Distributed Random Forest: Midway Report

Sumeet Mishra
sumish@iu.edu
Indiana University Bloomington

Srinithish K
skandag@iu.edu
Indiana University Bloomington

1 INTRODUCTION

Cluster computing becomes popular when the cost of commodity PC hardware and network equipment dropped. However, there are limitations to cluster computing. Network communication becomes a severe bottleneck, and there always exists the possibility of node failures within the cluster. The Distributed Random Forests algorithm specifically targets implementation in cluster environments and attempts to accommodate the inherent limitations and concerns of using cluster hardware by presenting an algorithm with sparse communication and fault tolerance. It is a powerful classification and regression tool. When given a set of data,it generates a forest of classification or regression trees, rather than a single classification or regression tree. Each of these trees is a weak learner built on a subset of rows and columns. More trees will reduce the variance. Both classification and regression take the average prediction over all of their trees to make a final prediction, whether predicting for a class or numeric value.

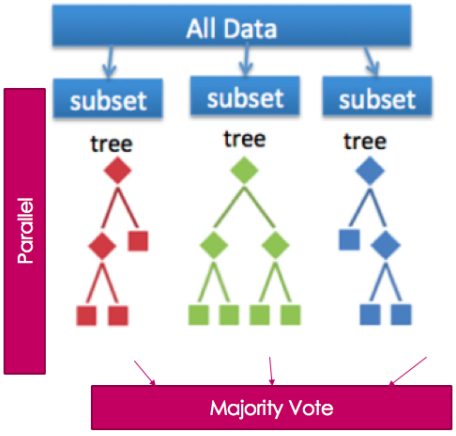

Figure 1: Distributed Random Forest

1.1 Apache Spark

Apache Spark has been one of the most widely used methods of performing large scale batch processing or stream data processing on distributed systems.

This open source analytics engine stands out for its ability to process large volumes of data significantly faster than MapReduce because data is persisted in-memory on Sparks own processing framework. It is highly flexible and scalable with support in many programming languages like Python, Scala, Java ,SQL,R etc and can run on standalone cluster mode, on EC2, on Hadoop YARN, on Mesos, or on Kubernetes. Also, Access data in HDFS, Alluxio, Apache Cassandra, Apache HBase, Apache Hive, and hundreds of other data sources. Due to these extensive support options along with built in libraries for running Machine Learning programs it has become an important platform to run algorithms on distributed systems.

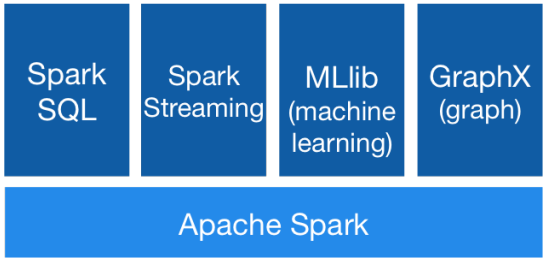

Figure 2: Apache Spark Framework

1.2 Tensor Flow

TensorFlow is an open source software library for high performance numerical computation. Its flexible architecture allows easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices. It creates the whole network with placeholders. Placeholders enable us to separate the network configuration phase and the evaluation phase. It actually helps to scale up the network without any modifications being made.

2 RELATED WORK

We found some related work as follows-
Tao et al. [6] and Lin et al. [7] introduced some classification algorithms for high-dimensional data to address the issue of dimension-reduction. Strobl [8] and Bernard [9] studied the variable importance measures of random forest and proposed some improved models for it. Taghi et al. [10] compared the boosting and bagging techniques and proposed an algorithm for noisy and imbalanced data. Yu et al. [11] and Biau [12] focused on RF for high-dimensional and noisy data and applied RF in many applications such as multi-class action detection and facial feature detection, and achieved a good effort. Basilico et al. [13] proposed a COMET algorithm based on MapReduce, in which multiple RF ensembles are built on distributed blocks of data. A Parallel Random Forest Algorithm for Big Data in a Spark Cloud Computing Environment by Jianguo Chen et al[14]. Random Forests for Big Data by RobinGenuera et al[15] disscusess how out-of-bag error is addressed in parallel adaptations

of random forests.

3 METHOD

3.1 Data

We used Airlines data of year 2008. Since the data does not have a specific target variable, we decided to predict if the flight got delayed at its arrival. Any flight that got delayed by 15 minutes or greater is considered as delayed. A variable 'isDelayed' was created to convert the problem into classification. Variables 'DepTime', 'CRSDepTime','ArrTime', 'CRSArrTime' all had a time stamp in HH:MM format from which we extracted the time in minutes. The variables 'Dest'(Destination) and 'Flight tail number' was dropped as they are unique and the distance travelled was already captured in another variable. Variables 'DepTimeInMins' was also dropped as we were predicting if the flight got delayed and hence we wouldn't know this variable in the real scenario.
After creating the variable IsDelayed we have the following distribution,

Not Delayed	5541790
Delayed	1313239

The total data set has approximately 7 Million rows.

CRSElapsedTime	AirTime	ArrDelay	DepDelay	Distance	TaxiIn	TaxiOut	DepTimeInMins	CRSDepTimeInMins	ArrTimeInMins	CRSArrTimeInMins	IsDelayed
150.0	116.0	-14.0	8.0	810	4.0	8.0	1203.0	1195	1331.0	1345	0
145.0	113.0	2.0	19.0	810	5.0	10.0	474.0	455	602.0	600	0
90.0	76.0	14.0	8.0	515	3.0	17.0	388.0	380	484.0	470	0
90.0	78.0	-6.0	-4.0	515	3.0	7.0	566.0	570	654.0	660	0
90.0	77.0	34.0	34.0	515	3.0	10.0	1109.0	1075	1199.0	1165	1

Figure 3: Few rows of the data

3.2 Spark

3.2.1 *Installation:*

Spark was installed on top of Hadoop's YARN cluster manager. Pyspark an interface to Spark's functions using Python was used to run programs. Configured the maximum data that could be returned to the driver as 6GB from the default 1024 MB.

3.2.2 *Decision Tree Implementation:*

We implemented a Decision tree from scratch as we wanted a greater control over the parameters and functions like getting the best splitting criteria over a variable. Of-the-shelf implementation like Sci-kit Learn, do not give this freedom or its too hard to tweak their functions due to dependencies. Hence we decided to write our own decision tree implementation. We have built decision tree for classification problems. Randomization of decision tree in a random forest can be achieved by sub sampling the data points and considering a subset of the variables or features for a split in a tree.

3.2.3 *Task Parallel approach:*

The most interesting part of Random forests is that the paralleization of the growing of the trees.Since its an ensemble technique once you have the data at every executor , the task of building trees in parallel can happen without much interaction between the executors as the splitting and growing of the tree is only related to that tree and independent of the other trees. Exploiting this feature of the random forest we implemented the paralleization of the building of each decision tree in the forest. Randomization was achieved by sub sampling (bagging) of the data points and storing their corresponding indices in an Resilient Distributed Datasets (RDDs).Note that we do not store the sub samples themselves, but only the indices there by reducing memory foot print. We now get the executors to build decision trees in parallel and get their vote on the test data and store them in an RDD. We now aggregate these votes get the majority by folding the RDD. To reduce the traffic of the data transferred between and to the executors the input data matrix was broadcasted such that the variable is available for all the executers without having to be shipped at every task. Hence building a Parallel Random Forest in Task parallel approach.

3.2.4 *Data Parallel approach:*

To be done:
Second approach is to achieve optimization using Data Parallel approach where we would be vertically partitioning the data on each feature and save it in a Resilient Distributed Datasets (RDD) and hence the best split computation can occur close to the data. We also intend to parallelize the computation of finding the best split.

3.2.5 *Scaling to multiple Nodes:*

To be done

3.3 Tensorflow

- To be done.

4 EXPERIMENTS

All the experiments are conducted on a single node. For each experiment we used 10 decision trees with maximum depth allowed in each tree as 5.

4.1 Sequential approach

We implemented the random forest without parallelzing the tree growth.This serves as benchmark for our experiments. Trees were grown in sequence on a single core.

F1Score	0.8115699
Time for computations(s)	886.382

4.2 Parallel approach

Implemented the tree growth in parallel by allowing spark to exploit all the cores in the node. The Juliet node has 48 cores in each node.

F1Score	0.811978
Time for computations(s)	198.502

4.3 Parallel with modified splitting criteria

As an attempt to be improve the accuracy, we changed the number of split points to be tried while calculating the best possible split. Initially for computational simplicity split thresholds were tried only at the median of the data present at each split. But to improve the accuarcy we allowed the tree to be split at 3 quantiles of the variable. Namely at 25%, 50% and 75%. We improved the accuracy

by 3.5% but at a slightly higer computation time about 30 seconds higher.

F1Score	0.83897
Time for computations(s)	232.465

5 CONCLUSION

Consolidated results so far.

Approach	F1Score	Computation time (s)
Sequential approach	0.81157	886.382
Parallel approach	0.81198	198.502
Parallel with modified splitting criteria	0.83897	232.465

Paralleization of the forest decreased the computation time by a factor of 4. Modifying the splitting criteria allowed to improve the accuracy by 3.5%

Conclusions and experiments for the rest of the proposed methods are yet to be done.

6 ACTIVITIES

6.1 Old Responsibilities

1. Random Forest Implementation: Sumeet and Srinithish
2. Spark Implementation: Srinithish
3. Tensorflow Implementation: Sumeet

6.2 Revised Responsibilities

Completed tasks

1. Random Forest Implementation: Sumeet and Srinithish
2. Spark Installation: Sumeet
3. Spark Implementation : Srinithish

To be done

4. Scaling spark to multi nodes: Sumeet (March 10,2019)
5. Tensorflow Implementation of random forest: Sumeet(March 30,2019)
6.Scaling Tensorflow to multi nodes: Srinithish(April 3,2019)
4.Comparing performance between them:Sumeet and Srinithish(April 15,2019)

7 REFERENCES

(1) http://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/drf.html
(2) http://cs229.stanford.edu/ proj2005/AziziChaiChui- DistributedRandomForests.pdf
(3) https://spark.apache.org/
(4) https://www.tensorflow.org/
(5) http://www.adaltas.com/en/2018/05/29/spark-tensorflow-2-3/
(6) Q. Tao, D. Chu, and J. Wang, âĂIJRecursive support vector machines for dimensionality reduction,âĂİ Neural Networks, IEEE Transactions on, vol. 19, no. 1, pp. 189 to 193, January 2008.
(7) Y. Lin, T. Liu, and C. Fuh, âĂIJMultiple kernel learning for dimensionality reduction,âĂİ Pattern Analysis and Machine Intelligence, IEEE Transactions on, vol. 33, no. 6, pp. 1147 to 1160, June 2011.
(8) C. Strobl, A. Boulesteix, T. Kneib, and T. Augustin, âĂIJConditional variable importance for random forests,âĂİ BMC Bioinformatics, vol. 9, no. 14, pp. 1 to 11, 2007.
(9) S. Bernard, S. Adam, and L. Heutte, âĂIJDynamic random forests,âĂİ Pattern Recognition Letters, vol. 33, no. 12, pp. 1580âĂŞ1586, September 2012.
(10) T. M. Khoshgoftaar, J. V. Hulse, and A. Napolitano, âĂIJComparing boosting and bagging techniques with noisy and imbalanced data,âĂİ Systems, Man and Cybernetics, IEEE Transactions on, vol. 41, no. 3, pp. 552âĂŞ568, May 2011.
(11) G. Yu, N. A. Goussies, J. Yuan, and Z. Liu, âĂIJFast action detection via discriminative random forest voting and top-k subvolume search,âĂİ Multimedia, IEEE Transactions on, vol. 13, no. 3, pp. 507 to 517, June 2011.
(12) G. Biau, âĂIJ Analysis of a random forests model,âĂİ Journal of Machine Learning Research, vol. 13, no. 1, pp. 1063 to 1095, January 2012.
(13) J. D. Basilico, M. A. Munson, T. G. Kolda, K. R. Dixon, and W. P. Kegelmeyer, âĂIJComet: A recipe for learning and using large ensembles on massive data,âĂİ in IEEE International Conference on Data Mining, October 2011, pp. 41 to 50.
(14) A Parallel Random Forest Algorithm for Big Data in a Spark Cloud Computing Environment Article in IEEE Transactions on Parallel and Distributed Systems Âů August 2016.
(15) Random Forests for Big Data RobinGenuera, Jean-MichelPoggib, ChristineTuleau-Malotc, NathalieVilla-Vialaneixd.