SPARK MLLIB VS APACHE MAHOUT (PERFORMANCE COMPARISON)

This is submitted as the final project report of the course CSE 60490: Big Data Analytics

by

Muhammad Amin Ghias

Presented to
Dr. Tariq Mahmood
Professor
Department of Computer Science
School of Mathematics and Computer Science (SMCS)
Institute of Business Administration (IBA), Karachi

Spring Semester 2022
Institute of Business Administration (IBA), Karachi, Pakistan

SPARK MLLIB VS APACHE MAHOUT (PERFORMANCE COMPARISON)

This is submitted as final project report of course CSE 60490: Big Data Analytics

by

Muhammad Amin Ghias
(ERP ID: 25366)

Submitted to:

Dr. Tariq Mahmood

Professor

School of Mathematics and Computer Science (SMCS)
Institute of Business Administration (IBA), Karachi

Spring Semester 2022
Institute of Business Administration (IBA), Karachi, Pakistan

Dedication

This project report is dedicated to our parents, whose full support has enabled us to complete this project timely and effectively

Acknowledgement

First of all, thanks to Allah for allowing us to complete this project successfully and build an effective ML model for comparing Apache Spark MLlib and Apache Mahout performance.

Furthermore, we would like to thank our teacher Dr. Tariq Mahmood for extending his knowledge, experience and assistance throughout the course.

Also, we are thankful to our families and friends for their continuous support and encouragement.

Table of Contents

Abstract

With advancement of technology data has also increased, this big data is noy analyzed in order to get insights form it. In addition to that Machine Learning algorithms are now also implemented on big data to get predictive results. This project will focus on the performance of Spark MLlib and Apache Mahout and compare them.

Keywords:
Targeted, big data, Machine Learning, Spark Mllib, Mahout, prediction models

1. Introduction

In this modern time with increasing data, we not only have to analyze big data but also implement Machine Learning models on it. On big data Machine learning can be done on Spark Mllib and Apache Mahout. Both these ML tools have their own characteristics which will be studied here.

1.1 Business Problem

The performance of any technology is very important to us. With the Spark Mllib and Apache Mahout we need to find which is better with respect to results and time and compare them both.
.

1.2 Main Objective

The main purpose of our project is to do a performance comparison of SPark MLib and Apache Mahout, find out advantages and disadvantages of both and come up with conclusive results.

1.3 Background

We can have a brief introduction of Spark MLlib and Mahout before going into their implementation and comparison

1.3.1 Spark MLlib

MLlib is Spark's machine learning (ML) library designed to make practical machine learning scalable and easy. It is built for the purpose simplicity, scalability, and easy integration with other tools. data scientists can now concentrate more on data with the scalability, language compatibility, and speed of Spark instead of spending time to solve infrastructure, configurations problems of distributed data. Built on top of Spark, MLlib is a scalable machine learning library consisting of common learning algorithms and utilities, including classification, regression, clustering, collaborative filtering, dimensionality reduction, and underlying optimization primitives

1.3.2 Apache Mahout

Mahout is a Machine Learning library of Hadoop and is mostly used for regression, clustering and classification. Apache Mahout is an open-source project to create scalable, machine learning algorithms. Mahout operates in addition to Hadoop, which allows you to apply the concept of machine learning via a selection of Mahout algorithms to distributed computing via Hadoop. Mahout's core algorithms include recommendation mining, clustering, classification, and frequent item-set mining.

Apache Mahout is an open-source project that is primarily used for creating scalable machine learning algorithms. It implements popular machine learning techniques such as:

- Recommendation
- Classification
- Clustering

2. Data acquisition and understanding

The dataset used for this project is acquired from UCI Machine Learning repository:

https://archive.ics.uci.edu/ml/datasets/Gas+sensor+array+temperature+modulation

2.1 About the dataset

The data of " Gas sensor array temperature modulation Data Set was obtained from UCI Machine Learning repository, is provided by Javier of Institute of Bioengineering of Catalonia (IBEC).

2.2 Dataset main characteristics

The data contains 3.8 million instances of data from a chemical detection platform consisting of various gas sensors.

	Time (s)	humidity_percentage	Temperature (C)	flow_rate	Heater voltage (V)	R1 (MOhm)	R2 (MOhm)	R3 (MOhm)	R4 (MOhm)	R5 (MOhm)	R6 (MOhm)	R7 (MOhm)	R8 (MOhm)	R9 (MOhm)	R10 (MOhm)
0	0.000	49.7534	23.7184	233.2737	0.8993	0.2231	0.6365	1.1493	0.8483	1.2534	1.4449	1.9906	1.3303	1.4480	1.9148
1	0.309	55.8400	26.6200	241.6323	0.2112	2.1314	5.3552	9.7569	6.3188	9.4472	10.5769	13.6317	21.9829	16.1902	24.2780
2	0.618	55.8400	26.6200	241.3888	0.2070	10.5318	22.5612	37.2635	17.7848	33.0704	36.3160	42.5746	49.7495	31.7533	57.7289
3	0.926	55.8400	26.6200	241.1461	0.2042	29.5749	49.5111	65.6318	26.1447	58.3847	67.5130	68.0064	59.2824	36.7821	66.0832
4	1.234	55.8400	26.6200	240.9121	0.2030	49.5111	67.0368	77.8317	27.9625	71.7732	79.9474	79.8631	62.5385	39.6271	68.1441

```
1 df.shape
```

CVE

```
(3843160, 20)
```

The 20 columns features are
Time (s),

- CO concentration (ppm),
- Humidity (%r.h.),
- Temperature (Â°C),
- Flow rate (mL/min),
- Heater voltage (V),
- the resistance of the 14 gas sensors: R1 (MOhm),R2 (MOhm),R3 (MOhm),R4 (MOhm),R5 (MOhm),R6 (MOhm),R7 (MOhm),R8 (MOhm),R9 (MOhm),R10 (MOhm),R11 (MOhm),R12 (MOhm),R13 (MOhm),R14 (MOhm)
-

•

2.2.1 Dataset Size

The size of the csv is **553 MB**
The no. of rows = 3843160
Number of columns = 20

2.2.2 Data types

All the features are of float type.

Target Variable to predict **“CO (ppm)”**

```
1  df.dtypes

Time (s)                 float64
humidity_percentage      float64
Temperature (C)          float64
flow_rate                float64
Heater voltage (V)       float64
R1 (MOhm)                float64
R2 (MOhm)                float64
R3 (MOhm)                float64
R4 (MOhm)                float64
R5 (MOhm)                float64
R6 (MOhm)                float64
R7 (MOhm)                float64
R8 (MOhm)                float64
R9 (MOhm)                float64
R10 (MOhm)               float64
R11 (MOhm)               float64
R12 (MOhm)               float64
R13 (MOhm)               float64
R14 (MOhm)               float64
CO (ppm)                 float64
dtype: object
```

2.3 Data Exploration

Data set was explored, and no null values were found

2.3.1 Data cleaning and filtering

The dataset didn’t have any outliers and null values, so no cleaning was required.
The dataset is filtered based on correlation with target variable CO
(Note: Later on, for Mahout and SpaekMllib comparision we used this filtered reduced dataset)

.

3. Implementations

To start with our project, we will have to implement ML models on Spark MLlib and Apache Mahout

3.1 Working Machine Specification

Windows edition

Windows 10 Pro

© Microsoft Corporation. All rights reserved.

System

Processor:	11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz 2.80 GHz
Installed memory (RAM):	24.0 GB (23.7 GB usable)
System type:	64-bit Operating System, x64-based processor
Pen and Touch:	No Pen or Touch Input is available for this Display

3.2 Building Docker Environment for making multi container Application

For implementing MLlib and Mahout we will use 2 docker containers. For this project we will built a multi container environment using docker compose file

3.2.1 Building docker compose File

In compose file we will build 2 services containers for:

- Apache Mahout
- Spark MLlib (pyspark)

The main features of compose file are:
Volume: Assigning volume to each container, to help in data sharing with containers and host computers
Ports: Allocating containers port
Image: Which image from docker hub will be used to build conatiner
Name: Giving containers name

Extra features:

- Stdin- open: was used to prevent mahout container from closing which was closing and restarting
- Restart: to run container if it stops or crashes
- Resources are also allocated for each container__

```yaml
version: "3.8"

services:

  mahout:
    image: michabirklbauer/mahout:latest
    container_name: mahout-container

    deploy:
      resources:
        limits:
          memory: 10000M
        reservations:
          memory: 10000M
    stdin_open: true

    volumes:
      - 'F:/BDA-project/data:/data/'
    restart: always
    ports:
      - 9000:9000

  mllib:
    image: jupyter/pyspark-notebook
    container_name: mllib-container
    deploy:
      resources:
        limits:
          memory: 10000M
        reservations:
          memory: 10000M

    volumes:
      - 'F:/BDA-project/data:/home/jovyan/notebooks'
    restart: always
    ports:
        - "10000:8888"
```

3.2.2 Transferring Data to containers

Data is transferred copied to the directory of volume as mentioned in compose file of each container. Compose file is specifying the directory of host computer and docker container. For our convenience and to allow same data to be used by both containers the host directory of volume is same for both containers

That is, the directory F:/BDA-project/data folder can be accessed by both MLlib and Mahout container

Images of containers

Mahout: michabirklbauer/mahout:latest
Link : https://hub.docker.com/r/michabirklbauer/mahout

Container Image runs the following Apache services:

- Apache Mahout 0.14.2
- Apache Maven 3.6.3
- Apache Hadoop 3.2.1
- Apache Spark 3.1.1

Spark MLlib : jupyter/pyspark-notebook

Link: https://hub.docker.com/r/jupyter/pyspark-notebook

Container Image runs the following services:

- Apache Hadoop
- Apache Spark
- Jupyter Notebook

3.2.3 Making and Running containers

Steps:

1. Make directory as mentioned in compose file
   F:/BDA-project/data

2. Make docker compose file in F:/BDA-project/

3. Run this command in cmd from same working directory this will make the multi container application with 2 conatiner

   Docker-compose up -d

3.3 Implementing Mahout in docker container

Now we will run the mahout container we built using docker compose file

Start container with
docker start mahout-container

Run container
docker exec -it mahout-container/bin/bash

Start Shell
./mahout/bin/mahout spark-shell

3.3.1 Problems faced in Mahout

When using full data csv it gave memory error, also with full features OLS mahout gave faulty results. Mahout OLS worked on smaller filtered datasest

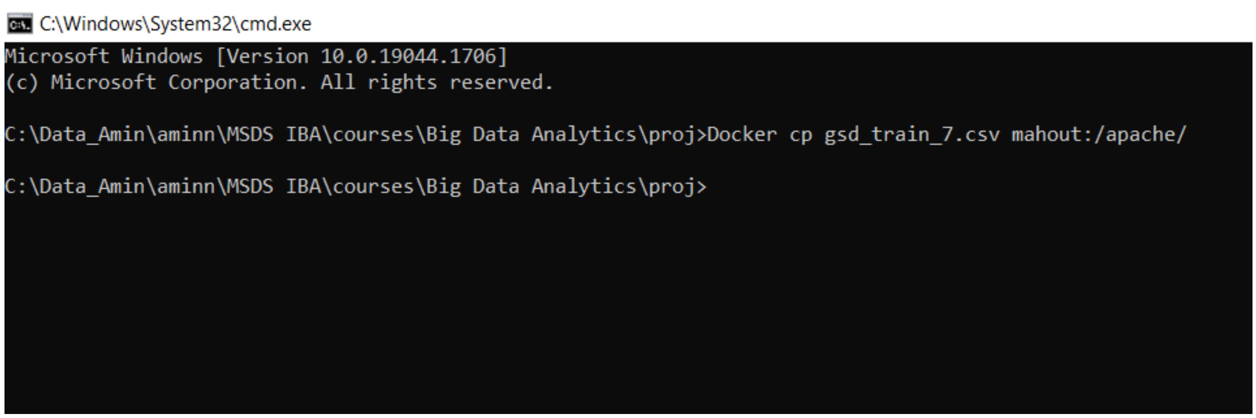
3.3.2 OLS Model on Mahout

Steps

1. Import data in mahout

   Docker cp gsd_train_7.csv mahout-container:/data/

[Note: This will not be required as we have copied data to the volume already]



2. Importing Libraries

import org.apache.mahout.math.algorithms.regression.OrdinaryLeastSquares
import org.apache.spark.sql.types.{StringType, StructType}
import spark.implicits._
import Array.ofDim

3. Read data in mahout

val dft = spark.read.option("header","true").csv("/data/gsd_train_7.csv")

```
scala> val dft = spark.read.option("header","true").csv("gsd_train_7.csv");
dft: org.apache.spark.sql.DataFrame = [R8 (MOhm): string, R9 (MOhm): string ... 6 more fields]

scala> dft.count
res1: Long = 192158

scala> val rowDF = dft.select(array(dft.columns.map(col):_*) as "row")
rowDF: org.apache.spark.sql.DataFrame = [row: array<string>]
```

4. Converting dataframe into Mahout matrix

val rowDF = dft.select(array(dft.columns.map(col):_*) as "row")
val mat = rowDF.collect.map(_.getSeq[String](0).toArray)
val train = mat.map(_.map(_.toDouble))
val rddA = sc.parallelize(train)
val drmRddA: DrmRdd[Double] = rddA.map(a => new DenseVector(a)).zipWithIndex().map(t => (t._2, t._1))
drmRddA.collect
val train = drmWrap(rdd= drmRddA)

```
scala> val rowDF = dft.select(array(dft.columns.map(col):_*) as "row")
rowDF: org.apache.spark.sql.DataFrame = [row: array<string>]

scala> val mat = rowDF.collect.map(_.getSeq[String](0).toArray)
mat: Array[Array[String]] = Array(Array(28.6584, 23.6376, 25.6229, 26.5903,
3.4763, 4.44), Array(28.7735, 23.1497, 29.9089, 34.8037, 27.3364, 19.1965,
y(18.5531, 12.7976, 14.3303, 21.8977, 18.2805, 17.6031, 18.2135, 6.67), Ar
9.7462, 62.0947, 55.4749, 58.9429, 71.2524, 0.0), Array(22.1794, 12.8416,
.1017, 0.1079, 6.67), Array(0.117, 0.1021, 0.1252, 0.1151, 0.1201, 0.1129,

scala> val train = mat.map(_.map(_.toDouble))
train: Array[Array[Double]] = Array(Array(28.6584, 23.6376, 25.6229, 26.59
 53.4763, 4.44), Array(28.7735, 23.1497, 29.9089, 34.8037, 27.3364, 19.196
ray(18.5531, 12.7976, 14.3303, 21.8977, 18.2805, 17.6031, 18.2135, 6.67),
 59.7462, 62.0947, 55.4749, 58.9429, 71.2524, 0.0), Array(22.1794, 12.8416
 0.1017, 0.1079, 6.67), Array(0.117, 0.1021, 0.1252, 0.1151, 0.1201, 0.112

scala> val rddA = sc.parallelize(train)
rddA: org.apache.spark.rdd.RDD[Array[Double]] = ParallelCollectionRDD[21]

scala> val drmRddA: DrmRdd[Double] = rddA.map(a => new DenseVector(a)).zip
22/06/01 21:51:17 WARN TaskSetManager: Stage 4 contains a task of very lar
drmRddA: org.apache.mahout.sparkbindings.DrmRdd[Double] = MapPartitionsRDD

scala> drmRddA.collect
22/06/01 21:51:18 WARN TaskSetManager: Stage 5 contains a task of very lar
res2: Array[org.apache.mahout.math.drm.DrmTuple[Double]] = Array((0.0,{0:2
.3693,2:50.0936,3:52.0519,4:48.1165,5:45.7841,6:53.4763,7:4.44}), (2.0,{0:
```

```
scala> val train = drmWrap(rdd= drmRddA)
train: org.apache.mahout.math.drm.CheckpointedDrm[Double] = org.apache.mahout.sparkbindings.d

scala> train.collect
22/06/01 21:51:19 WARN TaskSetManager: Stage 6 contains a task of very large size (1536 KiB).
22/06/01 21:51:19 WARN TaskSetManager: Stage 7 contains a task of very large size (1536 KiB).
22/06/01 21:51:19 WARN TaskSetManager: Stage 8 contains a task of very large size (1536 KiB).
res3: org.apache.mahout.math.Matrix =
{
 0 =>   {0:28.6584,1:23.6376,2:25.6229,3:26.5903,4:25.8244,5:21.4715,6:28.2509,7:15.56}
 1 =>   {0:41.2509,1:47.3693,2:50.0936,3:52.0519,4:48.1165,5:45.7841,6:53.4763,7:4.44}
 2 =>   {0:28.7735,1:23.1497,2:29.9089,3:34.8037,4:27.3364,5:19.1965,6:31.3205,7:13.33}
 3 =>   {0:34.3044,1:28.1199,2:30.5588,3:30.998,4:31.9957,5:29.3116,6:33.8511,7:11.11}
 4 =>   {0:18.5531,1:12.7976,2:14.3303,3:21.8977,4:18.2805,5:17.6031,6:18.2135,7:6.67}
 5 =>   {0:19.9659,1:14.8416,2:12.295,3:16.9441,4:18.3179,5:13.1824,6:16.9013,7:20.0}
 6 =>   {0:70.1254,1:61.8514,2:59.7462,3:62.0947,4:55.4749,5:58.9429,6:71.2524}
 7 =>   {0:22.1794,1:12.8416,2:23.3104,3:30.2138,4:21.7739,5:17.9536,6:27.2227,7:6.67}
 8 =>   {0:0.1033,1:0.0982,2:0.118,3:0.1098,4:0.1098,5:0.10...
```

5. Dividing dataset into x and y

val x_train = train(::, 0 until 7)
x_train.collect

```
scala> val x_train = train(::, 0 until 7)
x_train: org.apache.mahout.math.drm.DrmLike[Double] = OpMapBlock(org.apache.mahout.sp
mbda$4044/904067302@5af1fa64,7,-1,true)

scala> x_train.collect
22/06/01 21:51:20 WARN TaskSetManager: Stage 9 contains a task of very large size (15
res4: org.apache.mahout.math.Matrix =
{
 0 =>   {0:28.6584,1:23.6376,2:25.6229,3:26.5903,4:25.8244,5:21.4715,6:28.2509}
 1 =>   {0:41.2509,1:47.3693,2:50.0936,3:52.0519,4:48.1165,5:45.7841,6:53.4763}
 2 =>   {0:28.7735,1:23.1497,2:29.9089,3:34.8037,4:27.3364,5:19.1965,6:31.3205}
 3 =>   {0:34.3044,1:28.1199,2:30.5588,3:30.998,4:31.9957,5:29.3116,6:33.8511}
 4 =>   {0:18.5531,1:12.7976,2:14.3303,3:21.8977,4:18.2805,5:17.6031,6:18.2135}
 5 =>   {0:19.9659,1:14.8416,2:12.295,3:16.9441,4:18.3179,5:13.1824,6:16.9013}
 6 =>   {0:70.1254,1:61.8514,2:59.7462,3:62.0947,4:55.4749,5:58.9429,6:71.2524}
 7 =>   {0:22.1794,1:12.8416,2:23.3104,3:30.2138,4:21.7739,5:17.9536,6:27.2227}
 8 =>   {0:0.1033,1:0.0982,2:0.118,3:0.1098,4:0.1098,5:0.1017,6:0.1079}
 9 =>   {0:0.117,1:0.1021,2:0.1252,3:0.11...
```

val y_train = train(::, 7 until 8)
y_train.collect

```
scala> val y_train = train(::, 7 until 8)
y_train: org.apache.mahout.math.drm.DrmLike[Double] = OpMapBlo
mbda$4044/904067302@342e0afd,1,-1,true)

scala> y_train.collect
22/06/01 21:51:21 WARN TaskSetManager: Stage 10 contains a tas
res5: org.apache.mahout.math.Matrix =
{
 0 =>   {0:15.56}
 1 =>   {0:4.44}
 2 =>   {0:13.33}
 3 =>   {0:11.11}
 4 =>   {0:6.67}
 5 =>   {0:20.0}
 6 =>   {}
 7 =>   {0:6.67}
 8 =>   {0:6.67}
 9 =>   {}
... }

scala>

scala> val t1 = System.nanoTime
t1: Long = 48605253544188
```

6. Fitting model

val t1 = System.nanoTime
val model = new OrdinaryLeastSquares[Double]().fit(x_train, y_train)
val duration_fit = (System.nanoTime - t1) / 1e9d
println("Model Fitting time in seconds : ", duration_fit)

```
scala> val t1 = System.nanoTime
t1: Long = 48605253544188

scala> val model = new OrdinaryLeastSquares[Double]().fit(x_train, y_train)
22/06/01 21:51:22 WARN TaskSetManager: Stage 11 contains a task of very larg
22/06/01 21:51:22 WARN TaskSetManager: Stage 13 contains a task of very larg
22/06/01 21:51:23 WARN TaskSetManager: Stage 15 contains a task of very larg
22/06/01 21:51:23 WARN TaskSetManager: Stage 17 contains a task of very larg
22/06/01 21:51:24 WARN TaskSetManager: Stage 18 contains a task of very larg
22/06/01 21:51:24 WARN TaskSetManager: Stage 19 contains a task of very larg
22/06/01 21:51:24 WARN TaskSetManager: Stage 20 contains a task of very larg
22/06/01 21:51:25 WARN TaskSetManager: Stage 21 contains a task of very larg
22/06/01 21:51:25 WARN TaskSetManager: Stage 22 contains a task of very larg
model: org.apache.mahout.math.algorithms.regression.OrdinaryLeastSquaresMode

scala> val duration_fit = (System.nanoTime - t1) / 1e9d
duration_fit: Double = 4.049513869

scala> println("Model Fitting time in seconds : ",  duration_fit)
(Model Fitting time in seconds : ,4.049513869)
```

The fitting time of model is 4.04951 seconds

7. Result summary

println(model.summary)

```
scala> println(model.summary)

Coef.           Estimate                Std. Error              t-score                 Pr(Beta=0)
X0              +0.30081                +0.00372                +80.81008               +0.00000
X1              +0.39936                +0.00412                +97.04073               +0.00000
X2              -0.88286                +0.00463                -190.80585              +0.00000
X3              +0.08705                +0.00383                +22.72937               +0.00000
X4              +0.76703                +0.00505                +151.87493              +0.00000
X5              -0.68081                +0.00541                -125.79324              +0.00000
X6              -0.12107                +0.00424                -28.52346               +0.00000
X7              +11.87872               +0.01987                +597.82723              +0.00000
F-statistic: 24104.970268970916 on 7 and 192150 DF,  p-value: 0.009545

Mean Squared Error: 21.992365168814732
R^2: 0.46755861060932247

scala>
```

8. Predicting the model

val t1 = System.nanoTime
val ypred = model.predict(x_train)
val duration_predict = (System.nanoTime - t1) / 1e9d
println("Model predicting time in seconds : ", duration_predict)

```
scala> val t1 = System.nanoTime
t1: Long = 48609647561774

scala> val ypred = model.predict(x_train)
ypred: org.apache.mahout.math.drm.DrmLike[Double] = OpAx(OpCbindScalar(OpMa
rm.DrmLikeOps$$Lambda$4044/904067302@5af1fa64,7,-1,true),1.0,false),{0:0.30
615,5:-0.6808103928134337,6:-0.12107128036286793,7:11.878724665974477})

scala> val duration_predict = (System.nanoTime - t1) / 1e9d
duration_predict: Double = 0.229774948

scala> println("Model predicting time in seconds : ",  duration_predict)
(Model predicting time in seconds : ,0.229774948)
```

3.4 Implementing Spark MLlib (pyspark) on container

MLlib is implement using Pyspark on jupyter notebook
Start the docker container
Docker start mllib-conatiner

Run the docker conatiner with
docker exec -it mllib-container jupyter lab
this results in

```
C:\Users\Dell>docker exec -it mllib-container jupyter lab
[I 2022-06-02 13:27:45.940 ServerApp] jupyterlab | extension was successfully linked.
[W 2022-06-02 13:27:45.946 NotebookApp] 'ip' has moved from NotebookApp to ServerApp. This config wil
 release.
[W 2022-06-02 13:27:45.946 NotebookApp] 'port' has moved from NotebookApp to ServerApp. This config w
xt release.
[W 2022-06-02 13:27:45.946 NotebookApp] 'port' has moved from NotebookApp to ServerApp. This config w
xt release.
[I 2022-06-02 13:27:45.956 ServerApp] nbclassic | extension was successfully linked.
[I 2022-06-02 13:27:46.176 ServerApp] notebook_shim | extension was successfully linked.
[I 2022-06-02 13:27:46.228 ServerApp] notebook_shim | extension was successfully loaded.
[I 2022-06-02 13:27:46.232 LabApp] JupyterLab extension loaded from /opt/conda/lib/python3.9/site-pac
[I 2022-06-02 13:27:46.232 LabApp] JupyterLab application directory is /opt/conda/share/jupyter/lab
[I 2022-06-02 13:27:46.242 ServerApp] jupyterlab | extension was successfully loaded.
[I 2022-06-02 13:27:46.249 ServerApp] nbclassic | extension was successfully loaded.
[I 2022-06-02 13:27:46.249 ServerApp] The port 8888 is already in use, trying another port.
[I 2022-06-02 13:27:46.250 ServerApp] Serving notebooks from local directory: /home/jovyan
[I 2022-06-02 13:27:46.250 ServerApp] Jupyter Server 1.16.0 is running at:
[I 2022-06-02 13:27:46.250 ServerApp] http://47176e291082:8889/lab?token=45afbace9f84ed5b19ef77fe93ff
[I 2022-06-02 13:27:46.250 ServerApp]  or http://127.0.0.1:8889/lab?token=45afbace9f84ed5b19ef77fe93f
[I 2022-06-02 13:27:46.250 ServerApp] Use Control-C to stop this server and shut down all kernels (tw
[C 2022-06-02 13:27:46.256 ServerApp]

    To access the server, open this file in a browser:
        file:///home/jovyan/.local/share/jupyter/runtime/jpserver-15851-open.html
    Or copy and paste one of these URLs:
        http://47176e291082:8889/lab?token=45afbace9f84ed5b19ef77fe93ff74210e7424e319205193
     or http://127.0.0.1:8889/lab?token=45afbace9f84ed5b19ef77fe93ff74210e7424e319205193
```

Start the notebook by copy pasting the url in browser
In notebook environment get in notebook folder where data csv file is stored as irt is volume direction mentioned in compose file

3.4.1 Data Processing

1. Read CSV

```
[9]:  df = spark.read.csv('gas_sensor_data_f.csv',header=True)

[10]: type(df)

[10]: pyspark.sql.dataframe.DataFrame

[11]: df.count()

[11]: 3843160
```

2. Change datatype to double

Change data type from string double

```
16]: df = df.withColumn("Time (s)",col("Time (s)").cast( DoubleType()))
df = df.withColumn("CO (ppm)",col("CO (ppm)").cast( DoubleType()))
df = df.withColumn("humidity_percentage",col("humidity_percentage").cast( DoubleType()))
df = df.withColumn("Temperature (C)",col("Temperature (C)").cast( DoubleType()))
df = df.withColumn("Temperature (C)",col("Temperature (C)").cast( DoubleType()))
df = df.withColumn("flow_rate",col("flow_rate").cast( DoubleType()))
df = df.withColumn("Heater voltage (V)",col("Heater voltage (V)").cast( DoubleType()))
df = df.withColumn("R1 (MOhm)",col("R1 (MOhm)").cast( DoubleType()))
df = df.withColumn("R2 (MOhm)",col("R2 (MOhm)").cast( DoubleType()))
df = df.withColumn("R3 (MOhm)",col("R3 (MOhm)").cast( DoubleType()))
df = df.withColumn("R4 (MOhm)",col("R4 (MOhm)").cast( DoubleType()))
df = df.withColumn("R5 (MOhm)",col("R5 (MOhm)").cast( DoubleType()))
df = df.withColumn("R6 (MOhm)",col("R6 (MOhm)").cast( DoubleType()))
df = df.withColumn("R7 (MOhm)",col("R7 (MOhm)").cast( DoubleType()))
df = df.withColumn("R8 (MOhm)",col("R8 (MOhm)").cast( DoubleType()))
df = df.withColumn("R9 (MOhm)",col("R9 (MOhm)").cast( DoubleType()))
df = df.withColumn("R10 (MOhm)",col("R10 (MOhm)").cast( DoubleType()))
df = df.withColumn("R11 (MOhm)",col("R11 (MOhm)").cast( DoubleType()))
df = df.withColumn("R12 (MOhm)",col("R12 (MOhm)").cast( DoubleType()))
df = df.withColumn("R13 (MOhm)",col("R13 (MOhm)").cast( DoubleType()))
df = df.withColumn("R14 (MOhm)",col("R14 (MOhm)").cast( DoubleType()))
```

3. Make vectors

```
]: from pyspark.ml.feature import VectorAssembler
vectorAssembler = VectorAssembler(inputCols = ['Time (s)', 'humidity_percentage', 'Temperature (C)', 'flow_rate', 'Heater voltage (V)','R1 (MOhm)','R2 (MOhm)',
                                     'R3 (MOhm)', 'R4 (MOhm)', 'R5 (MOhm)', 'R6 (MOhm)', 'R7 (MOhm)', 'R8 (MOhm)', 'R9 (MOhm)','R10 (MOhm)','R11 (MOhm)',
                                     'R12 (MOhm)', 'R13 (MOhm)', 'R14 (MOhm)'], outputCol = 'features')
data = vectorAssembler.transform(df)

data = data.withColumn('label', col('CO (ppm)'))

data = data.select(['features', 'label'])
data.show(3)
```

```
+--------------------+-----+
|            features|label|
+--------------------+-----+
|[0.0,49.7534,23.7...|  0.0|
|[0.309,55.84,26.6...|  0.0|
|[0.618,55.84,26.6...|  0.0|
+--------------------+-----+
only showing top 3 rows
```

4. Split data

Splitting Data ¶

```
9]: splits = data.randomSplit([0.7, 0.3])
    train_df = splits[0]
    test_df = splits[1]
```

```
9]: train_df.show(3)

+--------------------+-----+
|            features|label|
+--------------------+-----+
|[0.0,49.21,26.38,...|  0.0|
|[0.0,50.25,26.54,...|  0.0|
|[0.0,54.6258,25.3...|  0.0|
+--------------------+-----+
only showing top 3 rows
```

```
1]: test_df.show(3)

+--------------------+-----+
|            features|label|
+--------------------+-----+
|[0.0,49.7534,23.7...|  0.0|
|[0.31,52.63,25.3,...|  0.0|
|[0.618,55.84,26.6...|  0.0|
+--------------------+-----+
only showing top 3 rows
```

3.4.2 Linear Regression on MLlib

5. Tarin data
6. Predict data

Linear Regression

```
from pyspark.ml.regression import LinearRegression

lr_f = LinearRegression(featuresCol = 'features', predictionCol='pred_CO (ppm)',
                        maxIter=10, regParam=0.0, solver="normal", standardization=False)

start = time.perf_counter()
lr_modelf = lr_f.fit(train_df)
end = time.perf_counter()
duration_fit = format((end-start),'.4f')
print("Model Fitting Time Duration - {}".format(duration_fit))

print("Coefficients: " + str(lr_modelf.coefficients))
print("Intercept: " + str(lr_modelf.intercept))

start = time.perf_counter()
yfpredictions = lr_modelf.transform(test_df)
end = time.perf_counter()
duration_pred = format((end-start),'.4f')

trainingSummary = lr_modelf.summary

print("numIterations: %d" % trainingSummary.totalIterations)
print("objectiveHistory: %s" % str(trainingSummary.objectiveHistory))
trainingSummary.residuals.show()
print("RMSE: %f" % trainingSummary.rootMeanSquaredError)
print("r2: %f" % trainingSummary.r2)
print("Model Fitting Time Duration - {}".format(duration_fit))
print("Model predicting Time Duration - {}".format(duration_pred))
```

7. Scores

```
RMSE: 4.186850
r2: 0.575813
Model Fitting Time Duration - 41.8281
Model predicting Time Duration - 0.0338
```

3.4.3 Decision Tress regressor on MLlib

Similarly decision tree model is trained and then predicts

```
[31]: from pyspark.ml.regression import DecisionTreeRegressor
dt = DecisionTreeRegressor()

start = time.perf_counter()
model_dt = dt.fit(train_df)
end = time.perf_counter()
duration_fit = format((end-start),'.4f')
print("Model Fitting Time Duration - {}".format(duration_fit))

start = time.perf_counter()
ypred_dt = model_dt.transform(test_df)
end = time.perf_counter()
duration_pred = format((end-start),'.4f')

print("Model Fitting Time Duration - {}".format(duration_fit))
print("Model predicting Time Duration - {}".format(duration_pred))
```

```
Model Fitting Time Duration - 77.9203
Model Fitting Time Duration - 77.9203
Model predicting Time Duration - 0.0411
```

Scores

```
[ ]: evaluator = RegressionEvaluator()
print("R2 :",evaluator.evaluate(ypred_dt,
{evaluator.metricName: "r2"}))

print("MSE :",evaluator.evaluate(ypred_dt,
{evaluator.metricName: "mse"}))

print("RMSE :",evaluator.evaluate(ypred_dt,
{evaluator.metricName: "rmse"}))

print("MAE :",evaluator.evaluate(ypred_dt,
{evaluator.metricName: "mae"}))
```

```
R2 : 0.7484396758793779
MSE : 10.385553255624137
RMSE : 3.2226624482908752
MAE : 2.1642999933938785
```

3.4.4 Random Forrest Regressor in MLlib

For random forest data trained and then tested, normal default parameters used

```
from pyspark.ml.regression import RandomForestRegressor

# Define LinearRegression algorithm
rf = RandomForestRegressor() # featuresCol="eatures",numTrees=2, maxDepth=2, seed=42

start = time.perf_counter()
model_rf = rf.fit(train_df)
end = time.perf_counter()
duration_fit = format((end-start),'.4f')
print("Model Fitting Time Duration - {}".format(duration_fit))

start = time.perf_counter()
ypred_rf = model_rf.transform(test_df)
end = time.perf_counter()
duration_pred = format((end-start),'.4f')

print("Model Fitting Time Duration - {}".format(duration_fit))
print("Model predicting Time Duration - {}".format(duration_pred))
```

```
Model Fitting Time Duration - 104.6386
Model Fitting Time Duration - 104.6386
Model predicting Time Duration - 0.1611
```

Scores

```
evaluator = RegressionEvaluator()
print("R2 :",evaluator.evaluate(ypred_rf,
{evaluator.metricName: "r2"}))

print("MSE :",evaluator.evaluate(ypred_rf,
{evaluator.metricName: "mse"}))

print("RMSE :",evaluator.evaluate(ypred_rf,
{evaluator.metricName: "rmse"}))

print("MAE :",evaluator.evaluate(ypred_rf,
{evaluator.metricName: "mae"}))
```

```
R2 : 0.7633407364675332
MSE : 9.770369764966192
RMSE : 3.1257590702045785
MAE : 2.06678014299914
```

4. Comparison of Spark MLlib and Apache Mahout

As for Mahout only OLS algorithm which was available could be implemented on on smaller filtered dataset size (due to memory error), Same smaller reduced filtered dataset was also used on pyspark MLlib to find performance comparison

Pyspark results applying same linear regression which was applied for whole dataset

```
]: df.count()

]: 192158

]: df.show(2)

+---------+---------+----------+----------+----------+----------+----------+--------+
|R8 (MOhm)|R9 (MOhm)|R10 (MOhm)|R11 (MOhm)|R12 (MOhm)|R13 (MOhm)|R14 (MOhm)|CO (ppm)|
+---------+---------+----------+----------+----------+----------+----------+--------+
|  28.6584|  23.6376|   25.6229|   26.5903|   25.8244|   21.4715|   28.2509|   15.56|
|  41.2509|  47.3693|   50.0936|   52.0519|   48.1165|   45.7841|   53.4763|    4.44|
+---------+---------+----------+----------+----------+----------+----------+--------+
only showing top 2 rows
```

```
RMSE: 4.689602
r2: 0.467559
Model Fitting Time Duration - 2.4237
Model predicting Time Duration - 0.0358
```

4.1 Score performance

The results of R2 scores is shown int table below

	R2
Pyspark Mllib	0.467559
Mahout	0.4675

This shows that R2 score of pyspark MLlib is nearly similar to Mahout

4.2 Time performance

Time of mahout and pyspark is shown in table below

	Time (s)	
	Model Fitting Time Duration	**Model predicting Time Duration**
Pyspark Mllib	2.4237	0.0358
Mahout	4.0495	0.2297

Table Shows that Pyspark takes less time in model fitting and predicting as well

5. Result and Conclusion

We can combine the results and reach a conclusion

		Time (s)	
	R2	**Model Fitting Time Duration**	**Model predicting Time Duration**
Pyspark Mllib	0.467559	2.4237	0.0358
Mahout	0.4675	4.0495	0.2297

It can be seen from the results that Pyspark Mllib is faster than Mahout in both fitting and predicting and gives nearly similar values of R2.

Pyspark Mllib is 1.67 times faster in fitting data than Mahout

Pyspark Mllib is 6.42 times faster in predicting data than Mahout

Also, a major drawback which was observed in this project was that Mahout (working in container) was not able to work on big datasets and gave memory error while Spark MLlib worked on full dataset satisfactorily

Hence MLlib is performing better than Mahout as it is much faster in performance

6. Problems Faced

Following problems were faced in this project:

- Finding relevant docker images for Pyspark and Mahout took time
- For Mahout very little online resources are available
- Mahout is based on JAVA and Scala the algorithms of Mahout for Scala for Machine learning are not available
- Configuration of Mahout in container to run JAVA based codes and build jar files is very challenging
- Map reduced based Mahout ML models take difficult long configuration to run
- Mahout OLS was not able to be performed on full dataset, it performed on smaller reduced filtered datasest
- Mahout OLS with 20. features on smaller dataset gave faulty results of R2 in negatives
- Mahout OLS only performed on less features 8 features (after filtering) and smaller size dataset

7. Future Recommendation

- Other Pyspark ML models can be run
- Parameter optimization can be done on Pyspark models
- Performance of Pyspark can be compared with python sklearn

References

- https://spark.apache.org/docs/latest/ml-guide.html
- https://databricks.com/glossary/what-is-machine-learning-library
- https://www.ibm.com/docs/en/spectrum-symphony/7.2.0?topic=mapreduce-apache-mahout
- https://www.tutorialspoint.com/mahout/mahout_introduction.htm
-