Machine Learning Performance on Spark

Mengying Bi
University of Washington
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 machine.

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

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. As the quantity of data available for analytics 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. 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.

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 1.

For this paper, I would implement Logistic Regression using scikit-learn and MLlib library to test the runtime of different functions with different size of the data, and explore which setup works the best under which condition.

2. EXPERIMENTS

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.

Datasets. I experimented with real-world datasets that are publicly available online in different size.

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
Concatenated	1.2GB	11,534,325	31

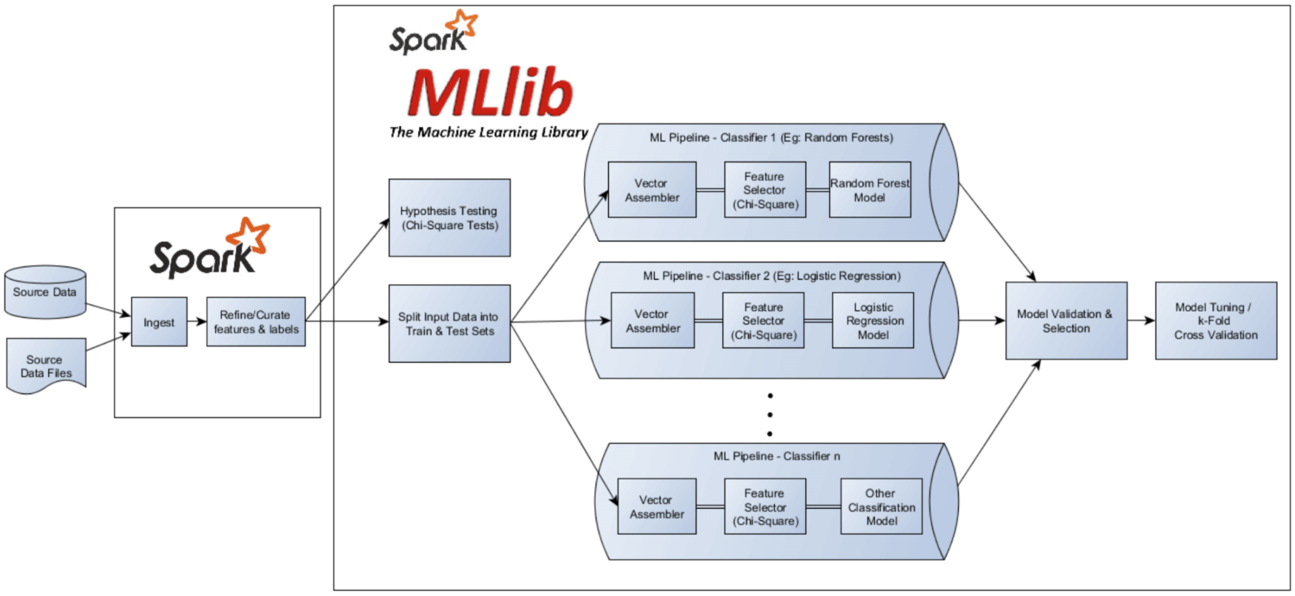

Spark
MLlib
The Machine Learning Library
Source Data
Source Data Files
Spark
Ingest
Refine/Curate features & labels
Hypothesis Testing (Chi-Square Tests)
Split Input Data into Train & Test Sets
ML Pipeline - Classifier 1 (Eg: Random Forests)
Vector Assembler
Feature Selector (Chi-Square)
Random Forest 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 1. Spark MLlib workflow. (Source: qubole)

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

1) Download dataset to local disk;

2) Read the csv data file as data frame using python Pandas library;

3) Pre-process data, including dropping missing values, transform categorical values into numerical values using one hot encoding, and standardizing the data;

4) Choose features and train test split

5) Train the model.

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

1) Upload datasets to AWS S3 bucket;

2) Initialize a Spark session;

3) Read the dataset into a Spark data frame;

4) Pre-process data, including casting all integer values to floats, handling missing values, transforming dataset using vector assembler, and scalarizing the features;

5) Train test split and imbalance handling;

6) Train the model;

7) Performance evaluation.

The main differences between the two methods are:

1) The way to access data,
2) Spark requires all the features to be floats,
3) Spark needs to assemble all selected features into a feature column using Vector Assembler, and
4) 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.

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.

3. Results

Accessing Data. 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 AWS instance.

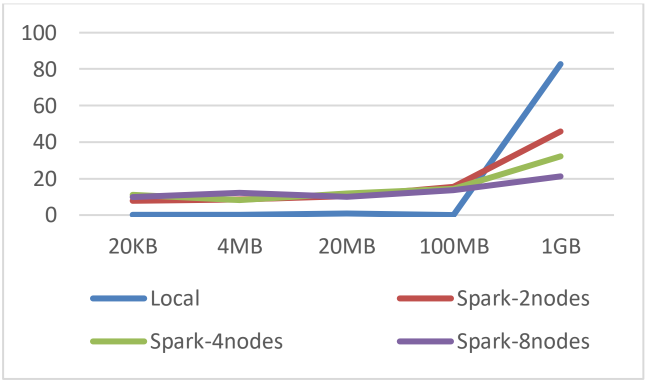


Figure 2. Runtime for Accessing the Data (in seconds)

Training the Model. Training Logistic Regression model on local machine with small data file is faster than training on AWS using Spark. When the data size is greater than or equal to 100MB, it would take a 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 (over 24 hours in this experiment).

Table 2. 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

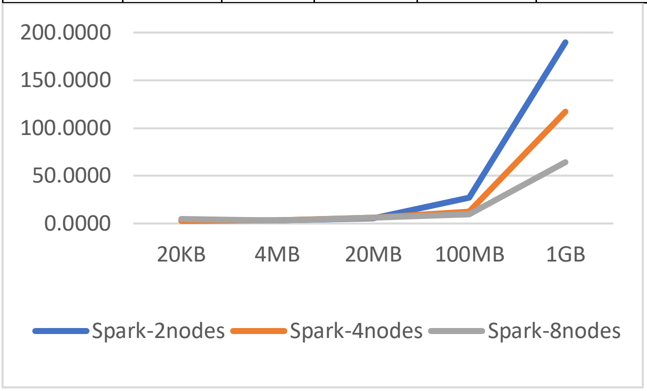
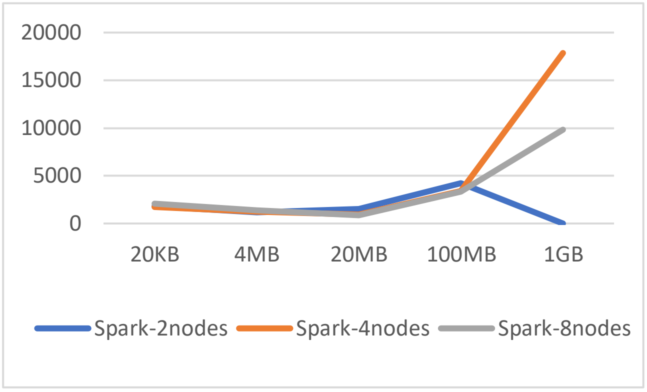


Figure 3. Runtime for Model Training on AWS (in seconds)

Therefore, we look at the training time on AWS 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 1GB and over, training on EMR instance with more nodes become more efficient.

Cross Validation. When performing cross validation on the model, it is foreseeably that it would take a considerable amount of time because of the number of models that we're creating and testing. Similar as previous experiments, when the data size reaches 100MB, the larger the node size, the short the runtime.



4. Summary

/TODO

REFERENCES

[1] Joseph Bradley, Xiangrui Meng, and Denny Lee. Why you should use Spark for machine learning. InfoWorld. FEB 11, 2016. DOI= https://www.infoworld.com/article/3031690/why-you-should-use-spark-for-machine-learning.html.

[2]