ML in Big Data: Experimenting Logistic Regression Performance Using Scikit-Learn and Spark MLlib

Mengying Bi
University of Washington
+1(206) 953 8981
mybi@uw.edu

ABSTRACT

I report on the efficiency of performing traditional Machine Learning tasks on local machine via Python Scikit-learn package (sklearn) versus on AWS via Spark MLlib package, from the aspects of runtime and scalability, on different size of datasets. The project uses the big data system Apache Spark on Amazon Elastic Map Reduce with different cluster size setups as an aggregate query to run typical Machine Learning algorithm (gradient descent via Logistic Regression in this paper) until convergence, and compare this to the standard approach of using python for gradient descent on local computer.

The result turns on to be that stand-alone Scikit-learn on local machine is faster on accessing local data and training Logistic Regression when the dataset is small for dataset sizes larger than 15MB, Spark cluster with 8 nodes are noticeably faster than 4 nodes, while Spark cluster with 4 nodes does not have large distinction comparing to 2 nodes cluster until the dataset size gets to 100MB and up. Spark would throw OutofMemory error due to too large of memory for dataset that is greater than 10GB in file size, therefore we consider it out of the project scope with limit time and resource.

Keywords

Machine Learning; Logistic Regression; Apache Spark; Amazon Elastic Map Reduce; MLlib; Scikit-learn; Python.

1. INTRODUCTION

With the introduction of Internet of Things, cloud storage, and cloud computing system, the quantity of data available and analysis capacity has been growing exponentially, machine learning is being learned and used to enable automated the building of analytical models. The algorithms learn iteratively from the data and allow hidden insights without explicitly specifying what to compute. The scikit-learn package in Python is implemented for this purpose and has been extensively used among Data Scientists. This method does help lots of data scientists to train their models within a descent amount of time when the data size is not considerably large. However, in the era of Big Data, using standalone scikit-learn would easily take hours even days to train a rather simple model over a dataset of a size over 100MB which limits the efficiency of machine learning model training. Apache Spark, as an open source big data processing framework, is used to reduce the runtime and increase scalability of the model training process.

In this paper, I evaluate the runtime and scalability of using scikit-learn and MLlib to train binary classification models using Logistic Regression algorithm over different size of datasets. By recording the runtime of different functions with different size of the data, I was able to explore which setup works the best under what kind of condition.

Section 2 describes the overall architecture of Spark and MLlib. Section 3 describes the problem statement and methods I use in this project. Section 4 presents a comprehensive comparative analysis. Section 5 summarizes the conclusion addressed from comparing Spark MLlib on AWS EMR to Python Scikit-learn on local computer.

2. SYSTEM EVALUATION

2.1 Spark and MLlib

Spark is a distributed processing framework and programming model and has an optimized directed acyclic graph (DAG) execution engine and actively caches data in-memory, which can boost performance, especially for certain algorithms and interactive queries.[1]

Basically, Spark uses a cluster manager to coordinate work across a cluster of instances. To initiate a spark session, we need to first set up a spark application consist of a driver process and executor processes (master node and cores): master runs the main function and distributes work across the cores, therefore enables the parallel processing capability and increase the task efficiency.

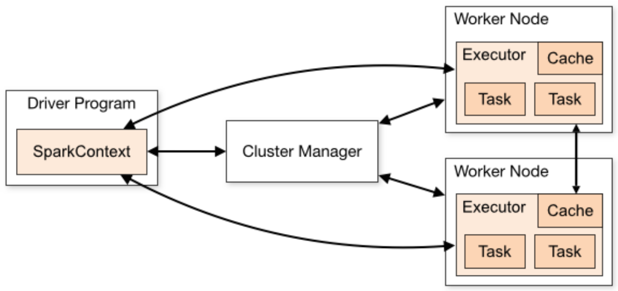

Figure 1. Spark Cluster Mode Architecture [2]

In Apache Spark, we use MLlib, a scalable machine learning library in Spark, consists of common learning algorithms and utilities, including classification, regression, clustering, etc., which is parallel to sklearn in Python. The workflow of Apache Spark MLlib library is showed in Figure 2.

2.2 Python Scikit-Learn

Scikit-learn is a simple and efficient tool to predictive data analysis, which is built on NumPy, SciPy, and matplotlib. Unlike Spark MLlib, Python and Scikit-Learn do in-memory processing in a non-distributed fashion. In practice, to optimize the code

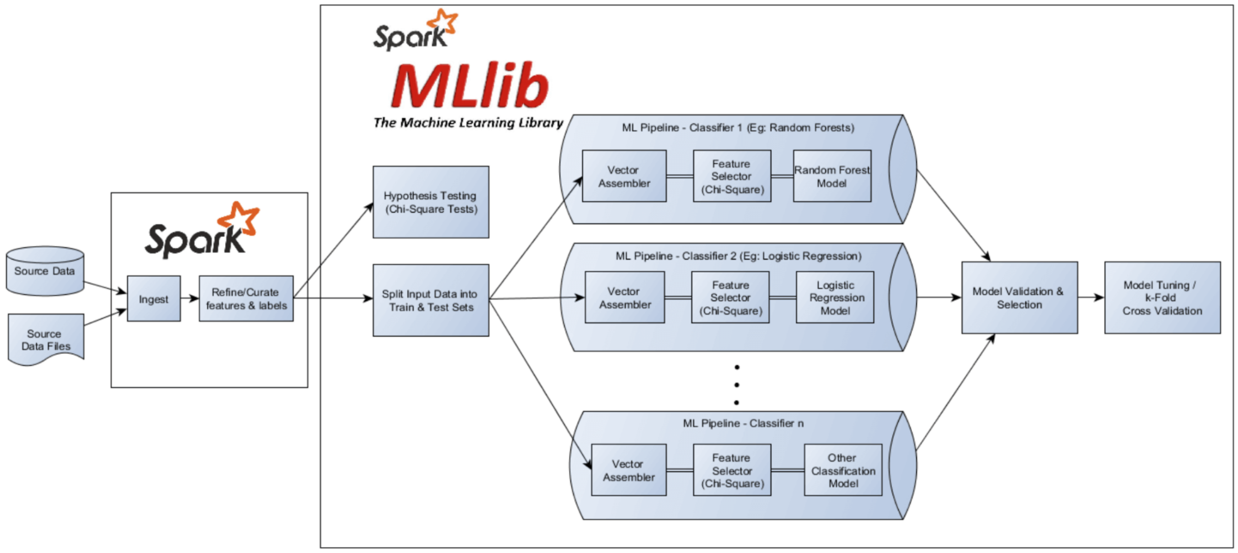
Spark
MLlib
The Machine Learning Library
Source Data
Source Data Files
Spark
Ingest
Refine/Curate features & labels
Hypothesis Testing (Chi-Square Tests)
Split Input Data into Train & Test Sets
ML Pipeline - Classifier 1 (Eg: Random Forests)
Vector Assembler
Feature Selector (Chi-Square)
Random Forest Model
ML Pipeline - Classifier 2 (Eg: Logistic Regression)
Vector Assembler
Feature Selector (Chi-Square)
Logistic Regression Model
ML Pipeline - Classifier n
Vector Assembler
Feature Selector (Chi-Square)
Other Classification Model
Model Validation & Selection
Model Tuning / k-Fold Cross Validation

Figure 2. Spark MLlib workflow. (Source: qubole)[2]

performance in Scikit-learn, we would try to replace nested for loops by calls to equivalent Numpy array methods. The goal is to avoid the CPU wasting time in the Python interpreter rather than crunching numbers to fit the statistical model by expressing an algorithm in simple vectorized Numpy code.[3]

2.3 MLlib and Scikit-learn Comparison

Both MLlib and Scikit-learn offer very convenient tools for building text vectors. Sometimes when using some parallel algorithms on vectorized data, the number of cores used might be reduced in a corresponding parameter and slow down the system. Scikit-learn is already a high-performance tool for machine learning as long as there's enough RAM to allocate the data. Otherwise, when the data file is huge, switching to Spark MLlib would be able to speed up the system.

2.4 AWS EMR

Since I run Spark on AWS EMR clusters, I would briefly talk about the architecture of EMR clusters.

Amazon EMR has a central component cluster, which is a collection of Amazon Elastic Compute Cloud (EC2) instances. Each instance is called a node, and has a role within the cluster (master, core and task). [4]

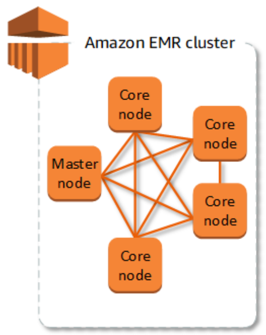

Figure 3. An EMR cluster with one master node and four core nodes

3. PROBLEM AND METHOD

How does Spark MLlib with different cluster sizes perform comparing with Python Scikit-learn on Machine Learning tasks? How is the ease of configuration process & data preprocessing of requirements of the two methods? What are the runtimes for accessing the data, training the model? How is the Scale-up and speed-up performance?

With these questions in mind, I researched on whether capability of parallel processing of Spark does enable shorter processing time, and if modifying the cluster size would give different results on runtime.

3.1 Experimental Setup

The experiments on local machine used an Intel Core i5 1.8GHz/64bit /4GB with macOS Mojave 10.14.6 Macbook Air.

The experiments on AWS used EMR clusters with 2, 4, and 8 nodes respectively with master node of type M5.xlarge instance type (emr-5.27.0, Hadoop 2.8.5, and Spark 2.4.4).

I report the runtime of accessing the data, training the model (and k-fold cross validation only on EMR instances, since the runtime is too long on local machine) on each setup respectively by running each system once and then report the average of three runs with warm cache using the Python time package.

3.2 Data Profiles

I experimented with real-world datasets that are publicly available online in different size. The 1.2GB dataset came from concatenating the 100MB flight dataset 10 times.

Table 1. Size of the datasets

Name	File Size	# of Columns	# of Rows
Diabetes	23.3KB	768	9
Income	3.8MB	32,561	17
Poker-hand	18MB	800,000	13
Flight	100MB	1,048,575	29
Flight Data Concatenated	1.2GB	11,534,325	31

3.3 Steps and Process

When training logistic regression model on local computer, I took the steps as follow:

- Download dataset to local disk;
- Read the csv data file as data frame using python Pandas library;
- Pre-process data, including dropping missing values, transform categorical values into numerical values using one hot encoding, and standardizing the data;
- Choose features and train test split
- Train the model.

The steps I used to develop a binary classification model in Spark using a User Defined Function (UDF)involved are as follows:

- Upload datasets to AWS S3 bucket;
- Initialize a Spark session;
- Read the dataset into a Spark data frame;
- Pe-process data, including casting all integer values to floats, handling missing values, transforming dataset using vector assembler, and scalarizing the features;
- Train test split and imbalance handling;
- Train the model;
- Performance evaluation;
- Resize the cluster and rerun the code above.

The main differences between the two methods are:

- The ease of accessing data,
- Spark MLlib has more requirements regarding data type of the variables
  E.g. It throws exceptions "Column label must be of type DoubleType but was actually IntegerType" if label/variables are of integer type.
- Spark needs to assemble all selected features into a feature column using Vector Assembler, and
- I added the model performance evaluation step to the Spark process using K-fold cross validation function since its capability of parallel processing enables faster processing time, which would take standalone python hours even days to process.

3.4 Data Points Recorded

On local machine, I recorded the runtime of loading the datasets into data frame and training logistic regression model; while on AWS EMR instance, I experimented the data loading, model training, and cross validation and recorded the runtime respectively.

4. RESULTS

4.1 Accessing Data

From the plot, it is safe to say that accessing data from local is extremely fast when data size is small, however the runtime increases tremendously as data size gets larger. When the data size is less than or equal to 100MB, local assessing data is around 700 times faster than spark loading data from S3. When data size gets to 1GB, loading data on local machine takes 2 times or more than any cluster size of that on EMR cluster. The performance improvement as node size increase began to show when the data size gets to 1GB.

Table 2. Runtime of Accessing the Data (in seconds)

	20KB	4MB	20MB	100MB	1GB
Local	0.02	0.12	0.77	4.89	82.75
Spark-2nodes	7.81	8.47	10.22	15.52	45.83
Spark-4nodes	10.99	8.21	11.78	14.42	32.19
Spark-8nodes	9.86	12.14	10.03	13.64	21.20

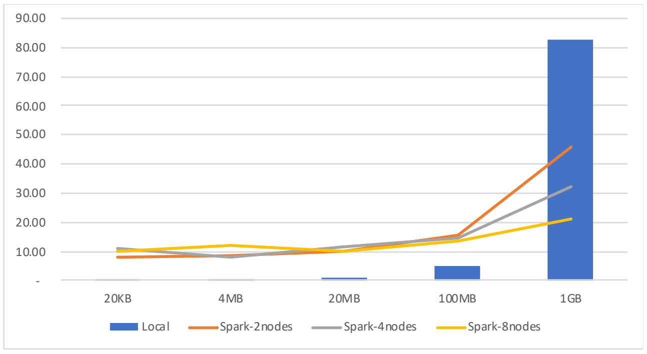

Figure 2. Runtime for Accessing the Data (in seconds)

4.2 Training the Model

Training Logistic Regression model on local machine with small data file is faster than training on EMR using Spark. When the data size is greater than or equal to 100MB, it would take a local computer 300 times or more of time to train a logistic regression model; and as the data size continue to increase, it becomes impossible to train it within a reasonable time on local machine (over 24 hours in this experiment).

Table 3. Runtime of Training the Model (in seconds)

	20KB	4MB	20MB	100MB	1GB
Local	0.01	0.09	31.49	7551.18	>24hrs
Spark-2nodes	2.84	3.07	5.75	27.23	189.77
Spark-4nodes	2.81	3.51	6.11	12.12	117.14
Spark-8nodes	4.64	3.11	5.91	9.42	64.21

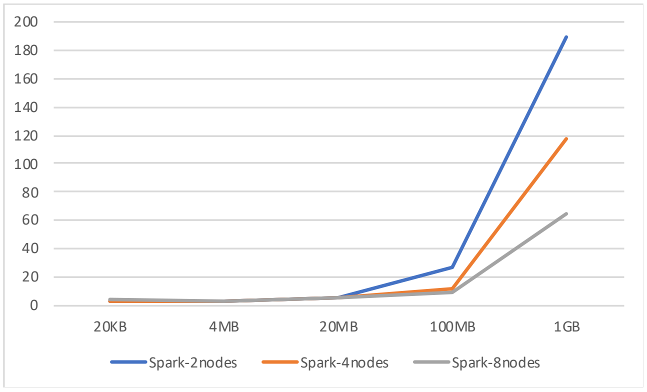

Figure 4. Runtime for Model Training on AWS (in seconds)

For the next step, we look at the training time on EMR and compare its scalability. It turns out that when size is less than 100MB, model training time has no obvious distinction among different node sizes; the ones with more nodes may even take more time to train depend on the time of the day and how the master node assigns the work. When data size reaches 100MB and over, training on EMR instance with more nodes become more efficient. However, there is not a linear speed up when the data size scales.

4.3 Cross Validation

For this part of the experiment, I only did it on EMR clusters, as the number of models being created and tested has exceed the limit for the capacity of a local machine. Because it is anticipated that the runtime for this function would be very long, I started the experiment with 8 nodes, and for the 1GB data, I did not do the 2 nodes experiment because of the time and budget constrain. Therefore, it is shown in the figure as a dotted line. From the data, we can conclude that similar to the previous experiments, when the data size reaches 100MB, the larger the node size, the short the runtime. If one tries to perform cross validation for a model, it would be smart to forgo trying using Scikit-learn and directly do it with a parallel distributed system.

Table 4. Runtime of Cross Validation on EMR (in seconds)

	20KB	**4MB**	**20MB**	**100MB**	**1GB**
Spark 2node	1785.18	1194.82	1510.5	4222.73	NA
Spark 4node	1724.43	1271.09	940.95	3438.98	17859.75
Spark 8node	2083.3	1388.87	863.15	3333.65	9828.273

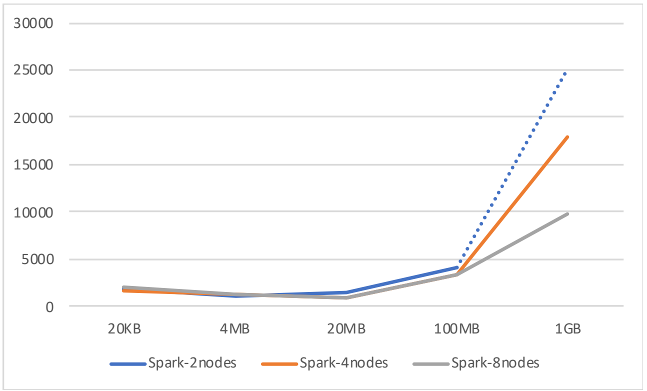

Figure 5. Runtime for Cross Validation on EMR (in seconds)

5. Summaries

In the application of machine learning, we should choose wisely on which computing platform, distribution method to use based on goals, data sizes, and budgets. Since the runtime difference gets too huge to show in the plot, I plot it in log scale in order to see the performance. From the experiment results showed above, I derived the following conclusions:

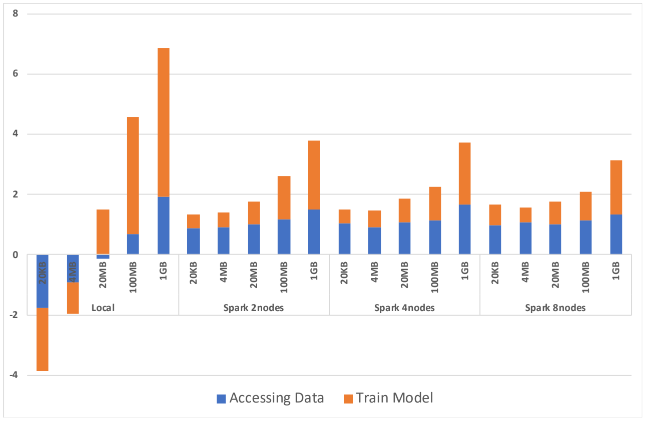

Figure 6. Runtime for comparison in log scale.

- Scikit-learn is faster than Spark MLlib when accessing data and train Logistic Regression model for dataset size <=20MB
- For dataset sizes up and over 100MB, Spark MLlib outperforms Scikit-learn on runtime.
- Spark cluster shows a diminishing return to speed-up and non-linear scale-up; with large amount of data (>=1GB) and complex computation, cluster with more nodes are significantly faster than the ones with less.
- On the code side, Python Scikit-learn is easier to use and setup, even for beginners, while Spark MLlib needs some extra setup and configuration steps on AWS. Furthermore, Spark MLlib has more requirements regarding data type of the variables for training. Even labels "1" and "0" needs to be cast to float or double type, otherwise, it would throw exceptions such as "Column label must be of type DoubleType but was actually IntegerType" if label/variables are of integer type. So, it would need more work to get familiar with the code style.
- The speed comes with a price. Even using Spark on EMR does speed up significantly, it cost more than running on local (money versus time). Since EMR does not allow spot instances, or temporarily stop the machine, even seconds you have the cluster on counts. Running models on EMR would require some serious planning to keep the cost low.

Future works. It would be interesting to implement spark on local machine through pyspark, so we could get a more accurate comparison between performance with and without parallel processing while isolate the difference in CPU performance. Another aspect that would be interesting to address is the prediction accuracy between MLlib and Scikit-learn.

REFERENCES

[1] Apache Spark Documentation. AWS. https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-spark.html.

[2] Machine Learning Workflow on Qubole. https://www.qubole.com/developers/spark-getting-started-guide/workflow/.

[3] How to Optimize for Speed. Scikit-learn documentation.
https://scikit-learn.org/stable/developers/performance.html.

[4] Overview of Amazon EMR. Amazon EMR Management Guide.https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-overview.html.