Comparing Machine Learning Algorithms Across Various Data Systems

Grace Davenport, Hayden French, Silas Hayes, and Ryan Lipps
University of Virginia
mha4rh, hmf9kx, sjh7yg, rhl8pk@virginia.edu

Abstract

This paper explores the potential of various data systems for implementing machine learning algorithms, focusing on their performance in terms of speed and scalability. With a particular interest in decreasing compute time for machine learning tasks, we compare the efficacy of two data systems–Dask and Spark–in deploying three common algorithms: Linear Regression, Logistic Regression, and K-Means. The experimental setup involves deploying four EC2 t3.large instances, each with 2 CPUs and 8GB of RAM, and running the algorithms separately on each system. We consider speed and scalability in our comparison of these two systems. We find that Dask has lower runtimes than Spark across all algorithms for up to 2 GBs of data but the systems differ in their scalability. This paper aims to provide quantitative insights into the trade-offs between different distributed computing systems, thereby enhancing understanding of their strengths and weaknesses.

1 Introduction

Machine learning algorithms often encounter prolonged compute times when operating on large datasets, presenting a significant challenge for data scientists. As M.S. in Data Science students, we find it extremely practical to optimize computational resources to improve machine learning workflows. In this study, we evaluate the suitability of different distributed computing systems for implementing typical machine learning algorithms.

To systematically investigate the performance of these data systems, we implement an experimental design deployed on Amazon Web Service's EC2 instances. Leveraging Ubuntu as the operating platform, we execute three prominent machine learning algorithms—Linear Regression, Logistic Regression, and K-Means—within a Jupyter Notebook environment.

Existing theory and prior knowledge suggest that distributed computing, by leveraging parallelization, holds promise in reducing the runtime of tasks involving large datasets. In particular, the algorithms that use gradient descent such as Linear Regression and Logistic Regression, benefit greatly from parallel execution.

Through this empirical study, we provide evidence on the comparative performance of Dask and Spark. By conducting identical experiments across these systems, we report their respective strengths and weaknesses, thereby aiding data scientists in making informed decisions regarding their choice of data systems for machine learning tasks. Ultimately, we hope that our findings will enhance the efficiency of machine learning workflows in real-world applications.

2 Methodology

To compare performance across the software implementations, we used a single dataset consistent across algorithms and data systems. The dataset represents 20 million flight searches between June 2022 and August 2022 and is 2.0 GBs. Each row is one instance of a flight search on a specific date, returning base fare, distance and time traveled, and how many seats are remaining. For consistency, we performed data preprocessing on the entire dataset before transferring it to the cluster. After cleaning, we centered each variable by Z-scaling in order not to favor algorithms that perform better on non-centered, non-scaled data. We then performed an 80:20 train/test split.

For Linear Regression, we predicted the base price of a flight. For logistic regression, we split base price into a binary variable, cheap vs. expensive. Flights below the mean were labeled "cheap" and flights above the mean were labeled "expensive". We performed unsupervised learning via K-means to cluster flights. All of our predictor variables are quantitative.

At first, we sought to assess all of the algorithms and systems using Python's built-in "timeit" function, and did so for Dask. This provided a mean and standard deviation for the runtime on 7 iterations. Across all of the models, at varying proportions of the dataset, the standard deviation was typically around 2% of the mean. Because this value was low across all of the models and running "timeit" increases the overall

run-time seven-fold, we decided that timing each model once would suffice.

Originally, we attempted to implement these three algorithms with sklearn, as we have extensive prior experience with this package and the documentation for Dask, Spark, and Ray promised easy integration. In fact, there was a lack of compatibility across some of the system-algorithm pairings, forcing a switch to Dask and Spark's own machine learning backend systems. As Ray does not have its own ML implementation for these algorithms, we compared the three algorithm performance times on Dask and Spark only.

To reflect typical modeling workflow, we calculated model performance metrics for all three algorithms on both Dask and Spark, including these calculations in our runtime measurements. The difference between these metrics was negligible when comparing algorithm performance between distributed systems.

We deployed each of our experiments on a cluster of four EC2 instances. It is important to note that Dask and Spark utilize the machines in the cluster differently. Dask assigns one machine to a scheduler and deploys three workers; whereas Spark uses one machine as a master node to generate a Directed Acyclic Graph (DAG) outlining the task workflow, and then deploys all four machines as workers. This difference in resource allocation had a significant impact on our results below. For the Dask implementation, the data were copied onto each machine, whereas with Spark the data were stored in Hadoop Distributed File System (HDFS) hosted on the four-machine cluster.

3 Experimental Results

3.1 Dask

Figure 1 displays the runtimes for each of the three algorithms for increasing proportion of the dataset using Dask. As expected, as the dataset size increases, the computational requirement, and therefore runtime, also increases. Out of the three algorithms tested, K-Means has the highest total runtime, at 326.70 s (Table 1). All algorithms scale in a linear fashion, but K-Means appears to be the least scalable. The algorithm runtime increases at a higher rate for K-Means compared to Linear and Logistic Regression.

3.2 Spark

Figure 2 displays the runtimes for each of the three algorithms for increasing proportion of the dataset using Spark. Similar to Dask, the larger the dataset, the longer the Spark runtime. Linear Regression has the lowest total runtime (144.91s) and K-Means has the highest total runtime (691.83 s) (Table 1). Both Linear and Logistic Regression algorithms scale linearly. Whereas, the K-Means rate of change is more variable as the

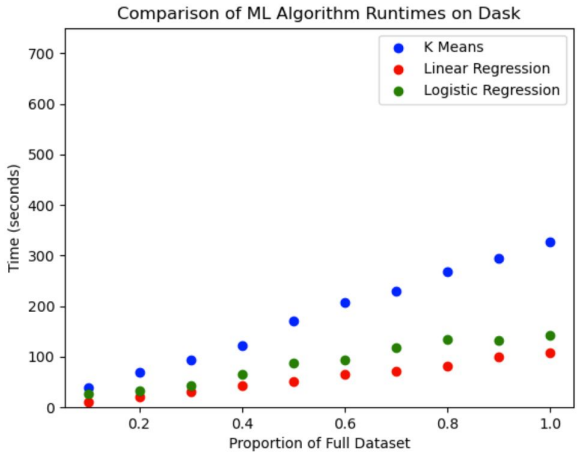

Figure 1: Dask Algorithm Comparison. All algorithms appear to scale linearly, but K-Means is the least scalable.

Table 1: Time comparison between Dask and Spark

Model	Dask	Spark
Linear Regression	108.04 s	144.91 s
Logistic Regression	142.80 s	279.03 s
K-Means	326.70 s	691.83 s

dataset size increases. At 10% of the total data, all three algorithms are around a 100 second runtime. As the dataset size increases, Linear Regression runtime only slightly increases, indicating a large Spark overhead time cost.

3.3 Comparison

For direct comparison of performance between Dask and Spark, we provide graphs for each of the three algorithms with times for each system in Figures 3-5. These figures have linear regression best-fit lines with their corresponding equations, with Table 2 showing the R^2 values for these equations. Most of the R^2 values are greater than 0.95, with the exception of K-Means on Spark, which has an R^2 of 0.883. This indicates that most of the variance in runtime for our experiments can be explained by dataset size.

For K-Means, Dask appears to scale better, as the Spark slope, 550.39, is larger than the Dask slope, 325.26 (Figure 3). For Linear Regression, Spark appears to scale better, as the Spark slope is 42.97, compared to the Dask slope, 107.23 (Figure 4). For Logistic Regression, Dask and Spark appear to scale similarly, as both slopes are around 150 (Figure 5).

4 Discussion

The variance in Spark's performance on K-Means could be due to different initializations of the algorithm. K-Means is

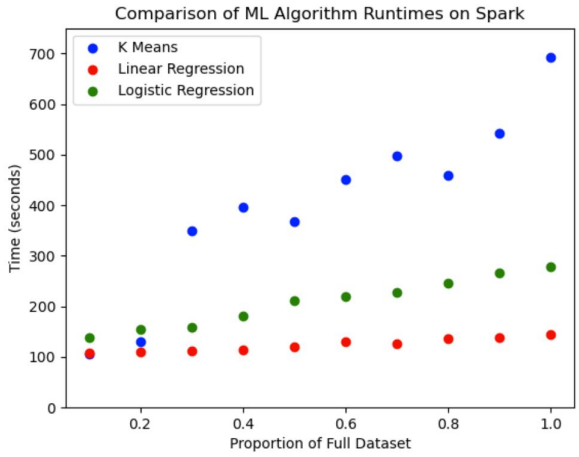

Figure 2: Spark Algorithm Comparison. Linear and Logistic Regression appear to scale linearly. K-Means efficiency is highly sporadic as the dataset size increases.

especially sensitive to different initializations, as random starting points with cluster centers far from the actual centers in the data will take longer to converge. Interestingly, both Spark and Dask's documentations cite *k-means||* as the method of initialization. [1] [2] Nonetheless, given our experiments only consist of 10 runs for each system-algorithm pair, it is possible that Dask was lucky or Spark was unlucky in randomizing initializations. We offer some solutions to this issue in the "Future Work" section below.

Across all models, Spark has a roughly 95-second overhead, likely due to job scheduling and network I/O times. Since the data for Spark were stored on HDFS without explicit partitioning, each initialization of the model would require each machine to retrieve the relevant data from the distributed file system, incurring this network overhead. In contrast, each machine had a local copy of the full dataset in the Dask implementation. This introduces an interesting trade-off in the performance of these systems: while Spark takes longer to run than Dask across the board, Dask requires more space in order to store the data on each machine. For the purposes of our experiment, this storage space is negligible; however, this would be an important consideration in production computing clusters. It is also possible this overhead is due to Spark computing the workflow on the master node before launching workers, and then deploying all machines on computation; whereas, Dask appears to begin computation immediately, at the cost of assigning a scheduler node that does not carry out any computation.

Despite the startup overhead causing Spark to run longer than Dask for all of our experiments, each algorithm scales differently on both systems. As shown in Figure 3, K-Means appears to scale better on Dask than Spark. Figure 4 shows that Linear Regression scales better on Spark, and Figure 5

Table 2: R^2 Values for Regression Equations

Model	**Dask R^2**	**Spark R^2**
Linear Regression	0.996	0.956
Logistic Regression	0.971	0.987
K-Means	0.997	0.883

shows Logistic Regression scales similarly on both systems, with Dask scaling slightly better. It is unclear why this difference in scalability exists.

Lastly, it is especially notable that Dask outperforms Spark across all experiments despite having one less worker than Spark. As mentioned above, Dask utilizes three workers with one scheduler, whereas Spark eventually deploys all machines as workers after generating the workflow DAG. Dask's better performance across the board is especially interesting given the difference in scaling between algorithms. Because Spark leverages more workers, we would expect Spark to scale better but our results show this is true for only one algorithm, linear regression.

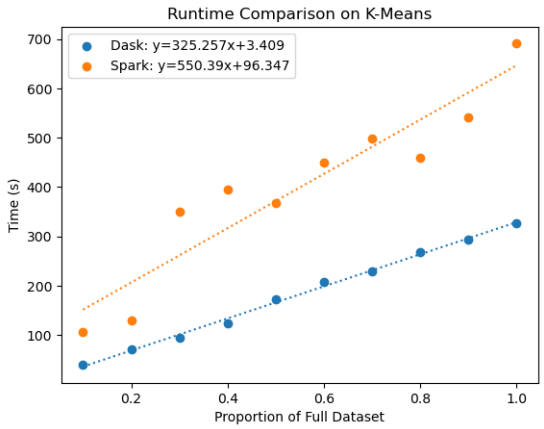

Figure 3: K-Means Comparison. Dask scales better for input size than Spark on this dataset. Spark has more variance than Dask, possibly due to different cluster center initializations.

5 Future Work

To further assess these systems, varying the number of virtual machines, and the size of the data, will provide valuable insights into optimizing the number of machines. This will especially be insightful as different systems allocate virtual machines in different ways. For example, Dask requires a head node but performs only scheduling work on that node, whereas Spark performs both scheduling and compute work on the head node.

Additionally, as discussed above, Sklearn support on these systems is sparse at best. To adequately assess Sklearn's parallelization across these systems, a custom-built backend that

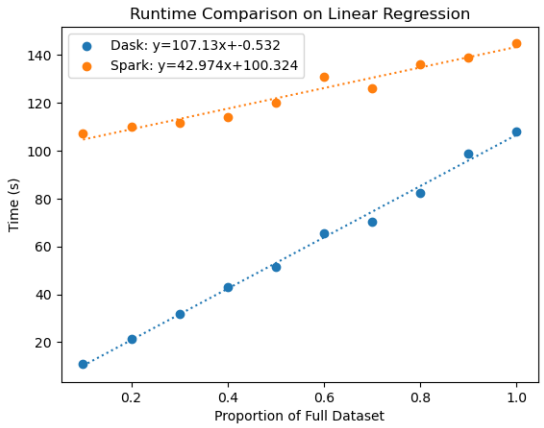

Figure 4: Linear Regression Comparison. Spark scales better than Dask for this data, though Spark's additional overhead means it takes longer than Dask for all sizes.

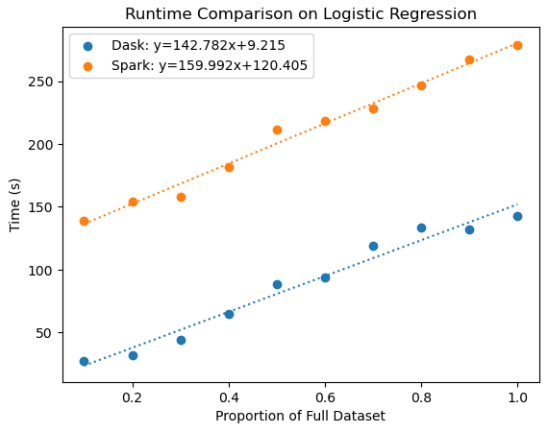

Figure 5: Logistic Regression Comparison. Both systems scale similarly for Logistic Regression, with Dask having a slightly lower slope. Spark's additional overhead persists, causing it to take longer than Dask for all sizes.

utilizes all available virtual machines evenly would give us better insight into how Sklearn performs in a best-case setup for each system.

Finally, to improve the experimental design used in this paper, running multiple trials would provide a mean time for each system, as well as a standard deviation. While we believe these standard deviations would be small based on our previous findings, it would increase confidence in the results. Increasing the size of the data would also confirm or refute the trends we see in the scalability of each system—which are only analyzed from 0.2 to 2.0 GBs currently.

6 Conclusion

This research addresses the challenge of optimizing computational resources for efficient machine learning workflows when dealing with large datasets. By examining the performance of two prominent distributed systems, Dask and Spark, in deploying common machine learning algorithms—Linear Regression, Logistic Regression, and K-Means—we qualitatively assess the speed and scalability of these systems.

The evidence demonstrates that while both Dask and Spark offer advantages in distributed computing, they exhibit differences in their scalability depending on the algorithm. Dask has the lower runtimes across all algorithms and dataset sizes with its approach that minimizes overhead, copying data onto each machine. On the other hand, Spark exhibits slightly longer runtimes due to its job scheduling and network overhead. The scaling for these platforms differs depending on the algorithm tested. Spark scales best for Linear Regression, Dask best for K-Means, and they scale equally for Logistic Regression. These findings reveal the importance of considering dataset size and machine learning algorithm when choosing a distributed computing system for machine learning tasks.

7 Metadata

The presentation of the project can be found at:

https://drive.google.com/file/d/1A9o7MDaDPks_SJzND7uMLERy8NmXb818/view?usp=sharing

The code/data of the project can be found at:

https://github.com/rlipps/DS5110_Final_Project

References

[1] Dask ML Developers. Dask ML: KMeans Documentation. https://ml.dask.org/modules/generated/dask_ml.cluster.KMeans.html. Accessed: [April 28, 2024].

[2] Apache Spark. MLlib: Clustering - Spark 3.1.2 Documentation. https://spark.apache.org/docs/latest/ml-clustering.html. Accessed: [April 28, 2024].