**SCALABLE IMPLEMENTATION OF DISTRIBUTED DECISION TREE**

**By**

**K. MONICA BHARGAVI 411635**

**V. L. P. GAYATRI 411681**

**Ch. BHARGAVI 411614**

***Under the esteemed guidance of***

**Dr. K Hima Bindu**

****

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**NATIONAL INSTITUTE OF TECHNOLOGY ANDHRA PRADESH**

**TADEPALLIGUDEM-534102, INDIA**

**May – 2020**

Page i

Page ii

**SCALABLE IMPLEMENTATION OF DISTRIBUTED DECISION TREE**

**By**

**K. MONICA BHARGAVI 411635**

**V. L. P. GAYATRI 411681**

**Ch. BHARGAVI 411614**

***Under the esteemed guidance of***

**Dr. K HimaBindu**

****

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**NATIONAL INSTITUTE OF TECHNOLOGY ANDHRA PRADESH**

**TADEPALLIGUDEM-534102, INDIA**

**May – 2020**

Page iii

**DECLARATION**

We declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Page iv

**CERTIFICATE**

It is certified that the work contained in the thesis titled “**SCALABLE IMPLEMENTATION OF DISTRIBUTED DECISION TREE**”, done by K.Monica Bhargavi, V.L.P Gayatri, Chinthapatla Bhargavi bearing roll no. 411635, 411681 and 411614 respectively has been carried out under my/our supervision and that this work has not been submitted elsewhere for a degree.

**Dr. K. Hima Bindu**

**Dept. of Computer Science and Engineering**

**N.I.T. Andhra Pradesh**

**May, 2020**

Page v

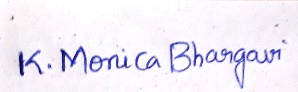
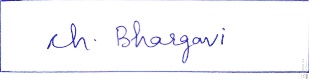
**ACKNOWLEDGMENTS**

We take this opportunity to express our profound gratitude and deep regards to our supervisor Dr. K.HimaBindu, Assistant professor, Department of Computer Science and Engineering, NIT Andhra Pradesh for her exemplary guidance, monitoring and constant encouragement throughout the project.

We avail ourselves of this proud privilege to express our regard to all the faculty of the Department of Computer Science and Engineering at NIT Andhra Pradesh for empathizing and providing all the necessary facilities throughout the work.

We offer our sincere thanks to all other persons who knowingly or unknowingly helped me complete this project.

K. Monica Bhargavi V. L.P. Gayatri Ch.Bhargavi

Roll No: 411635 Roll No: 411681 Roll No: 411614   

(Signature) (Signature) (Signature)

Page vi

**LIST OF TABLES**

Table Page

1. Comparative analysis of decision tree algorithms…………………………………..08

**LIST OF FIGURES**

Figure Page

1. Flow chart of proposed approach……………………………………………………12
2. Multiprocessing Pool………………………………………………………………...16
3. Comparison of time taken for serial and parallel execution for different datasets……………………………………….............................................................22
4. Comparison of accuracy of different datasets for serial and parallel execution……………………………………………………………………………..23

Page vii

**NOTATIONS**

**Name Meaning**

Chefboost framework Light weight decision tree framework

.json Java script object notation

JSON Human readable serialization format

pickle binary serialization format

Ak  kth attribute

Page viii

**ABSTRACT**

Data mining is a bustling area of research in this contemporary world. Innumerable data generated every day is comprehended and analyzed by using data mining to discern inherently useful and understandable patterns in data. Data mining comprises of numerous techniques to mine copious amount of data However, among all classification is used continually. Classification is the most prevailing problem in machine learning having vast applications in statistics, decision making and many other fields. Among them, decision trees are one of the most powerful and popular tools used for classification and regression problems in supervised learning as they are easy to elucidate. Decision trees are used to solve variety of implications in data mining process and used in various areas like statistics, machine learning.

The contemporary world has a thriving need of building classifiers from myriads of data. However ignoring the typical demon i.e. the curse of dimensionality is unavoidable because high dimensional data is pretty prevalent in science domain. This problem can be alleviated through parallelism. One of the striking reasons to employ parallelism is it handles the growing amount of work by utilizing more number of resources.

Several approaches had been proposed by many researchers for the construction of decision tree overcoming different constraints and problems such as memory restrictions, high-dimensional data, over-partitioning using different parallelization techniques like multithreading, instruction level parallelism. Among them multiprocessing would be a cut above choice because of its advantages such as increased throughput, cost saving, attaining speed up by taking the advantage of idle cores. Hence, a C4.5 decision tree using multiprocessing was implemented in the proposed approach to observe the performance and time comparisons of the models built in serial and parallel approach with increase in scalability of datasets.

The main feature of the proposed approach is to take advantage of the idle processors in the multiprocessing pool to perform computationally expensive CPU tasks like building the decision tree nodes with innumerable amount of data. For this purpose, idle processors in the pool were assigned to each newly created node in the tree structure. As a result, the load was distributed among all the processors in the pool instead of the single processor being burdened with heavy load.

Page ix

Another significant feature of the proposed approach is to observe how multiprocessing parallelism can handle increase in scalability and give better performance. For this purpose, three datasets of different sizes were taken for experimentation to draw inferences. The observations were done by proliferating the size of datasets gradually to see whether the increase in scalability was handled by the processors available in the pool without deteriorating the performance of the models built.

Accuracy is used as the metric to measure the performance of the models built using three different datasets. Furthermore, time comparisons were made by building the models in serial and parallel manner for the three datasets.

The results obtained have shown that with increase in scalability, performances have not been deteriorated or compromised in parallel mode. Better performance is retained due to multiprocessing. Our proposed approach maintained a tradeoff between performance and time taken to build the models with increase in scalability

Page x

**TABLE OF CONTENTS**

Content Page

1. TITLE……………………………………………………………………………… …i
2. DECLARATION……………………………………………………………………..iv
3. CERTIFICATE………………………………………………………………………..v
4. ACKNOWLEDGEMENTS…………………………………………………………..vi
5. LIST OF TABLES……………………………………………………………………vii
6. LIST OF FIGURES…………………………………………………………………...vii
7. NOTATIONS…………………………………………………………………………viii
8. ABSTRACT…………………………………………………………………………...ix
9. TABLE OF CONTENTS……………………………………………………………...xi

Page xi

**CONTENTS**

1. INTRODUCTION…………………………………………………………………….1
2. LITERATURE REVIEW……………………………………………………………..3

2.1 Classical C4.5 decision tree algorithm……………………………………………3

2.2 SPRINT…………………………………………………………………………....3

2.3 SLIQ………………………………………………………………………………4

2.4 PLANET…………………………………………………………………………..4

2.5 Parallel C4.5 implementation based on map reduce……………………………....5

2.6 XGBoost…………………………………………………………………………..6

2.7 Yggdrasil……………………………………………………………………….…7

1. PROPOSED APPROACH
   1. Problem Statement……………………………………………………………11
   2. Classification Algorithm
      1. DecisionTree……………………………………………………..13
      2. C4.5 DecisionTree………………………………………….........13
      3. Parallel C4.5 DecisionTree…………………………………………..................13
      4. Preprocessing………………………………………………...…..14
      5. Multiprocessing vs Multithreading…………………………........................................14
      6. Training……………………………………………………...…...15
      7. Prediction……………………………………………………...…17
      8. Performance and time evaluation with increase in scalability……………………………………………………...…17
2. EXPERIMENTAL PROCEDURE
   1. Dataset Description…………………………………………………………..…18
   2. Multiprocessing……………………………………………………………..…..20
   3. Training Model……………………………………………………………........20
   4. Implementation Details………………………………………………….............21
   5. Testing…………………………………………………………………………...21

Page xii

1. RESULTS AND DESCRIPTION……………………………………………….......22
2. CONCLUSION AND FUTURE SCOPE……………………………………………24
3. REFERENCES………………………………………………………………………25
4. APPENDIX……………………………………………………………………….....26

Page xiii

**CHAPTER 1**

**INTRODUCTION**

It is estimated that petabytes of data is generated every day. So, extraction of knowledge from this myriads of data is indispensable for applications in diverse domains to comprehend useful patterns in data which in turn enables in proliferating the profits of businesses. Among all the mining techniques, classification is used continually.

Decision trees are commonly used in operations research which is a key discipline in the management of organizations to make better decisions through mathematical analysis. Furthermore, decision trees provide well-structured delivery of resolutions and augment the quality of customer services. In addition, they are prevalent in health care systems, fraud detection in insurance and bank sectors and predict risk scoring in financial services.

So, decision trees are predominantly used supervised learning methods that are pliable at solving both classification and regression problems. They are easy to elucidate i.e., visually illustrate a documented thought process and do not require any feature scaling. Apart from predictive modelling, decision trees are used in data exploration stage of organization projects to ease the understanding of different variables and discern the non-trivial variables.

In spite of their simplicity, there are several impediments for constructing decision tree classifiers due to high dimensionality, over partitioning, memory restrictions, time complexity and data complexity of the data. Consequentially, it is not feasible to implement the decision tree algorithm in a serial manner as it is highly time consuming because tree construction is computationally expensive when there is innumerable data. So, it necessitated the construction of parallel decision tree. For this purpose, multiprocessing was used to build classifier for the parallel decision tree algorithm because of its various advantages.

Furthermore, multiprocessing proliferates reliability because a single processor failure will not disturb other processors in the machine and the work of the processor which had debacle will be distributed among alive processors. Not having shared memory unlike threads is also one of the reasons for increase in reliability in multiprocessing.

There are mainly three challenges to be addressed for building a C4.5 decision tree using multiprocessing. One of them is to employ branch level parallelism by taking advantage of the idle processors present in the multiprocessing pool so that the load of building the nodes of decision tree will not be fallen on a single node. The second challenge is performance of the model built should not be deteriorated while proliferating the scalability of datasets. The third challenge is a tradeoff has to be maintained between performance and time taken to build the model because consuming interminable time to build the model is not feasible to use in various applications or very low performance is also not acceptable for applications.

Page 1

To address the first challenge, decision tree nodes need to be distributed among all the available processors in the multiprocessing pool. For the second and third challenges, computationally expensive task i.e., each newly formed decision tree node has to be given to the idle processors in the pool and the decision tree should handle all the missing values. Furthermore, tree termination conditions should be followed to build a transparent model to increase performance.

The aim of the proposed approach is to achieve a tradeoff between performance and time consumed to build the models with the increase in scalability. Furthermore, the performance should not be deteriorated with increase in scalability of datasets

Page 2

**CHAPTER 2**

**LITERATURE REVIEW**

Decision tree algorithms are one of the predominantly used algorithms in classification as well as regression problems. Many decision tree algorithms have been proposed by various researchers to easily comprehend and interpret the useful patterns in different application domains in health care centers, customer services, banks and insurance sectors.

**2.1 Classical C4.5 decision tree algorithm**

Classical C4.5 decision tree is a greedy approached learning decision tree algorithm. This algorithm selects the best splitting attribute recursively using information gain ratio as attribute measure and follows top down induction down the tree.C4.5 algorithm is feasible when data set and decision tree are small and fit into memory. It is also viable when mixed attributes are present.C4.5 builds a decision tree from a set of training data similar to ID3 using the concept of information entropy. Training data is a set of pre-classified samples. Each sample has a p-dimensional vector, where the properties of the sample represent values or attributes, as well as the class in which they fall. C4.5 chooses an optimal attribute which divides its set of samples into subsets enriched in one class or another at each node of the tree. The bifurcation criterion is the generalized information gain (difference in entropy).The attribute with the highest generalized information gain is chosen for decision making. The C4.5 algorithm then iterates over the small sublist.

**2.2 SPRINT**

When decision tree does not fit in the memory two disk based approaches namely SPRINT and SLIQ are used to facilitate tree construction. Both use new data structures and can handle categorical and continuous valued attributes. SPRINT and SLIQ remove all the memory restrictions and enables to build a decision tree for large data.

SPRINT is fast and scalable parallel classifier for data mining which makes use of an attribute list data structure that holds the class and RID information. The attribute lists are partitioned and distributed among the resulting child nodes according to the split node. The order of the records in the list is maintained when a list is partitioned. Hence, partitioning lists does not require resorting. SPRINT was designed to be easily parallelized as well as scalable. It also requires the use of a hash tree proportional in size to the training set which may become expensive as the training set size grows.

Page 3

**2.3 SLIQ**

SLIQ is supervised learning in quest which employs disk-resident attribute lists and a single memory-resident class list. Each tuple is depicted by an association of one entry from each attribute list to an entry in the class list, which in turn is associated to its corresponding leaf node in the decision tree. As the class list is often accessed and modified in the building and pruning phases, it remains in memory. The size of the class list grows directly proportionally to the number of tuples in the training set. The performance of SLIQ decreases when a class list cannot fit into memory.

**2.4 PLANET**

The above mentioned pitfalls extends the learning to a scalable distributed framework over large datasets for learning tree models. PLANET is parallel learner for assembling numerous ensemble trees. It is a learner for training decision trees built on map reduce. PLANET is feasible when the data set is large and tree fits into memory. PLANET scales to very large datasets and supports ensemble techniques like bagging and boosting. Master is the main system component of PLANET which controls the entire process and determines the state of the tree and grows it. If there is relatively little data entering a node, master launches an In Memory Map Reduce job to grow the entire subtree.

For larger nodes, master launches a Map Reduce job to find candidate best splits. Master collects results from Map Reduce jobs and chooses the best split for a node and then it updates the model. Another component namely initialization map reduce component identifies all the attribute values which need to be considered for splits. If the attribute is continuous then it computes an approximate equi-depth histogram and the boundary points of histogram are used for potential splits. If the attribute is categorical then it identifies attribute’s domain.

This component also generates an “Attribute file” to be loaded in memory by other tasks. Find Best Split Map Reduce component’s job is to find the best split when there is too much data to fit in memory. Initially, mapper initializes by loading attribute file from initialization task and current model file. For each record run the map algorithm. Consider a number of possible splits (Xi, v) on its subset of the data. For each split, it stores partial statistics. Partial statistics means mapper computes quality of split based on attribute and attribute value. These partial statistics are sent to reducers. Finally, reducer collects all the partial statistics and determines best splits. Map Reduce In Memory Build component grows the entire subtree once the data fits in memory. Mapper initializes by loading current model file. For each record, identify whether the node is to be grown. If yes then it outputs <Node Id, Record>.Reducer then initializes by loading attribute file from initialization task. For each <Node Id, Record> run basic decision tree algorithm on records. Finally, reducer outputs the best splits for each node in the subtree.

Page 4

To leverage the performance of the model, ensemble techniques like bagging and boosting are used. Bagging constructs multiple trees in parallel, each on a sample of the data and obtains results by combining output from each tree. Boosting constructs multiple trees in series, each on a sample of the data and procures results by combining output from each tree.

However, PLANET is not suitable for high dimensional data. If number of splits proliferate then mapper runs out of memory. Per map reduce overhead is also highly pronounced for deep trees.

**2.5 Parallel C4.5 implementation based on map reduce**

Due to flaws in PLANET, learning process should be extended to parallel C4.5 implementation based on map reduce which comprises of two parallelized methods to build the tree nodes.

MR-C4.5 tree approach handles Memory restrictions, time complexity, data complexity and over- partitioning problem.MR-C4.5 tree approach comprises of the following parts:

* Attribute Selection
* Dataset Splitting
* MR-C4.5 tree construction

**Attribute Selection:**

Attribute selection algorithm is based on MapReduce framework of Hadoop which is named as MR-A-S algorithm. Initially the training data X with n attributes { Ak }k=1n is divided into m subsets such that each subset has N instances. In mapper phase for each subset,

For each attribute, Ak

if Ak is numerical then

all its values are sorted and all possible cut points are calculated.

if Ak is nominal then

each cut point will be each distinct value of attribute Ak

The information gain for each cut point is calculated and the cut point with maximum information gain is selected as the optimal cut point. The split information of the optimal cut point is enumerated. The obtained values help in enumerating the gain ratio of Ak.In the reducer phase, the sum of information gain ratios of an attribute from all the subsets is calculated and the attribute with maximum information gain ratio is selected as best splitting attribute.

**Dataset Splitting:**

Once the best splitting attribute and the cut points cpk are conformed the next work is splitting the dataset into several subsets. The data splitting algorithm (MR-D-S) does this job using map reduce framework. In the mapper phase at each worker an id value is assigned to the best splitting attribute value of each instance.

Page 5

This id value helps in assigning each instance to its corresponding child node while constructing tree. In the reducer phase the over partitioning problem is alleviated by setting some conditions such as minimum number of samples, maximum class probability and depth of constructed decision tree. When one of these termination conditions are met the construction of tree is stopped.

**MR-C4.5 tree construction:**

Based on the previous reckonings of the attribute selection and dataset splitting MR C4.5 tree is constructed. For each tree node building we search the optimal splitting attribute by MR-A-S algorithm and split the node into several child nodes by MR data splitting algorithm. When one of the termination conditions are met further construction of child nodes is inhibited.

**2.6 XGBoost**

XGBoost is a scalable tree boosting system. It is a greedy and sparsity aware tree boosting system. Tree boosting is highly effective and pervasively used machine learning method. Boosting is a mechanism of transforming weak learners into strong learners. In boosting, each new tree is fit on a modified version of an original dataset. XGBoost is viable when the data set is large and tree fits in memory. XGBoost scales billions of examples using minimal amount of resources and it also solves the curse of high dimensionality where tree boosting does not reply on any distance metric. However, XGBoost only works with numeric features.

**Split finding algorithms in XGBoost:**

**Basic exact greedy algorithm:**

One of the key problems in tree learning is to find the best split. In order to find the best split efficiently the algorithm must first sort the data according to feature values and visit the data in sorted order to accumulate the gradient statistics for the structure score. This algorithm is very powerful however it is impossible to efficiently find the best split when the data does not fit entirely into memory.

**Approximate algorithm:**

This algorithm initially proposes candidate splitting points based on the percentiles of feature distribution. Later, the algorithm maps the continuous features into buckets split by using these candidate points, aggregates the statistics and finds the optimal solutions among proposals according to the aggregated statistics. Depending on when the proposal is given, there are two variants of the algorithm. During the initial phase of tree construction, the global variant proposes all the candidate splits and utilizes the same proposals for split finding at all levels. The local variant re-proposes after each split. The global method utilizes less proposal steps than the local method. The local proposal filters the candidates after splits. Also, local proposal can potentially be more appropriate for deeper trees.

Page 6

**Weighted quantile sketch:**

The prime step in the approximate algorithm is to propose candidate split points. Usually percentiles of a feature are used to make candidates distribute evenly on the data. The general idea is to propose a data structure that supports merge and prune operations, with each operation to be maintained a certain accuracy level.

**Sparsity aware split finding:**

Presence of missing values in the data or persistent zero entries in the statistics are various reasons for scarcity. The instance is classified into the default direction, when a value is missing

in the sparse matrix X. The optimal default directions are learnt from the data and furthermore, this algorithm treats the non-presence as a missing value and learns the optimal direction to handle missing values.

**2.7 Yggdrasil**

Yggdrasil is an optimized system for training deep decision trees at a scale and it also scales well to high dimensional data. Yggdrasil is viable when the data set is large and tree fits in memory. It scales well to high dimensional data. Yggdrasil is based on vertical partitioning of the data unlike PLANET which is based on horizontal partitioning of the data, along with the set of optimized data structures to reduce CPU and communication costs of training. However, more splits are required for deeper trees which results in more time complexity.

Training on compressed data without decompression, efficient training on uncompressed data and minimal communication between nodes are the three novel optimizations on top of the basic idea of vertical partitioning.

**Working:**

At iteration t, we compute the optimal splits for all nodes on the tth level of the tree via two round trips of communication between the master and the workers. All splits for a single depth t are computed at once like PLANET. For each node i at depth t, the following steps are performed:

**ComputeBestSplit:**

The jthworker locally computes fj from Split (i) = arg maxjfj and sends this to the master. The master selects s=Split(i). Let fj\*denote the optimal feature selected for s\*, and let Wj\*be the worker containing this optimal feature: fj\* belongs to Wj\*.

**bitVector = CollectBitVector(Wj\*):**

In order to ascertain which child node each training point x belongs to I should be assigned to, the master requests a bit vector from Wj\*

Page 7

**BroadcastSplitInfo(bitVector):**

The master then broadcasts the bitVector to all the k workers and each worker later updates their internal state to prepare for the next epoch of training.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Comparison Parameter** | **C4.5** | **SPRINT & SLIQ** | **PLANET** | **Parallel C4.5 implementation** | **XGBoost** | **Yggdrasil** |
| **Advantages** | Memory efficient than ID3 | Memory efficient than C4.5 | More performance than ID3,SPRINT and SLIQ | More performance than C4.5,avoids overfitting | Solves the curse of high dimensionality | handles high dimensional data |
| **Disadvantages** | High training samples are needed | Lists or hash trees cannot fit into memory | Cannot handle high dimensional data | Increase in communication cost with increase in depth | works with numeric features | More splits for deep trees |
| **Measure** | Information gain ratio | Gini index | Information gain ratio | Information gain ratio | Information gain | Information gain ratio |
| **Procedure** | top down decision tree induction | top down decision tree induction | top down decision tree induction | top down decision tree induction | top down decision tree induction | top down decision tree induction |
| **Approach** | greedy | greedy | greedy | greedy | greedy | greedy |

Table1. Comparative analysis of decision tree algorithms

Page 8

With the increase in data in many application domains, parallelism is required to augment the scalability in implementing machine learning algorithms. As our proposed approach is based on decision tree, parallelism in decision tree can be effortlessly achieved by building decision tree nodes in parallel or by dispensing the training data among the computing elements. However building a decision tree is tedious task for some reason which will be delineated below.

Firstly, shape of the tree is very intermittent which can be determined only at runtime and the amount of processing required at each node may vary conspicuously. Secondly, decision tree construction phase is most computationally expensive stag of the algorithm.

Parallelism can be employed while building decision tree nodes or during the enumerations of best splitting attribute in attribute selection phase. Dynamically distributing decision tree nodes among the processors for further expansion is known as task parallelism. The decision tree construction phase is started by a single processor using the complete training set. When the number of processor nodes is equal to the number of decision tree nodes, then the nodes are split among the processors. Each processor continues constructing sub decision trees rooted at the nodes it has been assigned.

Executing the same set of instructions in the algorithm with all the processors involved is known as data parallelism. Each processor possess distinct pat of data as the complete training dataset is disseminated among all the processors. The distribution of training dataset among all the processors can be done in a horizontal or vertical manner. The data distribution in a vertical manner involves splitting the training data by assigning different set of attributes to each processor. All the instances for the set of attributes assigned to the processor and the values of the classes of all instances are kept in processor’s memory. Each processor is superintendent only for the evaluation of its attributes during any possible splits in the algorithm.

The dataset distribution in a horizontal manner involves distributing the instances equally by the processors. All the processors are involved in the evaluation of all the possible splits of the instances associated to a node which are communicated among all the processors to find the global values used and then decide the best split. However, the split is performed locally by each processor.

Both the parallelisms give good performance and better scalability results as they are capable of processing for very large datasets. However, data parallelism requires high communication load because the calculated gain ratios by each processor have to be communicated among all the processors to find the best split globally.

Page 9

Constructing a decision tree using thread level parallelism is an intuitive process. Starting with the complete training dataset at root node, data enumerates the gain ratio of attributes and decides the best split attribute. After discovering the optimal attribute, branch is created for each possible value. However, using thread level parallelism consumes a lot of time in node building at the branch because it is CPU bound task.

Instruction level parallelism can also be exploited while building decision tree models by utilizing global knowledge of the program which is not possible for the processor to analyze at runtime due to availability of limited resources. Decision tree scheduling can be used due to its absence of join points and side entries. There is only one point in a decision tree which is the root and side exits from the interior basic blocks of decision tree are prohibited. However this parallelism has few short comings like data dependence, name dependence and control dependence.

Page 10

**CHAPTER 3**

**PROPOSED APPROACH**

**3.1 Problem Definition**

As the days are passing, the amount of data generated by several industries, devices, business organizations and technologies is proliferating in leaps and bounds. Identifying, grouping and properly classifying this large data is very essential for the classification models to efficiently work in contemporary world. However the conventional classification algorithms are proven to be ineffective in handling the big data generated in real time scenarios. The main bottlenecks in learning from big data are

* Memory restrictions: It is difficult to keep the whole training dataset or most of it in memory on a single computer.
* Time complexity: Finishing the computation within a tolerable time is very hard.
* Data complexity: The high-dimensional and multi modal features of the data make an extensive influence on the performance and efficiency of results.

Because of above bottlenecks the parallelization of algorithms becomes a prevalent and reliable choice.

Multiprocessing is one of the specific implementations of parallelism. Multiprocessing is very expedient for parallelization which abstracts away from many of the difficulties in parallelizing computationally expensive calculations like decision tree construction in our problem statement.

The problem definition can be summarized as follows:

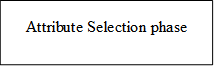
Building a C4.5 decision tree using multiprocessing parallelism mode to observe the performance and time comparisons of the models built in serial and parallel approach with increase in scalability of datasets. A tradeoff between performance and time consumed to build the models should be achieved with the increase in scalability. Furthermore, the performance should not be deteriorated with increase in scalability of datasets

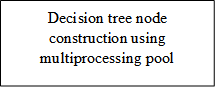
Page 11

**3.2 Proposed approach**

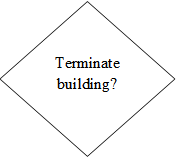
**Flow chart of proposed approach:**

****

****

****

**Yes**

**No**

****

Page 12

**3.2.1 Decision Trees**

Decision trees provide a simple and comprehensible method to classify the data based on the former knowledge. Decision trees are made up of internal nodes and leaf nodes. Each internal node can be split into one or more children based upon the attributes of the training dataset.

This decision of which attribute to be chosen for splitting the data is taken based on some metrics such as information gain, gain ratio etc. In this way the internal nodes are constructed in a recursive fashion until all the tuples of the training data are classified. The leaf nodes at the end of the decision tree are associated with a class label or an outcome. Commodious and transparent characteristics of the decision trees make them propitious tools for designing a parallelized learning algorithm.

**3.2.2 C4.5 Decision Tree**

C4.5 adopts greedy approach where decision trees are constructed in a top down recursive divide and conquer manner. In case of C4.5 decision tree, attribute selection measure is information gain ratio. It differs from other attribute selection metrics in a way that “It takes into account the number of branches that would result before making the split”, this would eventually reduce the bias by taking into consideration the intrinsic information of a split. Hence C4.5 decision tree is usually the prime choice.

E(s)=

Information Gain(T,X)=Entropy(T)-Entropy(T,X)

Information gain ratio=Information gain/split information

In addition to the pros mentioned above, another advantage of choosing C4.5 decision tree is that it handles both continuous and discrete attributes. As a result, it creates a threshold and later splits the list into those whose attribute value is above the threshold and those that are less than or equal to it in order to handle continuous attributes.

Missing attributes are also handled by C4.5 decision tree by not including them in gain and entropy calculations.

**3.2.3 Parallel C4.5 Decision Tree**

Parallelization of C4.5 decision tree is achieved by multiprocessing using a pool of worker processes. In this type of parallelism all the cores are utilized and the possibility of any core remaining idle is very less. Several methods are present that allow tasks to be offloaded to the worker processes.

Page 13

The pool distributes the task of construction of the decision tree to the available processes using a FIFO scheduling. Here the task of construction of decision tree is divided in such a way that each branch of decision tree is constructed by one of the worker processes in the pool according to the FIFO scheduling policy such that no process is left idle.

The pool waits for all the tasks to finish, collects output from all the processes and returns the final output.

**3.2.4 Preprocessing**

Data preprocessing is very crucial as it directly impacts the performance of the model. It reduces the complexity of the data and helps to resolve issues such as unclean and incomplete data. After preprocessing the data becomes more suitable for further processing.

NumPy library is used for the sophisticated calculations involved in preprocessing. The preprocessing phase in the construction of parallel C4.5 decision tree deals with handling missing values and mixed attributes.

If the attribute having one or more missing values is numerical then the minimum of the attribute values is calculated and every missing value is replaced with minimum value minus 1(minimum value-1) whereas if the attribute id is categorical then the missing values are replaced by NULL.

Decision trees are classified into two types based on the target variable. The first type is categorical target variable decision tree which has a categorical target variable as the name suggests. Another type is continuous target variable decision tree which has a continuous target variable.

Categorical attributes do not require processing for attribute selection. However, for continuous attributes, processing is required before calculating the information gain ratio to decide the best splitting attribute. In case of continuous attributes, the attribute values are sorted and the the midpoints between the attributes are evaluated based on information gain ratio metric in order to come up with split points. Later attribute selection is performed using information gain ratio metric to decide the best splitting attribute.

**3.2.5 Multiprocessing VS Multithreading**

Multiprocessing is CPU/GPU bound whereas multithreading is IO bound. CPU/GPU bound tasks involve encryption and decryption, sophisticated mathematical operations involving multidimensional arrays, algorithms which involve continual enumerations like sorting, binary search etc. IO bound tasks involve counting number of lines in a file, web scraping, processing data from disk etc.

Multiprocessing launches different processes which are independent to each other whereas multithreading launches threads which are still dependent on the parent process.

Page 14

**3.2.6 Training**

Deciding the best splitting attribute is the primary and significant step in building a decision tree classifier. Choosing an attribute without the best metric value impacts the accuracy and decision tree that will be constructed. The attribute selection measure which we have chosen is information gain ratio because it minimizes the bias towards multi-valued attributes by dividing the information gain with the intrinsic information.

After the step of preprocessing, all the attributes of the training dataset are evaluated based on the information gain ratio metric to decide the best splitting attribute. Before calculating the information gain ratio, entropy of each attribute has to be enumerated first.

Entropy is calculated for each attribute parallely. Branch level parallelism is used to build the decision tree classifier which means the branches of the decision tree will be created parallely.

Beginning with all the training instances at the root node which is assigned to an idle processor available in the pool, data is evaluated to determine the optimal split attribute. After discerning the optimal attribute, a branch is created for each possible value. Based on the values of the chosen attribute, dataset is divided into mutually exclusive subsets of instances and new nodes are formed in a tree structure. Thus, for each newly created node, an idle processor available in the pool is assigned. As a result, a single processor will not be burdened with heavy load and the tasks are distributed among the available processors in the pool. At subsequent levels of the tree, these steps are repeated for every node until termination. Then a leaf node is created and it is given a class value. Thus a decision tree model is built.

Mean absolute error, root mean square error are enumerated to know the training error in regression trees whereas accuracy is computed in classification trees to know the training error.

Page 15

(root =1) (dataset,root,leafid=0,parents=0,dataset\_features)

Sample rule:

current level: root+1

leafid

parents

decision rule

best splitting attribute

instances

Core 1

Sample rule:

current level

leafid

parents

decision rule

best splitting attribute

instances

Sample rule:

current level

leafid

parents

decision rule

best splitting attribute

instances

Core 2 Core3 (subdataset,root,leafid=uuid1(),0,dataset\_features)(subdataset,root,leafid=uuid1(),0,dataset\_features)

**Figure 2: Multiprocessing pool**

Page 16

**3.2.7 Prediction**

Predictions are made on unseen instances using the transparently built decision tree classifier. The model built is tested on a test dataset consisting of unseen instances by traversing the tree based on the attribute values and assigning it the class at the corresponding leaf.

Accuracy is enumerated to know the deviation of the predicted labels from the true labels.

**3.2.8** **Performance and time evaluation with increase in scalability**

Accuracy is used as the metric to measure the performance of the models built using three different datasets. Furthermore, time comparisons should be made by building the models in serial and parallel manner for the three datasets.

The size of the datasets is gradually proliferated to observe the scalability and performance of the models built. A tradeoff between performance and time consumed to build the models should be achieved with the increase in scalability. Furthermore, the performance should not be deteriorated with increase in scalability of datasets.

Page 17

**CHAPTER 4**

**EXPERIMENTAL PROCEDURE**

**4.1 Dataset Description**

We used three datasets to comparatively analyze the accuracy of our parallel decision tree model. We chose the datasets such that we can analyze and deduce the changes in the accuracy of the model and how the size of the dataset effects the accuracy.

The datasets we have chosen are Tic-Tac-Toe of small size, HIV-1 protease cleavage sites of moderate size and Cardiovascular Disease dataset which is of relatively high size.

Tic-tac-toe dataset is about all the configurations possible for the tic-tac-toe board game. It has 958 instances and nine attributes each depicting the value {x,o,b} in respective squares of the board game. It is assumed in this dataset that ‘x’ started first and the target concept is to learn whether x won the game or not. All the attributes are categorical in nature and the classification problem is binary classification problem because the outcomes of the class label are either positive or negative. The dataset size is 28kb.

1.top-left-square:{x,o,b}  
2.top-middle-square:{x,o,b}  
3.top-right-square:{x,o,b}  
4.middle-left-square:{x,o,b}  
5.middle-middle-square:{x,o,b}  
6.middle-right-square:{x,o,b}  
7.bottom-left-square:{x,o,b}  
8.bottom-middle-square:{x,o,b}  
9.bottom-right-square:{x,o,b}  
10. Class: {positive,negative}

Hiv-1 protease cleavage sites dataset is of 84kb size. It has 1 categorical attribute having sub parts which denote the octamers of amino acids and the class label denotes flag value whose value is +1 or -1. It has moderately good number of instances i.e., 6590 without any missing values.

Page 18

Each attribute is a letter denoting an amino acid

1. G (Glycine, Gly);
2. P (Proline, Pro);
3. A (Alanine, Ala);
4. V (Valine, Val);
5. L (Leucine, Leu);
6. I (Isoleucine, Ile);
7. M (Methionine, Met);
8. C (Cysteine, Cys);
9. F (Phenylalanine, Phe);
10. Y (Tyrosine, Tyr);
11. W (Tryptophan, Trp);
12. H (Histidine, His);
13. K (Lysine, Lys);
14. R (Arginine, Arg);
15. Q (Glutamine, Gln)
16. ; N (Asparagine, Asn);
17. E (Glutamic Acid, Glu);
18. D (Aspartic Acid, Asp);
19. S (Serine, Ser);
20. T (Threonine, Thr).
21. Target label (+1 or -1)

Cardiovascular Disease dataset is relatively larger dataset of 2.79 mb of size. It consists of as high as 70,000 instances. It has eleven features and two target values viz. 0 or 1 specifying whether the patient smokes or not. The main advantage of selecting this dataset is it has mixed valued attributes. All types of attributes such as continuous, non-continuous, numerical and categorical are present in it which tests the models potential to handle mixed attributes problem.

1. Age(in days)-continuous attribute
2. Height(in cm)-continuous attribute
3. Weight(in kg’s)-continuous attribute
4. Gender-categorical attribute
5. Systolic blood pressure-continuous attribute
6. Diastolic blood pressure-continuous attribute
7. Cholesterol-categorical attribute
8. Glucose-categorical attribute
9. Smoking-categorical(binary) attribute
10. Alcohol intake-categorical(binary) attribute

Page 19

1. Physical activity-categorical(binary) attribute
2. Presence or absence of cardio vascular disease-target variable-categorical(binary) attribute

All the datasets are divided into 4:1, training data and test dataset splits.

**4.2 Multiprocessing**

In the proposed approach, multiprocessing is the optimal parallelism mode because python global interpreter lock restricts multiple threads from running at a time to prevent memory dirty writes. For this purpose, multiprocessing library in python is installed. Multiprocessing library allows to perform work in a parallel fashion to augment performance by taking advantage of multiple cores inside a processor. Multiprocessing pool in python facilitates creation of number of workers that run in child processes. Pool enables distribution of work across multiple processes without involving writing of process creation and process teardown methods.

Pool maps the data to the respective workers and aggregates it back to display the final result and it also gives the freedom to indicate the number of worker processes required for your execution. For instance, if you indicate 8 workers and you have 16 jobs to complete, then 8 of them would start first and as and when a worker completes its job, it, without further ado, starts executing the next process.

Pool class of python creates worker processes which are daemonic in nature. Consequentially, it doesn’t enable creation of pools within pools and raises an error which is “daemonic processes are not allowed to have children if the programmer tries to create them. Python pool class raises an error to disenable children processes to create a pool so that an army of zombie grandchildren is prevented.

pool.starmap() maps a function whose parameters are iterables to a multiprocessing pool. This function distributes input data across processes to be run with the referenced function.

**4.3Training model**

C4.5 decision tree training model:

* No of epochs = 10
* Learning rate=1
* Training dataset sizes =28 KB, 84 KB, 2.79 MB.
* Sample rules created in .txt files are merged into .json file.
* Decision tree model rules are stored in .py file.
* Accuracy is the metric used to determine training data performance in classification. Mean absolute error, root mean square error are the metrics to know the training error in regression.
* Number of processors used for multiprocessing=3

Page 20

**4.4 Implementation details**

* OS: Windows.
* Language: python 3.
* Libraries and packages: Chefboost framework, python version more than 3.6, multiprocessing, pickle, imp, JSON libraries.
* pip version required:20.0.1.

**4.5 Testing**

Apart from predicting class labels of single unseen instances, testing is also done on three datasets namely Tic-tac-toe, HIV1 protease cleavage and cardiovascular disease. The accuracies and time taken to build the models for the three test datasets were measured in serial and parallel mode.

During the testing phase, the performance and time taken to build the models were observed by proliferating the size of datasets. A graph was plotted for accuracies of the models in serial and parallel mode. In addition, a bar chart was plotted for time consumed to build the models in serial and parallel mode. The comparison of accuracies in serial and parallel manner was illustrated by increasing the size of datasets gradually through bar chart and graph plotted.

Page 21

**CHAPTER 5**

**RESULTS AND DESCRIPTION**

**Figure3: Comparison of time taken for serial and parallel execution for different datasets.**

Page 22

**Figure4: Comparison of accuracy of different datasets for serial and parallel execution**

The results in the nascent stages showed that the performance of the model built is higher in parallel mode than in serial mode and the time taken to build the model in parallel mode is more than that of serial mode. Later, after increasing the size of the datasets, the performance of the serial approach has been retained in the parallel approach and the time taken to build the model in parallel mode is less than that of serial mode. The results connote that communication overheads can be a key factor in limiting the performance of the parallel approach same as that of the serial approach. Therefore, a tradeoff is maintained between performance and time taken to build the models with the increase in scalability**.**

Page 23

**CHAPTER 6**

**CONCLUSION AND FUTURE SCOPE**

The proposed approach handled the increase in scalability by utilizing the idle processors available in the multiprocessing pool without deteriorating the performance of the model. In the proposed approach, the main features of the C4.5 decision tree algorithm such as handling continuous attributes, dealing with missing values were preserved.

The results in the nascent stages showed that the performance of the model built is higher in parallel model than in serial mode and the time taken to build the model in parallel mode is more than that of serial mode. This denotes that the use of branch level parallelism for building the nodes for small size datasets has led to increase in performance in parallel approach due to low communication overheads. The increase in time taken in parallel approach indicates that there are no communication overheads in serial approach whereas the parallel approach has few communication overheads.

Later, after increasing the size of the datasets, the performance of the serial approach has been retained in the parallel approach and the time taken to build the model in parallel mode is less than that of serial mode. This indicates that the use of branch level parallelism for building the nodes for large datasets has led to retaining the performance in parallel approach same as that of the serial approach due to more communication overheads instead of increasing the performance like that of the preliminary stages. The decrease in time taken in parallel approach evinces that if the datasets are very large, the serial approach takes large amount of time in processing. As a result, though the communication overheads are more in parallel approach with increase in data, time taken to build the model in parallel mode is less than that of serial mode.

In a nutshell, the results connote that communication overheads can be a key factor in limiting the performance of the performance of the parallel approach same as that of the serial approach. Therefore, our proposed approach maintained a tradeoff between performance and time taken to build the models with increase in scalability.

In future, we would like to proliferate the scalability of the model even further by using more number of computational machines instead of just one machine as used in the proposed approach. Increase in the number of machines will completely remove memory restrictions pertaining to storage.

Page 24

**REFERENCES**

[1]J. Shafer, R. Agrawal, and M. Mehta, “Sprint: A scalable parallel classifier for data mining,” in Vldb, vol. 96, pp. 544–555, Citeseer, 1996.

[2] M. Mehta, R. Agrawal, and J. Rissanen, “Sliq: A fast scalable classifier for data mining,” in International conference on extending database technology, pp. 18–32, Springer, 1996.

[3] B. Panda, J. S. Herbach, S. Basu, and R. J. Bayardo, “Planet: massively parallel learning of tree ensembles with mapreduce,” 2009.

[4] Y. Mu, X. Liu, Z. Yang, and X. Liu, “A parallel c4. 5 decision tree algorithm based on mapreduce,” Concurrency and Computation: Practice and Experience, vol. 29, no. 8, p. e4015, 2017.

[5] T. Chen and C. Guestrin, “Xgboost: A scalable tree boosting system,” in Proceedings of the 22nd acmsigkdd international conference on knowledge discovery and data mining, pp. 785–794, 2016.

[6] F. Abuzaid, J. K. Bradley, F. T. Liang, A. Feng, L. Yang, M. Zaharia, and A. S. Talwalkar, “Yggdrasil: An optimized system for training deep decision trees at scale,” in Advances in Neural Information Processing Systems 29 (D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, eds.), pp. 3817–3825, Curran Associates, Inc., 2016.

[7] H. Chauhan and A. Chauhan, “Implementation of decision tree algorithm c4. 5,” International Journal of Scientific and Research Publications, vol. 3, no. 10, pp. 1–3, 2013

[8] B. Hssina, A. Merbouha, H. Ezzikouri, and M. Erritali, “A comparative study of decision tree id3 and c4. 5,” International Journal of Advanced Computer Science and Applications, vol. 4, no. 2, pp. 13–19, 2014.

[9] K. Kubota, A. Nakase, H. Sakai, and S. Oyanagi, “Parallelization of decision tree algorithm and its performance evaluation,” in Proceedings Fourth International Conference/Exhibition on High Performance Computing in the Asia-Pacific Region, vol. 2, pp. 574–579, IEEE, 2000.

[10] N. Amado, J. Gama, and F. Silva, “Exploiting Parallelism in decision tree induction,” in Proceedings from the ECML/PKDD Workshop on Parallel and Distributed computing for Machine Learning, pp. 13–22,2003.

[11] S. Bejai, “Compilation techniques for increasing instruction-level parallelism for the scad machine,”

[12] J. P. Bradford and J. A. Fortes, “Characterization and parallelization of decision-tree induction,” Journal of Parallel and Distributed Computing, vol. 61, no. 3, pp. 322–349, 2001.

[13] K. J. Steinhaeuser and N. V. Chawla, Scalable learning with thread-level parallelism. PhD thesis, University of Notre Dame, 2007.

Page 25

**APPENDIX**

* **Root Node:**

**Root node represents the total population which is further divided into two or more homogenous sets.**

* **Splitting:**

**A node is divided into two or more sub nodes based on an attribute selection metric.**

* **Branch / Sub-Tree:**

**Subsection of the built decision tree is known as branch/sub-tree.**

* **Chefboost:**

Chefboost is a light weight decision tree framework which enables building of decision trees using different algorithms like ID3, C4.5, CART,CHAID and regression tree algorithms.

* **Mean absolute error:**

It is a model evaluation metric used in regression trees to measure the resulting between two continuous variables.

* **Root mean square error:**

It is a model evaluation metric used in regression trees which is the standard deviation of the predicted errors. It lets us know how concentrated the data is around the line of optimal fit.

* **Accuracy:**

It is a model evaluation metric used in classification trees.

* **Dataset link:**

<https://archive.ics.uci.edu/ml/datasets/Tic-Tac-Toe+Endgame>

<https://archive.ics.uci.edu/ml/datasets/HIV-1+protease+cleavage>

<https://www.kaggle.com/sulianova/cardiovascular-disease-dataset>

* **Multiprocessing in python:**

<https://blog.mbedded.ninja/programming/languages/python/python-multiprocessing/>

<https://medium.com/contentsquare-engineering-blog/multithreading-vs-multiprocessing-in-python-ece023ad55a>

Page 26