# Multi-Resolution Hierarchical Structure for Efficient Data Aggregation and Mining of Big Data

*Korra Bichya, Research Scholar and Dr. Suribabu Potnuri, Professor*

*Department of Computer Science and Engineering,*

*J.S. University, Shikohabad, U.P. India email: korra.bichya@gmail.com*

## ABSTRACT

For modern applications in a variety of disciplines, including healthcare, assistive technology, intelligent transportation, environment, and climate monitoring, the use of big data analysis is very necessary. With regard to the management of enormous volumes of data, the conventional methods that are used in the area of data mining and machine learning are not effective. It is necessary to have effective methods for mining and learning from massive amounts of data, even if there is a possibility that such methods would compromise accuracy. A framework for data aggregation has been developed by us in order to reduce the amount of data that is comprised of a large number of instances and comes from a variety of data sources. There are many different degrees of detail that are used to gather and aggregate data, and the level of detail that is selected requires a careful balancing act between being efficient and accurate. After the initial construction, the building is then progressively modified over the course of subsequent years. It performs the role of a common data input for a variety of learning and mining algorithms simultaneously. The algorithms used in data mining are modified so that they can handle aggregated data as input. For the purpose of analyzing and extracting useful information from enormous datasets, hierarchical data aggregation is a system that integrates innovative data representations and algorithms. For the purpose of determining how successful it is, we used a multi-resolution Naive Bayes Classifier on the structure of the data aggregation. The outcomes of the experiments show that the proposed framework makes it possible for the classifier to reduce the amount of time it takes to calculate by an average of 75% and to limit the amount of memory it uses while still retaining the accuracy of the results.

***Keywords***—*Big data reduction, data aggregation, multiresolution data mining.*

## 1. Introduction

In a number of applications, including as healthcare, assistive technology, intelligent transportation, and monitoring of the environment and climate change, the study of large-scale data is an essential component. During a very short period of time, many applications need the gathering of millions of data instances at once. Conventional data mining approaches, on the other hand, run into limitations when trying to manage large-scale data sets because of limited memory and central processing unit capabilities.

The extraction of useful information from data that was previously unknown or implied is accomplished via a process known as data mining, which is a complicated and repeated process. As a result of the countless repetitions of data scanning required by various data mining methodologies. Because of this, the process of obtaining massive amounts of data on an individual basis for the goal of mining is very expensive. As a result, researchers have been compelled to investigate creative methods for expediting the study of big data. These techniques encompass a wide variety of approaches, such as parallel algorithms, which are used to make efficient use of multi-core processors; compression algorithms, which are used to minimize the amount of data storage that is required; dimensionality reduction algorithms, which are used to reduce the number of variables in the data; and data summarization techniques, which are used to reduce the number of data instances in big data.
When compared to dealing with raw data that is excessively redundant, inconsistent, and noisy, doing data mining on data that has been summarized, minimized, and is relevant is more efficient. On the other hand, there has been a dearth of empirical research that has been conducted with the intention of reducing the number of data instances via the process of summarizing, and then using the condensed data for the purposes of data mining. In order to accomplish the goal of data mining, the use of condensed data involves a number of technical challenges.

Condensed data should be used as universal measurements in order to accommodate a wide variety of data mining techniques. This is the first challenge. The selection of relevant techniques for summarizing is dependant upon the particular properties that are needed by each data mining methodology for the construction of its model. In order to accomplish this goal, the condensed data should provide each data mining technique with a collection of information that is not only succinct but also complete.

When it comes to the second challenge, the duty of organizing the summaries in a variety of resolutions is involved. The user is able to choose the resolution that is most appropriate for their needs and find a balance between time, memory, and accuracy depending on the resources that are available to them and the requirements of the program. This is made possible by the availability of a large number of resolutions.The efficient generation and maintenance of the condensed data is the third issue that has to be addressed. Despite the fact that the creation process should only take place once, it is of the utmost importance that it be carried out at a rapid pace and, more importantly, that it be able to be updated gradually. Presented in this research is a tree structure that is shared across many resolutions. The proposed tree structure has as its primary objective the provision of scalability and dependability in the process of summarizing enormous amounts of data, both in real time and in the past, in a format that supports several resolutions. This is done with the intention of facilitating efficient data mining. A representative and condensed collection of enormous data is provided by the structure for the purpose of usage by mining and learning algorithms. This enables these algorithms to execute their model more effectively and with reduced memory consumption, which may result in a loss of accuracy that is minimal.

It has been decided to divide the remaining amount of the job into five distinct components. The first section offers background information and previous studies. An explanation of the data cube lattice, which was developed expressly for the purpose of managing vast amounts of data, is presented in the next section. In the third section, the framework that was recommended is presented, together with extensive information on its implementation and a study of its performance. A description of the tests that were carried out and the discoveries that were brought about by those trials can be found in the fourth section of the text. This paper concludes with a discussion of possible subjects that may be the subject of more investigation.

## Background And Previous Works

The number of instances that are included inside the dataset is a factor that determines the amount of time that is required to complete various data mining and machine learning techniques. In its quadratic solution, the Support Vector Machine (SVM) has a time complexity of O(N3), and when it is used in conjunction with the sequential minimum optimization (SMO) technique, it has a best complexity of O(N2) [5]. Here, N is the number of occurrences. On the other hand, similarity-based classifiers, such as KNN, have a time complexity of O(Ndk), while conventional decision trees, such as CART, hjjave a time complexity of O(NlogN). When a significant number of events are taken into consideration, these approaches, which include mining algorithms that may need several scans of data, become computationally onerous. Even though it may result in a possible drop in accuracy, some mining and learning algorithms need a preprocessing step in order to reduce the number of instances N. This is something that is necessary in order for them to be able to manage big data sets.

Based on the findings of a recent study, it has been shown that random sampling is the only method that data scientists often use in order to quickly extract insights from a huge dataset. As a result of the highly unique and irregularly distributed attribute values, it is difficult to get a representative random sample for a big dataset that is not steady.

A method of instance reduction known as instance selection includes picking a smaller set of instances from a larger collection of raw data in order to reduce the amount of data that is collected. The subset processes severely diminish the quality of data and contain a computational cost of at least O(NlogN). Because of this, instance selection is not suitable for use with large-scale data sets. It is essential to have a clear understanding that the ideas of sampling and instance selection are two separate things. Sampling is the process of picking data points from the original dataset in a random fashion, while instance selection is the process of selecting the instances that are the most informative by taking into consideration the associations between them.

By combining instances that are connected to one another into a single instance, data aggregation is a technique that may be used to reduce the number of instances that are redundant. In order to support complex analysis and informed decision-making, advanced approaches for aggregation include the use of multidimensional data cubes. These cubes store aggregated data in subspaces, which allows for efficient data storage. Within the realm of business intelligence, it is used to a significant degree. OLAM, which stands for "online analytical mining," is a technique that enhances the effectiveness and scalability of data mining by combining mining models with a multidimensional data cube and an online analytical processing (OLAP) engine. The size of a multidimensional data cube grows exponentially as the number of dimensions on the cube increases, which is the primary reason for its enormous size. There have been a number of different approaches proposed as potential solutions to the problem. As an example, the iceberg cube does not take into consideration data cells that have a lower number of data instances than a threshold that the user specifies. There are a significant number of cells in the data cube that are poorly occupied, which indicates that there are places in the multidimensional data space that are unavailable. It may be possible to get rid of these cells in order to improve the effectiveness of the educational and mining operations.

or the purpose of facilitating quick cross-referencing, it is necessary to link data cells that have been integrated at different levels of detail. Among the many potential solutions to the problem, one of the most famous strategies is to adopt a certain index structure. In order to alleviate the inefficiencies that were experienced by the multidimensional data cube, a tree-based index structure was implemented. In order to accomplish this goal, several versions of the KD-tree, the aR-tree, and the R+-tree have been used. When it comes to working with data that has a high dimensionality, however, K-DTree-based approaches are notorious for their complicated creation process and their slow execution. In addition to this, it does not have the capability to do incremental updates. The aR-tree method is responsible for handling areas that overlap, which results in an unnecessary load of processing for our particular scenario, in which we only deal with regions that do not overlap (as mentioned in Section III). On the other hand, the R+-tree method is not able to deal with data that has been aggregated. In addition, the CF tree that is employed in the BIRCH methodology and the R* tree that is utilized in the DBSCAN method are also examples of different categories of index structures. On the other hand, these methods are tailored to the particular sorts of geographical data, and their use is contingent upon the query. Once the tree has been deployed for a particular query (data mining operation), it is not possible to reuse it for any other future searches. As a consequence of this, several indexing strategies require using multiple scans of the raw data.

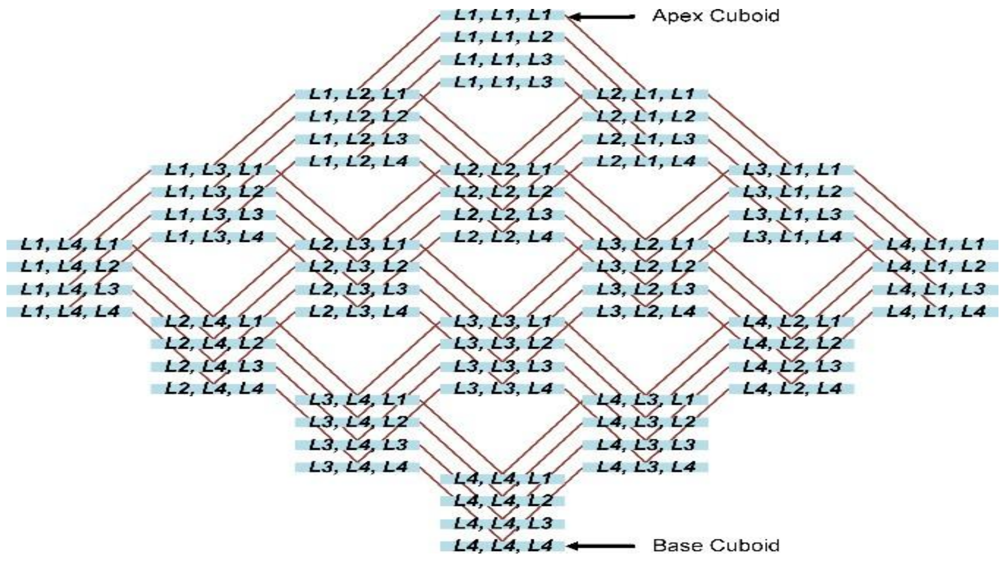
## DATA CUBE

For the purpose of this investigation, a dataset with dimensions N x d is investigated. Here, N stands for the number of occurrences, and d is the number of variables. The presumption is that N is much more than 2d. This set of variables has values that are both continuous and numerical in nature.
One kind of data that is often seen is continuous data, which includes the information that is generated by sensors such as temperature, air pressure, gyroscope, accelerometer, GPS, gas sensors, and water sensors. Due to the lack of a concept hierarchy in continuous data, the process of aggregating continuous data into a multi-resolution hierarchical structure is a difficult one to do.

The use of a hierarchical multilevel grid summarizing approach is what we propose The hierarchical multilevel grid on multiple dimensions can be assumed as a

as a means of reducing the total number of data instances. In the disciplines of geographic data mining and robotic mapping, an approach that is analogous to this one has been used recently. This strategy entails dividing each dimension in the dataset into a limited number of bins that are all the same width and do not overlap with one another. A summary of the information that pertains to the raw data instances that are included inside each area is included in each single area. In order to effectively distinguish between the various areas, we provided a number designation to each one. Numerous smaller regions are joined to produce a bigger region at a higher level, which is made possible by the multilevel structure, which enables the aggregation of data at a variety of different levels of detail. It is our hypothesis that the size of each level as a whole is equal to half of the size of the level immediately below it. For continuous data, this technique provides a hierarchical structure to take use of.

multidimensional data cube. Fig. 1 shows an example of a multidimensional data



cube lattice generated from three dimensions, each has four levels of aggregation.

**Fig. 1. The lattice of multidimensional data cube that generated from three dimensions.**

Every single node in the lattice is a rectangular solid that represents a different combination of aggregation levels (one level from each dimension). The cuboid structure starts with a base cuboid (L4, L4, L4), which is made up of the lowest levels (levels 4), each of which has dimensions of 1, 2, and 3 accordingly. It then moves on to the top cuboid, also known as the apex cuboid (L1, L1, L1), which is produced by the highest levels (levels 1), which have dimensions of 1, 2, and 3, respectively. In light of this, the data cube lattice allows for the consolidation of data by taking into account all of the various combinations of dimension levels that exist between the two cuboids.

Each and every cuboid is made up of several cells. Through the use of a multi-value index, which is comprised of a collection of region numbers (one from each dimension), the cells of the data cube are recognized and retrieved successfully. An example of a basic cuboid of the cube lattice is shown in Figure 2. Assuming that there are eight areas in each of the three dimensions, we will suppose that there are four levels. In addition, it depicts a cell that is referred to as <1, 7, 8>. This cell is made up of regions 1, 7, and 8 that are located in dimensions 1, 2, and 3, respectively.

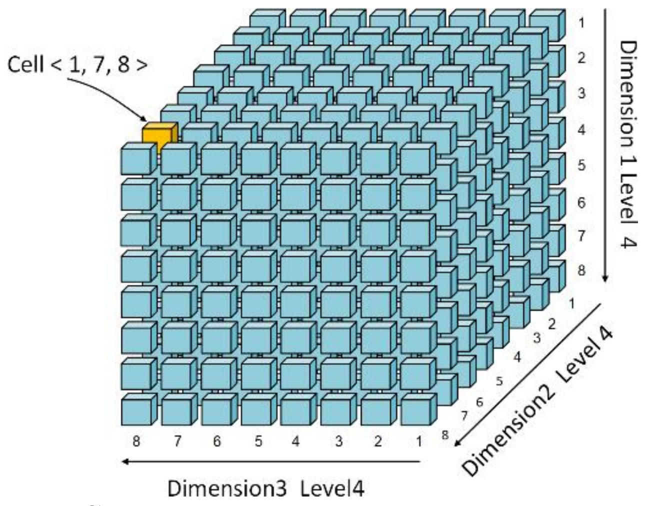
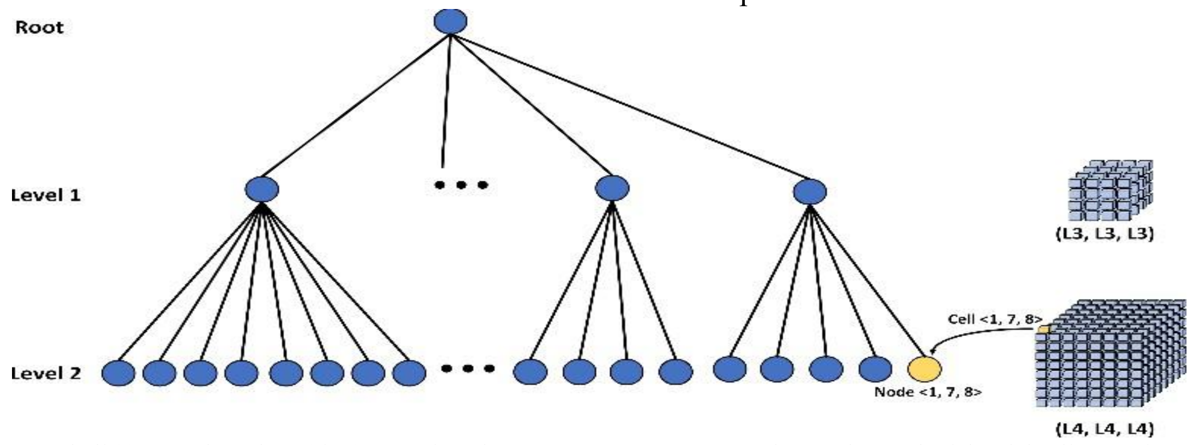


**Fig. 2. The base cuboid (*L*4, *L*4, *L*4)**

## 2. Proposed Multi-Resolution Tree Structure

The usage of essential tree operations, such as the search operation, is made possible by the storage of aggregated data in a tree structure. Compared to looking through the massive dataset that has not been processed, searching using the hierarchical tree structure produces results that are both more efficient and speedier. An example of a strategy that enables the calculation of features at higher levels in a tree structure based on the characteristics at lower levels is the Apriori approach. This technique eliminates the need to rescan the raw data, which is a common practice in the information technology field. There is also the possibility of using the tree structure for parallel and distributed aggregating methods.



Several different levels of information have been merged into the data that is stored in the tree that has been proposed. The leaves of the tree contain the data cubes with the lowest degree of aggregation (L4, L4, L4), and as we get closer to the base of the tree, the amount of aggregation grows. An expert in the field is responsible for determining the number of tree levels, denoted by the letter h.

All dimensions are concurrently subjected to aggregation when using the tree that has been proposed. In the prior example, which was presented in Section III, the tree has the potential to combine data only for the cuboids that are labeled (L1, L1, L1), (L2, L2, L2), (L3, L3, L3), and (L4, L4, L4). This is to understand the situation. Consequently, the data at each level of the tree is condensed to half of its size in two dimensions and then transmitted on to the next level (given that it is predicted that each dimension would be reduced to half of its size at the next level). It should be brought to your attention that the apex cuboid (L1, L1, L1) is not produced since it does not provide any mining benefits to merge all of the datasets into a single node. A tree structure that consolidates data into two layers is seen in Figure 3, which is represented by the symbol h equivalent to 2. It is important to note that the first level corresponds to the cuboid with dimensions (L3, L3, L3), whereas the second level represents the base cuboid with dimensions (L4, L4, L4).

Every node in the tree, with the exception of the root, represents a cell in the congruent cuboid that does not include any empty space. The statistical metrics that aggregate the data instances contained inside an intermediate node are often included within the node. When new data instances are introduced to the data collection and mapped to their appropriate parent and children nodes, the measurements have to be updated progressively after the addition of the new data instances.

**Fig. 3. Typical multi-resolution tree**

**with two levels of aggregation and root node. Each tree node corresponds to a nonempty cell in the cuboid**

As can be seen in Figure 4, each and every element in the map table has a pointer that identifies the proper node (location). Leaf nodes may have as little as 0 children or as many as 2d children, depending on the situation. There is just one item in the map table of a leaf node that links to its parent node. There are no entries for the leaf node's children in the map table. When it comes to statistical metrics, the root node is the sole node that is missing. In light of this, it does not point to any degree of consolidation. Specifically, it only includes the mapping table for the nodes that are at the very top of the tree. We are operating under the assumption that the root node can be easily uploaded into memory. One such assumption is that the tree, or at least a piece of it, can be loaded into memory in a short amount of time.

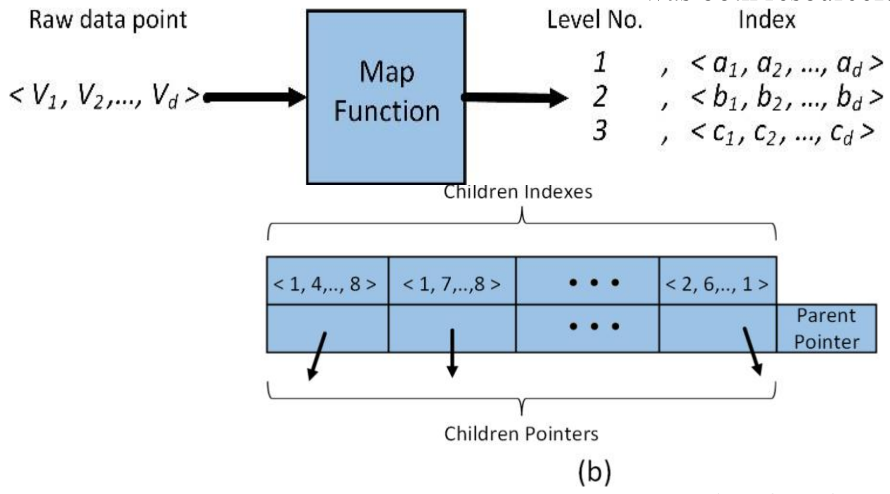


**Fig. 4. Typical intermediate node content. (a) Incremental statistical features (b) Map table.**

### *Multi-indexing and Mapping*

One might think of a tree node as being comparable to a data cube cell. A tree node is a tuple that constitutes attributes from dimensions. The index of this object is comprised of many values, which are organized in the pattern < a1,l, a2,l, …, ad,l >, where ap,l is the number of the area with dimension p at tree level l. Each node is given a name that is comprised of the level number of the tree in which it is situated, in addition to the index of the node. The objective of the node name is to facilitate access to and differentiation amongst nodes that are included inside the tree.

We made use of a mapping function in order to link each individual data instance with its associated nodes in the tree hierarchy. This allowed us to create the tree making use of a dataset in a way that was both resourceful and efficient. The mapping function generates a large number of keys, also known as indexes, for each data instance. Each key corresponds to a distinct level in the tree. For example, in the case that has been shown, where the tree has a root and three levels, and the raw data has d dimensions, the mapping function generates three keys, as can be seen in Figure 5.

**Fig. 5. Map function.**

It is very necessary to construct the map functions with a complexity of O(d) in order to ensure that the mapping method is not only quick but also effective. For this particular criterion to be satisfied, the map function ought to be constructed in a fashion that is not reliant on any information that calls for further calculation. When determining the index of a dimension based on Z-score, for example, it may be required to employ a map function that depends on the mean of the values. This is because the Z-score is used to determine the index. When dealing with a huge streaming dataset, calculating the average may be a difficult and troublesome task. It is our recommendation that the map function be designed using the information that is already there for the data collection. As an illustration, in the field of Human Activity Recognition (HAR), we are aware that some sensors, like the gyroscope, are capable of producing continuous actual data that falls between the range of -20 to 20 for example. Due to the fact that this method does not call for any extra processing resources, it is appropriate to construct the map function by taking into consideration the maximum and minimum distances. In order to generate a tree with several resolutions, we need the lowest value of p for dimension p, the number of tree levels h, and a matrix W. For each entry wp, l ϵ W in the matrix W, the width of dimension p at level l is represented respectively. Both the values of h and W are able to be determined by an expert in the subject who is experienced and informed. The data instance < v1, v2,..., vd > may be mapped using Equation (1) as the map function, as seen in the preceding example. This can be accomplished by using the map function.

## A. *Sufficient Statistical Measures*

One cannot simply calculate the average of these elements in order to combine several data instances into a single node in the multiresolution tree structure. This needs additional calculations. As part of the process of developing the multiresolution tree structure, we need to calculate statistical measures that are not only informative but also succinct, that are widely used, and that can be continually updated from the initial dataset. There are three metrics that are maintained by the BIRCH data clustering approach. These metrics are the total, the sum of squares, and the number of instances for each data cluster. We utilize the same metrics because of their incremental nature and the fact that we are able to update them directly for each node. This is because we are building upon the concepts that BIRCH has established. Consequently, this makes it possible to update the multi-resolution structure in a flexible manner, either from the top-down or bottom-up perspective. Additionally, we have supplied measurements for both the lowest and maximum values, which serve to illustrate the boundaries of the dimensional range that is connected with this node. The essential summary statistics that were generated for each node of the proposed framework are shown at the beginning of Table I. The total, the sum of squares, the lowest, and the maximum values are all computed for each dimension. This is something that should be taken into consideration. There is a possibility that the level number (l), region number (i), and dimension (p) will be provided by the node name.

**TABLE I. Summarization Feature**

| Feature | Calculation | Comments |
|---|---|---|
| $n$ | | The number of data instances in the node. |
| $S_p$ | $\sum \vec{x_p}$ | The sum of values of dimension $p$. |
| $SS_p$ | $\sum (\vec{x_p} \cdot \vec{x_p})$ | Square Sum of element-by-element values of dimension $p$. |
| $Max_{i,p,l}$ | $Min_p + (w_{p,l} * i)$ | Maximum value of dimension $p$ at region $i$ and tree level $l$. |
| $Min_{i,p,l}$ | $Max_{i,p,l} - w_{p,l}$ | minimum value of dimension $p$ at region $i$ and tree level $l$. |

**Measures**

A wide variety of essential statistical features that are often used in data mining are included in the measurements that are reported in Table I's table. In order to calculate basic statistical measures like the mean μ, variance σ2, and standard deviation σ in a step-by-step manner, these tools may be used. The correlation and distribution of variables are two examples of the many additional statistical computations that need these measures as an important component. These essential statistical values are shown in Table II. These values may be calculated from the measurements that are presented in Table I.

**TABLE II. Basic Statistical Measures**

| Statistic | Calculation | Comments |
|---|---|---|
| $\mu$ | $S_p/n$ | Mean |
| $\sigma^2$ | $\frac{SS_p}{n} - (S_p/n)^2$ | Variance |
| $\sigma$ | $\sqrt{\frac{SS_p}{n} - (S_p/n)^2}$ | Standard Deviation |

It is possible to calculate cluster means, sizes, and distances in a variety of modes using these metrics if they are adequate. The Euclidean, Manhattan, L1, and L2 distances that are determined between cluster centers are some examples. Additionally, the average distances between clusters are also included in this category. For the purposes of clustering algorithms and learning based on similarity, this information is quite useful.

By include class labels in the aggregated data, we would be able to calculate the probabilities associated with each class. The construction of classifiers, such as Bayesian classifiers and Hidden Markov Models (HMM), requires the use of class probabilities as an essential component. Additionally, they are used in the computation of entropy, gain ratio, and Gini index for the purpose of creating decision trees devoid of the need of accessing individual data items.

There are only a certain number of values or labels that may be assigned to the dimension of class labels, which is a discrete feature. There is the possibility of dividing the dimension into several regions, with each region reflecting a different class label. The class property does not go through any kind of aggregation and continues to be constant throughout the whole of the multi-resolution structure that is being suggested. As a result, each node is free of any pollution and has a condensed representation of data that is only relevant to a particular class. One scan of each level in the tree is all that is required in order to compute Equation (2), which is a representation of the class probability P(C) for a certain class label C. This may be accomplished by utilizing the tree structure that has been provided. In order to do this scan, the count measure n

### B. *Implementation and Performance Analysis*

C. The implementation of the proposed framework is carried out in a hierarchical fashion, beginning with the primary node and working its way down to the subordinate nodes. In order to conduct an analysis of the worst-case scenario for updating the structure whenever a new data instance is received, we assume the assumption that the data is spread uniformly throughout all regions and that there is no region that is devoid of data. Therefore, in the multi-resolution structure, each node at level h - 1 has a maximum of 2d offspring, which are the leaf nodes at level h. This applies

$$N_{l=} \left\lceil N_h/2^{(h-l)d} \right\rceil \quad (5)$$

to all of the nodes in the structure. We are able to calculate the total number of leaf nodes (Nh) by entering the height of the structure (h), the width matrix (W) of the region, as well as the highest and lowest values for each dimension (d).

$$N_h = \left\lceil \prod_{p=1}^{d} (max_p - min_p)/w_{ph} \right\rceil$$

Where $w_{p,h} \epsilon W$, is the regions width for

and the class attribute value C are used for each node that is located at, respectively.

Let's say that the set of nodes that are inside a certain level of the tree is denoted by the letter q. The total number of instances that are combined in node i is denoted by the parameter ni, whereas the number of examples that are combined in node i and contain the class label C is represented by the parameter ni,C.

dimension $p$ at tree level $h$. The values $max_p$ *and* $min_p$ are the maximum and minimum values of dimension $p$ respectively.

In case class labels $C$ is included in the structure, the number of nodes at level $h$ is:

$$N_h = \left\lceil \prod_{p=1}^{d-1} [\,(max_p - min_p)w_{ph}] \ * \ |C| \right\rceil \quad (4)$$

The $|C|$ is the cardinality of class labels dimension $C$. The cardinality equal to the number of distinct values of class labels.

Each level is aggregated to $2^{-d}$ of the size of its immediate lower level. Thus, the number of nodes in any intermediate level $l$ depends on leaf level is:

Then, the maximum total number of nodes in the multiresolution structure with one root node and $h$ levels is:

$$N_{Tree} = 1 + \left\lceil \sum_{l=1}^{h} (N_h/2^{(h-l)(d-1)}) \right\rceil \quad (6)$$

The space that is necessary to store the proposed multi-resolution tree structure is actually less than the space that is required to store the raw data. This is the case despite the fact that the size of a tree node is more than the size of an instance of raw data. This is owing to the fact that the number of occurrences in the raw data, which is represented by the letter N, is very high and continues to increase from time to time as a result of streaming data. On the other hand, the size of the tree structure that has been suggested, which is symbolized by the symbol NTree, is substantially lower than N.

The multi-resolution tree structure is constructed using Algorithm 1 by doing a single examination of the data that is being received. The map function is used in order to generate h index keys throughout the process of getting ready for an upcoming data instance. From what can be seen in Figure 5, each key corresponds to a different level in the tree. After that, the computer does an analysis on the root map table in order to locate an item that serves as a match for the first key. In the event that the entry is found, the algorithm will first update the node by invoking it with the appropriate pointer and then updating it.

uploaded to memory. These nodes are the root node and a single node from each level of the tree.

**Algorithm 1:** Build the Multi-resolution Tree

```
Input  : The raw dataset with $N$ data objects, $h$ number
         of levels in the output multi-resolution tree $T$,
         and $W$ matrix of regions width.
Output: Multi-resolution tree $T$.
procedure UPDATE($Index, CurrentNode$)
if $index$ not in $CurrentNode.MapTable$ then
    1: Create a new entry in $CurrentNode.MapTable$;
    2: Insert $index$ in the new entry;
    3: Create a new node $E$;
    4: Create a new entry in the $CurrentNode.children$,
       correspond to previously created entry;
    5: Make the new entry point to E;
    6: $E.parent \leftarrow CurrentNode$;
    7: $CurrentNode \leftarrow E$;
    8: Initialize $CurrentNode$ materialization and entries
else
    1: $CurrentNode \leftarrow$ Child pointed by
       $CurrentNode.children$ entry corresponding to the
       $index$ entry in the $CurrentNode.MapTable$;
    2: Update $CurrentNode$ materialization and entries;
end
end procedure
for $i \leftarrow 1$ to $N$ do
    for $l \leftarrow 1$ to $h$ do
        $index[l] \leftarrow$ generate index of data object $O_i$ for
        level $l$ using $W_l$;
    end
    if $T.root$ is $Null$ then
        Create a $root$ node ;
    else
        $CurrentNode = root$ ;
        UPDATE($Index, CurrentNode$);
    end
    for $l \leftarrow 2$ to $h - 1$ do
        UPDATE($Index, CurrentNode$);
    end
end
```

Algorithm 1 is more effective than a data cube because it eliminates the need to generate empty nodes. As a consequence, the tree structure that it generates is more compact than the one that a data cube generates. In addition to this, it avoids the need of searching through all of the nodes. Instead, it only calls (h + 1) nodes, which results in a penalty of O(h + 1) disk operations having to be performed.

Algorithm 1 is shown in Figure 6 by utilizing the same example that was illustrated in Figure 5. The path that the algorithm takes from the root node to the leaf nodes is shown by the dotted curve. Additionally, the only nodes that are visited are the red nodes, which includes the root node. The cost of searching the map tables of the nodes that have been visited is represented by the symbol $\sum h-1$ $2(h-l)$. One node is traversed at each level, beginning with level 1 and continuing all the way down to level h minus 1. In spite of this, we do not do a scan on the leaf nodes at level h since they do not produce any progeny. While using the Birch technique to update the CF tree, using Algorithm1 to update the recommended tree is more efficient than using the Birch approach to update the CF tree since it only requires a single pass over the tree. The top-down pass is used to determine which nodes are appropriate, and the bottom-up pass is used to update the nodes that have been determined to be suitable.

.

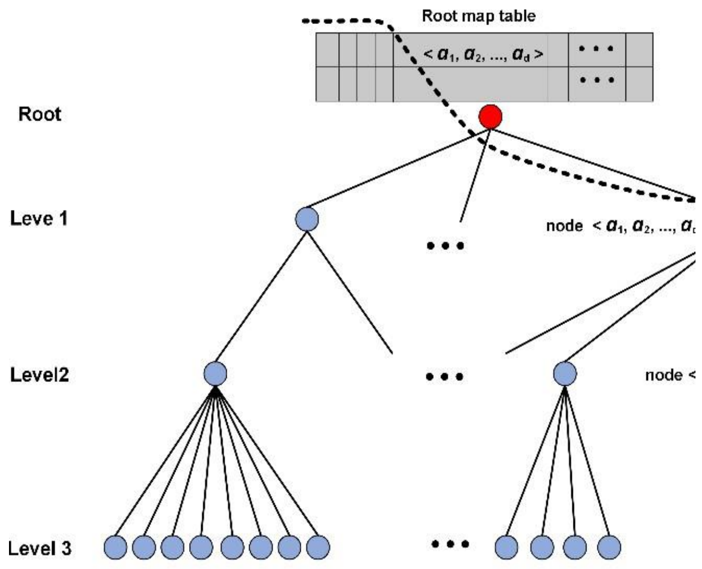


**Fig. 6. Updating the multi-resolution tree**

## Experimental Evaluation

Within this section, we carry out a multitude of tests on the framework that we have proposed in order to improve the effectiveness of the Naive Bayes classifier. The Naive Bayes classifier is a data mining approach that is both efficient and reliable. It is often used for classification tasks, such as the categorization of text and the identification of medical conditions. Naive Bayes is a technique that is used to estimate the class label Þ Eq. (7) for an instance x that has not yet been categorized. The Bayes theorem is used to compute the class probability P(C), and then the class label Ck is assigned to the class with the greatest posterior probability. This is how it accomplishes the intended purpose. Every instance in the training dataset is assigned to one of k distinct classes, while the dataset itself is composed of N examples that are located in a space with d dimensions.

If you want to determine the probability of continuous variables, you may do so by using the summary characteristics that are provided in Table II and then replacing them in the equation that came before it. In the next step, we use the Multi-resolution Naive Bayes (MRNB) technique on the tree topologies and compare it to the traditional Naïve Bayes method (NB) that is applied to datasets that have not been treated. The results, which are shown in Table III, illustrate that our MRNB technique consistently beats NB by a significant margin across all datasets. A time saving of around 25 percent is shown by MRNB in comparison to regular NB. In addition to this, it has the capability of operating at a higher level (level 1) of the tree, which results in a lower memory need. The number of instances in the raw dataset, which is marked by the letter N, is much more than the number of nodes that are present in this level, which is denoted by the letter N1. Additionally, it retains a degree of prediction accuracy that is comparable to that of normalization.

As seen in Equation (8), the Naive Bayes method makes the assumption that variables are independent of one another and use the normal distribution probability in order to calculate the class probability of variables that are normally distributed.

$$P(x_p|C_k) = \frac{1}{\sigma_{p,k}\sqrt{2\pi}} e^{\frac{-(x-\mu_{p,k})^2}{2\sigma_{p,k}^2}} \quad (8)$$

In order to do an analysis on raw data, the Naive Bayes classifier examines each and every occurrence of the data in order to compute the conditional mean, variance, and standard deviation. In order to calculate the probability for continuous variables, these values are required. The effectiveness of training huge datasets is hampered as a result of this.

A large number of datasets of varied sizes have been constructed for the purpose of this experiment. These datasets include 104, 105, 106, 107, 2 x 107, 22 x 107, 23 x 107, and 108 occurrences. There are two continuous predictor variables and one response variable included in each dataset to be considered. Both the predictor variables and the response variable have a normal distribution, with the response variable having two class labels: 0 and 1. A training set consisting of eighty percent and a testing set consisting of twenty percent have been randomly separated from the datasets. Using the parameter values h = 4 and W = {0.1, 0.2, 0.4, 0.8}, the training set is used in order to generate the multiple resolution tree structure.

In the beginning, we make use of the Naive Bayes classifier, which takes the multi-resolution tree that was provided as input in order to compute the

**TABLE III: Time in Seconds for MRNB and NB**

| Dataset | MRNB | NB |
|---|---|---|
| $10^4$ | 0.241 | 0.421 |
| $10^5$ | 1.758 | 2.685 |
| $10^6$ | 21.730 | 29.650 |
| $10^7$ | 235.618 | 296.726 |
| $2 \times 10^7$ | 489.188 | 608.424 |
| $2^2 \times 10^7$ | 927.986 | 1178.559 |
| $2^3 \times 10^7$ | 1744.030 | 2164.098 |
| $10^8$ | 2410.937 | 3010.914 |

## Conclusions and Future Works

We have developed a universal multi-resolution indexing tree structure that guarantees the scalability and reliability of summarizing both real-time and historical huge datasets. This structure was built thanks to our efforts. The construction of this structure is done just once, and then it is progressively updated. Its purpose is to reduce the amount of computational time and memory that mining and learning algorithms need. When it comes to creating, organizing, retrieving, and conserving a progressive update of the tree levels and contents, we have developed approaches that are both efficient and effective. A number of data mining and learning approaches would not be possible without the information that is provided by the properties of structural summarization. After using the Naive Bayes classifier to utilize the tree structure as input, we examined its performance on a variety of datasets to see how well it performed it. According to the results, the algorithm demonstrates a reduction in the amount of time and memory that it consumes while it is operating on the multi-resolution tree that was recommended, as compared to when it is working on the data that has not been processed.

The use of this framework for summarization, which makes use of data mining methods, is more efficient than directly retrieving material that has not been processed. A two-dimensional dataset, in which the data is organized in a matrix or CSV file format, is the only dataset that can be mined using traditional mining methods. Following the transformation of raw data into a tree structure with many resolutions, the data is no longer appropriate for use with traditional mining techniques. The implementation of further data mining algorithms that are capable of receiving a multi-resolution summarized tree structure as input will be the primary focus of our future work to be done. Once we have gathered the data of the tree structure, we will proceed to estimate the parameters of their models using those observations. Approximation similarity-based classifiers, Bayesian belief networks, classic tree classification, and online estimated Support Vector Machines (SVM) are some of the many ways that may be used.

## References

1) Bifet, "Mining Big Data in Real Time," Informatica 37, pp. 15–20, 2013.
2) C.-W. Tsai, C.-F. Lai, H.-C. Chao, and A. V. Vasilakos, "Big data analytics: a survey," Journal of Big Data, vol. 2, no. 1, Dec. 2015.
3) W. J. Frawley, "Knowledge Discovery in Databases: An Overview," Knowledge Discovery in Databases, 1991.
4) M. H. ur Rehman, C. S. Liew, A. Abbas, P. P. Jayaraman, T. Y. Wah, and
5) S. U. Khan, "Big Data Reduction Methods: A Survey," Data Science and Engineering, vol. 1, no. 4, pp. 265–284, Dec. 2016.
6) Feng Cai and V. Cherkassky, "Generalized SMO Algorithm for SVMBased Multitask Learning," IEEE Transactions on Neural Networks and Learning Systems, vol. 23, no. 6, pp. 997–1003, Jun. 2012.
7) J. A. R. Rojas, M. Beth Kery, S. Rosenthal, and A. Dey, "Sampling techniques to improve big data exploration," in 2017 IEEE 7th Symposium on Large Data Analysis and Visualization (LDAV). Phoenix, AZ: IEEE, Oct. 2017, pp. 26–35.
8) S. Garca, S. Ramrez-Gallego, J. Luengo, J. M. Bentez, and F. Herrera, "Big data preprocessing: methods and prospects," Big Data Analytics, vol. 1, no. 1, Dec. 2016.
9) S. Garca, J. Luengo, and F. Herrera, "Instance Selection," in

Data Preprocessing in Data Mining. Cham: Springer International Publishing, 2015, vol. 72, pp. 195–243.

10) Y. Fu, X. Zhu, and B. Li, “A survey on instance selection for active learning,” Knowledge and Information Systems, vol. 35, no. 2, pp. 249– 283, May 2013.

A. Cuzzocrea, “OLAP Data Cube Compression Techniques: A Ten-Year-Long History,” in Future Generation Information Technology, ser. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, Dec. 2010, pp. 751–754.

11) J. Han, “Towards on-line analytical mining in large databases,” ACM Sigmod Record, vol. 27, no. 1, pp. 97–107, 1998.

12) S. Alwajidi and L. Yang, “3d Parallel Coordinates for Multidimensional Data Cube Exploration,” in Proceedings of the 2018 International Conference on Computing and Big Data - ICCBD ’18. Charleston, SC, USA: ACM Press, 2018, pp. 23–27.

13) F. Heine and M. Rohde, “PopUp-Cubing: An Algorithm to Efficiently Use Iceberg Cubes in Data Streams,” in Proceedings of the Fourth