications is one of the challenging tasks. Recently, more contributions were made to the data mining algorithms, such as clustering and classification. One of the most commonly-used data mining schemes is Sequential Pattern Mining (SPM). Here, statistically significant and relevant sequential patterns are used for classification purpose, but the complexity grows for the increasing data sizes. This paper introduces a novel approach, namely RadixTreeMiner, for mining sequential patterns from the sequence database, and to classify the data efficiently based on maximal sequential patterns. The proposed RadixTreeMiner algorithm constructs the radix tree from the sequences available in the input database, and then identifies the maximal sequences. Further, the Neural Network (NN) approach is employed in this work for the classification of database based on the maximal sequential patterns. Experimentation of the proposed RadixTreeMiner algorithm uses two standard gene-sequence databases and its performance is evaluated based on the metric Classification Accuracy (CA). From the achieved results, it is evident that the proposed algorithm has better performance with values of 0.9038 and 0.8628 as classification accuracy for both the datasets.

Keywords: Data mining, sequential patterns, radix tree, maximal sequences, neural network, classification accuracy

1 Introduction

Rapid development of web-based technologies has greatly contributed towards the increase in data growth. Also, digitization in every field has forced users to store data in large volumes [1]. In web-based applications, data is generated in huge volumes for different domains, and hence, developing mining-based algorithms for data mining is necessary. Data mining algorithms process the available data in the stored database, and they allow decision making. Data mining algorithms perform two major tasks in a database, which are clustering and classification. Pattern mining algorithms have gained more popularity, as these algorithms identify the interesting, useful and unexpected patterns from the database. Pattern mining algorithms identify the relation between data and thus retrieve interesting and novel patterns. Literature has introduced various forms of pattern mining algorithms, such as itemset mining, sequential pattern mining, correlated pattern mining and behavior mining.

Pattern mining algorithms do not consider the sequential order of events prevailing in the database. So, while using pattern mining algorithms for the database with sequential ordering information, it may lead to information loss. Ignoring sequential information during mining process results in loss of significant information. Some studies have made use of support-based approaches for mining the database with sequential information. It is important to consider sequential patterns for the experimentation, since it may contain significant information. Several domains use sequential information for the analysis. Using support-based approaches for considering sequential patterns may fail in some domain, as it depends on combinatorial search space [2].

For tackling issues related to data mining, SPM was introduced. For evaluating sequential data in large volume, SPM is more compact as it constructs a transaction database from sequences present in the database. Several streams such as web mining [3], classification [4], finding copy-paste bugs in large-scale software code [5] and mining motifs from biological sequences [6] use SPM-based techniques for data mining. The database provided as input to SPM contains data sequences, and each data sequence is represented as

[*] Corresponding author e-mail: poongodik79@gmail.com

ordered list of transactions. The transaction, also called itemset, has a set of literals. Ordering sequence as transaction details depends on time-stamp association. Besides, order also depends on non-time related items. SPM algorithm aims to identify suitable sequential patterns satisfying the user-defined minimum support. The minimum support defines the percentage of sequences in the database having the pattern as required by the user [7]. Even though SPM provided good performance on the database with sequential patterns, it faces over-fitting issues. SPM failed to recognize minor patterns and thus, this resulted in incorrect class label predictions [1]. Several applications, such as protein classification [8], text classification [9], speech recognition [10] and image identification [11] belong to this domain [1].

Sequence classification task is one of the commonly-used tasks in SPM, which allows classification of sequences through training of data samples. While classifying the tasks in the sequence database, it is necessary to assign the class labels to the sequences by gathering necessary knowledge from the domain. Also, knowledge gathering can be done efficiently through the training. In the literature, several pattern mining techniques related to the classification have been developed, and it can be enlisted as Classify-By-Association rules (CBA) [12], sequential pattern-based sequence classifier [13], Classify-By-Sequence (CBS) algorithm [14] and pattern-based sequence classification [15]. These compound sequence classification techniques combine several data mining techniques for improving the classification accuracy. Among various techniques, SPM provides improved classification accuracy, and it provides improved performance in complex environments. In SPM-based classification, pattern mining and classification are done in two stages. In the first stage, frequent sequential patterns are extracted from the database. These frequent patterns are then used for building the classification model in the second stage. Using iterative characteristics for training can significantly improve the mining process, and subsequently enhance the classification accuracy [1, 15].

The primary intention of this research is to design and develop an algorithm, called RadixTreeMiner, for sequential pattern mining. RadixTreeMiner mines the sequential patterns from the sequence database using three important steps. In the first step, a sequence database is recursively projected into a set of smaller radix trees, which is the alternate representation of the projected database utilized in the PrefixSpan algorithm. In the second step, sequential patterns are extracted from the smaller radix trees. In the third step, the maximal patterns are identified from the mined sequential patterns. To prove the maximal sequence is better for classification, the mined maximal sequences are applied to training data for sample selection. Here, the sequences matching with the maximal sequence are given for training the neural network, and finally, data classification is done.

The major contribution of this paper is the design of a RadixTreeMiner algorithm. The proposed RadixTreeMiner algorithm develops the radix tree for the itemsets present in the sequences and thus, mines the sequential patterns from the database. The mined sequential patterns are then used for data classification.

The organization of the paper structure is defined as follows: Section 2 surveys eight literary studies dealing with pattern mining, further challenges prevailing in these techniques are also discussed. Section 3 describes proposed RadixTreeMiner algorithm for classifying the database, and its results are discussed in Section 4. Finally, the conclusion to this research work is presented in Section 5.

2 Related work

This section presents various literary work dealing with pattern mining to classify a large amount of data.

2.1 Literature survey

Bao Huynh et al. [16] presented the parallel Dynamic Bit Vector Frequent Closed Sequential Patterns (pDBV-FCSP) scheme for parallel mining of data. This approach overcomes the common issues in parallel mining, such as overhead of communication, synchronization and data replication. This approach could not use other architectures to improve the efficiency of the mining process. Dmitriy Fradkin et al. [17] presented the direct sequential pattern mining approach and Bidirectional Extension-Discriminative Class (BIDE-DC) method for sequential mining of patterns. This method provides an efficient solution with good classification performance. BIDE-DC has few match patterns during mining and it also provides worse performance. The cost of accuracy can be high. Chieh-Yuan Tsai et al. [1] proposed two-stage SPM-based sequence classification scheme for classifying the sequential patterns. The algorithms achieved improved classification accuracy, but they do not handle time-series sequence datasets containing numerical values. Ana Palacios et al. [18] presented the sequence mining algorithm for Engine Health Monitoring (EHM). This pattern mining concept achieved improved results for the database containing sequences of alphabets with small length. If the length of the sequence increases, the algorithm resulted in information loss during mining.

Fabio Fumarola et al. [7] proposed Closed FAST (CloFAST) sequential pattern mining algorithm based on sparse id-lists for pattern mining. The proposed CloFAST model compensated the memory and time required for mining the long length sequences by improving the speed

of mining. Besides its improved speed, the CloFAST returns constraints of fixed memory for some decisive datasets. Akiz Uddin Ahmed et al. [2] proposed the Weighted Uncertain Interesting Pattern Mining (WUIPM) scheme for mining interesting patterns from a sequence database. The WUIPM model had improved speed and efficiency and also eliminated the number of false positive patterns in the database. Tung Kieu et al. [19] have proposed various mining schemes, namely Naive Approach Mining (NAM) algorithm, Vertical Approach Mining (VAM), and Vertical with Index Approach Mining (VIAM) algorithm for pattern mining. It had improved execution time and scalability while processing larger database. Besides its advantages, it failed to handle sequential processing during real-time analysis. Hwa Kyung Lim et al. [20] proposed an approach, BaggedLeast Absolute Shrinkage and Selection Operator method (B-LASSO) for pattern mining. This scheme used logistic regression scheme for pattern mining and hence, it had improved prediction accuracy. The scheme partitions the database before processing, and this requires large computation time.

2.2 Challenges

Challenges prevailing in data mining while using pattern mining techniques are discussed as follows:

- During sequential pattern mining, a large number of candidates is generated during the early stage of mining. Also, mining the sequential patterns from a huge database with a small value of minimum support increases memory requirement [1].
- Complexity issues prevail in the pattern mining environments while using distributed and multi-core processors. Also, pDBV-FCSP approach proposed by Bao Huynh et al. [16] fails to mine the sequences from complex databases by using hybrid environment.
- In several studies, the pattern mining concentrates on a particular application. For example, WUIPM suggested by Akiz Uddin Ahmed et al. [2] cannot be used to mine the database containing sequential itemsets, closed frequent itemsets, etc.
- The optimization-based schemes, such as NAM, and VAM discussed in [19], do not adapt to structures, like Tree and Lattices.
- The main drawback of the SPM is that the task of finding all frequent sequences in a huge database is more challenging as the search space is large. The information of a pattern is not just related to a single item, but the itemsets and the number of itemsets in a pattern and the number of items in the itemset are unknown prior to mining.

3 Proposed RadixTreeMiner algorithm for data mining

This section presents the description of the proposed RadixTreeMiner algorithm developed for data mining. As depicted in Fig. 1, sequences present in the database are subjected to the mining process using the proposed RadixTreeMiner algorithm. The proposed algorithm evaluates each sequence in the database and identifies the sequential patterns by fixing a threshold value for the minimum support computed. Sequential patterns identified by the proposed RadixTreeMiner algorithm are used for the identification of maximal sequential patterns. By matching the maximal sequential patterns with the training sample, the dimension of the database is reduced to make it suitable for classification. Finally, NN is employed for the data classification using the reduced database.

3.1 Constructing sequential patterns: RadixTreeMiner algorithm

This work introduces RadixTreeMiner algorithm for sequential pattern mining. The proposed RadixTreeMiner algorithm is inspired by PrefixSpan algorithm. In the existing PrefixSpan mining algorithm, interesting sequences are identified from sequence database by fixing user-defined minimum support. Here, sequences-providing values less than the minimum support is neglected for further mining process. The process continues until the maximal sequence is obtained for the sequence database. The complexity of the existing PrefixSpan mining algorithm is higher, as it constructs the transaction database for every change in the sequence. Also, while constructing the database, the PrefixSpan algorithm needs to check the minimal support for all the sequences within the database, which again increases the complexity. Thus, it is suitable only for small-sequence database.

Hence, for overcoming the challenges prevailing in the PrefixSpan algorithm, RadixTreeMiner algorithm is introduced in this work. In the proposed RadixTreeMiner algorithm, small-sized radix trees are constructed from the sequences in the sequence database. The minimum support is defined for constructing the sequential patterns from the radix tree, and this tree is formed as a base for pattern mining. Less complexity is one of the major advantages of the proposed RadixTreeMiner algorithm. The RadixTreeMiner algorithm reduces the time complexity while mining, even though the length of the sequences increases. Hence, the proposed RadixTreeMiner algorithm is suitable even for the database with large length sequences.

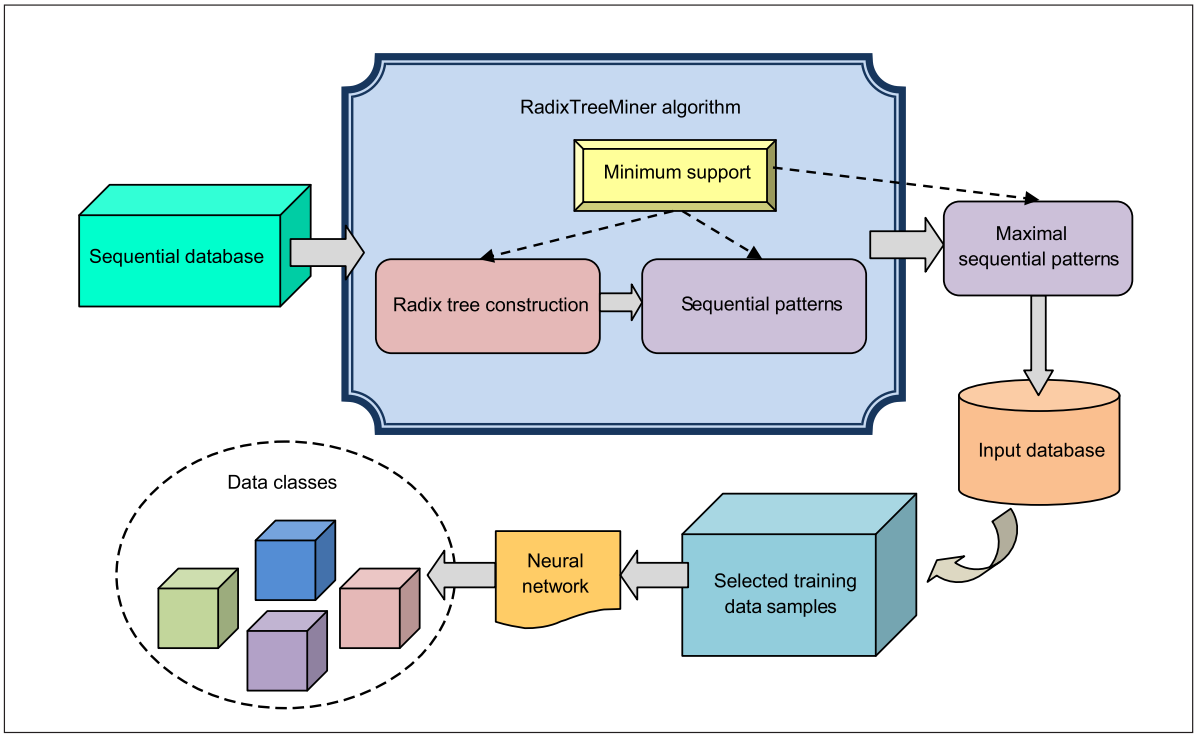


Fig. 1: Block diagram of the proposed RadixTreeMiner algorithm

3.2 Mining of 1-length sequence

The initial step in the proposed RadixTreeMiner algorithm is mining 1-length sequences from the sequence database. Consider a database D with N sequences, and the proposed RadixTreeMiner algorithm identifies 1-length sequences from the database. The 1-length sequence identified from the database forms the node for the radix tree.

The proposed RadixTreeMiner algorithm is implemented in gene database, which has large length sequences. The term 'length' of the sequence refers to the number of items within the sequence. Table 1 presents a representation of the sequence database used for constructing the radix tree. The sequences in the database have two parts and they are referred to as $<$ sequence_id, sequence $>$. Also, the sequences in the database have many subsequences each having length one.

Table 1: Sample sequence database

Sequence_id	Sequence
10	$< atcag >$
20	$< atgat >$
30	$< tagca >$
40	$< taat >$

In table 1, the database used for example has four sequences each of different length. Here, support z_w is calculated for identifying 1-length sequence from each sequence. The 1-length sequence is defined as unique items in the database, and it is identified by declaring minimum support Z set by the user. In table 1, there are four unique items in the database, namely $< a >$, $< t >$, $< c >$, and $< g >$. The proposed RadixTreeMiner algorithm calculates the support of unique itemsets, and the itemset having support z_w greater than minimum support Z is declared to be a 1-length sequence.

3.3 Projected radix tree formation

The sequences present in the database are provided to RadixTreeMiner algorithm. The proposed RadixTreeMiner algorithm identifies the maximal sequence from a sequence database. Similar to the PrefixSpan algorithm, the proposed algorithm utilizes a minimum support value for identifying the maximal sequences. Besides, the proposed RadixTreeMiner algorithm constructs small-sized radix tree for each itemset to identify the maximal sequence. Identifying the maximal sequence from the sequence database through the proposed algorithm is explained as follows:

i) Initially, the sample sequence database A is provided as input to the proposed RadixTreeMiner algorithm. The proposed algorithm mines the sequential

patterns from the database based on the frequency of occurrence of itemset. Thus, the term support, referred to as z_w is defined in the proposed RadixTreeMiner algorithm and is defined as the frequency of occurrence of the itemset in each sequence. Thus, when the database arrives, RadixTreeMiner algorithm calculates the support of each item available in the sequence. Then, based on the requirements of the user, minimum support Z is defined and itemset obtaining support less than the minimum support is removed.

ii) Then, the support z_w of each itemset is found in the database. For sequential pattern mining, the sequences having support z_w higher than the minimum support Z are considered to be a 1-length sequence, and hence, they are mined. Thus, itemset satisfying the condition $z_w \geq Z$ is considered to be unique items/nodes and is applicable for tree construction.

iii) In the next step, the radix tree is constructed by using the nodes obtained from the previous step. Consider the itemset $H\{s_1, s_2, \ldots, s_w, \ldots, s_H\}$ in the sequence that has support greater than minimum support, which is appended to radix tree.

iv) After the initial-stage construction of radix tree, itemsets in sequence A_1 to A_N are appended to each node $\{s_1, s_2, \ldots, s_w, \ldots, s_H\}$ of radix tree. Here, the first sequence acts as the first branch/subtree of the node $\{s_1, s_2, \ldots, s_w, \ldots, s_H\}$. For radix tree construction of node s_1, itemsets in sequences A_1 to A_N are compared with the node s_1, and the itemset following next to node s_1 in sequence A_1 to A_N are appended to the node s_1 as child nodes. Each sequence attached to node s_1 acts as branch or subtree. To construct the radix tree for the node s_1, the node s_1 is searched from the sequences A_1 to A_N in the database. After identifying the sequences with the node s_1, the items following node s_1 are appended as child node to s_1.

v) Step (iv) is repeated until all the unique items identified in step (iii) get its child nodes from the database. In each stage of construction of radix tree, another subtree is formed for child node. If same child nodes arrive in the sequence, the sequences are appended as next child nodes. Thus, if the child node is appended with further two child nodes, then it is given a weight 2.

vi) In the next step, the support z_w of itemset appended as a child node in radix tree corresponding to the node s_w is calculated. Now, the child nodes satisfying the condition $z_w \geq Z$ are appended with the node s_w to form a sequential pattern and they are subjected to further sub radix tree construction.

vii) The iteration continues until all the nodes in radix tree cannot be further subdivided. Finally, the sequential patterns are identified from each node of radix tree structure, and it is represented as $\{K_1, K_2, \ldots, K_r, \ldots, K_Q\}$.

The following steps illustrate the running example to signify the proposed RadixTreeMiner algorithm.

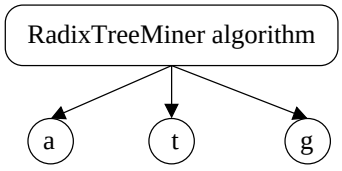
Step 1: Consider the sequence database of Table 1 with four 1-length sequences for the experimentation. As the subsequence present in the database has one itemset, it is referred to as a 1-length sequence. Initially, the frequency of itemset in each sequence is identified, and the support of each itemset is identified. For the considered database, there are four itemsets $< a >$, $< t >$, $< c >$, and $< g >$ in all sequences. Then, the support of each sequence is calculated based on the frequency of occurrence of each itemset in the database. The frequency of occurrence of various itemsets is referred to as z_w. The itemset $< a >$ occurs in all four sequences and hence, the support for $< a >$ is declared to be $z_w = 4$. Similarly, other itemsets $< t >$, $< c >$, and $< g >$ have the support z_w value of 4, 2, and 3 respectively.

a	t	c	g
4	4	2	3

Step 2: In the next step, minimum support is fixed based on the user requirement, and the items achieving support less than the minimum support are neglected from further steps. The minimum support value Z is chosen as 3 for this database, i.e. $Z = 3$. To identify the node for tree construction, itemset having support z_w greater than Z is chosen. As shown in the table below, itemsets $< a >$, $< t >$, and $< g >$ have support z_w greater than minimum support $Z = 3$ and hence, it is taken as unique items. As the item $< c >$ has $z_w = 2$, it is neglected from forming the radix tree.

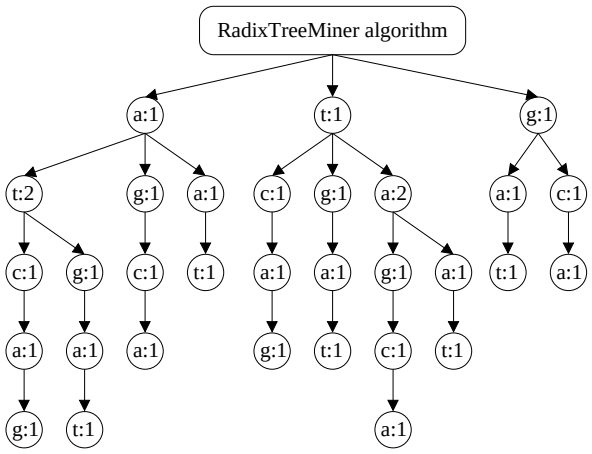
a	t	~~c~~	g
4	4	~~2~~	3

Thus, the initially-constructed radix tree with the unique items $< a >$, $< t >$, and $< g >$ is depicted below:



Step 3: In the next step, the sequences present in the database are appended to the nodes of radix tree. For constructing the radix tree for node $< a >$, the sequences $< atcag >$, $< atgat >$, $< tagca >$, and $< taat >$ are compared with the node $< a >$. The itemset arriving after the node $< a >$ in each sequence is appended to the node as child node. Thus, the first subtree of $< a >$ has the child nodes $< t >$, $< c >$, $< a >$, and $< g >$. The next sequence $< atgat >$ is appended to $< a >$ and since both the sequences $< atcag >$ and $< atgat >$ have their first child node as $< t >$, the sequence $< atgat >$ is appended

to the child node of $< t >$. Also, the weight of child node $< t >$ present in the first subtree of $< a >$ is assigned to be 2, as it has two subtrees. Similar to node $< a >$, sequences in the database are appended as the child node to other nodes $< t >$ and $< g >$. The radix tree constructed for each node $< a >$, $< t >$, and $< g >$ is depicted below.

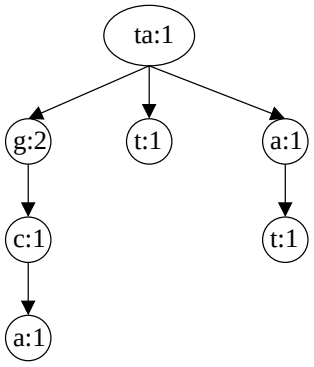


Step 4: In the next level, support of each itemset in individual tree is calculated. For simplification, the tree constructed under itemset $< t >$ is considered. Here, support z_w of child nodes attached to node $< t >$ is calculated. The child node occurs in all four branches and hence the support of child node $< a >$ in radix tree corresponding to $< t >$ is calculated as 4. The frequency count estimation is simplified by declaring weights. As shown in step 3, child node $< a >$ in radix tree constructed through $< t >$ has weight of 2 in one of its subtree. Hence, the algorithm doesn't need to check the fourth branch to search for availability of child node $< a >$. Thus, the RadixTreeMiner algorithm has less complexity compared to PrefixSpan algorithm. Similarly, support z_w of other child nodes in radix tree based on $< t >$ node is identified, and the frequency of child nodes appended under node $< t >$ is given below.

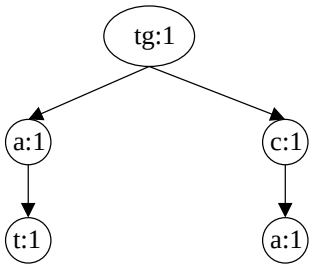
a	t	c	g
4	2	2	3

As the minimum support Z is chosen to be 3, itemsets $< a >$ and $< g >$ will be retained considering subtrees 't' and 'g'. Thus, the child nodes $< a >$ and $< g >$ are appended with the node $< t >$ for further processing.

Step 5: The sequential patterns $< ta >$ and $< tg >$ identified from radix tree $< t >$ is subjected for further processing. Again, the sequences in the database sequences $< atcag >$, $< atgat >$, $< tagca >$, and $< taat >$ are compared with the sequential pattern $< ta >$, and appended to node $< ta >$ forming a sub radix tree. As shown in the tree below, the first branch of subtree has only one child node $< g >$. Here, the child nodes other than the sequential pattern $< ta >$ are appended as the sub radix tree of $< ta >$. Similarly, the other itemsets in sequences of the database are appended as child node of $< ta >$.



The above tree indicates the sub radix tree formed from itemset $< ta >$. Similarly, the sequences from the database sequences $< atcag >$, $< atgat >$, $< tagca >$ and $< taat >$ are compared with the sequential pattern $< tg >$, and appended to node $< tg >$ forming a sub radix tree as shown below.



Step 6: In the next step, the support of child nodes in subtree $< ta >$ is identified, and it is expressed in the following table.

a	t	c	g
2	2	1	1

As specified in the above table, no itemset in the sequence $< ta >$ provides necessary support value for further processing. Hence, the sequence $< ta >$ cannot be further simplified, and it is taken as one of the sequential patterns of the radix tree. Similarly, the support of child nodes in subtree $< tg >$ is identified and it is represented below.

a	t	c	g
2	1	1	0

As shown in the table above, no itemset in the sequence

$< tg >$ provides necessary support value for further processing. Thus the sequence $< tg >$ cannot be further simplified and it is considered as another sequential pattern of the radix tree. Hence $< ta >$ and $< tg >$ are the sequential patterns.

Step 7: Similarly, sequential patterns are identified for nodes $< a >$ and $< g >$ present in the radix tree. The iteration is continued until the radix tree cannot be further subdivided into subtree for all nodes. Finally, the sequential patterns for the database are mined through RadixTreeMiner algorithm.

3.4 Termination

Radix tree formed through RadixTreeMiner algorithm terminates when the subtree cannot be further divided. When the algorithm is terminated, sequences that cannot be further formed as a tree are declared to be sequential patterns. At the end of the iteration, the maximal sequences are obtained for the sequential database. Finally, the procedure of RadixTreeMiner is followed to automatically identify the sequential patterns from the given sequence database.

Algorithm 1: RadixTreeMiner algorithm

```
1)  Input: Sequence database A
2)  Output: Sequential patterns K
3)  Parameters: Minimum support Z, support of item z_w
4)  Begin
5)      For each item 'w' in the sequence database
6)          Find the support z_w
7)          Declare the number of nodes 'H' for
                radix tree construction
8)          For ∀ H
9)              If radix tree has itemset 'w'
10)                 Compute support z_w of itemset
11)                 If (z_w ≥ Z)
12)                     Append the itemset 'w' with K
13)                 End if
14)             Else
15)                 Remove the itemset 'w' from
                        tree formation
16)             End if
17)             Construct radix tree for all 'H' nodes
18)             For each itemset 'w' in subtree
19)                 Compute support z_w of itemset
20)                 If (z_w ≥ Z)
21)                     Append the item 'w' with
                            sequential pattern K
22)                 Else
23)                     Remove the item 'w' from
                            tree formation
24)                 End if
25)                 Construct subtree of radix tree
                        for each node
26)             End For
27)         End For
28)     End For
29)     Terminate
30)     Return Sequential pattern K
31) End
```

3.5 Maximal sequential patterns

Maximal sequential patterns are identified from the sequential patterns obtained through the RadixTreeMiner algorithm. Using the proposed RadixTreeMiner algorithm, Q sequential patterns are identified, and they are represented as $\{K_1, K_2, \ldots, K_r, \ldots, K_Q\}$. From Q sequential patterns obtained, the maximal sequential patterns are selected. A sequential pattern K_r can be called as the maximal sequence if there is no other sequential pattern, such that K_k is considered to be a super pattern of the sequential pattern K_r [21]. The itemset present in maximal sequential patterns varies for the database, as it is constructed based on minimum support defined by the user.

3.6 Classifying the database by employing Neural Network

Maximal sequential patterns mined from the database are used for selecting the training samples for classification. The primary aim of this work is to select the training samples from the input database based on the identified maximal sequential patterns. This work employs NN for the classification task and backpropagation algorithm for the training process. Before classifying the database, the classification task is simplified by selecting few sequences related to the maximal sequence designated from the proposed RadixTreeMiner algorithm. Then, selected training sample is given as input to NN for training, and finally, the database is classified into different classes.

The training data sample subjected to classification is classified into a number of classes. NN is trained with the help of backpropagation algorithm and is comprised of three layers, namely input layer, hidden layer, and output layer. Each layer in NN constitutes neurons for performing the classification task, and also, several weight elements are included for refining the output.

Consider the NN with input layer having X number of neurons, and it is specified below as

$$R = \{R_1, R_2, \ldots, R_i, \ldots, R_X\} \quad (1)$$

where R_i indicates the i^{th} neuron in the input layer, and i ranges between 1 and X. The hidden layer in NN has Y

neurons and it is expressed as

$$S = \{S_1, S_2, \ldots, S_j, \ldots, S_Y\} \tag{2}$$

where S_j indicates the j^{th} neuron in hidden layer with the limit $1 \leq j \leq Y$. The sequences in the training sample are provided as input to the input layer, and it is multiplied by the weights, $w = \{w_1, w_2, \ldots, w_k, \ldots, w_J\}$. The weight vector w has a number of weights, which is equivalent to the total neurons in input and hidden layer, i.e. $J = X + Y$. The expression for the output of the hidden layer is specified as follows:

$$S_j = \frac{1}{X} \sum_{i=1}^{X} w_i * R_i \tag{3}$$

where w_i refers to the weight between the input and the hidden layers of NN. Finally, the output layer neurons are computed by multiplying the weights with the hidden neurons and it is expressed as

$$O_o = \sum_{j=1}^{Y} w_j * S_j \tag{4}$$

where O_o indicates the o^{th} neuron in the output layer and w_j indicates the weight between the hidden and the output layers. The output layer is comprised of G neurons as expressed below:

$$O = \{O_1, O_2, \ldots, O_o, \ldots, O_G\} \tag{5}$$

The steps for training the NN with backpropagation algorithm are given as follows:

(1) Random initialization of weights: To classify the sequences in the database, selection of weights is a necessity. In the initialization step, the weights are randomly assigned.

(2) Calculation of output and error value: In the next step, based on the randomly-initialized weights, the output values are computed as given in the following function:

$$O_o(t) = NN(w^T, R) \tag{6}$$

where w^T indicates the transpose of weight vector and R specifies the elements in the input layer. The error is then estimated for computed output and it is expressed as

$$E(t) = \sum_{o=1}^{G} (O_o(t) - \eta_o) \tag{7}$$

where η_o indicates the ground truth value for the o^{th} output neuron.

(3) Weight update based on backpropagation algorithm: In this step, the randomly-initialized weights are updated based on the backpropagation algorithm. The expression for the weight update is specified below:

$$w(t+1) = w(t) + \Delta w \tag{8}$$

where Δw indicates the adaptive weight vector and it is expressed as

$$\Delta w = -\rho \frac{\partial E}{\partial w} \tag{9}$$

where ρ indicates the learning rate of NN and it has a value of 0.1.

(4) Recalculation of output value with updated weight: In this step, based on the updated weight value, the output of NN is calculated and the error value is updated.

(5) Updating learning rate: The learning rate of NN gets updated based on the chosen delay value. Expression for updating the learning rate is specified as below:

$$\rho(t+1) = \rho(t) * d \tag{10}$$

where d indicates the delay value and it is chosen to be 0.1.

(6) Termination: The algorithm refines the output value until the end of the iteration. At the end of iteration, denoted as T_{max}, the output with minimum error value is taken as the classified output.

3.7 Sample selection using maximal sequential patterns

Maximal sequential patterns constructed from the previous section pave the way for selecting the training database. Consider the input database I having a number of sequences of size $a \times b$. The size of the database depends on the number of sequences in the database. For reducing the complexity of training process, it is more compact to reduce the number of sequences for training. Hence the maximal sequences are compared with each sequence in the input database I. Finally, the sequences related to maximal sequences are selected as the training sample database and thus the size of the data is reduced. After sample selection, the training sample of size $U \times V$ is provided for training the NN. Thus, for the training purpose, the database I of size $a \times b$ is reduced to $U \times V$. Here, the values of U and V is less than a and b. Fig. 2 presents the NN model for classifying the database I.

4 Results and discussion

Simulation results of the proposed RadixTreeMiner algorithm are evaluated in this section. Here, experimentation is done by evaluating the gene sequences available in two standard databases and evaluated with the metric classification accuracy.

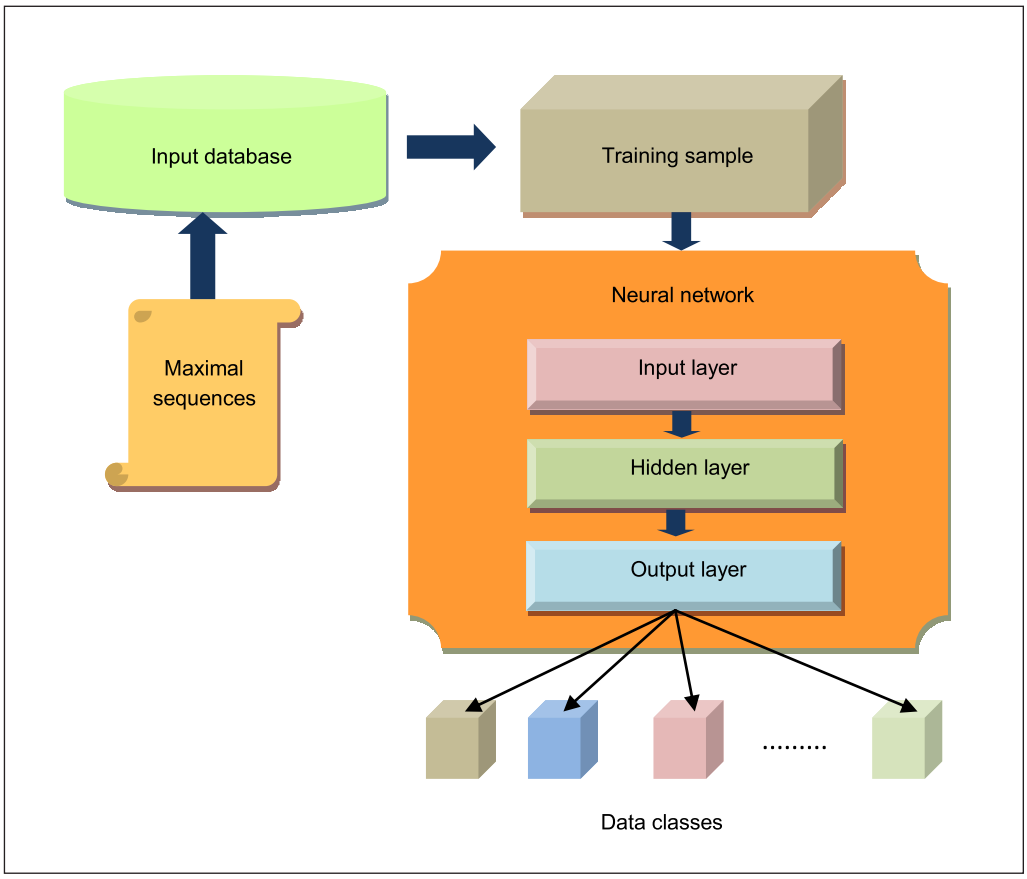


Fig. 2: Classifying the sequential database with NN model

4.1 Experimental setup

Experimentation of the proposed RadixTreeMiner algorithm is implemented in JAVA tool. The personal computer used for the experimentation purpose requires configurations such as Windows 10 OS, 4 GB RAM, and Intel I3 processor.

4.2 Database description

Experimentation of the proposed RadixTreeMiner algorithm uses two standard datasets, namely Molecular Biology (splice-junction gene sequences) [22] and Molecular Biology (promoter gene sequences) [23]. These datasets provide gene sequences for data mining. The splice-junction gene sequences have a total of 3190 instances with 61 attributes, whereas the promoter gene sequences has a total of 106 instances under 58 attributes.

4.3 Performance metrics

For the evaluation, this work uses the metric Classification Accuracy (CA), which is defined as given below:

Classification Accuracy: It defines the number of patterns correctly classified, and hence, it measures the closeness to the actual response. Classification accuracy can be expressed as follows:

$$CA = \frac{TP+TN}{TP+TN+FP+FN} \tag{11}$$

where TP, TN, FP, and FN indicate the true positive, true negative, false positive, and false negative respectively.

4.4 Comparative models

The simulation results achieved by the proposed RadixTreeMiner algorithm are compared with various existing models, such as PrefixSpan [24], CloSpan [25], and Top-k mining [19]. Description of these comparative techniques is given below:

PrefixSpan: The PrefixSpan algorithm mines the sequential patterns from the sequence database by

constructing the projected database. For each change in sequence, it constructs the projected database and hence, requires more memory.

CloSpan: In the existing CloSpan technique, only a closed group of sequences are analyzed for pattern mining.

Top-k mining: The Top-k mining process uses the Naive Approach Mining (NAV) to identify the top-ranked patterns, and to mine the sequential patterns accordingly.

4.5 Comparative analysis

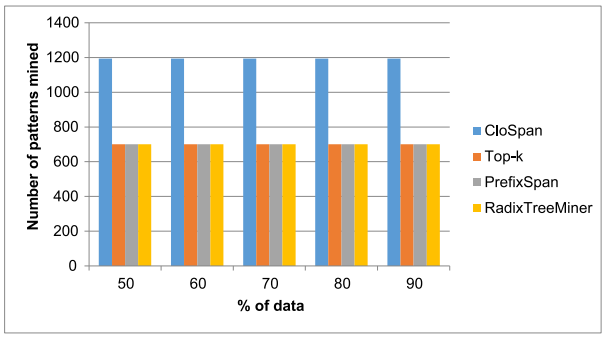
Here, the performance of the proposed RadixTreeMiner and the existing algorithms is evaluated by fixing various values for minimum support Z. The performance of the proposed RadixTreeMiner algorithm against various existing techniques is presented below:

4.6 Comparative analysis for splice-junction gene sequences for minimum support $Z = 50$

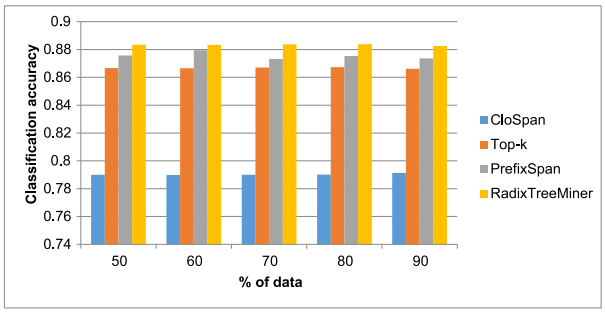
Fig. 3 presents the analysis of comparative techniques for splice-junction database with minimum support $Z = 50$. Fig. 3(a) shows the analysis based on the number of patterns mined for splice-junction database by varying the training percentages. While using 90% of the database, the existing CloSpan algorithm mines 1193 patterns while Top-k mining and PrefixSpan techniques achieve a high value of 701. Similar to Top-k mining and PrefixSpan, the proposed RadixTreeMiner algorithm also has the number of patterns mined as 701. The analysis is done based on CA using splice-junction database with minimum support $Z = 50$, as shown in Fig. 3(b), which depicts that the existing CloSpan, Top-k mining and PrefixSpan algorithms achieve CA values of 0.7899, 0.8666 and 0.8756 respectively, for 50% of data. The proposed RadixTreeMiner algorithm achieves a high value of 0.8834 for 50% of data. Fig. 3(c) shows the analysis based on the computational time for splice-junction database with the training percentages varying from 50% to 90%. While using 90% of the database, the existing CloSpan, Top-k mining and PrefixSpan algorithms have the computational time of 2207475 milliseconds (ms), 1483670 ms and 709170 ms respectively. The proposed method has the computational time of 579548 ms for 90% of data.

4.7 Comparative analysis for splice-junction gene sequences for minimum support $Z = 60$

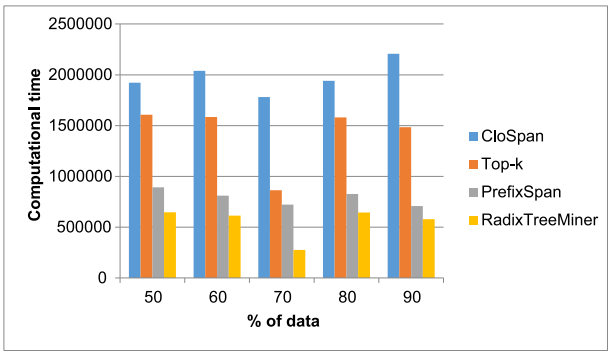
Fig. 4 presents the analysis of comparative techniques for splice-junction database with minimum support $Z = 60$.



(a)



(b)

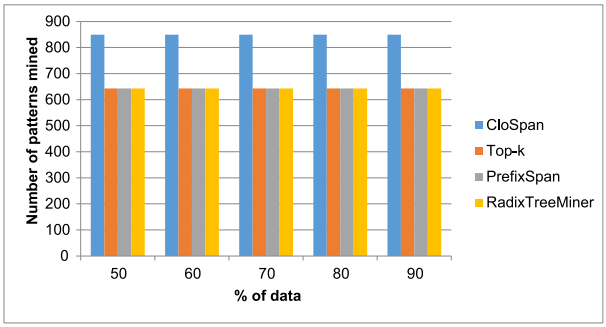


(c)

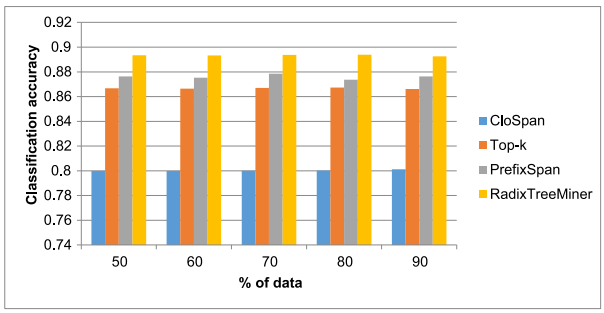
Fig. 3: Comparative analysis using splice-junction gene sequences with $Z = 50$ based on (a) Number of patterns mined, (b) Classification accuracy and (c) Computational time

Fig. 4(a) shows the analysis based on the number of patterns mined for splice-junction database by varying the training percentages from 50% to 90%. When using 90% of the database, the existing CloSpan algorithm has the number of patterns mined as 849, while Top-k mining, PrefixSpan and the proposed algorithm have achieved a high value of 643. Analysis is done based on CA using splice-junction database with minimum support $Z = 60$, as shown in Fig. 4(b), which depicts that the existing CloSpan, Top-k mining and PrefixSpan algorithms achieve classification accuracy values of 0.7999, 0.8666 and 0.8762 respectively, for 50% of data. The proposed RadixTreeMiner algorithm achieves better performance

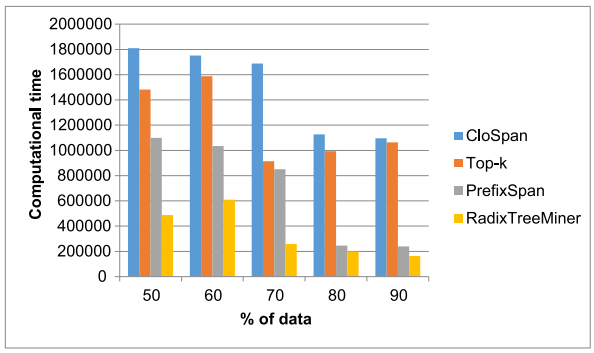
than the existing techniques with the value of 0.8934 for 50% of data. Fig. 4(c) shows the analysis based on the computational time for splice-junction database with the training percentages varying from 50% to 90%. The existing CloSpan, Top-k mining and PrefixSpan algorithms have the computational time of 1096151 ms, 1062805 ms and 239522 ms respectively, for 90% of data. The proposed method has the computational time of 162965 ms for 90% of data.



(a)



(b)



(c)

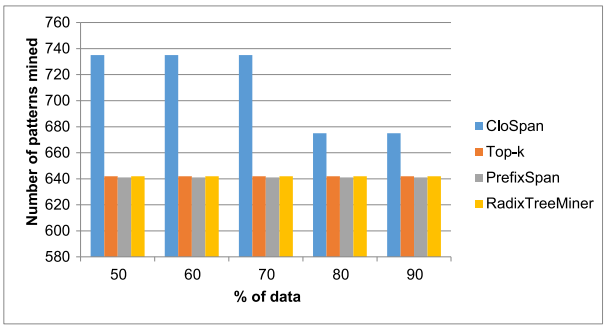
Fig. 4: Comparative analysis using splice-junction gene sequences with $Z = 60$ based on (a) Number of patterns mined, (b) Classification accuracy and (c) Computational time

4.8 Comparative analysis for splice-junction gene sequences for minimum support $Z = 70$

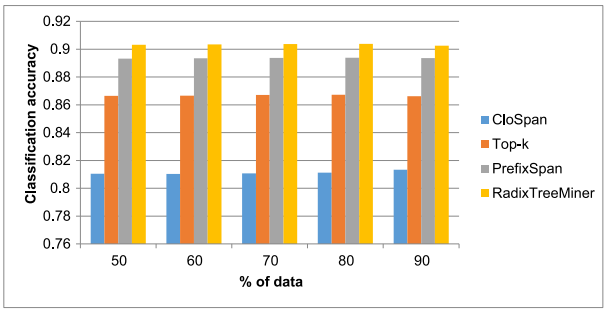
Fig. 5 presents the analysis of comparative techniques for splice-junction database with minimum support $Z = 70$. Fig. 5(a) shows the analysis based on the number of patterns mined for splice-junction database with the training percentages varying from 50% to 90%. While using 90% of the database, the existing CloSpan algorithm achieve the number of patterns mined as 675, while the proposed algorithm has 642 patterns mined. Analysis is done based on CA using splice-junction database with minimum support $Z = 70$ is shown in Fig. 5(b), which depicts that the existing CloSpan, Top-k mining and PrefixSpan algorithms have achieved CA values of 0.8104, 0.8664 and 0.8932 respectively, for 50% of data. The proposed RadixTreeMiner algorithm achieves a high value of 0.9032 for 50% of data. Fig. 5(c) shows the analysis based on the computational time for splice-junction database with the training percentages varying from 50% to 90%. While using 90% of the database, the existing CloSpan, Top-k mining and PrefixSpan algorithms have the computational time of 415557 ms, 121379 ms and 122095 ms respectively. The proposed method has the computational time of 79562 ms for 90% of data.

4.9 Comparative analysis for promoter gene sequences for minimum support $Z = 50$

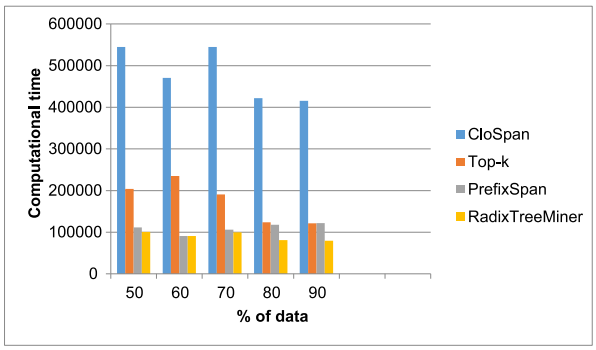
Fig. 6 presents the analysis of comparative techniques for promoter gene sequence database with minimum support $Z = 50$. Fig. 6(a) shows the analysis based on the number of patterns mined for promoter gene sequence database. While 90% of the database is used, the existing CloSpan algorithm mined 36658 patterns, while Top-k mining, PrefixSpan and the proposed RadixTreeMiner algorithm have a value of 35628. Analysis is done based on CA using promoter gene sequence database with minimum support $z = 50$, as shown in Fig. 6(b), which demonstrates that the existing CloSpan, Top-k mining and PrefixSpan algorithms achieve CA values of 0.6358, 0.6642 and 0.7971 respectively, for 90% of data. Meanwhile, the proposed RadixTreeMiner algorithm achieves a high value of 0.8071 for 90% of data. Fig. 6(c) shows the analysis based on the computational time for promoter gene sequence database with the training percentages varying from 50% to 90%. While using 90% of the database, the existing CloSpan, Top-k mining and PrefixSpan algorithms have the computational time of 2036541 ms, 2391846 ms and 1415537 ms respectively. The proposed method has the computational time of 1025695 ms for 90% of data.



(a)

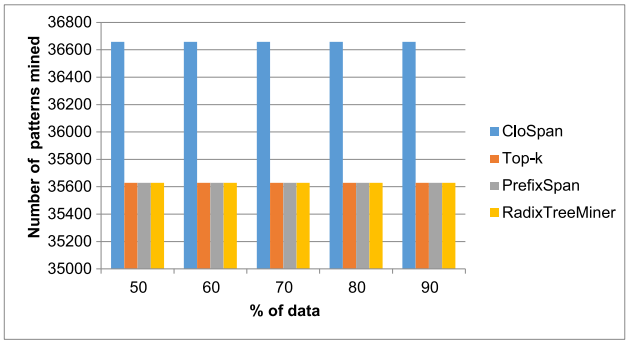


(b)

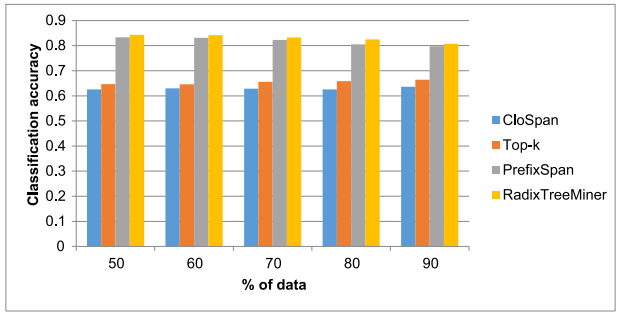


(c)

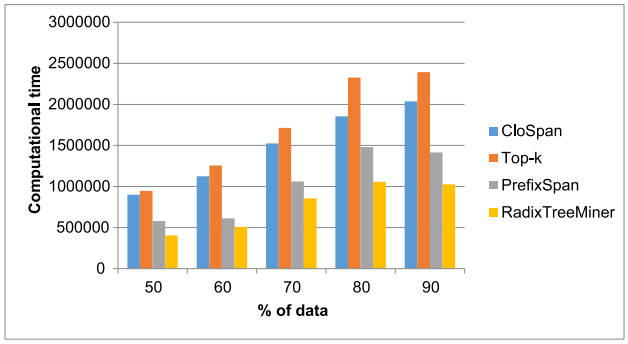
Fig. 5: Comparative analysis using splice-junction gene sequences with $Z = 70$ based on (a) Number of patterns mined, (b) Classification accuracy and (c) Computational time



(a)



(b)

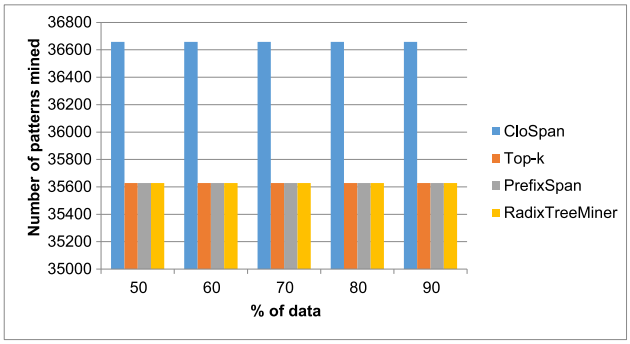


(c)

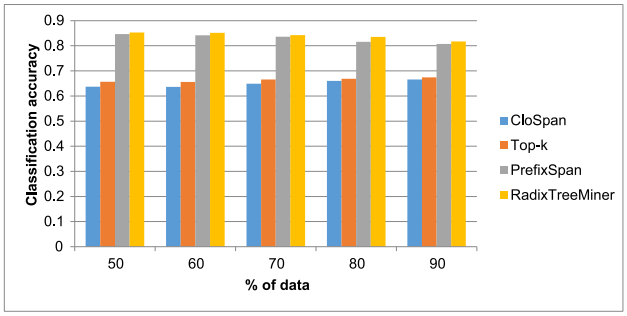
Fig. 6: Comparative analysis using promoter gene sequences with $Z = 50$ based on (a) Number of Patterns mined, (b) Classification accuracy and (c) Computational time

4.10 Comparative analysis for promoter gene sequences for minimum support $Z = 60$

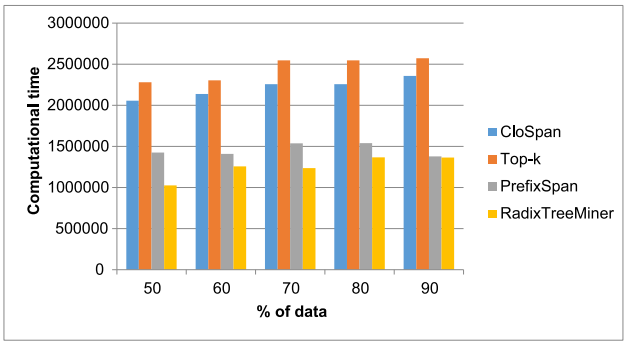
Fig. 7 presents the analysis of comparative techniques for promoter gene sequence database with minimum support $Z = 60$. Fig. 7(a) shows the analysis based on the number of patterns mined for promoter database. While using 90% of the database, the existing CloSpan algorithm achieve the number of patterns mined as 36658, while the number of patterns mined by the proposed RadixTreeMiner algorithm is 35628. Analysis is done based on CA for promoter gene sequence database with minimum support $Z = 60$, as pictured out in Fig. 7(b), which shows that the existing CloSpan, Top-k mining and PrefixSpan algorithms achieve CA values of 0.6655, 0.6742 and 0.8076 respectively, for 90% of data. However, the proposed RadixTreeMiner algorithm achieves a CA of 0.8171 for 90% of data. Fig. 7(c) shows the analysis based on the computational time for promoter gene sequence database with the training percentages varying from 50% to 90%. The existing CloSpan, Top-k mining and PrefixSpan algorithms have the computational time of 2356985 ms, 2572112 ms and 1378238 ms respectively, for 90% of data. The proposed method has the computational time of 1365222 ms for 90% of data.



(a)



(b)

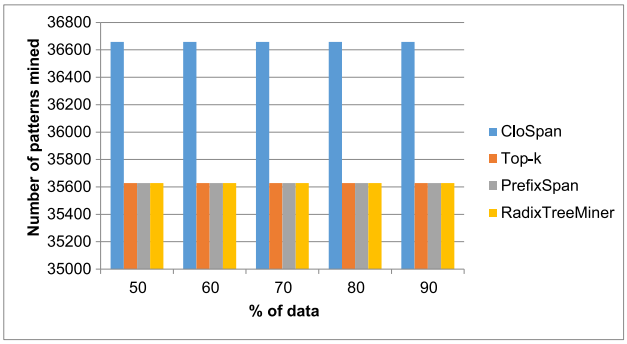


(c)

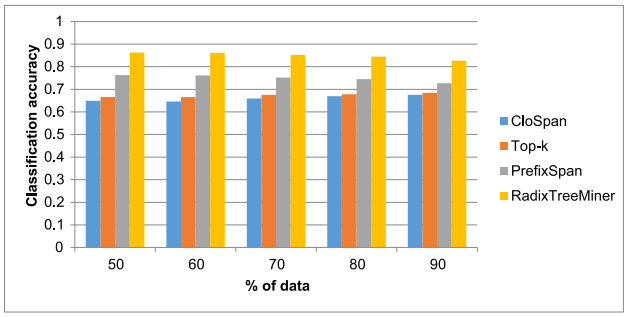
Fig. 7: Comparative analysis using promoter gene sequences with $Z = 60$ based on (a) Number of Patterns mined, (b) Classification accuracy and (c) Computational time

4.11 Comparative analysis for promoter gene sequences for minimum support $Z = 70$

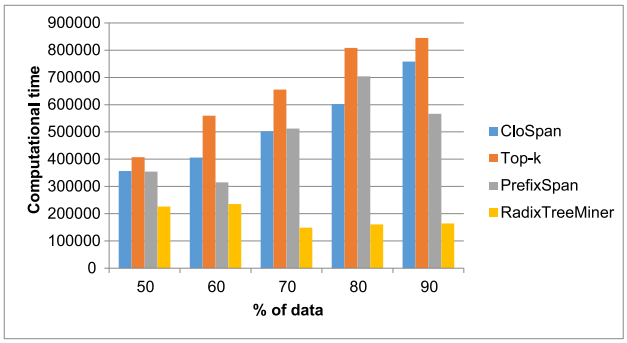
Fig. 8 presents the analysis of comparative techniques for promoter gene sequence database with minimum support $Z = 70$. Fig. 8(a) shows the analysis based on the number of patterns mined for promoter gene sequence database. While using 90% of the database, the number of patterns mined using existing CloSpan algorithm is 36658, while that by the proposed RadixTreeMiner, Top-k mining and PrefixSpan techniques are 35628. Analysis is done based on CA using promoter gene sequence database with minimum support $Z = 70$, as given in Fig. 8(b), which depicts that the existing CloSpan, Top-k mining and PrefixSpan algorithms achieve CA values of 0.6758, 0.6842 and 0.7271 respectively, for 90% of data. The proposed RadixTreeMiner algorithm achieves a high value of 0.8271 for 90% of data. Fig. 8(c) shows the analysis based on the computational time for promoter gene sequence database with the training percentages varying from 50% to 90%. While using 90% of the database, the existing CloSpan, Top-k mining and PrefixSpan algorithms have the computational time of 758156 ms, 844852 ms and 566703 ms respectively. The proposed method has the computational time of 164406 ms for 90% of data.



(a)



(b)



(c)

Fig. 8: Comparative analysis using promoter gene sequences with $z = 70$ based on (a) Number of Patterns mined, (b) Classification accuracy and (c) Computational time

4.12 Comparative discussion

Table 2 presents the comparative discussion of the proposed RadixTreeMiner algorithm against other existing techniques. From the table 2, it is evident that the proposed RadixTreeMiner algorithm has achieved improved performance over other models for both the databases. For the splice-junction database, the number of patterns mined by the proposed RadixTreeMiner algorithm is 642 and its CA is 0.9038. Similarly, for the promoter gene sequence database, the number of patterns mined by the proposed RadixTreeMiner algorithm is 35628, with the CA of 0.8628. The performance of databases based on computational time suggests that the proposed RadixTreeMiner algorithm achieves less computational time than other existing algorithms. For the splice-junction database, the RadixTreeMiner algorithm achieves low computational time of 79562 ms and for the promoter gene sequence database, it achieves computational time of 148916 ms.

5 Conclusion

This paper presents a novel approach, namely RadixTreeMiner, for pattern mining. The proposed RadixTreeMiner algorithm is constructed for sequential pattern mining and mines the sequential patterns by constructing the radix trees. The proposed RadixTreeMiner algorithm constructs small-sized radix tree for sequences in the database, and hence, mining becomes easier even though the length of the sequence is large. Then, from the sequential patterns, maximal sequential patterns are identified. Finally, the input database with a large number of sequences is compared with the generated maximal sequences, and the possible training samples are selected for the classification. NN is employed for classifying the training sample. The experimentation of the proposed RadixTreeMiner is done using splice-junction database and promoter gene sequence database. For the splice-junction database, the proposed RadixTreeMiner algorithm achieves CA value of 0.9038. Similarly, for the promoter gene sequence database, the proposed algorithm obtains a CA value of 0.8628.

Acknowledgement

K. Poongodi gratefully acknowledges the support provided by Centre for Research, Anna University, Chennai to carry out this research work by granting Anna Centenary Research Fellowship (ACRF 2016 - 2018).

References

[1] Chieh-Yuan Tsai and Chih-Jung Chen, A PSO-AB classifier for solving sequence classification problems, Applied Soft Computing, 27, 11-27 (2015).

[2] Akiz Uddin Ahmed, Chowdhury Farhan Ahmed, MdSamiullah, Nahim Adnan and Carson Kai-Sang Leung, Mining interesting patterns from uncertain databases, Information Sciences, 354, 60-85 (2016).

[3] Jingjun Zhu, Haiyan Wu and GuozhuGao, An efficient method of web sequential pattern mining based on session filter and transaction identification, Journal of Networks, 5, 1017-1024 (2010).

[4] Themis P. Exarchos, Markos G. Tsipouras, Costas Papaloukas and Dimitrios I. Fotiadis, A two-stage methodology for sequence classification based on sequential pattern mining and optimization, Data and Knowledge Engineering, 66(3), 467-487 (2008).

[5] Zhenmin Li, Shan Lu, SuvdaMyagmar and Yuanyuan Zhou, CP-miner: finding copy-paste and related bugs in large-scale software code, IEEE Transactions on Software Engineering, 32(3), 176-192 (2006).

[6] A. Turi, C. Loglisci, E. Salvemini, G. Grillo, D. Malerba and D. D'Elia, Computational annotation of UTR cis-regulatory modules through frequent pattern mining, BMC Bioinformatics, 10(6), (S25) (2009).

[7] Fabio Fumarola, Pasqua Fabiana Lanotte, Michelangelo Ceci and DonatoMalerba, CloFAST: closed sequential pattern mining using sparse and vertical id-lists, Knowledge and Information Systems, 48(2), 429-463 (2016).

[8] I.V. Sergienko, B.A. Biletskyy and A.M. Gupal, Predicting torsion angles in aminoacid protein sequences based on a Bayesian classification procedure on Markov chains, Cybernatics System Analysis, 46(5), 684-690 (2010).

[9] Dou Shen, Jian-Tao Sun, Qiang Yang, Hui Zhao and Zheng Chen, Text classification improved through automatically extracted sequences, in Proc. of 22^{nd} International Conference on Data Engineering, 121-123 (2006).

[10] Poonam Bansal, AmitaDev and ShailBala Jain, Role of different order ranges of autocorrelation sequence on the performance of speech recognition, WSEAS Transaction System, 9(1), 1-9 (2010).

[11] H.Y. Yang, X.Y. Wang and Z.K. Fu, A new image denoising scheme using support vector machine classification in shiftable complex directional pyramid domain, Applied Soft Computing, 12(2), 872-886 (2012).

[12] Bing Liu, Wynne Hsu and Yiming Ma, Integrating classification and association rule mining, in Proc. of the Fourth International Conference on Knowledge Discovery and Data Mining, 80-86 (1998).

[13] Neal Lesh, Mohammed J. Zaki and Mitsunori Ogihara, Scalable feature mining for sequential data, IEEE Intelligent Systems and their Applications, 15(2), 48-56 (2000).

[14] Vincent S. Tseng and Chao-HuiLee, Effective temporal data classification by integrating sequential pattern mining and probabilistic induction, Expert Systems with Applications, 36(5), 9524-9532 (2009).

[15] Cheng, Zhou, Boris Cule and Bart Goethals, Pattern based sequence classification, IEEE Transactions on Knowledge and Data Engineering, 28(5), 1-14 (2015).

[16] Bao Huynh, Bay Vo and Vaclav Snasel, An efficient parallel method for mining frequent closed sequential patterns, IEEE Access, 5, 17392-17402 (2017).

[17] Dmitriy Fradkin and Fabian Morchen, Mining sequential patterns for classification, Knowledge and Information Systems, 45(3), 731-749 (2015).

Table 2: Comparative discussion

Database	Evaluation metric	Comparative techniques			
		CloSpan	Top-k mining	PrefixSpan algorithm	Proposed RadixTreeMiner
Splice-junction Gene Sequences	Number of patterns mined	675	642	641	642
	Classification accuracy	0.8133	0.8672	0.8938	0.9038
	Computational time (ms)	415557	121379	90753	79562
Promoter Gene Sequences	Number of patterns mined	36658	35628	35628	35628
	Classification accuracy	0.6758	0.6842	0.8465	0.8628
	Computational time (ms)	356954	407090	315100	148916

[18] Ana Palacios, Alvaro Martinez, Luciano Sanchez and Ines Couso, Sequential pattern mining applied to aeroengine condition monitoring with uncertain health data, Engineering Applications of Artificial Intelligence, 44, 10-24 (2015).

[19] Tung Kieu, Bay Vo, Tuong Le, Zhi-Hong Deng and Bac Le, Mining top-k co-occurrence items with sequential pattern, Expert Systems with Applications, 85, 123-133 (2017).

[20] Hwa Kyung Lim, Yongdai Kim and Min-Kyoon Kim, Failure prediction using sequential pattern mining in the wire bonding process, IEEE Transactions on Semiconductor Manufacturing, 30(3), 285-292 (2017).

[21] Philippe Fournier-Viger, Cheng-Wei Wu, Antonio Gomariz and Vincent S. Tseng, VMSP: Efficient vertical mining of maximal sequential patterns, in Proc. of Canadian Conference on Artificial Intelligence, Springer, Cham, 83-94 (2014).

[22] Molecular Biology (Splice-junction Gene Sequences) Dataset from, https://archive.ics.uci.edu/ml/datasets/Molecular+Biology +(Splice-junction+Gene+Sequences).

[23] Molecular Biology (Promoter Gene Sequences) Dataset from, https://archive.ics.uci.edu/ml/datasets/Molecular+Biology +(Promoter+Gene+Sequences).

[24] Jian Pei, Jiawei Han, BehzadMortazavi-Asl, Jianyong Wang, Helen Pinto, Qiming Chen, UmeshwarDayal and Mei-Chun Hsu, Mining Sequential Patterns by Pattern-Growth: The PrefixSpan Approach, IEEE Transactions on Knowledge and Data Engineering, 16(11), 1424-1440 (2004).

[25] Xifeng Yan, Jiawei Han and Ramin Afshar, CloSpan: mining closed sequential patterns in large datasets, in Proc. of the 2003 SIAM International Conference on Data Mining, 166-177 (2003).

K. Poongodi received the MCA degree with distinction from Bharathiar University, Coimbatore in 2003. She was awarded M.E degree in Computer Science and Engineering with distinction from Anna University, Chennai in 2014. Currently, she is a Ph.D. scholar in the Faculty of Information and Communication Engineering, Anna University, CEG Campus, Chennai, Tamil Nadu, India. She has nine years of teaching experience and received 39^{th} University Rank in her M.E. degree. She received best teacher award in Coimbatore District in 2012 from Akshaya Institute of Management Studies, Coimbatore. Her research interests include Data Mining, Big Data and Database Management System.

A. K. Sheik Manzoor is currently working as Associate Professor in the Department of Management Studies, Anna University, CEG Campus, Chennai, Tamil Nadu, India. He received his B.E. degree in Computer Science and Engineering from Manonmaniam Sundaranar University in 1999, M.E. in Systems Engineering and Operations Research from Anna University, Chennai in 2016, MBA in Management Studies from Madurai Kamaraj University in 2001 and Ph.D. in the Faculty of Management Sciences from Anna University, Chennai in 2013. He has 16 years of teaching experience and his current research interests include Data Mining, Information Systems and Databases. His other areas of specialization are Medical Tourism and Enterprise Resource Planning. He has published several papers in reputed National and International Journals.