Proceedings of the Third International Conference on Trends in Electronics and Informatics (ICOEI 2019)
IEEE Xplore Part Number: CFP19J32-ART; ISBN: 978-1-5386-9439-8

Accuracy Prediction for Distributed Decision Tree using Machine Learning approach

Siddalingeshwar Patil
Department of Computer Science and Engineering
SDM College of Engineering and Technology
Dharwad, India
siddalingeshwar@gmail.com

Umakant Kulkarni
Department of Computer Science and Engineering
SDM College of Engineering and Technology
Dharwad, India
upkulkarni@yahoo.com

***Abstract*—Machine Learning is one of the finest fields of Computer Science world which has given the innumerable and invaluable solutions to the mankind to solve its complex problems. Decision Tree is one such modern solution to the decision making problems by learning the data from the problem domain and building a model which can be used for prediction supported by the systematic analytics. In order to build a model on a huge dataset Decision Tree algorithm needs to be transformed to manifest itself into distributed environment so that higher performance of training the model is achieved in terms of time, without compromising the accuracy of the Decision Tree built. In this paper, we have proposed an enhanced version of distributed decision tree algorithm to perform better in terms of model building time without compromising the accuracy.**

***Index Terms*—Machine Learning, Classification, Decision Tree, Distributed algorithm, Distributed Decision Tree, Spark, Map-Reduce, BigData**

I. Introduction

Supervised Learning is processing the dataset with known results/labels to learn the model/system, this process of learning the training data continues until the model learns correctly and achieves an acceptable level of performance. Once the model is ready, it is used to find out the answers or results of the new unknown data. Supervised Learning has a set of input variables X and an output variable Y where an algorithm is used to learn a mapping function from the input to the output.

$$Y = f(X) \tag{1}$$

Decision Tree is a prototypical *classification* and/or *prediction* algorithm in supervised machine learning, which can be used to categorise or classify the given data based on previous knowledge of the training data. **Classification** and **regression** are two fundamental ML tasks which use the labeled data to train and build a model. This model can be used to determine the label or class of any given data. The performance of such a model can be verified by using several metrics like precision, accuracy, recall among many other. After having the required performance of the model in place, the model can be put in place for real-time usage to arrive at many decisions. To build such a robust model which can support critical business decisions, adequate sized real-time data needs to be processed so that the data represent the real-time scenarios.

Classification is approximating a map function from X input variables to discrete Y output variables. **Regression** is approximating a map function from X input variables to continuous Y output variables. For example, *classification* can help to identify loan-defaulter based on multiple attributes of the customer data which can prevent the possible loss to particular financial institute. Similarly *regression* can help to detect fraudulent transaction online by providing numerical analytics of the values of various attributes of that transaction. Some classification methods are Naive Bayes, Decision Tree, Logistic Regression, Support Vector Machines.

Decision Tree (DT) algorithm constructs a tree structure where each non-leaf node represents an attribute evaluation and leaf-nodes represent class labels. This algorithm analysis the training data to identify the attributes with higher information gain then the rest of the attributes, as such an attribute can effectively classify or categorise the data. Likewise each attribute is considered one after the other in the increasing order of information gain recursively and builds the tree. The attribute with the highest information gain is considered as the root node because that classifies the entire dataset into different classes instantaneously. Subsequently the attributes with next lower-level information gain are considered to further divide the dataset into sub-part until each data record is assigned its designated class label.

DTs are constructed by analysing a set of training data examples for which the class labels are known. Such DTs are applied to classify the new unseen data examples. If DTs are trained on high-quality training data, they can predict the outcome very accurately. Thus used in many critical real-time systems like, "to find out the presence or type of cancer based on previous patients's data" and similar ones. A Decision Tree looks like Figure 1. A DT is drawn upside down with root at the top. The leaf nodes represent the class labels. Wind, Temp represent attributes whereas the texts on the edges represent the values of those attributes.

Distributed Decision Tree (DDT) is required to process big data as training set because DTs underperform or fail with such huge amount of data. In order to implement DDT, distributed environment is required and as part of this research work Apache Spark is used with Scala. This research work has been conducted to design and implement "Distributed Decision Tree Learning" model in a distributed environment which scales-out extensively for huge datasets (giga bytes) by parallelising time-consuming complex-computations and which performs better in terms of model building time compared to the existing solutions in the similar environment.

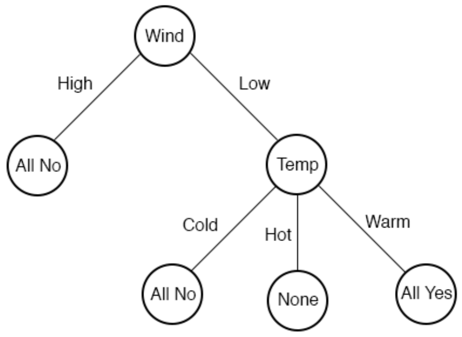

Fig. 1. Sample Decision Tree.

The remainder of this paper is organised as follows: Section 2 discusses the necessary background and related work to understand the existing systems. Section 3 details out the working of basic DT learning algorithm, Section 4 discusses the implementation of Distributed Decision-Tree Learning in detail, analyses the results and benchmarks for the work carried out. Finally section 5 concludes this paper and briefs about future work.

II. Background

Google's PLANET [2] uses MapReduce [3] to distribute the tree building computations and uses Hadoop architecture to store intermediate results on HDFS. This disk based distributed design although works faster than centralised tree building algorithm, has its limitations with respect to the speedup as it involves disk I/O operations for every data processing task. The Spark MLLib [5] implements PLANET's idea on Apache Spark platform with significant performance improvement over PLANET as Spark's in-memory data storage helps it to obtain speed-up. However this benefit of in-memory storage of intermediate results and processing is subjected to the amount of memory available for such task and also depends upon the size of the data that needs to be managed. So Spark MLLib can always have better or equal performance speed-up benefits as that of PLANET. Spark provides a PySpark [10] package for Python which is a wrapper to Spark MLLib. MLLib requires the training data to be converted into a specific object format, which takes a significant amount of CPU time for PySpark to convert the data into such format. Distributed Decision Tree [12] and v.2.0 [13] as proposed although show comparatively better performance results with that of other existing solutions, do not provide the implementation details. In this paper we have proposed to implement distributed version of decision tree learning algorithm which performs better than PLANET in terms of model building time by using in-memory data-processing facilitated by Spark. As we have not constrained the training data to be in a specific object format, this proposed implementation works better even if it is wrapped in Python. C45 [1] a newer version of Decision Tree algorithm and its MapReduce implementation [4] are referred for comparison as part of our research work. Similar research on YARN [14] and a survey [8] are also referred for better understanding of the existing implementations.

III. Basic Decision Tree Induction Algorithm

Algorithm 1: DecisionTreeInduction

```
  Input: D, dataset with attribute values

 1: Tree = {}
 2: if D is "pure" OR other termination condition satisfied
    then
 3:    terminate
 4: end if
 5: for all attribute $a \in D$ do
 6:    Compare information purity if we split on $a$
 7: end for
 8: Select attribute with best information purity, $a_{best}$
 9: Add a node that splits dataset on $a_{best}$
10: Induce subset of dataset D based on $a_{best}$ into $D_v$
11: for all $D_v$ do
12:    $Tree_v$ = DecisionTreeInduction($D_v$)
13:    Add the subtree $Tree_v$ to the respective branch of node
14: end for
15: return Tree
```

Algorithm 1 is the generalised form of DT induction, it is known as ID3 [6] as proposed by Quinlan. It is a recursive algorithm so it starts with an empty tree. It first analysis all the attributes in terms of information fain or purity or how best the particular attribute can split the given dataset. It then chooses the best attribute and designates it as the root node of the tree. Here criteria can be information gain purity - entropy, Gini index. After choosing the root node of the tree, it splits the input data into subsets with each subset having certain value of the best attribute - at root node. The algorithm takes up each subset of data and continues to recursively invoke the same algorithm with data subset and produces the sub-trees. These sub-trees will be added to create a complete tree at the end.

Entropy is the measure commonly used in Information theory which characterises the purity or impurity of a collection of data instances. The homogeneity of examples can be measured by entropy. Given a set of examples S, containing negative and positive instances of a target concept, the entropy of S relative to boolean classification can be measured as

$$Entropy(S) \equiv -p_{\oplus} log_2 p_{\oplus} - -p_{\ominus} log_2 p_{\ominus} \tag{2}$$

For multi-class dataset entropy can be calculated as

$$\sum_{i=1}^{c} -p_i log_2 p_i \tag{3}$$

To illustrate the calculation of entropy, suppose a collection of 20 examples of some two-class concept - positive and negative, including 12 positive examples and 8 negative examples,

ID	Outlook	Temperature	Humidity	Windy	Play
1	sunny	hot	high	false	no
2	sunny	hot	high	true	no
3	overcast	hot	high	false	yes
4	rainy	mild	high	false	yes
5	rainy	cool	normal	false	yes
6	rainy	cool	normal	true	no
7	overcast	cool	normal	true	yes
8	sunny	mild	high	false	no
9	sunny	cool	normal	false	yes
10	rainy	mild	normal	false	yes
11	sunny	mild	normal	true	yes
12	overcast	mild	high	true	yes
13	overcast	hot	normal	false	yes
14	rainy	mild	high	true	no

TABLE I
SAMPLE WEATHER DATASET

then the entropy of this collection with respect two boolean classification is

$$Entropy([12+, 8-]) = -(12/20)log_2(12/20) - (8/20)log_2(8/20) = 0.971 \quad (4)$$

A. Working of ID3

To illustrate the working of ID3, let us consider the dataset in table 1, given the weather condition we like to predict whether to play outside or not. Few attributes of weather are *Temperature, Outlook, Windy, Humidity*. Each attribute has its own values, for example *Temperature* may be *Hot, Cool, Mild*. The output class is *Play* and it has two values: *yes* and *no*.

There are four attributes ignoring ID attribute which can be considered to split the data at the root level. In the following section let us see how the algorithm works to construct a DT. **Information Gain:** One option to consider the best attribute to split the data at the root node is to consider the attribute with the highest *purity measure*. ID3 measures the purity of each attribute in terms of *information value*. To quantify such measure it uses entropy formula 3. This entropy defines a measure of how effectively an attribute can classify the training data. The information gain, Gain(S, A) of an attribute A, relative to collection of data examples S, is defined as,

$$Gain(S, A) \equiv Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v) \quad (5)$$

The entropy of the given collection of examples is calculated as below. There are 14 examples - including 9 positive examples and 5 negative examples, then the entropy of this collection with respect two boolean classification is

$$Entropy([9+, 5-]) = -(9/14)log_2(9/14) - (5/14)log_2(5/14) = 0.940 \quad (6)$$

The algorithm first considers *Outlook* attribute and calculates information gain.

1) *Outlook* can have three values: sunny, rainy and overcast. The algorithm calculates the information gain for each of these values.
2) Two instances with *Outlook* = sunny has class label yes and three instances of no.

$$Entropy(S_{sunny}) = -\frac{2}{5}log_2\frac{2}{5} - \frac{3}{5}log_2\frac{3}{5} = 0.971 \quad (7)$$

Outlook = overcast has four yes and zero no class labels.

$$Entropy(S_{overcast}) = -\frac{4}{4}log_2\frac{4}{4} - \frac{0}{4}log_2\frac{0}{4} = 0 \quad (8)$$

Outlook = rainy has three yes and two no class labels.

$$Entropy(S_{rainy}) = -\frac{3}{5}log_2\frac{3}{5} - \frac{2}{5}log_2\frac{2}{5} = 0.971 \quad (9)$$

3) The next step is to calculate the Information Gain using the aforesaid formula

$$Gain(S, Outlook) = Entropy(S) - \sum_{v \in \{sunny, rainy, overcast\}} \frac{|S_v|}{|S|} Entropy(S_v) \quad (10)$$

$$Gain(S, Outlook) = Entropy(S) - \frac{5}{14} Entropy(S_{sunny}) - \frac{5}{14} Entropy(S_{rainy}) - \frac{4}{14} Entropy(S_{overcast}) \quad (11)$$

$$Gain(S, Outlook) = 0.940 - \frac{5}{14} * 0.971 - \frac{5}{14} * 0.971 - \frac{4}{14} * 0 = 0.247 \quad (12)$$

Hence the Information Gain for the *Outlook* attribute is 0.247

4) Similarly Gain is calculated for all the attributes. Refer the Table 1, the algorithm produces the following Gains for each attribute:
Gain(Outlook) = 0.247
Gain(Temperature) = 0.029
Gain(Humidity) = 0.152
Gain(Windy) = 0.048
Among these four, *Outlook* attribute has the highest Gain, so this is considered as the first split node, that is the root node.

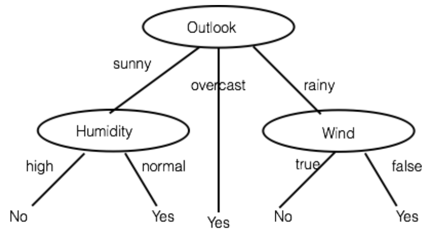

Fig. 2. Decision Tree: Play

Data-set	Instances	Class	Attributes	Size in MB
winequlaity	4898	10	12	0.25
skin-segmentation	196,045	2	4	3
credit-card-fraud	227,845	2	31	138
click-on-ad-predict	1,597,928	2	12	144

TABLE II
SELECTED DATASET

5) The algorithm further considers the subset of data that have been split at the root node and reiterates the above three steps to recursively to identify the next best feature and keeps on adding to the tree till each node ends in only one type of output value.

Thus built DT looks as in Figure 2.

B. Experiments with medium-large dataset

DT induction algorithm ID3 works as explained in the previous section, in this section ID3 is applied on small to medium real-time data-sets available on UCI ML repository [11]. For this experimental work four datasets namely, wine quality, skin segmentation, credit card fraud detection and click on ad prediction are selected. ID3 is implemented and it's performance is compared with that of Python's Scikit DecisionTreeClassifier. These datasets are listed and summarised in Table 2.

For example in *'click on ad prediction'* dataset the attributes are numeric, the attributes and sample values with target label are as under. Here *click* is the target attribute.

click, impression, url_hash, ad_id, advertiser_id, depth, position, query_id, keyword_id, title_id, description_id, user_id

1, 1, 1.4340390157469403E19, 9027213, 23808, 2, 1, 5, 1, 0, 0, 11808635

In the available dataset *eighty percent* of instances are used to train the model and remaining *twenty percent* are used to validate the model. Two parameters are considered to compare the performance of the *Scikit-library API* [15] and *ID3 algorithm* implementation. The first parameter is model building time and second is accuracy. Both the parameters are comparable and can be improvised by implementing the distributed algorithm. Experiments were performed on a single machine with 2.9 GHz Intel core i5 processor, 8 GB 1867 MHz DDR3 memory and 64 bit macOS.

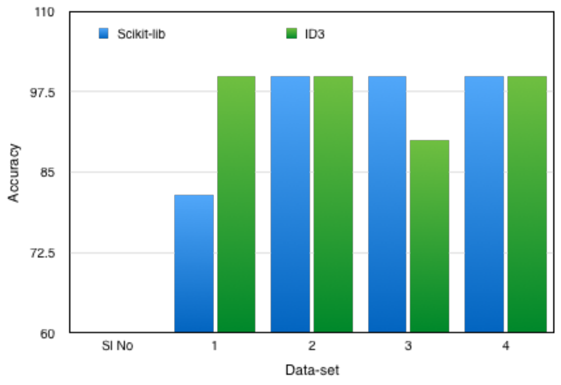

Fig. 3. Accuracy for selected datasets

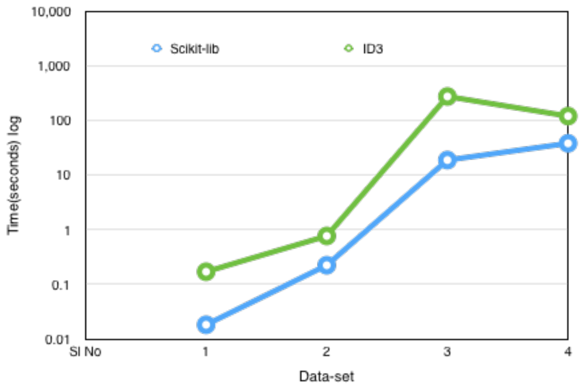

Fig. 4. Model Building Time for selected datasets

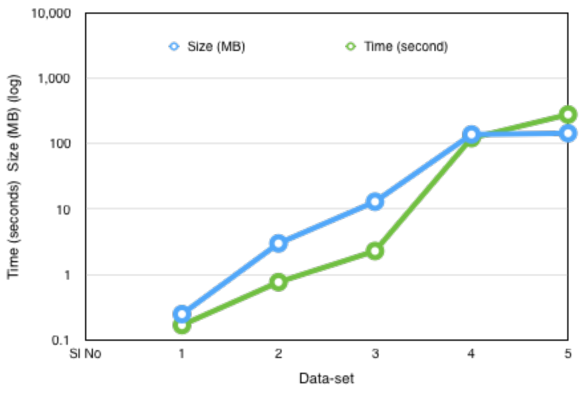

Fig. 5. Size of selected datasets versus Model Learning Time

Table 3 and Figure 3, 4 and 5 show the results of experiments.

Accuracy. It is as high as above 99 percent for all the numerical datasets using ID3 implementation. Performance is at-par with that of the scikit library as shown in the Figure 3.

Model learning time. For DT implementation it is lower compared to that of the library as shown in the Figure 4. However the time of validation of test dataset is not considered for performance comparison.

		Accuracy		Model Learning Time	
Data-set	Data Size(M)	Scikit-library	ID3	Scikit-library	ID3
wine-quality	0.25	81.445	100	0.0181	0.17
skin-segmentation	3	99.9	100	0.223	0.77
credit-card-fraud	138	99.9	90	18.957	277.79
click-on-add-predict	144	99.9	100	38.396	121.35

TABLE III
SIZE, ACCURACY, MODEL BUILDING TIME OF SELECTED DATASET.

IV. DISTRIBUTED SPARK DECISION TREE

In this paper we are proposing Distributed Spark Decision Tree (DST) algorithm which is a scalable CART [7]. Spark supports iterative operations to be handled very effectively. The training dataset is stored as Resilient Distributed Dataset (RDD) which is building block data structure of Spark and RDDs can be stored in-memory which reduce I/O operations. RDDs are partitioned so as to allow the data to be spread across a cluster. Each partition of RDDs is processed by a separate worker node in the Spark cluster so as to distribute the work-load and perform the computations in parallel. For the first iteration all the records of input dataset in the RDD are marked to be processed and are evaluated to identify the best node to split with maximum information gain ratio. That is how the root node of the tree is identified. Then onwards, the dataset is split and marked for processing for the subsequent iterations suitably based on the value of the splitting attribute. Likewise the best split attribute is identified at every level and that is added as a node in the tree hierarchically to build the DT model.

```
Algorithm 2: DistributedSparkTreeLearning
    Input: D, dataset with attribute values

1: D* = PrepareDataset(D)
2: Create empty "Root" node in the DST model.
3: repeat
4:     ClassLabelAggregateInfo =
       AggregateLabelInfoByEachFeature(D)
5:     bestAttribute = FindBestSplitAttributeAtEach-
       Level(ClassLabelAggregateInfo)
6:     UpdateDST(DST, bestAttribute)
7:     D* = IdentifyDataForNextLevel(D*, DST)
8: until no node left to expand OR depth of tree in DST
   has reached maxDepth
9: return DST
```

The Algorithm 2 described in this section is implemented in Apache Spark using Spark's Map-Reduce in its native language Scala. Spark's RDD and map-reduce are used to manage the input dataset and to execute the jobs of DDT learning respectively. Every function is mapped to respective map-reduce jobs. Spark carries out each Map job in the distributed fashion by delegating tasks to different worker nodes and consolidates the results of the Reduce jobs from the workers. The driver process runs as an executor on the master node of the Spark cluster and manages all the executors on

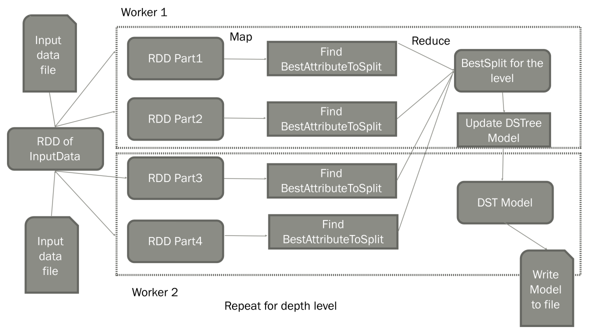

Fig. 6. Architecture View of DST Learning

the worker nodes. Figure 6 depicts the architecture view of the proposed algorithm.

A. Experimental Framework

Apache Spark [9] cluster is deployed in standalone mode and is used as a distributed environment to conduct the experiments of training DDT learning on large datasets both actual as well as synthetic dataset. This environment is also used to compare the performance of the Spark DDT with that of Python's PySpark Package [10] and Spark's MLLib [5]. The above mentioned experiments are performed on a standalone Spark Cluster with 2 nodes. One node with 2.9 GHz Intel core i5 processor, 8 GB 1867 MHz DDR3 memory and 64 bit macOS. This is designated as Master. It also hosts a Worker. Second node with 2.1 GHz Intel Core Duo Processor T8100, 4GB DDR2 memory and 64 bit Ubuntu Linux. It hosts a Worker. Each executor is run with 2GB memory. Small to medium real-time data-sets available on UCI ML repository are used. Synthetic data is prepared from the above real-time datasets to create the required file size and number of instances.

B. Datasets

For this experimental work four datasets namely, wine quality, skin segmentation, credit card fraud detection and click on ad prediction are selected from UCI ML repository [11]. From the above mentioned UCI datasets the large data-files of required size are synthesised as listed in Table 4 to evaluate the performance with respect to model building time of Distributed Spark Decision Tree (DST), PySpark (PT) and MLLib (MLT).

C. Results

Experiments are conducted on the dataset listed in Table 4 for following three different algorithms.

Sl. No.	No. of Features	No. of Instances(csv)	No. of Instances(labelled csv)	Size of Dataset
1	4	41 million	27 million	500 MB
2	4	82 million	54 million	1 GB
3	6	41 million	27 million	930 MB
4	6	41 million	41 million	930 MB and 1.4 GB

TABLE IV
LARGE SYNTHESISED DATASET

Trial	DST 41 Million 500 MB (minutes)	PySparkTree 27 Million 500 MB (minutes)	MLLIBTree 41 Million 900 MB (minutes)
Trial 1	4.3	13	3.4
Trial 2	4	13	3.1
Trial 3	3.9	12.5	3.1
Average	6.33	12.83	3.2

TABLE V
PERFORMANCE OF VARIOUS DT LEARNING ALGORITHMS - DATA WITH 4 FEATURES

Trial	DST 41 Million 1 GB (minutes)	PySparkTree 27 Million 1 GB (minutes)	MLLIBTree 41 Million 1.4 GB (minutes)
Trial 1	6.3	15	7
Trial 2	6.1	14	4.6
Trial 3	6.6	14	4.5
Average	6.33	14.33	5.37

TABLE VI
PERFORMANCE OF VARIOUS DT LEARNING ALGORITHMS - DATA WITH 7 FEATURES

1) Distributed Spark Tree (DST) – proposed algorithm
2) PySpark packages MLLib Tree (PT)
3) Spark MLLib's (MLT)

In all cases maximum tree depth was configured as 5. The performance figures are listed in the following Table 5 and Table 6 and are charted as shown in the Figure 7 and Figure 8.

The charts in Figure 7 and 8 clearly distinguish the fact that the proposed DST outperforms PySpark packages Decision Tree building API. The reason for this huge performance gap is that PySpark being a wrapper package for Spark's MLLib on Python needs the data in a LabledPoint (Spark MLLib Util) format, so the input dataset read as RDD[String] is converted to RDD[LabledPoint] a Python RDD. While invoking the actual API from Spark MLLib natively written in Scala, the above mentioned RDD in Python is retagged as RDD[LabledPoint] in Java. The above operations consume a significant time before actually working on tree building job, hence the gap in performance can be noted. Our DST is performing almost at the similar speed as that of MLLib's API with a difference of approximately 0.8 - 1 minutes on an average for three trials when tested for a whooping 41 million records of 6 features each. This clearly indicates that it is comparable with MLLib performance. More importantly it performs better than traditional DT when dataset is in the order of gigabytes. Also, Spark DDT is more suitable in real-time applications as it can handle the dataset that is spread across different nodes in a cluster. Our Spark DDT is designed to update the labels of data instances required to be processed in the next iteration but that leads to creation of new RDD of instances because of the basic nature of RDD which is immutable and every transformation on an RDD creates a new RDD. Due to this constraint, sometimes our algorithm could not outperform MLLib's DT. However we can have a Python wrapper around DST which would not restrict the input data format hence can save the data format conversion overhead incurred in PySpark. Thus, we have provided a python solution to distributed decision tree learning problem which is better than any existing ones on similar platform.

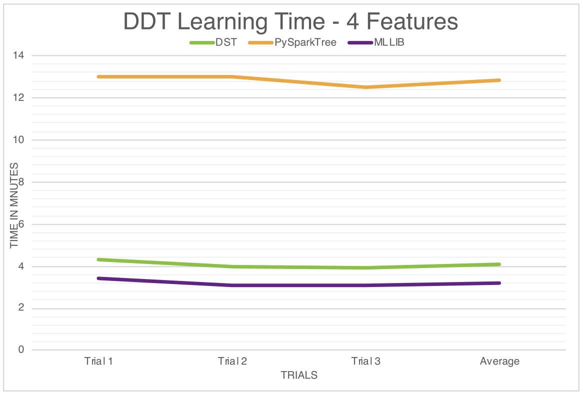

Fig. 7. Performance chart - data with 4 features

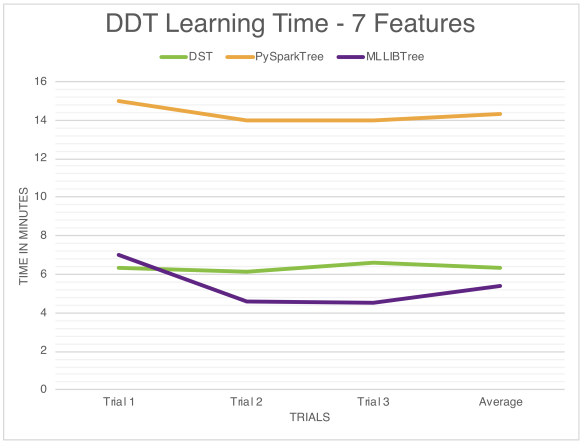

Fig. 8. Performance chart - data with 7 features

V. CONCLUSION AND FUTURE WORK

DT is a perfect classification tool with the highest accuracy suitable for small to medium sized numerical and nominal dataset. It performs better in-terms of model learning time for small to medium sized datasets as detailed in the previous section. Hence DT is the most widely used ML model for prediction and classification in real-time systems. The constraint that is mentioned in the previous subsection that creating new RDD for every subset of data to be processed should be dealt with and avoided unnecessary garbage collection, which shall improve the performance of the algorithm. As a continuation of this research work, need to consider vertical partitioning of the dataset to reduce the dimensionality so that insignificant features can be ignored to reduce the overfitting of the data and also parallelising the tree building on subset of features can be performed. MLLib implements binning of continuous features on a sub-sample of dataset to increase accuracy but it is a non-parallel task. We can consider parallelising this task to improve the performance. The experiments were performed on large datasets with seven features, should also increase the number of features in the dataset to compare the results. This work can be extended to implement distributed ensemble learning using Random Forests.

In this paper we propose an implementation of distributed decision tree in Spark environment. The implementation performs horizontal data parallelism in the shared-nothing architecture of Spark. It is scalable to large datasets and outperforms DT and PySpark's API in terms of model building time. Spark's in memory computations, RDD and map-reduce work coherently so well that any scalable algorithm can be effectively implemented in it. Research on distributed decision tree effectively established the fact that parallelising the time-consuming tasks in an algorithm and careful design considering the scalability can improve the performance significantly. Working with tremendously large datasets is becoming new normal in machine learning and while its effective usage in real-time. Hence scalable implementation of any machine learning algorithm should be considered as important can be done similar to the work carried out as part this research. This implementation of scalable classification tree is open for extension that with no medication to the core part can be used for regression. Final word before concluding, Spark undoubtedly performs better than native disk-based Hadoop implementations like Planet but there is a limit of this speed-up factor as available memory cannot be linearly increased proportional to the size of the dataset always. Beyond certain size of the dataset once the available memory is consumed by storing RDD, it spills out to disks storage. At that moment it is similar to Hadoop like disk access implementations neither better nor worse.

References

[1] Quinlan, J. R. C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, 1993.

[2] B. Panda, J. S. Herbach, S. Basu, and R. J. Bayardo. PLANETs: Massively parallel learning of tree ensembles with mapreduce. International Conference on Very Large Data Bases, 2009.

[3] Jeffrey Dean and Sanjay Ghemawat. Mapreduce : Simplified data processing on large clusters. Communications of the ACM, 51:1–13, 2008.

[4] Wu, G., Li, H., Hu, X., Bi, Y., Zhang, J., and Wu, X. (2009, August). MReC4. 5: C4. 5 ensemble classification with MapReduce. In ChinaGrid Annual Conference, 2009. ChinaGrid'09. Fourth (pp. 249-255). IEEE.

[5] Apache Spark Online Guide MLlib - Decision tree, found in: https://spark.apache.org/docs/1.1.0/mllib-decision-tree.html

[6] Quinlan, J. R. (1986). Induction of decision trees. Machine learning, 1(1), 81-106.

[7] Breiman, L., Friedman, J., Stone, C.J., Olshen, R.A.: Classification and Regression Trees. Belmont, California: Wadsworth (1984).

[8] S. R. Safavian and D. Landgrebe. A survey of decision tree classifier methodology. 1990.

[9] Apache Spark. https://spark.apache.org

[10] PySpark Package. http://spark.apache.org/docs/2.2.0/api/python/pyspark.html

[11] UCI Machine Learning Repository dataset. https://archive.ics.uci.edu/ml/index.php

[12] Ankit Desai and Sanjay Chaudhary. 2016. Distributed Decision Tree. In Proceedings of the 9th Annual ACM India Conference (COMPUTE '16). ACM, New York, NY, USA, 43-50. DOI: http://dx.doi.org/10.1145/2998476.2998478.

[13] Ankit Desai, Sanjay Chaudhary. Distributed Decision Tree v.2.0. IEEE International Conference on Big Data (BIGDATA). 2017.

[14] Hua Wang, Bin Wu, Shuai Yang, Bai Wang, and Yang Liu. Research of Decision Tree on YARN Using MapReduce and Spark.

[15] scikit-learn developers. scikit-learn user guide, Release 0.21.dev0.