Transactions on Machine
Learning and Data Mining
Vol. 10, No. 2 (2017) 67-77
© 2017, ibai-publishing,
ISSN: 1864-9734,
ISBN: 978-3-942952-54-5

Performance Analysis of Sparks Machine Learning Library

Seyedfaraz Yasrobi, Jakayla Alston, Babak Yadranjiaghdam, Nassehzadeh Tabrizi

East Carolina University, Department of Computer Science
Science and Technology Building
Greenville, NC 27858-4353 USA
yasrobis14@students.ecu.edu, alstonja14@students.ecu.edu, yadranjiagh-damb15@studentds.ecu.edu, tabrizim@ecu.edu

Abstract. This paper examines the performance of Apache Sparks machine learning library with reference to the optimal required resources such as the number of machines and cores to best perform popular machine learning algorithms. In order to achieve this, we have observed the training time of classification algorithms such as logistic regression, support vector machines, decision trees, random forests, and gradient boosted trees under different configurations on a sample dataset. Our research revealed that having an excessive number of resources does not necessarily decrease the training time of the machine learning algorithms, rather, it may even degrade the training time by up to 30 percent. Furthermore, this study confirms that methodologies such as tree ensembles can increase the training time of machine learning algorithms compared to that of typical decision trees.

1 Introduction

Machine learning with its automated learning power unleashes big data power to aid data scientists to gain knowledge in a variety of applications such as computer vision, speech processing, natural language understanding, neuroscience, health, and Internet of Things (IoT). The major challenges of using machine learning in big data is to perform the analysis in a reasonable time [28].

In-memory processing with its increased performance in response time has been proven beneficial in a range of areas such as fraud detection, risk management, and decision-making, where machine learning algorithms are used to conduct analytics. Together with real-time analytics, it provides unmatched productivity and profit gains that explains why companies such as Facebook, Twitter, and Amazon take advantage of their capabilities [9, 24].

In-memory processing involves querying data in the computers RAM instead of its hard disk. Being able to interact directly with RAM as opposed to the traditional method of hard disks I/O operations greatly increases response times and throughput. Acker et al. concluded that response times were 5 to 19 times better and throughput increased by 7 times when using in-memory processing platforms. It has been reported that in-memory processing is easier to set up and maintain which is ideal when analyzing large datasets [1], reduction in memory prices, and higher user satisfaction that enables faster data access that contributes to faster decision making. [5, 10, 12].

There are various big data frameworks that take advantage of in-memory processing and of those frameworks the one that is mostly used is Apache Spark, a descendent of Hadoop. Using its in-memory capabilities Apache Spark executes machine learning algorithms efficiently by keeping the dataset in RAM and eliminating the repetitious pulling of data from the hard disk [14]. For example, the machine learning algorithm known as logistic regression is over 100 times faster when using Spark instead of Hadoop [3].

Spark has caused a surge of interest due to the companies' desire to utilize real-time analytics, analysis of streaming IoT data sets, and efficient implementation of iterative algorithms. But regardless of its benefits, the unfamiliarity of many companies with big data cluster specifications such as the number of machines, number of cores per machine, and the interoperability of machines, often results in lower hardware usage efficiency.

Every data analysis ecosystem has two main components which are filesystem and processing system. Spark, handles files in the form of Resilient Distributed Datasets (RDD), and processes them with its in-memory processing engine. In the following sections, we will discuss more of these components as well as our choice of test data set and algorithms.

Resilient Distributed Dataset

RDDs are data structures that partition the dataset into multiple partitions, and aid with parallel computing by supporting iterative operations on the Spark. The resilient feature of RDD that logs the transformations used to build a dataset will capture enough information to re-compute a lost or damaged partition. Furthermore, RDD gives users control over how the data is partitioned, the storage strategy for each RDD, and indicate which RDDs will be reused. Due to its parallel computing capabilities, RDD significantly improves the performance of large dataset computations on an in-memory computing platform [26].

In-memory Processing

In-memory processing has been studied since the 1980s, but recently it has gained popularity due to the availability of ultra-fast memories, each with its massive capacity often offered at the lower cost [23, 13]. There are several significant differences between processing in main memory versus processing on hard disks. The most quantifiable difference is the fact that main memory processes computations notably faster than hard disks [11]. This is because in-memory processing places the computation near the data as a means to reduce data movement [2]. Despite the volatility and vulnerability of main memory, the processing speed and other benefits make using in-memory processing worthwhile.

Data

In this study, Higgs dataset from UCI machine learning repository [4] has been adopted and produced using Monte Carlo simulations. The first 21 features are kinematic properties measured by the particle detectors in the accelerator and the last seven features are functions of the first 21 features; these are high-level features derived to help discriminate between the two classes.

Machine Learning Algorithms

Machine learning is a type of artificial intelligence that provides computers with the ability to learn autonomously, using a combination of methodologies developed by the statisticians and computer scientists, to learn relationships from data while also placing emphasis on efficient computing algorithms. Machine learning is also used in the analysis and diagnosis of medical images in radiological medicine, predict the susceptibility of soil liquefaction, and forecast models of consumer credit risk. The techniques used within these countless applications of machine learning techniques each fall under one of two categories of supervised or unsupervised learning [8, 25, 20, 15].

In this study, we have examined the performance of the following five machine learning algorithms.

Logistic Regression: Linear regression attempts to fit a line to data that has only two levels or outcomes, whereas, logistic regression models the chance of an outcome based on a transformation known as a logit [19]. Spark uses two optimizers that are examined for logistic regression in this study. These optimizers are Stochastic Gradient Descent (SGD) and Limited Memory BFGS (LBFGS).

The SGD approximates the gradient by accessing a single element within the dataset during each iteration. It works best for machine learning algorithms that are large in magnitude because it does not require loading the entire dataset. It is also inexpensive in regard to computations and the dataset can be quickly processed. Those benefits are key factors that influence many to use SGD to optimize logistic regression and SVM algorithms [22]. LBFGS is a quasi-Newton method optimizer that uses prior iterations to estimate the Hessian matrix of the objective function [27]. These prior iterations have a sequence of gradient vectors that are used to solve unconstrained nonlinear minimization problems. Of the machine algorithms that were previously mentioned, logistic regression utilizes LBFGS to optimize its performance.

Support Vector Machine (SVM): The support vector machine algorithm uses training examples to create a hyperplane that separates the dataset into classes. The complexity of classes may vary, but the simplest form of the SVM algorithm has only two possible labels to choose from. To reduce misclassifications, a decision boundary is obtained while training the SVM algorithm. This decision boundary is known as the optimal separation hyperplane [22]. The only optimizer available for SVM on Spark is SGD optimizer.

3) Decision Trees: Classification via decision trees begins with a series of questions about the various features of the dataset. Each question is housed in a node that points to at least one child node that responds to the question. A hierarchal tree is formed as a result of these questions and thus, allowing the classification of an item based on how we answer the questions. A classification occurs once we have reached a leaf node [16].

4) Random Forests: Random forests and boosted trees are other forms of classifiers that are based on decision trees and yield more precise classifications by its multiple decision trees. Random forests are one of the methods of tree ensembles where tree predictors are combined in such a way that each tree prediction depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest. In the process of training a random forests predictor model, the root node corresponds to whole input space and this input space will be partitioned into multiple disjoint partitions. At each of these nodes, a decision tree model will be trained based on its data partition [6].

Gradient Boosted Trees: One of the common methods for improving the accuracy of any machine learning algorithm is by boosting it. This method had been introduced by Sapphire at 1990 and utilizes a combination of weak classifiers in order to create a sturdier classification model. Gradient boosted trees classification algorithm allows for first decision tree model to be trained based on the training dataset, improving the accuracy of the model iteratively by retraining the model. During each training, tuples will be reweighted, so the model can decrease its error rate. Boosting does not suffer from overfitting regularly, but because of its iterative nature takes a longer time to train the model [7].

2 Related Work

Until recently, there has not been much research conducted in regard to in-memory processing. Sparks in-memory capabilities have been shown to be faster and more flexible than Hadoop in a variety of areas [21]. For example, according to a study reported in [26], a workload that took 20 hours on Hadoop was reduced to 3 minutes on Spark. The performance of in-memory processing is analyzed with regard to the number of machines, amount of memory, number of cores per machine, and the amount of pool threads on a variety of workloads [2]. Machine learning is treated as a small aspect instead of a focal point.

Siegal et al. [21] presented a machine-learning library prototype called SmartMLib that allows machine learning implementations to be invoked from a program other than Spark. The authors tested the library's performance and built machine-learning applications that are also provided in Spark's MLlib and tested the performance of k-means clustering, linear regression, Gaussian mixture models, and support vector machines using MLib and Smart-Mlib.

Kumar et al. [17] compared the prediction accuracy of two groups of supervised machine learning ensembles based Meta learners. They showed that Random Forest performs well in both Meta learners: bagging and boosting than other learning algorithms. Lee et al. [18] evaluated the performance of three sampling-based algorithms, four ensemble algorithms, four instance-based algorithms, and two support vector machine algorithms. They conducted two experiments to compare algorithm performance using etching process data and chemical vapor deposition process data, which showed that increasing imbalance ratio leads to higher performance in the instance-based algorithms.

3 Big Data Infrastructure

3.1 Hardware

Big Data analytics require a scalable hardware infrastructure with parallel processing capability. This system should have enough memory, bandwidth, and throughput to run multiple tasks simultaneously, and perform parallel processing of advanced analytics algorithms in a matter of seconds. Since the main concept of Big Data computing is distributed processing, in this study the framework is implemented over a cluster of servers. We have arranged a cluster of 16 servers providing a powerful hardware base for big data analytics tasks. Four of these servers act as administrative nodes, and 12 servers operate as worker nodes. Each of the 16 servers has two Intel(R) Xeon(R) quad-core CPU 5620 2.40 GHz processors, meaning there are eight real cores or 16 virtual cores on each server. The servers are equipped with 16 GB DDR3 RAM and a 1 TB hard disk. The operating system used is a Linux Ubuntu server 14.04 64-bit distribution. The switch used is Juniper EX4200, which is a high-performance, low-latency, and provides a one Gigabit Ethernet (GbE) access environment.

3.2 Software

To analyze the performance of machine learning algorithms on Spark, we began by building a Hadoop cluster using YARN as the resource manager. YARN is a platform that provides consistent operations, security, and data governance tools across Hadoop clusters. We chose YARN as the resource manager because it outperforms Sparks standalone mode for handling clusters. YARN allows for multi-tenancy, dynamic allocation of cluster resources, and data center expansion [21]. A bash script was created that ran machine learning models on the HIGGS dataset with data size of 8 gigabytes, 28 attributes, and 11 million instances [4]. To test the performance of the machine learning algorithms, we utilized a maximum of 7 machines with 8 virtual cores and 16 gigabytes of RAM. The amount of data partitions was also considered when analyzing the performance of the machine learning algorithms. There was a range of 40 to 60 partitions used. The performance time of the machine learning algorithms was first averaged regarding the number of machines and then averaged by the number of cores. 1) Spark: Spark is a framework for the parallel processing of Big Data. Spark is designed to use Hadoop MapReduce with some modifications that enable it to perform more efficiently. Apache Spark has its own streaming API and independent processes for continuous micro-batch processing across intervals with varying, but short time duration. Spark runs up to 100 times faster than Hadoop in certain circumstances. Spark has some features for real-time analytics and is supportive of applications such as machine learning, stream processing, and graph computation [26]. MLlib is Spark's machine learning library, focusing on learning algorithms and utilities, including classification, regression, clustering, collaborative filtering, dimensionality reduction, as well as underlying optimization primitives. MLlib is built on Apache Spark, which is a fast and general engine for large-scale processing that is up to 100x faster than Hadoop MapReduce, or 10x faster compared to disk. It supports Java, Scala, and Python.

4 Evaluation

4.1 Experimental Condition

An experiment was setup to investigate the performance of the machine learning algorithms with 128 different configurations, a number of servers 3 through 10, and number of virtual cores 1 through 16 with data chunk size of 64 megabytes. The minimum number of servers needed to perform machine learning algorithms were selected to conform with the replication factor of 3 that refers to the storing of data on different machines.

4.2 Results

In the process of training the classification models, we observed that the split rate of selected data for training data set does not effect the training time of the machine learning algorithms. This means that the only parameters affecting the training time of classification models for particular dataset and infrastructure are software parameters and the resources that are being used. We have discovered that the analysis won't complete if we don't provide enough RAM for the execution, and on the other hand adding excessive RAM will not increase the performance.

In this experiment, we trained the logistic regression model with two LBFGS and SGD optimizers, with over 100 different configurations. The average training time of SGD was 207 seconds and the average time of LBFGS was 106 seconds, as shown in Fig. 1, so we can conclude that logistic regression model can be trained 49 percent faster using LBFGS optimizer compared to SGD while being trained with similar cluster configurations.

As expected, there was an observed increase in the performance of machine learning algorithms as the number of servers increased. It is noteworthy to state, however, that adding additional servers will increase the performance, but decaying decreasing performance with higher number of servers and adding these additional servers may outweigh the performance benefit. The results of this experiment are presented in Fig. 2.

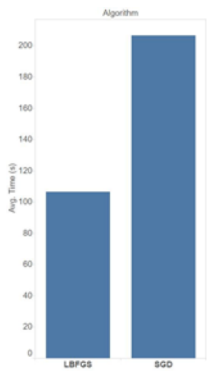

Fig. 1. Comparison of average training time for logistic regression using SGD and LBFGS

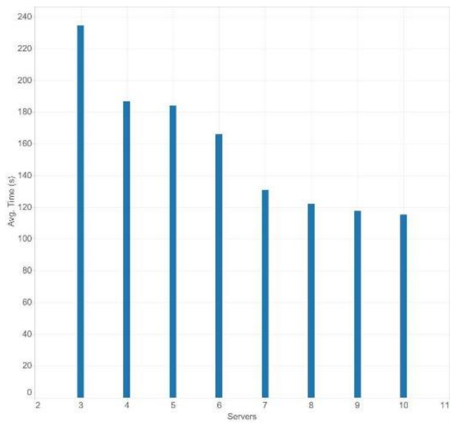

Fig. 2. Time comparison of classification algorithms on different number of servers

We observed that having more than one virtual core makes significant difference in the training time and having 2 through 8 virtual cores performs equally well. Furthermore, increasing the number of virtual cores used on each server to more than 8 degrades the performance of machine learning algorithm. Therefore, we conclude that the optimum number of virtual cores is between 2 and 8, independently selected algorithm, next we compared training time based on the number of servers.

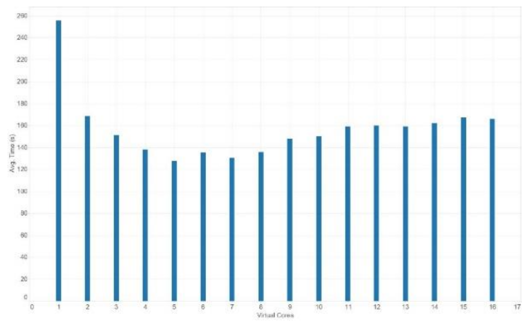

Fig. 3. Comparison of average training time of classification algorithms per number of virtual cores on each server

Interestingly, as shown in Fig. 3, the relationship between the number of cores and the performance of the machine learning algorithms did not follow a similar pattern to that of the number of machines and performance. This issue raises concern regarding Spark and YARNs ability to manage resources in training of machine learning algorithms.

We further analyzed results using tree ensembles and as was expected random forests capability to adapt with parallel processing, will allow training higher number of subtrees for training predictor model. Fig. 4 shows the average training time for this algorithm using 5, 25 and 50 subtrees.

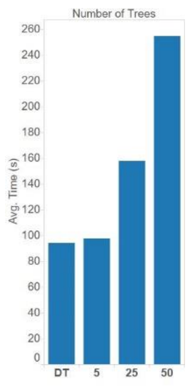

Fig. 4. Training time Comparison of decision tree and random forests with 5, 25, 50 subtrees

We Tested Gradient boosted trees with a different number of iterations and as expected an increasing number of iterations significantly increased its training time. Also, because of the iterative nature of this algorithm, parallelism and cluster computing only marginally helps with its boosting. As you can see in Fig. 5 the training time of gradient boosted tree with even few iterations increases rapidly.

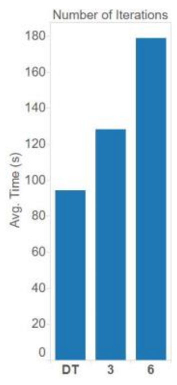

Fig. 5. Training time Comparison of decision tree and gradient boosted trees with 3 and 6 iterations

5 Conclusion

We have presented the optimal server and core configurations needed to the best performing machine learning models using Apache Spark on a specific dataset. These optimal configurations will not only save time but will also reduce the cost of infrastructure. The performance of these machine learning algorithms was analyzed in regards to time in seconds. Factors such as the number of cores, number of machines, and the number of partitions were taken into consideration when analyzing the performance of the machine learning algorithms. It was discovered that as the number of servers increased, the performance increased as well, but performance, however, decreased, as the number of cores increased. These results infer that more servers and fewer cores are needed for optimal performance of machine learning algorithms. From a business perspective, it is best to use the number of machines that make a significant impact on performance rather than a meager impact as means to save money. This justifies why we chose several servers that was less than the maximum number of machines that were available. We wanted to maximize benefits and resources while minimizing expenditures.

Acknowledgements

This research was supported in part by Grant #1262933 and #1560037 from the National Science Foundation.

References

1. Acker O., Gröne F., Blockus A., Bange C.: In-memory analytics strategies for real-time CRM, Journal of Database Marketing & Customer Strategy Management 18(2), 129–136 (2011).
2. Ahn J., Yoo S., Mutlu O., Choi K.: PIM-enabled instructions. In: Proceedings of the 42nd Annual International Symposium on Computer Architecture - ISCA, pp. 336–348, ACM Press, New York, USA (2015).
3. Apache Software Foundation (2015). Apache Spark - Lightning-fast cluster computing. URL http://spark.apache.org/
4. Baldi P., Sadowski P., Whiteson D.: Searching for exotic particles in high-energy physics with deep learning. Nature communications 5, 4308 (2014).
5. Bärenfänger R., Otto B., Osterle H.: Business value of in-memory technology-Multiple-case study insights. Industrial Management and Data Systems 114 (9), 1396–1414 (2014).
6. Cutler D.R., Edwards T.C., Beard K.H., Cutler A., Hess K.T., Gibson J., Lawler J.J.: Random forests for classification in ecology. Ecology 88(11), 2783– 2792 (2007).
7. Death G.: Boosted trees for ecological modeling and prediction. Ecology 88 (1), 243-251 (2007).
8. Deo R.C.: Machine learning in medicine. Circulation 132(20), 1920 – 1930 (2015).
9. Frenkiel E.: Want to Rise Up in Your Organization? Real-Time Big Data Analytics Is the Wave to Catch - Database Trends and Applications. Database Trends and Applications 28 (4), 27 (2014).
10. Garber L.: Using in-memory analytics to quickly crunch big data. Computer 45 (10), 16–18 (2012).
11. Garcia-Molina H., Salem K.: Main Memory Database Systems: An Overview. IEEE Transactions on Knowledge and Data Engineering 4(6), 509–516 (1992).
12. IBM Homepage, what is In-memory computing, https://www.ibm.com/analytics/us/en/technology/data-warehousing
13. Jin C., Kong Y., Kang Q., Qian W., Zhou A.: Benchmarking in-memory database. Frontiers of Computer Science 10 (6), 1067–1081 (2016).
14. Kestelyn J.: Putting Spark to Use: Fast In-Memory Computing for Your Big Data Applications. Cloudera Developer Blog, http://blog.cloudera.com/blog/2013/11/putting-spark-to-use-fast-in-memory-computing-for-your-big-data-applications (2013).
15. Khandani A.E., Kim A.J., Lo A.W.: Consumer credit-risk models via machine-learning algorithms. Journal of Banking and Finance 34(11), 2767– 2787 (2010).
16. Kingsford C., Salzberg S.L.: What are decision trees? Nature biotechnology 26(9), 1011–1013 (2008).
17. Kumar S., Pandey M.K., Nath A., Subbiah K.: Performance Analysis of Ensemble Supervised Machine Learning Algorithms for Missing Value Imputation. In: 2016 2nd International Conference on Computational Intelligence and Networks (CINE), IEEE, pp. 160–165 (2016).
18. Lee T., Lee K.B., Kim C.O.: Performance of Machine Learning Algorithms for Class-Imbalanced Process Fault Detection Problems. IEEE Transactions on Semiconductor Manufacturing 29(4), 436–445 (2016).
19. Sainani K.: Logistic Regression. PM&R 6(12), 1–28 (2006).
20. Samui P., Sitharam T.G.: Machine learning modelling for predicting soil liquefaction susceptibility. Natural Hazards and Earth System Science 11(1), 1–9 (2011).

21. Siegal D., Guo J., Agrawal G.: Smart-MLlib: A High-Performance Machine-Learning Library. In: IEEE International Conference on Cluster Computing (CLUSTER), IEEE, pp. 336–345 (2016).
22. Sopyla K., Drozda P.: Stochastic gradient descent with Barzilai-Borwein update step for SVM. Information Sciences 316(C), 218–233 (2015).
23. Tan K.L., Cai Q., Ooi B.C., Wong W.F., Yao C., Zhang H.: In-memory Databases. ACM SIGMOD Record 44(2), 35–40 (2015).
24. Trajano A.F.R., Fernandez M.P.: Two-phase load balancing of In-Memory Key-Value Storages through NFV and SDN. In: IEEE Symposium on Computers and Communication (ISCC) IEEE, pp. 409–414 (2015)
25. Wang S., Summers R.M.: Machine learning and radiology. Medical Image Analysis 16(5), 933–951 (2012).
26. Zaharia M., Chowdhury M., Das T., Dave A.: Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In: EECS 2011-82 (2012).
27. Zheng W., Bo P., Liu Y., Wang W.: Fast B-spline curve fitting by L-BFGS. Computer Aided Geometric Design 29(7), 448–462 (2012).
28. Zhou L., Pan S., Wang J., Vasilakos A.V.: Machine Learning on Big Data. Opportunities and Challenges. Neurocomputing 237, 350-361 (2017).