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

Safaa Alwajidi
Department of Computer Science
Western Michigan University
USA
safaakhalilmu.alwajidi@wmich.edu

Li Yang
Department of Computer Science
Western Michigan University
USA
li.yang@wmich.edu

***Abstract*—Big data analysis is essential for modern applications in areas such as healthcare, assistive technology, intelligent transportation, environment and climate monitoring. Traditional algorithms in data mining and machine learning do not scale well with data size. Mining and learning from big data need time and memory efficient techniques, albeit the cost of possible loss in accuracy. We have developed a data aggregation structure to summarize data with large number of instances and data generated from multiple data sources. Data are aggregated at multiple resolutions and resolution provides a trade-off between efficiency and accuracy. The structure is built once, updated incrementally, and serves as a common data input for multiple mining and learning algorithms. Data mining algorithms are modified to accept the aggregated data as input. Hierarchical data aggregation serves as a paradigm under which novel data representations and algorithms work together for analysis and mining of big data. To evaluate its performance, we have implemented a multi-resolution Naive Bayes Classifier on the data aggregation structure. Experimental results show that the proposed structure helps the classifier to reduce computation time to 25% on average and reduce the memory usage while preserving the accuracy of results.**

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

I. INTRODUCTION

The analysis of big data is essential for many applications such as health care, assistive technology, intelligent transportation, environment and climate change monitoring, where millions of data instances are gathered within a short period of time. However, traditional data mining algorithms have limits to scale up on big data with limited memory and CPU resources [1].

Data mining is an iterative and non-trivial process to extract implicit and previously unknown useful information from data [2] [3]. Many data mining algorithms require a scan of data multiple times. Therefore, accessing big data instance by instance for mining is a prohibitive process, which has motivated researchers to think about new techniques to speed up the analysis of big data. These techniques generally vary from parallel algorithms to utilize multi-core processors, compression algorithms to minimize the required space to store the data, dimensionality reduction algorithms to reduce the number of variables in the data, and data summarization techniques to reduce the number of data instances of the big data.

Data mining on summarized, reduced and relevant data is more efficient than working on raw, redundant, inconsistence and noisy data [4]. However, few empirical studies have been conducted to reduce the number of data instances by summarizing them and use the summarized data for data mining. Using summarized data for data mining has a few technical challenges.

The first challenge is to use the summarized data as common measures to serve many data mining algorithms. Determine the proper summarization measures depends on what parameters each data mining algorithm needs to build its model. For this purpose, the summarized data should provide minimal and yet sufficient information to each data mining algorithm.

The second challenge is to organize the summarization in multiple resolutions. multiple resolutions enable the user to choose the proper resolution and make a compromise between time, memory and accuracy depend on the available resources and the application requirement.

The third challenge is the efficient generation and maintenance of the summarized data. Although generated once, we want this to be fast enough and more importantly, can be updated incrementally.

In this paper, we propose a common multi-resolution tree structure. The overall objective of the proposed tree structure is to maintain scalable and reliable summarization of both on-the-fly and historical big data in a multi-resolution form for efficient data mining. The structure provides a representative and reduced set of big data to be used by mining and learning algorithms to implement their model faster and less memory usage with possible little loss in accuracy.

The rest of the paper is divided into five sections. The first section gives background and previous work. In the second section, we explain data cube lattice for big data. The third section introduces the proposed structure and gives its implementation and performance analysis. Experiments and results are discussed in the fourth section. The final section concludes this paper by giving areas of future work.

II. BACKGROUND AND PREVIOUS WORKS

Time complexity of many data mining and machine learning algorithms depends on the number of instances in the dataset. For example, Support Vector Machine (SVM) has a time complexity of $O(N^3)$ in its quadratic solution and its best complexity of $O(N^2)$ using sequential minimal optimization (SMO) technique [5], where N is the number of instances. Similarly, conventional decision tree such as CART has a time complexity of $O(NlogN)$ and similarity-based classifiers such as KNN has a time complexity of $O(Ndk)$. These algorithms and other mining algorithms which may require multiple scans of

data are computationally expensive when the number of instances is big [1]. To work on big data sets, many mining and learning algorithms need a preprocessing step to reduce the number of instances N albeit the cost of possible loss in accuracy.

A recent study [6] shows proof that random sampling is the only technique commonly used by data scientists to quickly gain insights from a big dataset. However, it is hard to find a representative random sample for non-stationary big dataset because the values of attributes are too distinctive and not evenly distributed.

Instance selection is one example of instance reduction techniques, that subsets the raw data set into a small group of instances. The subset techniques reduce the quality of data considerably and have a computational complexity of at least $O(NlogN)$ [7] [8]. That makes instance selection unsuitable for big data sets. Note that instance selection is different from sampling. Sampling is a random selection from the raw dataset while instance selection considers instance correlation to select most informative instances [9].

Data aggregation is another technique that is used to reduce the number of instances in which similar instances are combined into one instance to eliminate redundancy. More advanced methods for aggregation include multidimensional data cube, which holds aggregated data in subspaces to support advanced analysis and decision making. It is widely used in business intelligence [10]. Han [11] proposed. On-line Analytical Mining (OLAM) which integrates mining models on top of a multidimensional data cube and OLAP engine to gain efficiency and scalability for data mining. Multidimensional data cube suffers from its big size, which grows exponentially when the number of dimensions increases [12]. A few techniques have been proposed as solutions for the problem. For example, iceberg cube [13] ignores data cells with fewer data instances than a user-defined threshold. Most cells in the data cube are sparse cells that represent empty regions in multidimensional data space, these cells can be removed for efficient mining and learning process.

Data cells aggregated at different resolutions need to be linked for quick references. A specialized index structure is one of the major techniques that can be used as a solution for the problem [14] [15]. An index structure such as a tree has been used to overcome the inefficiency of the multidimensional data cube. Many variations of KD-tree [16], aR-tree [17], and R+-tree [18] have been used for this purpose. However, K-DTree based methods are known for their complicated creation and slow performance with high dimensional data. More importantly, it does not support incremental update. aR-tree deals with overlapped regions which add a computational overhead that we do not need since we deal with non-overlapped regions (as we will see in Section III), while R+-tree cannot deal with aggregated data.

In addition, CF tree that used in BIRCH method [19] [20] and R* tree which used in DBSCAN method [21] [22] are other types of index structure. However, these methods are dedicated only for spatial data type and are query dependent. Once we implement the tree depending on a specific query (data mining task), we cannot use it again for other queries. Therefore, these indexing techniques need to scan the raw data repeatedly [23].

III. DATA CUBE

In this paper, we consider a dataset of size N x d where N is the number of instances and d is the number of variables ($N >> 2^d$). These variables take continuous numeric type values.

Continuous data are common such as the data generated from sensors like temperature, air pressure, gyroscope, accelerometer, GPS, gas sensors and water sensors. Aggregating continuous data into a multi-resolution hierarchical structure is not a straightforward process because continuous data lacks concept hierarchy.

We propose using a hierarchical multilevel grid summarization approach to reduce the number of data instances. Similar method has been used in spatial data mining [23] [24] and robotic mapping [25]. In this method, each dimension in the data set is divided into a limited number of equal width and non-overlapping regions (bins). Each region stores the summarization information of the raw data instances falling inside it. We labeled each region by giving it a number to distinguish between them. The multilevel structure forms a different granularity of data aggregation, where multiple regions at a lower level are grouped to form one region in the next higher level. We assume aggregation of each level to half of the size of its immediate lower level. This method provides a concept hierarchy for continuous data.

The hierarchical multilevel grid on multiple dimensions can be assumed as a multidimensional data cube. Fig. 1 shows an example of a multidimensional data cube lattice generated from three dimensions, each has four levels of aggregation.

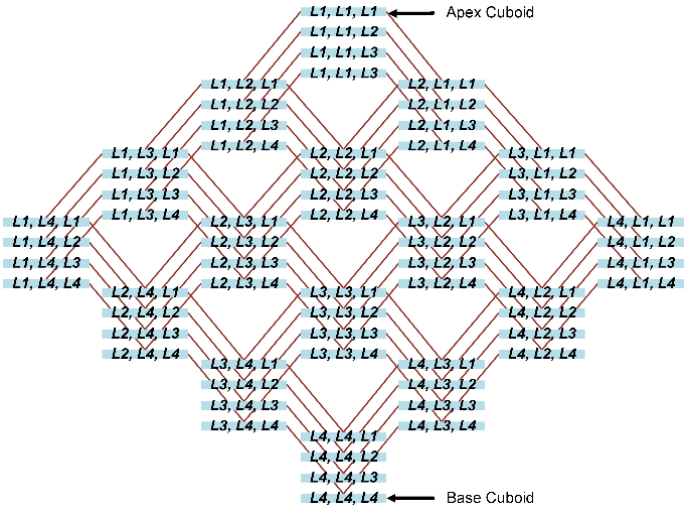

Fig. 1. The lattice of multidimensional data cube that generated from three dimensions.

Each node in the lattice is a cuboid that represents a single combination of aggregation levels (one level from each dimension). Starting from the bottom base cuboid (L4, L4, L4), which is a combination of lowest levels (levels 4) of dimensions 1, 2 and 3 respectively, toward the top cuboid/Apex cuboid (L1, L1, L1), which is a combination of highest levels (levels 1) of dimensions 1, 2 and 3 respectively. Thus, the data cube lattice aggregates data in all combinations of dimensions levels between these two cuboids.

Each cuboid contains a number of cells. Each cell in the data cube is defined and accessed by a multi-value index, which is a set of regions numbers (one region number from each dimension). For example, Fig. 2 shows a base cuboid of the cube lattice, we assume level 4 has 8 regions for each of the three dimensions, also it shows a cell indexed by < 1, 7, 8 > which is formed from regions 1,7 and 8 of dimensions 1, 2 and 3 respectively.

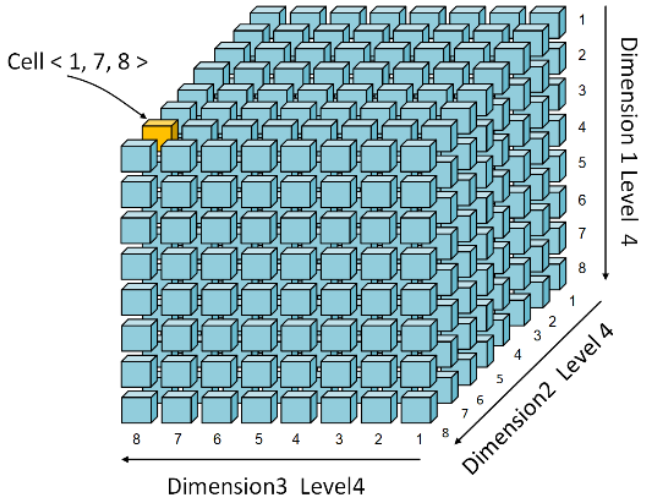

Fig. 2. The base cuboid (*L*4, *L*4, *L*4)

IV. Proposed Multi-Resolution Tree Structure

Storing aggregated data in a tree structure makes valuable tree operations such as search operation available. Searching the tree structure is faster and more efficient than searching the massive raw dataset. Apriori technique [26] is another example technique in which the features at high tree levels can be computed from features of low tree levels without the need to rescan the raw data. Parallel and distributed aggregation are also available using the tree structure.

The proposed tree holds aggregated data in multi-resolution levels. The lowest layer of data cube aggregation (L4, L4, L4) is stored in the tree leaves, and as we move toward the tree root the level of aggregation increase. The number of tree levels (denoted by h) is determined by the domain expert.

The aggregation in the proposed tree is carried out on all dimensions at the same time. In other words, for the previous example in Section III, the tree can aggregate data to only cuboids (L1, L1, L1), (L2, L2, L2), (L3, L3, L3) and (L4, L4, L4). Thus, the data in any level of the tree is aggregated to 2^{-d} of its size to the next upper level (because each dimension is assumed to be aggregated to half of its size to the next upper level). Note that, the apex cuboid (L1, L1, L1) is not generated as it does not benefit for mining purpose to aggregate all dataset in one node. Fig. 3 shows an example of a tree that aggregates data into two levels only, (h = 2), level two is the base cuboid (L4, L4, L4) and level one is the cuboid (L3, L3, L3).

Each node except the root in the tree represents a non-empty cell from the corresponding cuboid. The typical contents of an intermediate node include (I) statistical measures of aggregation of data instances in the node. These measures should be updated incrementally when new data instances join the data set (II) map table to its parent and children nodes.

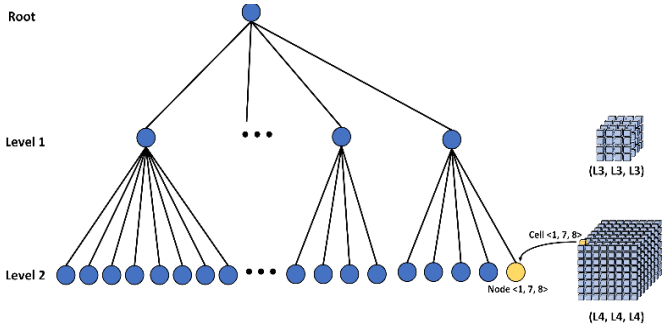

Fig. 3. Typical multi-resolution tree with two levels of aggregation and root node. Each tree node corresponds to a nonempty cell in the cuboid

Each entry in the map table has a pointer pointing to the corresponding node, as shown in Fig. 4. The minimum number of children is 0 for leaf nodes and the maximum number of children is 2^d. The map table of any leaf node has only a single entry that point to its parent node and no entries for children. The root node is the only node that does not have statistical measures. Thus, it does not represent any level of aggregation. It holds only the map table to the nodes of the highest level in the tree. We assume that the root node can be easily uploaded to memory (another assumption that the tree or part of it can be loaded easily to the memory).

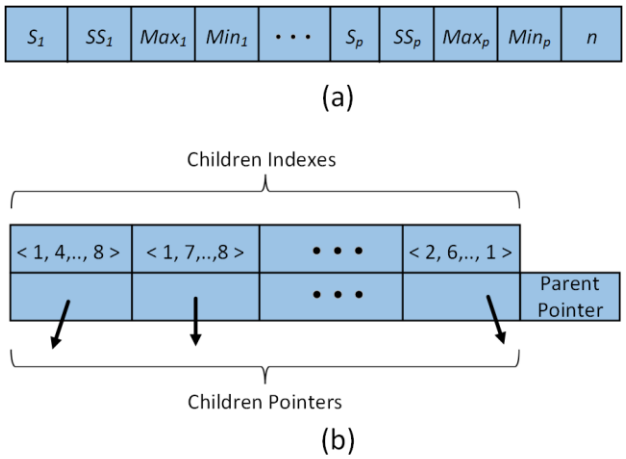

Fig. 4. Typical intermediate node content. (a) Incremental statistical features (b) Map table.

A. *Multi-indexing and Mapping*

A tree node, similar to a data cube cell, is a tuple over the attributes of dimensions, and its index has multiple values of the form < $a_{1,l}$, $a_{2,l}$, …, $a_{d,l}$ > where $a_{p,l}$ is a region number of dimension p at tree level l. We give each node a name which is the level number of the tree where the node resides, combined with its index. The node name is used to access and distinguish nodes of the tree.

To build the tree efficiently from a data set, we have used a mapping function to map each raw data instance to its corresponding nodes in the tree structure. For each data instance, the mapping function generates multiple keys

(indexes), one for each level in the tree. For example, if the tree has root and three levels, and the raw data has *d* dimensions, the mapping function generates three keys, as shown in Fig. 5.

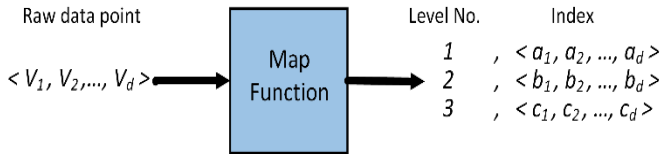

Fig. 5. Map function.

Designing the map functions of complexity $O(d)$ is crucial to ensure efficient and rapid mapping process. To satisfy this requirement the map function should be designed so that it does not depend on information that needs additional calculation. For example, when indexing a dimension depending on Z-score, this may need a map function that requires mean of the values. Getting the mean for streaming big data set is an overhead obstacle. We recommend design map function depending on information that is already available for the data set, for example, in HAR (Human Activity Recognition) we know previously some sensors like gyroscope could generate infinite real type data between -20 and 20 [27], so designing the map function depending on maxima and minima distance is appropriate and cost no overhead computation.

For multiple resolution tree implementation, we need to provide the minimum value min_p in the dimension p, the number of tree levels h and a matrix W where each element $w_{p,l} \in W$ is the regions width of dimension p at level l. Both h and W can be determined by a domain expert. Thus, for the above example the map function could map the data instance $< v_1, v_2, \ldots, v_d >$ using Eq. (1) as map function:

$$a_{p,l} = \frac{[v_p - min_p]}{w_{p,l}} \tag{1}$$

B. *Sufficient Statistical Measures*

Aggregating many data instances in one node in the multiresolution tree structure needs more than taking the average of these objects. We need to calculate statistical measures that are expressive enough, small in size, common and incrementally updated from raw dataset to materialize the multiresolution tree structure. BIRCH [19] is an example of a data clustering algorithm that keeps three measures sum, squares sum, and the number of instances for each cluster of data. Inspired by BIRCH, we use the same measures because they are incremental and can be updated directly for each node providing the ability of top-down or bottom-up update of the multi-resolution structure. In addition, we added minimum and maximum values measures that denote the boundary values of the dimension region corresponding to this node. Table I shows the basic summarization measures computed for each node of the proposed structure. Note that sum, squares sum, minimum and maximum are computed for each dimension. Level number l, region number i and dimension p can be obtained from node name.

TABLE I. SUMMARIZATION FEATURE MEASURES

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.

The measures in Table I can represent a wide spectrum of basic statistic features used for data mining. They can be used to calculate basic statistics like mean μ, variance σ^2, and standard deviation σ incrementally, which are necessary for many other statistical measures, such as variables correlation and distribution. Table II shows the basic statistics that can be computed from measures in Table I.

TABLE II. BASIC STATISTICAL MEASURES

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

These measures provide enough information to compute cluster means, sizes and distance in different modes. For instance, Euclidean, Manhattan, L1 and L2 distances between cluster centers and average inter-cluster distances. This information is useful for clustering algorithms and similarity-based learning.

If class labels are included in the aggregated data, this would allow us to calculate class probabilities. Class probabilities are essential in building a classifier such as Bayesian classifiers, Hidden Markov Model (HMM), and in defining Entropy, Gain ratio and Gini index for building decision trees without accessing individual data objects.

The dimension of class labels is a categorical attribute that has a finite number of values/labels. We can divide this dimension into regions where each region is a unique class label. There is no aggregation applied on the class attribute and it is maintaining the same regions at all levels of the proposed multi-resolution structure. Thus, each node is pure and has a summarization of data that belongs to only one class. To compute Eq. (2) the class probability $P(C)$, where C is a specific class label, from the proposed tree structure, we need only one scan of any level in the tree and use count measure n combined with class attribute value C for each node in the level.

$$P(C) = \sum_{i=1}^{q} n_{i,C} / \sum_{i=1}^{q} n_i \tag{2}$$

Where q is the set of nodes that belong to a specific level in the tree, n_i is the number of instances aggregated in node i while $n_{i,C}$ is the number of instances aggregated in node i that has class label C.

C. Implementation and Performance Analysis

The implementation of the proposed structure is a top-down process, starting from the root node toward leaf nodes. To analysis the worst performance case of updating the structure when a new data instance arrives, we assume the data is scattered over all the regions space and no region left without data. Thus, each node in level *h - 1* in the multi-resolution structure has the maximum 2^d children of leaf nodes (level *h*). Providing the structure height *h*, and region width matrix *W* along with maximum and minimum values for each of the *d* dimensions, we can calculate the number of leaf nodes N_h:

$$N_h = \left\lceil \prod_{p=1}^{d} (max_p - min_p)/w_{ph} \right\rceil \quad (3)$$

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

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

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

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

Although the size of a tree node is larger than the size of raw data instance, the space needed to store the proposed multi-resolution tree structure is less than the space of storing the raw data because $N_{Tree} << N$ and *N*, the number of instances in the raw data, is very big and increase over time due to streaming data.

Algorithm 1 builds the multi-resolution tree structure with a single scan of input data. For a coming data instance, it starts using the map function to generate *h* index keys, one key for each level in the tree such as the example in Fig. 5. The algorithm then examines the root map table for an entry that matches the first key. If the entry is found, the algorithm calls its node using the corresponding pointer and then update its measures. In case the entry is not found, the algorithm creates a new node and a new entry in root map table to hold its key (same as the first key generated by map function). The algorithm then makes that entry pointing to the newly created node and updates its measures. The algorithm repeats this process until it reaches the leaf node.

Fig. 6 explains Algorithm 1 using the same example from Fig. 5, the dotted curve represents the algorithm progress path from the root to leaf nods and only the red nodes are visited and 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 efficient than a data cube because it avoids creating empty nodes intuitively, and thus it generates a smaller tree structure than a data cube. Also, it avoids search all nodes. Instead it calls only (*h* + 1) nodes and cost *O* (*h* + *1*) disk operation. The worst case of this process cost $O\ (N_h/2^d + \sum_{l=1}^{h-1} 2^{(h-l)d})$ computation to update the tree due to arrive one data instance, where $N_h/2^d$ is the cost to search map table of the root node, and $\sum_{l=1}^{h-1} 2^{(h-l)d}$ is the cost to search the map tables of visited nodes. We visit one node at each level (levels 1 to *h* - 1), the level *h* or leaf nodes will not be searched as they do not have children. Updating proposed tree using Algorithm1 is more efficient than updating CF tree in Birch algorithm which requires two passes of the tree. Top-Down pass to find the appropriate nodes and Bottom-Up pass to update these nodes.

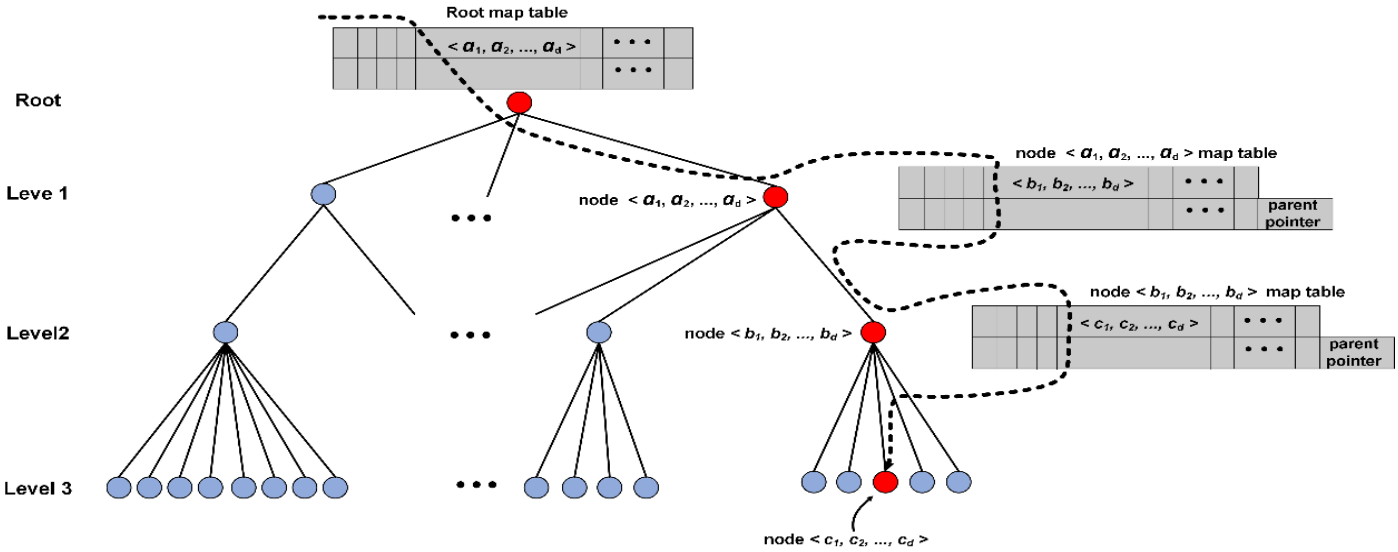

Fig. 6. Updating the multi-resolution tree

V. Experimental Evaluation

In this section, we run several experiments on our proposed structure to speedup Naive Bayes classifier. Naive Bayes classifier is a simple and powerful data mining algorithm that is used in classification problems such as text classification and medical diagnosis. Given a training dataset of N instances in d *dimensional* space and each instance belongs to one of k different classes, Naive Bayes finds estimated class label $\hat{y}$ Eq. (7) based on the Bayes theorem that used to compute class probability $P(C)$ and assigns the class label C_k with the highest posterior probability to unclassified instance x.

$$\hat{y} = \underset{k\in\{1,\ldots,K\}}{\operatorname{argmax}} P(C_k)\prod_{p=1}^{d} P(x_p|C_k) \tag{7}$$

Naive Bayes assumes that the variables are independent and uses the normal distributed probability to calculate Eq. (8) the class probability of normally distributed variables.

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

Naive Bayes classifier, working on raw data, scans all data instances to compute the conditional mean, variance and standard deviation that are required to compute the likelihood for continuous variables. This slows the training process for big dataset.

In this experiment, we have generated several datasets of different sizes 10^4, 10^5, 10^6, 10^7, 2 x 10^7, 2^2 x 10^7, 2^3 x 10^7 and 10^8 instances. Each dataset serves as a raw dataset and has two normally distributed continuous predictor variables and one response variable with two class labels 0 and 1. Each dataset has been split randomly into 80% training and 20% testing sets. We use the training set to implement the multiple resolution tree structure based on the following parameters values h = 4 and W = {0.1, 0.2, 0.4, 0.8}.

First, we implement Naive Bayes classifier that uses the proposed multi-resolution tree as input to compute the conditional mean, variance and standard deviation from summarization features as in Table II and substitute them in the previous equation to get the probability of continuous variables.

We then apply Multi-resolution Naive Bayes (MRNB) on the tree structures and compare it with the traditional Naïve Bayes algorithm (NB) applied on raw datasets. The results are given in Table III and show that our MRNB algorithm can achieve overall higher performance than NB in all datasets. MRNB consumes on average 25% less time than traditional NB and less memory since it can work on the higher level (level 1) of the tree which consists of a number of nodes N_1 less than the number of instances in the raw dataset $N_1 << N$. More importantly, it maintains the same accuracy of prediction as NB.

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
2×10^7	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

VI. Conclusions and Future Works

We have developed a common multi-resolution indexing tree representation that maintains scalable and reliable summarization of both on-the-fly and historical big data. This structure is built once, updated incrementally, and used to reduce computation time and memory usage for mining and learning algorithms. We have developed effective techniques to create, organize, access and maintain an incremental update of the tree layers and contents. The structure summarization features provide the necessary information for many data mining and learning algorithms. We have implemented the Naive Bayes classifier to use the tree structure as input, and then tested it on several datasets. The results show that the algorithm consumes less time and memory when using the proposed multi-resolution tree than working on the raw data.

Using this summarization structure by mining algorithm is more efficient than accessing raw data. However, traditional mining algorithms limited to work on a flat dataset, in which the data is organized in a matrix or CSV like file format. The change in the structure of raw data after aggregation into the multi-resolution tree structure makes it unavailable for traditional mining algorithms. Our future work will be setup to implement more data mining algorithms to accept multi-resolution summarized tree structure as input and to use its summarized measures to estimate parameters of their models. Candidate methods include approximate similarity-based classifiers, Bayesian belief networks, conventional tree classification and online approximated Support Vector Machines (SVM).

REFERENCES

[1] A. 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 S. U. Khan, "Big Data Reduction Methods: A Survey," Data Science and Engineering, vol. 1, no. 4, pp. 265–284, Dec. 2016.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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 IEEE/ACM International Conference on Big Data Computing, Applications and Technologies - BDCAT '17. Austin, Texas, USA: ACM Press, 2017, pp. 11–20.

[14] Y. Tao, D. Papadias, and J. Zhang, "Aggregate Processing of Planar Points," in Advances in Database Technology EDBT 2002, G. Goos, J. Hartmanis, J. van Leeuwen, C. S. Jensen, S. altenis, K. G. Jeffery, J. Pokorny, E. Bertino, K. Bhn, and M. Jarke, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, vol. 2287, pp. 682–700.

[15] W. F. Pan, D. R. Y. Cui, and Q. D. W. Perrizo, "Efficient OLAP Operations for Spatial Data Using Peano Trees," in Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery, San Diego, California, 2003, pp. 28–34.

[16] Y. Chen, L. Zhou, Y. Tang, J. P. Singh, N. Bouguila, C. Wang, H. Wang, and J. Du, "Fast neighbor search by using revised k-d tree," Information Sciences, vol. 472, pp. 145–162, Jan. 2019.

[17] Y. Hong, Q. Tang, X. Gao, B. Yao, G. Chen, and S. Tang, "Efficient RTree Based Indexing Scheme for Server-Centric Cloud Storage System," IEEE Transactions on Knowledge and Data Engineering, vol. 28, no. 6, pp. 1503–1517, Jun. 2016.

[18] L.-Y. Wei, Y.-T. Hsu, W.-C. Peng, and W.-C. Lee, "Indexing spatial data in cloud data managements," Pervasive and Mobile Computing, vol. 15, pp. 48–61, Dec. 2014.

[19] T. Zhang, R. Ramakrishnan, and M. Livny, "BIRCH: an efficient data clustering method for very large databases," in ACM Sigmod Record, vol. 25. ACM, 1996, pp. 103–114.

[20] L. Kovacs and L. Bednarik, "Parameter optimization for BIRCH preclustering algorithm," in 2011 IEEE 12th International Symposium on Computational Intelligence and Informatics (CINTI). Budapest, Hungary: IEEE, Nov. 2011, pp. 475–480.

[21] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise," in Proceedings 2nd International Conference on Knowledge Discovery and Data Mining, 1996, pp. 226–231.

[22] K. Mahesh Kumar and A. Rama Mohan Reddy, "A fast DBSCAN clustering algorithm by accelerating neighbor searching using Groups method," Pattern Recognition, vol. 58, pp. 39–48, Oct. 2016.

[23] W. Wang, J. Yang, and R. Muntz, "STING: A Statistical Information Grid Approach to Spatial Data Mining," in Proceedings of the 23rd VLDB Conference, 1997, p. 10.

[24] M. Huang and F. Bian, "A Grid and Density Based Fast Spatial Clustering Algorithm," in 2009 International Conference on Artificial Intelligence and Computational Intelligence. Shanghai, China: IEEE, 2009, pp. 260–263.

[25] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: an efficient probabilistic 3d mapping framework based on octrees," Autonomous Robots, vol. 34, no. 3, pp. 189–206, Apr. 2013.

[26] C. Aflori and M. Craus, "Grid implementation of the Apriori algorithm,"Advances in Engineering Software, vol. 38, no. 5, pp. 295–300, May 2007.

[27] J. W. Lockhart, G. M. Weiss, J. C. Xue, S. T. Gallagher, A. B. Grosner, and T. T. Pulickal, "Design considerations for the WISDM smart phone-based sensor mining architecture," in Proceedings of the Fifth International Workshop on Knowledge Discovery from Sensor Data. ACM, 2011, pp. 25–33.