Unified Framework for Data Mining using Frequent Model Tree

Shahid Zaman, Yumnam Jayanta Singh
Department of Computer Science & Engineering and Information Technology
School of Technology, Assam Don Bosco University, Guwahati-781017, INDIA
shahidpt0982219[at]gmail.com, jayanta[at]dbuniversity.ac.in

Abstract: *Data mining is the science of discovering hidden patterns from data. Over the past years, a plethora of data mining algorithms has been developed to carry out various data mining tasks such as classification, clustering, association mining and regression. All the methods are ad-hoc in nature, and there exists no unifying framework which unites all the data mining tasks. This study proposes such a framework which describes a data modelling technique to model data in a manner that can be used to accomplish all kinds of data mining tasks. This study proposed a novel algorithm known as Frequent Model (FM)-Growth, based on Frequent pattern (FP)-Growth algorithm. The algorithm is used to find frequent patterns or models from data. These models will then be used to carry out various data mining tasks such as classification, clustering. The advantage of these frequent models is that they can be used as it is with any data mining task irrespective of the nature of the task. The algorithm is carried out in two stages. In the first stage, we grow the FM-tree from the data and in the second stage, we extract the frequent models from the FM-tree. The accuracy of the proposed algorithm is high. However, the algorithm is computationally expensive when searching for frequent models in high volume and high dimensional data. The reason of expensiveness is that it needs to travel all the nodes of a tree. The study suggests measures to be taken to improve the efficiency of the overall process using dictionary data structure.*

Keywords: *Data Mining, Frequent Pattern Recognition Unified Framework, Classification, Clustering, FP-Growth tree.*

1. Introduction

The earliest digital databases were in the form of flat files where the logic to store and then manipulate data such as searching, updating and deleting were to be defined by the application developer. As a result, a small change in the structure of data led to a change of the application logic altogether. Also, the logic for storing and manipulating data remained inconsistent across different types of applications. This gave rise to the need for a consistent model where the underlying logic for storage and manipulation of data could be abstracted from the application developer so that he could focus more on the application logic rather than concentrating on how to store or manipulate the data needed by the application. The search for such consistent model came to end with Codd's relational model [1]. The relational model defines how the data is to be stored and also manipulated irrespective of the applications using the data. This enabled developers to focus on enhancing the functionality of their applications leaving the data logic to the relational model to handle. The model quickly gained widespread popularity and eventually a whole industry sprung around the relational model. With the introduction of SQL (Structured Query Language), more and more data intensive applications such as ERPs, CRMs, e-commerce came to prominence. The relational model provided a theory to the science of data storage and manipulation and encouraged developers to visualize various kinds of applications without having to worry about how to handle the application data.

A typical data mining is basket analysis where purchase records of customers can be analyzed to determine which products are purchased together. The results of the analysis can then be used to organize the layout of products in an offline store or to recommend products to customers in case of an online store. Other uses include sales prediction, credit card fraud detection, weather forecast, threat detection in computer networks and many more. Over the years, an enormous plethora of methods has been developed to accomplish various data mining tasks. Also, the state of the art data mining tools offer different methods ranging from statistics, probability and machine learning to realize various data

mining activities [2]. However, despite the presence of such huge resource for mining data, the science is still constrained to data scientists, analysts, researchers and academicians. The sole reason behind is the ad-hoc nature of the methods which are designed to carry out individual data mining tasks. So, in order to discover hidden patterns from data, one needs to know which appropriate method to use in finding a solution to a particular problem. The simplicity and ease of use of SQL have triggered the development of data-intensive applications on top of the relational model. But, going by the nature of data mining goals, this querying approach fits even better for mining tasks than data manipulation tasks. Though we follow such querying approach to mine data using OLAP (online analytical processing) technology [3] but such query language for data mining tasks is not exclusively defined.

The absence of a unifying model such as the relational model for databases that provides an ideal platform to carry out all kinds of data mining tasks without having to worry about the underlying logic is still missing. So, there is a need for a unifying model capable of doing several data mining tasks. It could open up new opportunities for developers to develop analytical applications which are the demand of this data rich era. With humongous amounts of data collected today, such applications can help realize goals of trending future technologies such as IoT (Internet of Things), Smart Cities, Smart Homes and many more. The study proposes such a model and an underlying framework with an aim of giving a "*Theory*" to data mining as a science. This study mainly focuses on the unifying algorithm running in the background which makes such unifying model conceivable. The unifying framework, when applied to given data, generates frequent patterns, which we refer to as models. The models are then used to carry out various data mining tasks. So, with the help of the unifying framework, all data mining tasks can be conducted in two stages viz. first generate the frequent models (patterns) and secondly, use these generated models to accomplish the task at hand.

In Section 2 we present some related studies. Section 3 provides the proposed framework and its components. Section 4 shows the data modelling algorithm. The experimental result is shown in section 5. Section 6 provides the summary of the drawbacks of the FM-growth approach and we conclude with Section 7 with a brief plan for carrying out our future research.

2. Related work

Such unifying framework had initially been proposed in the year 2000 [4]. The proposed framework was called Inductive Database. It was suggested to be a database that not only could store the data but also the frequent patterns observed in the data in a separate pattern store. Another such framework was proposed in [5]. Other researchers also proposed inductive database architecture inspired by MolFea [6, 7]. The researchers propose a string domain for the inductive database where both data and patterns are strings. Boulicat *et.al.* (1998) in [8] propose a query language for the inductive database which is based on SQL. VINLEN [9] and cINQ European Project [10] are two projects that focus on the development of such inductive databases.

Most of the real world data is stored in databases. Mining information and knowledge from such large databases has been identified as a key research topic. The topic has been identified by major industrial companies as a very crucial ad important area that has shown promising opportunities for major revenue generation. Data mining from the perspective of databases has been discussed in [11, 12, 13]. The frequent usages data marts are developed by using frequent pattern search algorithm [21].

Many frequent pattern mining algorithms have been developed to date. However, all such algorithms either follow an Apriori [14] approach or FP-Growth [15] approach. In Apriori, the frequent patterns are generated iteratively based on a support count (minimum frequency threshold). At each iteration, a candidate set of (k + 1) – length patterns is generated from the k-length frequent patterns generated in the previous iteration. Based on support count, infrequent (k + 1) – frequent patterns are pruned, and the remaining patterns are used to generate the candidate set for the next iteration. The process ends when no more frequent patterns can be generated and returns the set of frequent patterns. The primary drawbacks of the Apriori approach which make it computationally expensive are (1) Generating candidate set of (k + 1) – patterns in each iteration and (2) Multiple scans of the dataset for counting the support. To overcome the drawbacks faced by the Apriori approach, the FP-Growth approach was

introduced. In the FP-Growth approach, FP-tree is generated from the dataset in the first run. The frequent items can then be generated from the FP-tree. Another advantage of FP-tree is that it maintains the frequency or support count of all the different subsets of the frequent data itemsets and hence can be used to mine frequent itemsets at different hierarchical levels.

The FP- Growth approach is computationally efficient and also allow for the interactive and iterative mining of the data as per user preference. However, the FP-Growth Algorithm is much suited for itemset or sequence mining rather than other forms of mining. However, there is no common framework which describes a data modelling technique to model data in a manner that can be used to accomplish all kinds of data mining tasks.

3. Proposed Unifying Framework

To understand the requirements of the solution, we have to draw inspiration from the relational data model for databases. Merely storing the data is not enough; we need ways to search the data, sort the data, modify data and remove data. There are a number of algorithms to search or sort data. Which searching or sorting algorithm to use depends on the data structure used to store data such as arrays, lists, trees or graphs? The relational model introduced the concept of using relations (a tabular data structure) to store data. Any kind of data could easily be stored in relations. Simple searching and sorting algorithms could be used to efficiently search or sort data stored in relations. This simplicity of the system added to its acceptability and soon RDBMS (Relational Database Management System) entered the database market which included the relational model to store data, algorithms to search and sort data and a query language that enabled users to use the underlying storage model and algorithms from a high-level abstraction. RDBMS provides an interface which could be used by applications written in any popular programming languages to connect to and utilize its capabilities to accomplish their functionality. To take advantage of the features and capabilities of RDBMS, one needed to know the basic concepts of the relational model and working knowledge of SQL. Thus, the introduction of relational model and advent of RDBMS marked the shift of databases from a science domain to a mainstream technology.

In the field of data mining, the typical tasks can be classification, clustering, regression or association mining. There are corpora of algorithms to perform each of the tasks. Despite the diversity of methods, they all share at least one similarity. This similarity is that the end product of algorithms is a predictive model. For classification algorithms, this model is called classifier. In clustering, the model is called a cluster and for association mining, it is called a rule. Our desired solution requires a data modelling method or algorithm to be defined whose outcome is a unifying model that can be used as a classifier to classify new data or a cluster for clustering data or a rule whose support and confidence can be calculated.

Next, we need a source from which data is to be fed into the data modelling algorithm. Still many organisations have their data stored in relational databases. So, one of the primary sources of data for our solution is a relational database. But again, there a number of RDBMSs present in the market and each one is slightly different from the others. So, we need to define an interface through which our intended solution can interact with all major RDBMSs and extract data from these sources. Besides RDBMS, the solution could be extended to support other kinds of data sources ranging from flat files to web data.

We look that our solution to be a platform upon which many different kinds of data analytical applications can be developed. Again, we can get inspiration from RDBMSs in such a scenario. RDBMS provides SQL as a medium through which all different kinds of applications can communicate with it. So, our solution also needs a query language like SQL that can be used by applications to interact with it. After analyzing the requirements, we can define the solution as a framework that defines where to get the data from (data sources), a data modelling method and a simplistic query language as a medium of communication for other applications to communicate with it. Figure 1 (a) and (b) illustrates a basic proposed framework and different views respectively. The next section discusses in details about the core of the framework i.e. the data modelling method.

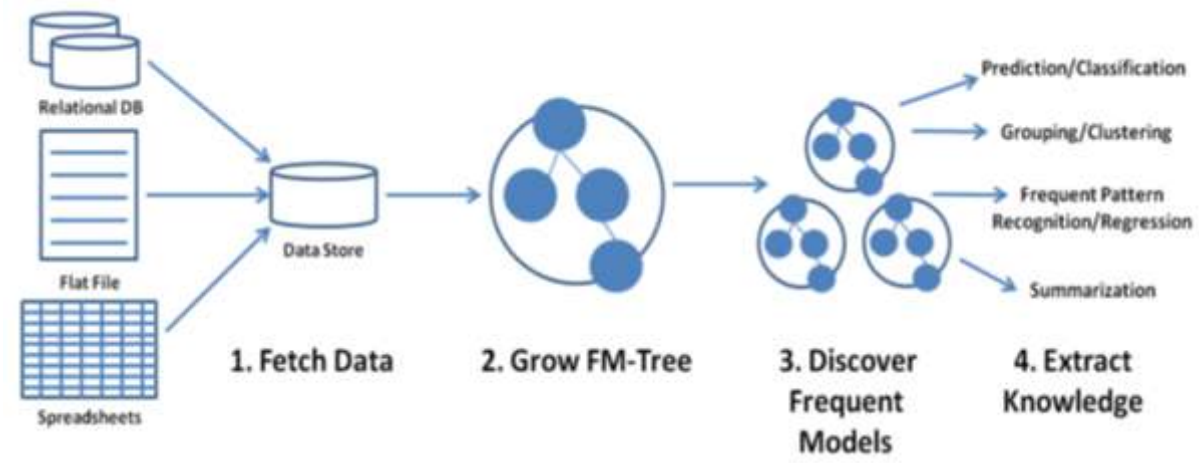

Figure 1. (a): Proposed Unifying Framework for Data Mining

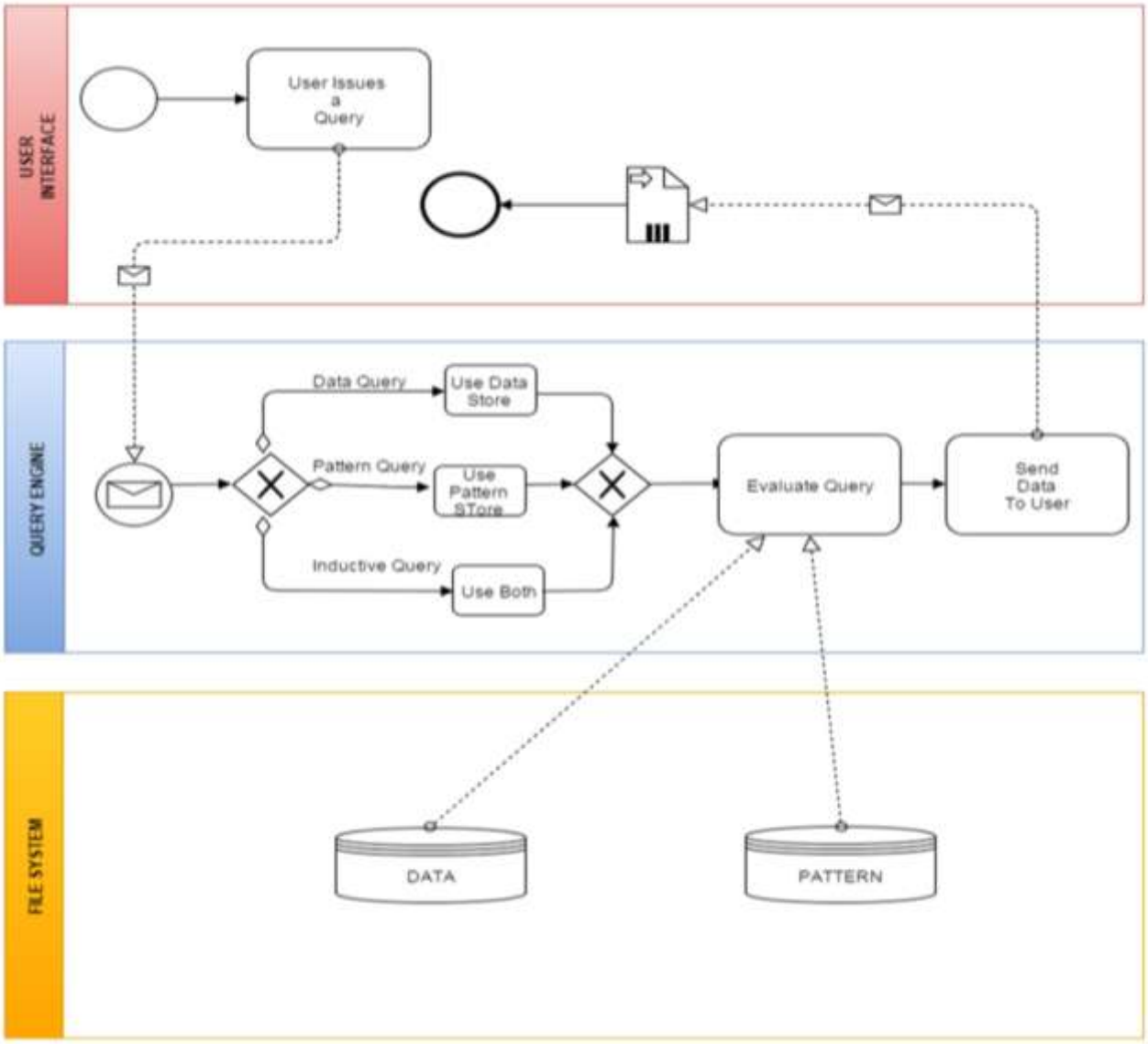

Figure 2. (b): Proposed Unifying Framework for Data Mining: Different layers

This proposed frame consists of several components. This frame can be seen from different layers such as user interface, query engine and file system as provided in Fig.1(b). The query engine plays a major role in modelling the data. The important components are described below.

The major components that constitute the framework are discussed below-

- Data Sources

The framework can accept data from multiple data sources such as relational data, data stored in flat files and spreadsheets.

- Data Store

The framework maintains a data store where all data from the different sources are accumulated. It acts as a warehouse for the framework.

- Data Modelling Algorithm

The data modelling algorithm accepts data and finds out the frequent patterns, referred to as models in the framework, from the data based on a minimum frequency threshold.

- Model Store

The frequent models generated in the data modelling step are stored in the model store for use by various data mining algorithms.

- Mining Algorithms

The mining algorithms are mainly simple programs that calculate the similarity of data instances with the generated models. Based on this similarity scores, the data instances are accordingly classified, clustered or association rules generated from the same.

- Query Interface

Users interact with the framework using declarative queries which specify what we want rather than specifying how to do it.

In this study, we mainly focus on designing the data modelling technique. In the following sub-sections, we discuss our proposed data modelling technique and also justify why it is suitable for various data mining tasks.

3.1 The Unified Data Model

The relational model describes the structure of the data whereas the data model provides a description of the data itself. There can be many descriptions of the same data. So we need many such models to get more accurate results. Thus, we need a model store which stores all the descriptions of the data (data models) in our proposed framework.

The data model is a description of the data. In RDBMS, each record or tuple is defined by a set of attributes or data items and can be considered as a data itemset. Data is described or summarised by the most frequently found data itemsets in the dataset. So, the data model is a frequently occurring data itemset or a subset of the data itemset. By adjusting the minimum threshold frequency, we can obtain different data models. Minimum threshold frequency signifies the minimum number of records in the data in which the attribute/feature values must occur to be considered a frequent model. The best choice for minimum threshold frequency is determined by running the algorithm several times under different minimum threshold frequency settings.

Table (1): A sample dataset

Buying	Maint	Doors	Persons	Lug_boot	Safety	Class
vhigh	vhigh	2	2	small	low	unacc
vhigh	vhigh	2	2	small	med	unacc
vhigh	vhigh	2	2	small	high	unacc
vhigh	vhigh	2	2	small	low	unacc
vhigh	vhigh	2	2	small	med	unacc

In the example dataset shown in Table (1), if we set the minimum frequency threshold to be 1 (i.e. the values must atleast occur in 1 record to be considered frequent model), then each data itemset is a data model. If we increase the minimum frequency threshold frequency to be 2, we get two data models which are subsets of the data itemsets or tuples as shown in Fig.(2) and so on. From this example, we can generate the idea of the data model and can verify if this data model can be used to carry out all data mining tasks or not. In the next sections, we are describing how we can use this data model as a basis for various data mining tasks. The examples to use the generated model in different mining tasks are provided below with exemplary basic data.

Buying	Maint	Doors	Persons	Lug_boot
vhigh	vhigh	2	2	small
vhigh	vhigh	2	2	small

Buying	Maint	Doors	Persons	Lug_boot
vhigh	vhigh	2	2	med
vhigh	vhigh	2	2	med

Figure (2): Frequent subsets of data itemset

3.2 Classification Using Unified Data Model

We are aiming to use the above-generated data model (as shown in Figure (2)) for a quick classification. In classification [16], we have a training dataset which is used to learn the classifier. A test data set is used to assess the quality or accuracy of the learned classifier. The classifier is then used to classify new data points into predefined classes. Let us use the dataset shown in Table (1) as our training dataset. By using a minimum frequency threshold of 4, we get only one data model as shown in Figure (3) which serves as our classifier in this case.

Buying	Maint	Doors	Persons
vhigh	vhigh	2	2
vhigh	vhigh	2	2

Figure (3): Learned Classifier

Now, we use a sample dataset as illustrated in Table.(2) to test the accuracy of our classifier. The accuracy is the ratio of the total number of correct classifications to the total number of data items in the test dataset.

$$\text{Accuracy} = \frac{Total\ number\ \ of\ correct\ \ classifications}{Total\ number\ \ of\ data\ \ ites} \quad .....(1)$$

The accuracy obtained, in this case, is 1. Thus, we see that our data model fits well in carrying out classification tasks. But whether or not it is suitable for other kinds of data mining tasks is yet questionable. So, in the next section, we consider another data mining task, clustering and assess the usability and efficiency of our data model.

Table (2): Test Dataset

Buying	Maint	Doors	Persons	Lug_Boot	Safety	Class
vhigh	vhigh	2	2	med	high	unacc
vhigh	vhigh	2	2	big	high	unacc
vhigh	vhigh	2	2	big	low	unacc
vhigh	vhigh	2	2	med	high	unacc
vhigh	vhigh	2	2	big	med	unacc

3.3 Clustering Using Unified Data Model

We are aiming to use the above-generated data model (as shown in Figure (3)) for a quick clustering. Clustering [17] can be identified as the unsupervised version of classification. In classification, we know the classes in advance, and any unclassified data object is classified into one of the classes using a classifier. Now, if we group all data objects according to their classes, we form k-clusters where k is equal to the number of classes available. The intra-cluster similarity between objects is high, and the inter-cluster similarity between objects is low. Each cluster is defined by the most frequent data itemset in the cluster. So, there will be at least as many frequent data itemsets as the number of clusters. Each new data object is compared with each of the frequent data itemset and based on similarity closeness; the object is assigned a cluster. Thus, we observe that our data model is well compatible for clustering data. In the following section, we look into association rule mining and assess if our data model is suitable for the task or not.

3.4 Association Mining Using Unified Data Model

We are also aiming to use the above-generated data model for a quick mining the association of data.

Association mining [18] or association rule mining is extensively used in market basket analysis. For mining association rules from a transaction database, all the frequent itemsets are first generated and then association rules are generated from the frequent itemsets. The rules are of the form X →Y where X and Y are subsets of the frequent itemsets. Going by the very nature of association mining, we find that our data model is suitable for mining association rules because our data model is a frequent itemset. The rules that can be generated from the data models illustrated in Figure (3) are:

{'Buying':'high','Maint':'vhigh','Doors':'2','Persons':'2','Lug_boot':'small'}→{'Class':'unacc'}

{'Buying':'vhigh','Maint':'vhigh','Doors':'2','Persons':'2','Lug_boot':'med'}→{'Class':'unacc'}

Thus, we observe that our data model can be best fit for all popular data mining tasks. In the next section, we define a baseline algorithm to generate the frequent data itemsets or models.

4. The Data Modelling Algorithm for Model Generation

The FP- Growth Tree approach is computationally efficient and allows for the interactive and iterative mining of the data. However, the FP-Growth algorithm is much suited for itemset or sequence mining rather than other forms of mining. So, we introduce our Frequent Model (FM)-tree growth algorithm which is derived from the FP-Growth Tree algorithm which may suit to work with all kinds of mining tasks. The outline of the data modelling algorithm using the FM-Growth tree approach is as follows:-

Input: Dataset, Minimum Frequency Threshold (support count)

Output: FP-tree and frequent patterns satisfying the frequency threshold.

Stage 1: Grow the FM –tree from the dataset.

Stage 2: Generate the frequent models from the FM-tree with the input support count. The stages of the algorithm are provided.

4.1. Stage 1: Grow the FM-tree

Input: The Dataset containing the data,
Output: The FM-Tree.
Method: Call GrowModelTree procedure.

Procedure GrowModelTree(dataset):

1. Initialize the root node of FM-tree to NULL
2. for each data instance/record in dataset do
3. for each attribute of record do
4. if a node with attribute value does not exist then
5. Add the node
6. Initialize its value to 1
7. else
8. Increment the node value by 1
9. endif
10. rof
11. rof
12. return FM-tree

4.2. Stage 2: Find the Frequent Models from FM tree

Input: Node of Interest (NoI), FM-tree and minimum support.

Output: FList (list of all the frequent models)

Method: Call FindFrequentModels procedure

Procedure FindFrequentModels(NoI,FM-Tree, Min. Support):

1. Find the Conditional Bases for Node of Interest (NoI)
2. Calculate the support counts of all nodes in the Conditional Bases of NoI
3. Prune all nodes that do not satisfy the minimum support threshold
4. Add the remaining to FList
5. return FList

5. The Experiments

For carrying out our experiments, we considered the Car Evaluation Dataset from UCI Machine Learning Repository [19]. It is a multivariate dataset with categorical attributes. The description of the dataset and its attributes is discussed below. The dataset consists of

1728 data instances. We divide the dataset into three equal halves. We use 2/3 of the dataset as training data and the remaining 1/3 as testing data. So, our training dataset consisted of 1152 instances and testing dataset 576 instances. We compare the proposed algorithm with k-Nearest Neighbours (kNN) [20] which is one of the best known algorithms for classifying data.

In kNN, each new unlabelled data item is compared with all the data instances in the training sample. Then, **k** nearest neighbours of the data item are identified and the unlabelled data item is assigned the majority class of the k nearest neighbours. We selected different values of k and calculated the accuracy and execution time of kNN for each case. The experimental results obtained are demonstrated in the table below.

Table (3): Experimental results of kNN

k	Accuracy	Execution Time
20	71.06%(approx)	458 secs(approx)
50	71.18%(approx)	482 secs(approx)
100	71.057%(approx)	500 secs(approx)

Here, we can observe that as we increase the value of k, the execution time increases but the accuracy remains almost the same in all three cases. We then apply the proposed algorithm on the data at various levels of minimum support threshold and evaluate the accuracy and execution time in each case. The detailed functioning of the algorithm with the help of suitable example is discussed below.

Let us try to understand the algorithm with the help of an example. Consider the following dataset in Table (3). The dataset consists data about six features of a car. These six features can have values from a given domain. The features and their values are shown in the Table (4). Based on the values of these features, each of the data instances is either classified as Acceptable (acc), Unclassified (unacc), Good (good) and very good (v-good). Now, the values of the features which occur most frequently for the class to be either unacc or acc are the frequent models we are searching for.

Table (4): Example Dataset

Buying	**Maint**	**Doors**	**Persons**	**Lug_boot**	**Safety**	**Class**
vhigh	vhigh	2	2	small	low	unacc
vhigh	vhigh	2	2	small	med	unacc
vhigh	vhigh	2	2	small	high	unacc

Table (5): Domain of possible value of attributes

Attribute/Feature	Domain of possible values
Buying	vhigh,high,med,low
Maintenance	vhigh,high,med,low
Doors	2, 3, 4, 5-more
Persons	2,4,more
Luggage Boot	small,med,big
Safety	low,high,med

First, we start growing the FM-tree from the dataset. We choose the root as 'NULL'. The FM-tree has the same number of levels as the number of attributes in the dataset plus the root (level 0). Since there are seven features in our dataset, our tree will have seven plus one levels. Each level might have a different number of nodes. Nodes at each level are *key: value* pairs. The keys represent the domain of possible values for the attribute at that level, and the values represent the lists of records from the dataset where the key occurs.

For growing the FM-tree, the data modelling algorithm chooses one record from the dataset at a time and grow one branch of the FM-tree. We start at the root and add nodes as we proceed. Let's start with the first record (Figure (4)) in our example dataset. So, we start at the root at our level 0. Now, the attribute at level 1 is ***'Buying'***. There can be four possible nodes at this level which is equal to a number of possible values that *Buying* can take and the keys of these four nodes can be **vhigh, high, med, low.** The values of these nodes indicate the count or number of times the key has occurred as the value of *Buying* attribute in the dataset. Initially, all the counts are 0 (zero). Whenever a particular key is encountered as the value of the attribute at that level, the count for that key (node) is incremented by 1. For the first record in the example dataset, the value of *Buying* is *vhigh* and so the count of the node with *vhigh* as the key is incremented by 1. Similarly, we repeat the process for each of the attributes of the first record in the dataset to obtain the first branch of the FM-tree. Figures 5(a), 5(b), 5(c) illustrate visually how the first branch of the FM-tree is constructed. Each branch is represented by a tabular structure where each row represents a level of the tree. Each row consists of two columns where the first column represents the value of node at that level and the second column represents how many times the node takes this particular value in the data.

Buying	Maint	Doors	Persons	Lug_boot	Safety	Class
vhigh	vhigh	2	2	small	low	unacc

Figure (4): First Record in Example Dataset

Attribute & Value	Frequency
Buying = vhigh	1
Maint = ?	1
Doors = ?	1
Persons = ?	1
Lug_Boot = ?	1
Safety = ?	1
Class = ?	1

Figure (5)(a): FM-tree

Attribute = Value	Frequency
Buying = vhigh	1
Maint = vhigh	1
Doors = ?	1
Persons = ?	1
Lug_Boot = ?	1
Safety = ?	1
Class = ?	1

Figure (5)(b): FM-tree

Attribute = Value	Frequency
Buying = vhigh	1
Maint = vhigh	1
Doors = 2	1
Persons = 2	1
Lug_Boot =small	1
Safety = low	1
Class = unacc	1

Figure (5)(c): FM-tree

Buying	Maint	Doors	Persons	Lug_boot	Safety	Class
vhigh	vhigh	2	2	small	med	unacc

Figure (6): Second Record in Example Dataset

Next is the construction of the next branch of the FM-tree from the next record in the dataset. The value of *Buying* attribute at Level 1 is *vhigh* and hence, the node's count value is incremented by 1. Similarly, the counts for levels 2, 3, 4, 5 are incremented since the values are the same as a previous branch. But when we get to level 6, we find that no node with key *'med'* exists. So, we introduce a new node with key *med* and initialize its value to 1. Figures (7) illustrate the growth of the second branch of the FM-tree. We repeat the same process with the third record in the example dataset to grow the third branch of the FM-tree. Figure (8) illustrates the complete FM-tree which is generated after constructing the third branch from the third record in the data.

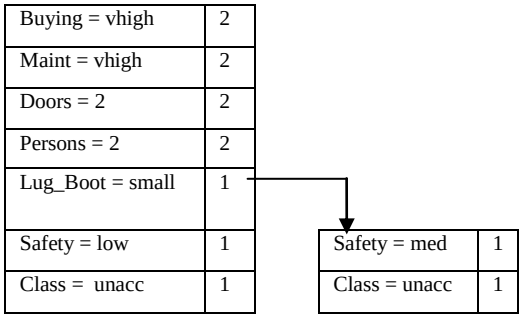

Figure (7): Growth of the second branch of the FM-tree

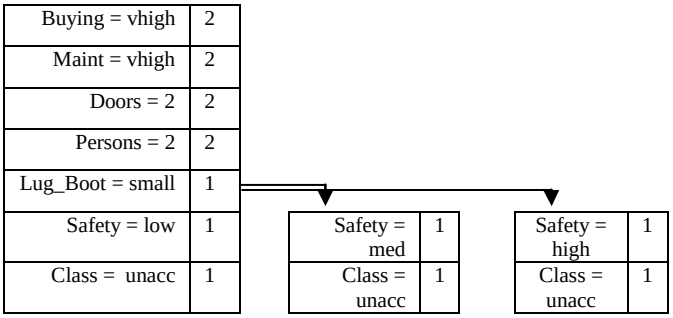

Figure (8): The complete FM-tree

The next stage of the algorithm is to find frequent models from the FM-tree. Note that growing the FM-tree is a one-time process while finding frequent models is a repetitive process. Once the FM-tree has been grown, one can generate frequent models from the FM-tree iteratively.

To find the frequent models, we need to provide the algorithm with a Node of Interest (NoI) and a minimum threshold support as inputs. The NoI is a *key:value* pair where the key is the attribute at a particular level of the FM-tree and the value is one from the domain of possible values for that attribute. The algorithm then calculates the frequent models from the FM-tree. Frequent models are the most frequently occurring sequence of node values of the branches containing

the NoI. For example, in our example dataset, we want to find all the frequent models that lead to the *Class* attribute taking the value *unacc*. In simple words, we want to find all the frequent values of *Buying, Maint, Doors, Persons, Lug_boot* and *Safety* for which the value of *Class* is *unacc*. Remember to be considered as frequent the values must occur at least equal to or more than the provided minimum support threshold.

Let's find the frequent models from our FM-tree. We consider *'Class':'unacc'* as the node of interest (NoI). Suppose; the minimum support threshold is 2 i.e. the attribute values must occur at least 2 times in the data to be considered frequent. To generate the frequent models, we need to find the *Conditional Bases* for the NoI. The Conditional Base of a node is the sequence of node preceding the node in the FM-tree. In our FM-tree, notice that all three branches end in the leaf node, '*Class':'unacc'* (NoI) and so we will have three conditional bases for NoI and these are but the sequence of node values preceding NoI. So, the *Conditional Bases* for NoI are-

vhigh, vhigh, 2, 2, small, low
vhigh, vhigh, 2, 2, small, lmed
vhigh, vhigh, 2, 2, small, high

Now, calculate the support of all the nodes at all levels of FM-tree in the conditional bases of the node to get our FList which is a list of all the frequent node values satisfying the minimum threshold support. The support count of node values at several levels of FM-tree in the conditional bases of NoI are-

Level 1 (Buying)→vhigh:3,
Level 2 (Maint)→ vhigh:3,
Level 3 (Doors)→2:3,
Level 4 (Persons)→ 2:3,
Level 5 (Lug_Boot)→small:3,
Level 6 (Safety)→ low:1, med:1, high:1.

Now, we notice that the values at level six do not satisfy the minimum support threshold of 2, and hence, these are pruned, and we obtain our FList-

{'Buying':'vhigh', 'Maint':'vhigh', 'Doors':2, 'Persons':2, Lug_Boot':'small','Class':'unacc'}

This is the frequent model/model and can now utilize it to classify or cluster new data instances. Similarly, we can generate as many frequent models as required by varying the NoI and/or minimum support threshold. Any new unlabeled data instance can be compared with these frequent models to find its similarity with the models and is assigned the class or cluster of the model it shares the maximum similarity with. It can be noticed that by choosing low threshold support results in the generation of many weak models. The prediction is performed through a majority vote of all these weak models. Thus, we use an ensemble learning approach for prediction where we compare the data instance with many weak models instead of one strong model. This reduces overfitting and hence induces more accurate prediction.

In such a way the dataset is executed, and the following results as shown in Table (6) are obtained-

Table (6): Summary of results of study

Min. Support	Accuracy	Execution Time
30%	68%(approx)	5 secs(approx)
40%	61.07%(approx)	5 secs(approx)
60%	53%(approx)	5 secs(approx)

After analysing the results of both kNN and FM-Growth algorithm, we see that both algorithms attain similar accuracy rates while selecting low threshold minimum support for FM-Growth algorithm. But we can clearly observe that the execution time of FM-Growth algorithm is much more efficient than kNN. Hence, both kNN and FM-Growth algorithms exhibit nearly similar accuracy rates however FM-Growth algorithm clearly emerges the better one when considering the execution times of the algorithms. Thus, we can conclude that FM-Growth algorithm is much more efficient than kNN while considering both the factors viz. accuracy and execution times.

6. Further Enhancements

Though the FM-Growth Tree approach for frequent model generation is better than the kNN in terms of efficiency but during our experimentation, we came across some issues where the approach shows lagging. First, when, we try to investigate the frequent models corresponding to a particular NoI, we must first locate all the branches where the NoI exists. For this, we have to search the entire FM-tree. But it is known that the worst case time for searching a tree data structure is ***n*** where ***n*** is the height of the tree. So, if we have many levels in our FM-tree, which is very common, the search for branches from which the frequent models are to be generated itself becomes very expensive. So, if we can reduce the search time to unit time, we can

significantly increase the efficiency of the overall process.

Secondly, we know that frequent models generated are based on the selection of Node of Interest. As discussed above, the frequent models are just the sequence of nodes and their values that precede the NoI. For better prediction, it is always recommendable to consider all the nodes in the branch. As such, we obtain the best results when NoI considered is a leaf node. But this is not an ideal case. Any node irrespective of whether it is a leaf node or internal node must get equal opportunity to be selected as NoI. In such a case, where an internal node needs to be considered as NoI, we need to shift its position with the leaf node which can be expensive and thus affect the efficiency of the overall process. So, if we can reduce this shifting step, we can increase the efficiency of the overall process. After examining the nature of the drawbacks, we suggest the use of data structure other than a tree to reduce the search time. Dictionary which stores data as *key:value* pairs seem to be a promising data structure that reduces the search time to unit time and since data is stored as *key: value* pairs, the order or sequence doesn't affect the search time of the algorithm and we can easily get done with the issue of shifting.

7. Conclusion

The proposed algorithm FM-Growth finds frequent patterns or models from data to used to carry out various data mining tasks. This is an unifying modelling framework which generate general models that can then be used to further carry out various data mining tasks. So, with the help of the unifying framework, all data mining tasks can be conducted in two stages viz. first generate the frequent models (patterns) and secondly, use these generated models to accomplish the task at hand. It has the advantages that the frequent models can be used as it is with any data mining task irrespective of the nature of the task. The algorithm is carried out in two stages. In the first stage, we grow the Frequent Model (FM)-tree from the data and in the second stage, we extract the frequent models from the FM-tree. The proposed algorithm is compare with kNN and found that the computation time for the FM-growth has better than kNN even though they have almost similar accuracy rate. Such study can save lot of execution time. This reduces over fitting and hence induces more accurate prediction.

In the near future, we would test our proposed framework on benchmarks datasets. The above-mentioned model and algorithm mainly assume data to be categorical in nature. We would try to extend the framework to accommodate different types of data such as continuous, boolean or ordinal. We would focus on the design the basic structure of the query interface. While doing so, we would strive to draw inspiration from SQL so that learning of the new query language is a cakewalk for those already accustomed to SQL at the most beginner level or expert level. Data mining requires data to free from noise and data pre-processing is essential for obtaining better results from mining data. We would extend our framework to include such pre-processing on data through querying. Further research can be carried out to suggest some optimization measures that the framework should adopt for more efficient performance.

References

[1] EF Codd, "A relational model of data for large shared data banks." Communications of the ACM 13.6 (1970): 377-387.

[2] U Fayyad, G Piatetsky-Shapiro, P Smyth, From data mining to knowledge discovery in databases, "From data mining to knowledge discovery in databases." AI magazine 17.3 (1996): 37.

[3] S Chaudhuri, U Dayal, "An overview of data warehousing and OLAP technology." ACM Sigmod record 26.1 (1997): 65-74.

[4] H Mannila, "Theoretical frameworks for data mining." ACM SIGKDD Explorations Newsletter 1.2 (2000): 30-32.

[5] DM Khan, N Mohamudally, DKR Babajee,"A unified theoretical framework for data mining." Procedia Computer Science 17 (2013): 104-113.

[6] C Helma, S Kramer, L De Raedt,"The molecular feature miner MolFea." Proceedings of the Beilstein-Institut Workshop. May, 2002.

[7] L.De Raedt, "A perspective on inductive databases." ACM SIGKDD Explorations Newsletter 4.2 (2002): 69-77.
[8] JF Boulicaut, M Klemettinen, H Mannila, "Querying inductive databases: A case study on the MINE RULE operator." European Symposium on Principles of Data Mining and Knowledge Discovery. Springer Berlin Heidelberg, 1998.
[9]KA Kaufman, RS Michalski, "The development of the inductive database system VINLEN: A review of current research." Intelligent Information Processing and Web Mining. Springer Berlin Heidelberg, 2003. 267-276.
[10] JF Boulicaut, "Inductive databases and multiple uses of frequent itemsets: the cInQ approach." Database Support for Data Mining Applications. Springer Berlin Heidelberg, 2004. 1-23.
[11] U Fayyad, G Piatetsky-Shapiro, P Smyth,"From data mining to knowledge discovery in databases." AI magazine 17.3 (1996): 37.
[12]T Imielinski, H Mannila, Imielinski, Tomasz, and Heikki Mannila. "A database perspective on knowledge discovery." Communications of the ACM 39.11 (1996): 58-64.
[13] MS Chen, J Han, PS Yu, "Data mining: an overview from a database perspective." IEEE Transactions on Knowledge and data Engineering 8.6 (1996): 866-883.
[14]C Borgelt, R Kruse, "Induction of association rules: Apriori implementation." Compstat. Physica-Verlag HD, 2002.
[15] C Borgelt, "An Implementation of the FP-growth Algorithm." Proceedings of the 1st international workshop on open source data mining: frequent pattern mining implementations. ACM, 2005.
[16]TN Phyu, "Survey of classification techniques in data mining." Proceedings of the International MultiConference of Engineers and Computer Scientists. Vol. 1. 2009.
[17] P Berkhin, "A survey of clustering data mining techniques." Grouping multidimensional data. Springer Berlin Heidelberg, 2006. 25-71.
[18] A Ceglar, JF Roddick, "Association mining." ACM Computing Surveys (CSUR) 38.2 (2006): 5.
[19]A Asuncion, D Newman, "UCI machine learning repository." (2007).
[20] LE Peterson, "K-nearest neighbor." Scholarpedia 4.2 (2009): 1883.
[21] SZ Barbhuiya, B Kumar, Z Azim, Y.J Singh, "Suggestive Local Engine for SQL Developer: SLED." ADBU Journal of Engineering Technology 4 (2016).

Authors Profile

Shahid Zaman Barbhuiya, is MTech (CSE) student of School of Technology, Assam Don Bosco University. He completed his Bachelors in Engineering from APIIT SD India in 2013 and currently pursuing his Masters in Technology in Computer Science and Engineering. His specialization is Data Mining.

Yumnam Jayanta, is working as Professor and Head of Dept of Computer Science & Engineering and IT, School of Technology, Assam Don Bosco University, Guwahati. He has received his PhD from Dr. B.A Marathwada University in 2004. He has worked with Swinburne University of Technology (AUS), Misurata University, Keane (India and Canada), TechMahindra, Skyline University College (UAE) etc. His research areas are ETL, Data Warehouse and Mining, Real-time Database system, and Image processing. He has produced several papers in International and National Journals and Conferences.