Big Data Analytics
Term Project

Topic: Machine Learning with Spark MLlib and Apache Mahout and their comparison

Name: Muhammad Salman Khan
ERP# 25373

Section 1: Machine Learning with Spark MLlib

PySpark:

PySpark, the Spark Python API, makes the Spark programming model available to Python. PySpark is based on the Java API for Spark and employs Py4J. Py4J, a Java-Python bridge, allows Python programs running in a Python interpreter to dynamically access Java objects in a Java Virtual Machine, according to Apache (JVM). The JVM caches and shuffles the data that is processed in Python.

We will be using PySpark to train our machine learning model.

Docker:

Docker allows developers and IT to design, manage, and secure mission-critical applications without concern of being locked onto proprietary technology or infrastructure.

To work on a Machine Learning project using Spark MLlib, we will be using Docker for Desktop.

At this point, I have Docker for Desktop installed on my computer.

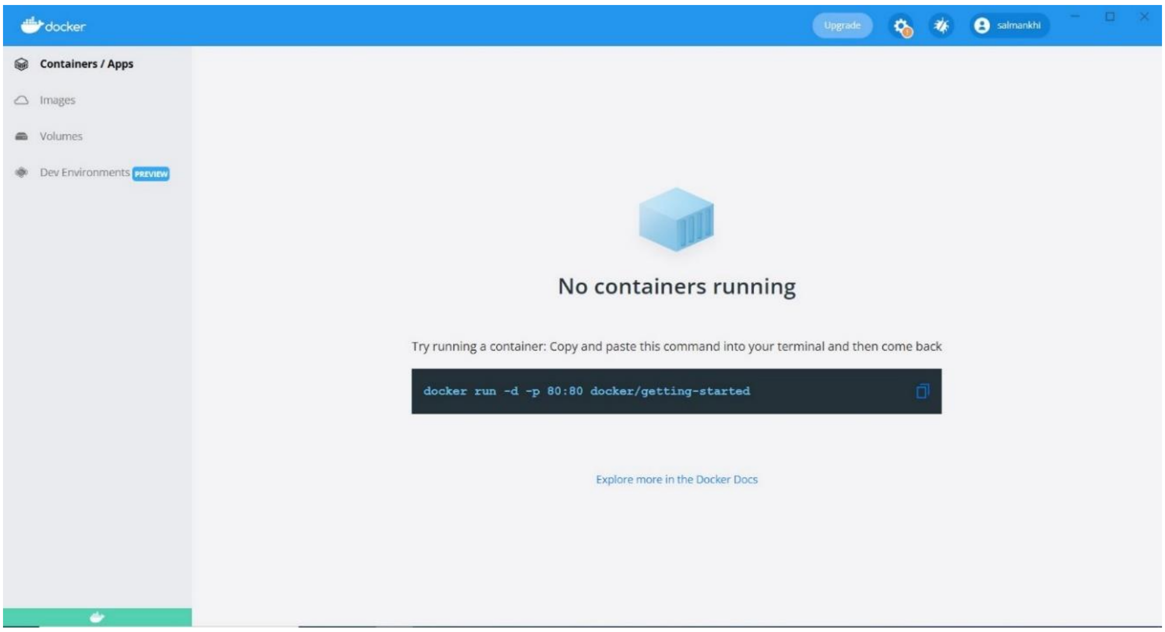
Jupyter:

Jupyter Docker Stacks were built by Project Jupyter to provide rapid and easy access to Jupyter Notebooks. The stacks are Docker images with Jupyter apps and supporting technologies that are ready to execute.

Among them is ‘jupyter/all-spark-notebook’ which contains all the dependencies to support all PySpark functionalities that we need for this machine learning project.

1. Start docker daemon:

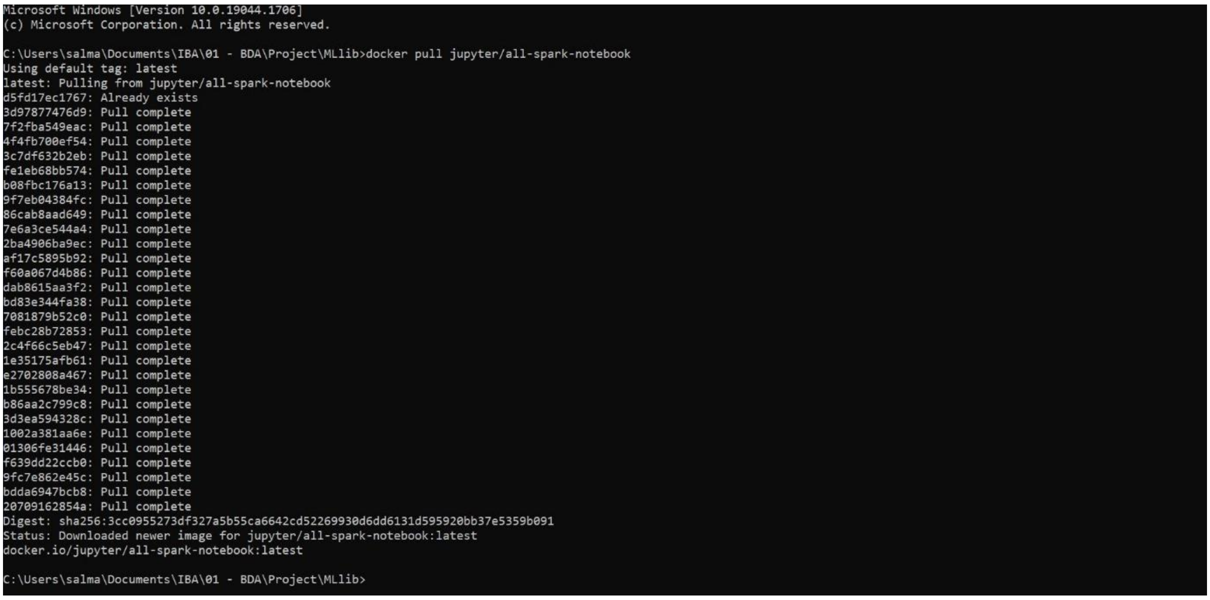
We start by running Docker for Desktop application on the computer.



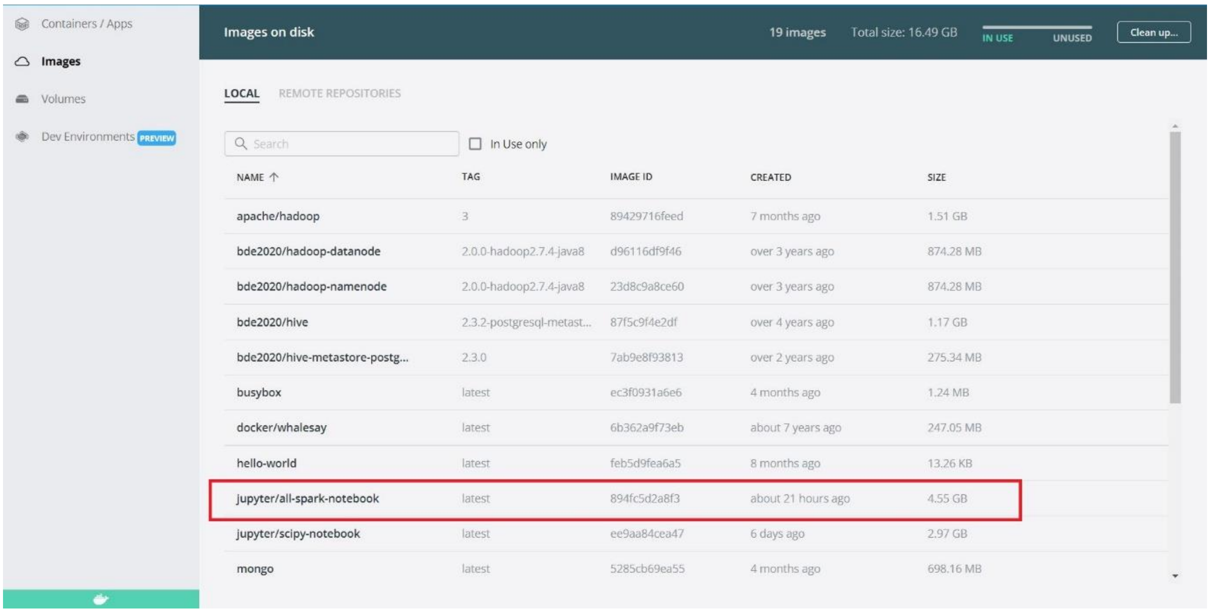
2. Pulling ‘jupyter/all-spark-notebook’ image:

We pull the image using the following command:

docker pull jupyter/all-spark-notebook



After pulling this image, we can find it in our docker daemon under images.



3. Creating a container from the pulled image:

Now we can use the pulled image to create a container, in which we will be training our machine learning model using PySpark and Spark MLlib.

We will use the following command to create a container:

docker run --rm -p 8888:8888 -v D:\BDA_project:/home/jovyan/work jupyter/all-spark-notebook

--rm: When we exit the process, this flag tells Docker to remove the container.

-p: Docker will publish a port from the container to a port on the host machine if this flag is set. The port number on the host system is the integer before the colon, and the port number in the container is the integer after the colon.

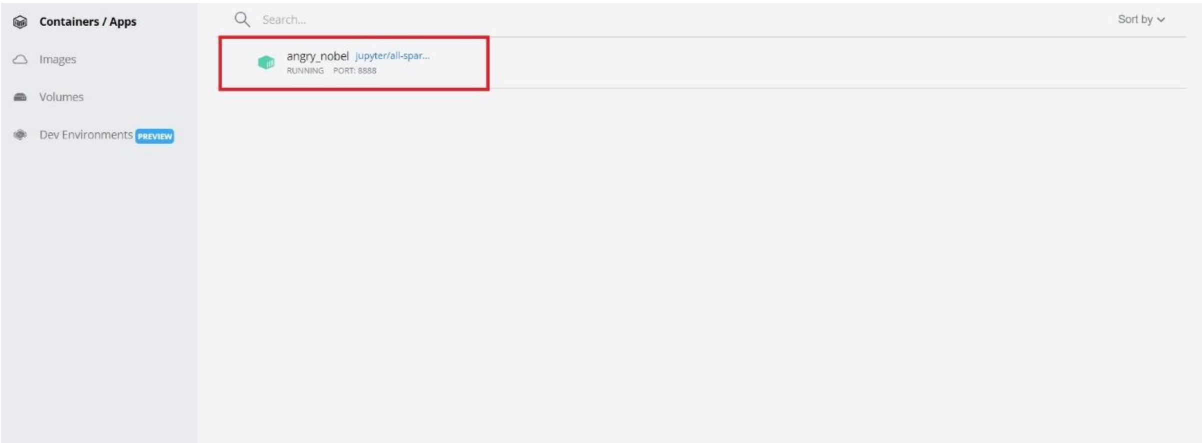
-v: This will connect a volume to the container so that we can access files from our system from the container (i.e., -v HOST PATH: CONTAINER PATH).

jupyter/all-spark-notebook: Name of the image, using which we want to create the container.

Executing the command:

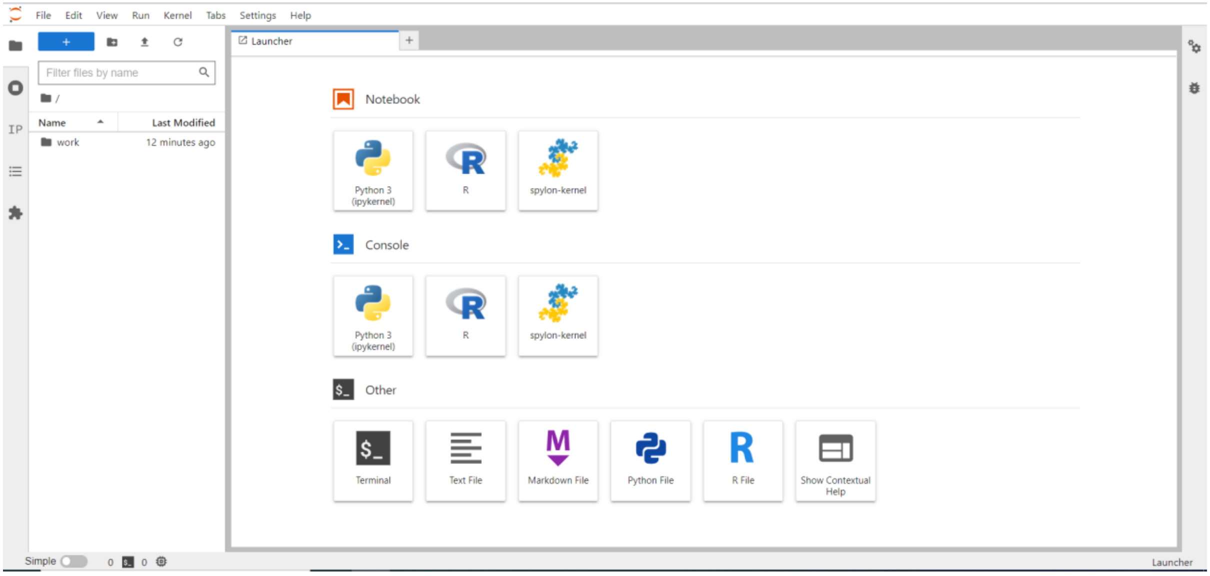
```
C:\Users\salma>docker run --rm -p 8888:8888 -v D:\BDA_project:/home/jovyan/work jupyter/all-spark-notebook
Entered start.sh with args: jupyter lab
/usr/local/bin/start.sh: running hooks in /usr/local/bin/before-notebook.d as uid / gid: 1000 / 100
/usr/local/bin/start.sh: running script /usr/local/bin/before-notebook.d/spark-config.sh
/usr/local/bin/start.sh: done running hooks in /usr/local/bin/before-notebook.d
Executing the command: jupyter lab
[I 2022-05-31 11:11:20.491 ServerApp] ipyparallel | extension was successfully linked.
[I 2022-05-31 11:11:20.517 ServerApp] jupyterlab | extension was successfully linked.
[W 2022-05-31 11:11:20.527 NotebookApp] 'ip' has moved from NotebookApp to ServerApp. This config will be passed to ServerApp. Be sure to update your config before our next release.
[W 2022-05-31 11:11:20.527 NotebookApp] 'port' has moved from NotebookApp to ServerApp. This config will be passed to ServerApp. Be sure to update your config before our next release.
[W 2022-05-31 11:11:20.527 NotebookApp] 'port' has moved from NotebookApp to ServerApp. This config will be passed to ServerApp. Be sure to update your config before our next release.
[I 2022-05-31 11:11:20.544 ServerApp] nbclassic | extension was successfully linked.
[I 2022-05-31 11:11:20.548 ServerApp] Writing Jupyter server cookie secret to /home/jovyan/.local/share/jupyter/runtime/jupyter_cookie_secret
[I 2022-05-31 11:11:20.560 ServerApp] notebook_shim | extension was successfully linked.
[I 2022-05-31 11:11:20.590 ServerApp] notebook_shim | extension was successfully loaded.
[I 2022-05-31 11:11:20.592 ServerApp] Loading IPython parallel extension
[I 2022-05-31 11:11:20.594 ServerApp] ipyparallel | extension was successfully loaded.
[I 2022-05-31 11:11:20.596 LabApp] JupyterLab extension loaded from /opt/conda/lib/python3.10/site-packages/jupyterlab
[I 2022-05-31 11:11:20.597 LabApp] JupyterLab application directory is /opt/conda/share/jupyter/lab
[I 2022-05-31 11:11:20.608 ServerApp] jupyterlab | extension was successfully loaded.
[I 2022-05-31 11:11:20.615 ServerApp] nbclassic | extension was successfully loaded.
[I 2022-05-31 11:11:20.615 ServerApp] Serving notebooks from local directory: /home/jovyan
[I 2022-05-31 11:11:20.616 ServerApp] Jupyter Server 1.17.0 is running at:
[I 2022-05-31 11:11:20.616 ServerApp] http://934ae8212251:8888/lab?token=a5c0d3f4e3e11f66017083ee535d3cc170d7792513452bfd
[I 2022-05-31 11:11:20.616 ServerApp]  or http://127.0.0.1:8888/lab?token=a5c0d3f4e3e11f66017083ee535d3cc170d7792513452bfd
[I 2022-05-31 11:11:20.616 ServerApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 2022-05-31 11:11:20.624 ServerApp]

    To access the server, open this file in a browser:
        file:///home/jovyan/.local/share/jupyter/runtime/jpserver-8-open.html
    Or copy and paste one of these URLs:
        http://934ae8212251:8888/lab?token=a5c0d3f4e3e11f66017083ee535d3cc170d7792513452bfd
     or http://127.0.0.1:8888/lab?token=a5c0d3f4e3e11f66017083ee535d3cc170d7792513452bfd
```

In docker daemon, we can see the created container:



4. Launching jupyter-lab:

Now, using the URL, given after executing the command in cmd we can access jupyter notebook and start working in it.



5. Testing Spark Setup:

Testing the spark setup, by creating a new notebook in the python environment, importing the PySpark library, and creating a spark session and data frame.

6. Training our machine learning model using spark:

6.1. Importing relevant libraries:

```
[1]: # importing required modules

from pyspark.sql import SparkSession
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import RandomForestRegressor
import time
```

6.2. Initiating spark session:

```
[2]: # initiating spark session

spark = SparkSession.builder.appName('Dataframe').getOrCreate()
```

6.3. Loading train and test dataset:

```
[3]: # loading train and test datasets

start = time.time()

df_train = spark.read.csv('trainX.csv',header=True,inferSchema=True)
df_test = spark.read.csv('testX.csv',header=True,inferSchema=True)

end = time.time()
duration = round(end-start, 2)

print(f"Time taken to load train and test datasets of size ~ 1GB: {duration} seconds")
```

```
Time taken to load train and test datasets of size ~ 1GB: 31.84 seconds
```

6.4. Transforming all predictors of the train and test data set into a vector variable:

This is to make all features compliant with the class, which will train our model in the next step.

```
[4]: # creating a list of all predictors

features_list = []
for col in df_train.dtypes:
    if col[0] != 'price_doc':
        features_list.append(col[0])
```

```
[5]: # transforming all predictors of train dataset into features, which is supported by pyspark for regression

vector_assembler = VectorAssembler(inputCols=features_list, outputCol='features')
output = vector_assembler.transform(df_train)
data = output.select("features", "price_doc")
```

```
[6]: # transforming all predictors of test dataset into features, which is supported by pyspark for regression

vector_assembler = VectorAssembler(inputCols=features_list, outputCol='features')
output = vector_assembler.transform(df_test)
test = output.select("features")
```

6.5. Training of model and predicting outcomes:

Training random forest regressor and predicting outcomes of a test dataset using that.

[7]:
```
# training random forest regressor on train dataset and then predicting results for test dataset

start = time.time()

# creating random forest object and fitting it on the train data
rf = RandomForestRegressor(featuresCol = 'features', labelCol = 'price_doc')
rfModel = rf.fit(data)

end = time.time()
duration = round(end-start, 2)

# predicting target variable for test dataset
predictions = rfModel.transform(test)

print(f"Time taken to train model on train dataset: {duration} seconds")
```

```
Time taken to train model on train dataset: 84.38 seconds
```

6.6. Performance of model:

Converting predictions to dataframe and then to CSV to find out the performance of the model.

[9]:
```
import pandas as pd
```

[10]:
```
predictions.select("prediction").toPandas().to_csv("spark_rf_pred.csv", index=False)
```

[11]:
```
row_id = pd.read_csv("ml1ch_test.csv")
row_id = row_id["row ID"]
```

[13]:
```
pred = pd.read_csv("spark_rf_pred.csv")
```

[15]:
```
pred["row ID"] = row_id
```

[17]:
```
pred = pred[["row ID", "prediction"]]
```

[19]:
```
pred.rename(columns = {'prediction':'price_doc'}, inplace = True)
```

[21]:
```
pred.to_csv('spark_rf.csv', index=False)
```

[22]:
```
pred.head()
```

[22]:

	row ID	price_doc
0	Row3	1.232080e+07
1	Row6	6.532913e+06
2	Row11	5.854936e+06
3	Row12	6.328924e+06
4	Row14	6.347550e+06

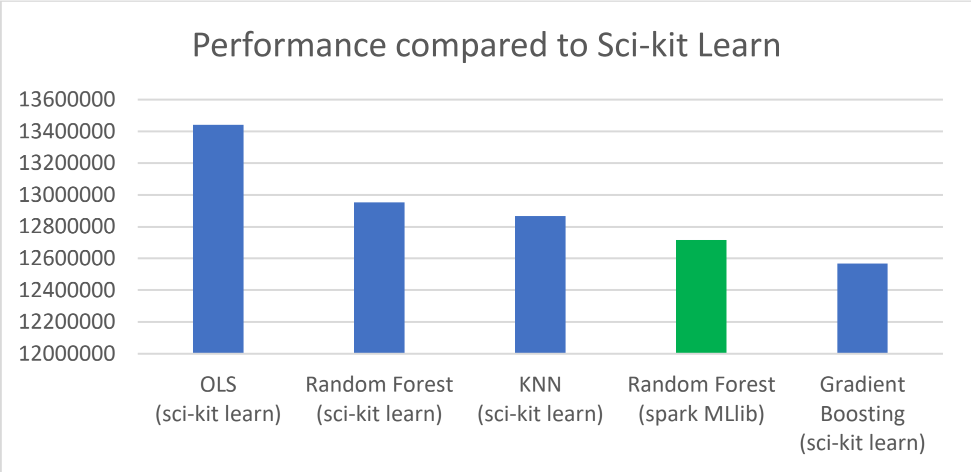
RMSE:

spark_rf.csv 12715389.48185

a minute ago by Muhammad Salman Khan

6.7. Is it good?

For comparison, I have tried this same problem using the sci-kit library before, and I had gotten the following results. (I have made these Scikit Learn submissions on a Kaggle competition, so these are legit results.)



Apart from the fact that random forest regressor has performed well, it took the least time also. I do not have figures for the other four regression models, but from experience, I can tell the only faster model than it was OLS, but it had a very weak performance.

6.8. Repeating the process with gradient boosting regressor:

We did not find better accuracy in repeating the machine learning process with gradient boosting regressor this time, and the time to train the model also increased a little.

```
[7]: # training gradient boosting regressor on train dataset and then predicting results for test dataset

start = time.time()

# creating gradient boosting object and fitting it on the train data
gb = GBTRegressor(featuresCol = 'features', labelCol = 'price_doc')
gbModel = gb.fit(data)

end = time.time()
duration = round(end-start, 2)

# predicting target variable for test dataset
predictions = gbModel.transform(test)

print(f"Time taken to train model on train dataset: {duration} seconds")

Time taken to train model on train dataset: 90.01 seconds
```

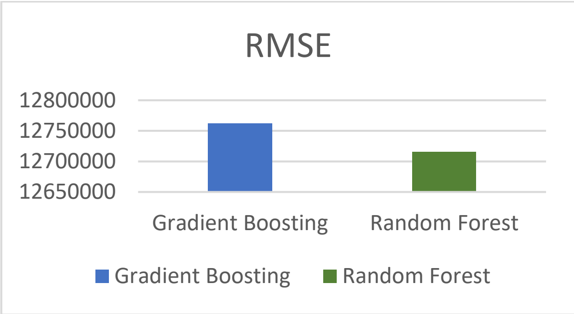
RMSE:

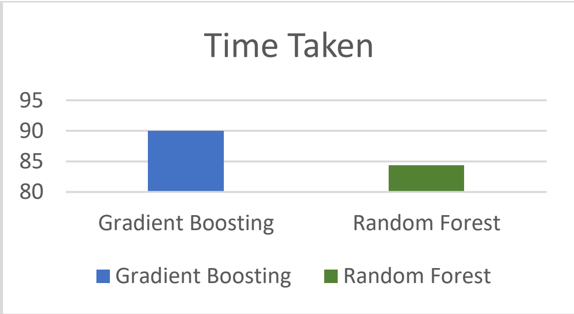
spark_gb.csv 12762382.49877

a few seconds ago by Muhammad Salman Khan

add submission details

Model comparisons:





Section 2: Machine Learning with Apache Mahout

Mahout:

Apache Mahout is an Apache Software Foundation project that employs the MapReduce paradigm and is built on top of Apache Hadoop.

It's also utilized to construct scalable, distributed machine learning algorithms for clustering, collaborative filtering, and classification. Mahout includes Java libraries for popular math algorithms and operations, as well as foundational Java collections, with a concentration on statistics and linear algebra.

Zeppelin:

Zeppelin is a web-based notebook that enables data-driven, interactive data analytics and collaborative documents with SQL, Scala, Python, R, and more.

1. Pulling apache/mahout-zeppelin:

Start docker daemon and pull the image using the following command.

docker pull apache/mahout-zeppelin:14.1

```
D:\Mahout>docker pull apache/mahout-zeppelin:14.1
14.1: Pulling from apache/mahout-zeppelin
6aa38bd67045: Pull complete
981ae4862c05: Pull complete
5bad8949dcb1: Pull complete
ca9461589e70: Pull complete
9a36db54646b: Pull complete
aadce748f6f7: Pull complete
569e366c5275: Pull complete
67c0dc51d817: Pull complete
1f35a839e5e1: Pull complete
6762bb966749: Pull complete
01b31218f3c5: Pull complete
772782988819: Pull complete
23d6794df0e1: Pull complete
12febdbbded1: Pull complete
4c6554c2c00b: Pull complete
0891cba5073e: Pull complete
4356de2ef818: Pull complete
682169bd35ac: Pull complete
14c3c3d0bdc6: Pull complete
86c0036e591c: Pull complete
143a9f22daf4: Pull complete
c48956470276: Pull complete
29d153e4f882: Pull complete
6f8e3d4514d2: Pull complete
Digest: sha256:5c3d6c99cb835383d17153b5278f5eede1214ee3967a8226833e76d426747f61
Status: Downloaded newer image for apache/mahout-zeppelin:14.1
docker.io/apache/mahout-zeppelin:14.1

D:\Mahout>
```

2. Creating and running a container with pulled image:

We will now create and run a container using the 'apache/mahout-zeppelin' image and the following command.

docker run -p 8080:8080 --rm --name mahoutContainer apache/mahout-zeppelin:14.1

```
C:\Users\salma>docker run -p 8080:8080 --rm --name mahoutContainer apache/mahout-zeppelin:14.1
OpenJDK 64-Bit Server VM warning: ignoring option MaxPermSize=512m; support was removed in 8.0
 WARN [2022-06-02 10:49:09,097] ({main} ZeppelinConfiguration.java[create]:159) - Failed to load configuration, proceeding with a default
 INFO [2022-06-02 10:49:09,170] ({main} ZeppelinConfiguration.java[create]:171) - Server Host: 0.0.0.0
 INFO [2022-06-02 10:49:09,171] ({main} ZeppelinConfiguration.java[create]:173) - Server Port: 8080
 INFO [2022-06-02 10:49:09,171] ({main} ZeppelinConfiguration.java[create]:177) - Context Path: /
 INFO [2022-06-02 10:49:09,171] ({main} ZeppelinConfiguration.java[create]:178) - Zeppelin Version: 0.9.0-preview1
 INFO [2022-06-02 10:49:09,209] ({main} Log.java[initialized]:193) - Logging initialized @1643ms to org.eclipse.jetty.util.log.Slf4jLog
 WARN [2022-06-02 10:49:09,627] ({main} ZeppelinConfiguration.java[getConfigFSDir]:631) - zeppelin.config.fs.dir is not specified, fall back to local conf directory zeppelin.conf.dir
 INFO [2022-06-02 10:49:09,671] ({main} Credentials.java[loadFromFile]:121) - /zeppelin/conf/credentials.json
 INFO [2022-06-02 10:49:09,716] ({ImmediateThread-1654166949621} PluginManager.java[loadNotebookRepo]:60) - Loading NotebookRepo Plugin: org.apache.zeppelin.notebook.repo.GitNotebookRepo
 INFO [2022-06-02 10:49:10,077] ({ImmediateThread-1654166949621} VFSNotebookRepo.java[setNotebookDirectory]:70) - Using notebookDir: /zeppelin/notebook
 INFO [2022-06-02 10:49:10,606] ({main} ZeppelinServer.java[setupWebAppContext]:488) - warPath is: /zeppelin/zeppelin-web-0.9.0-preview1.war
 INFO [2022-06-02 10:49:10,621] ({main} ZeppelinServer.java[setupWebAppContext]:501) - ZeppelinServer Webapp path: /zeppelin/webapps
 INFO [2022-06-02 10:49:10,900] ({main} ZeppelinServer.java[setupWebAppContext]:488) - warPath is: /zeppelin/zeppelin-web-angular-0.9.0-preview1.war
 INFO [2022-06-02 10:49:10,902] ({main} ZeppelinServer.java[setupWebAppContext]:501) - ZeppelinServer Webapp path: /zeppelin/webapps/next
 INFO [2022-06-02 10:49:11,073] ({ImmediateThread-1654166949621} GitNotebookRepo.java[init]:77) - Opening a git repo at '/zeppelin/notebook'
 INFO [2022-06-02 10:49:11,082] ({main} NotebookServer.java[<init>]:153) - NotebookServer instantiated: org.apache.zeppelin.socket.NotebookServer@3098cf3b
 INFO [2022-06-02 10:49:11,094] ({main} NotebookServer.java[setNotebook]:164) - Injected NotebookProvider
 INFO [2022-06-02 10:49:11,095] ({main} NotebookServer.java[setServiceLocator]:158) - Injected ServiceLocator: ServiceLocatorImpl(shared-locator,0,1859039536)
 INFO [2022-06-02 10:49:11,098] ({main} NotebookServer.java[setNotebookService]:171) - Injected NotebookServiceProvider
 INFO [2022-06-02 10:49:11,100] ({main} NotebookServer.java[setAuthorizationServiceProvider]:178) - Injected NotebookAuthorizationServiceProvider
 INFO [2022-06-02 10:49:11,101] ({main} NotebookServer.java[setConnectionManagerProvider]:184) - Injected ConnectionManagerProvider
 INFO [2022-06-02 10:49:11,104] ({main} ZeppelinServer.java[setupClusterManagerServer]:439) - Cluster mode is disabled
 INFO [2022-06-02 10:49:11,150] ({main} ZeppelinServer.java[main]:251) - Starting zeppelin server
 INFO [2022-06-02 10:49:11,157] ({main} Server.java[doStart]:370) - jetty-9.4.18.v20190429; built: 2019-04-29T20:42:08.989Z; git: e1bc35120a6617ee3df052294e433f3a25ce7097; jvm 1.8.0_265-8u265-b01-0ubuntu2~16.04-
b01
 INFO [2022-06-02 10:49:11,370] ({ImmediateThread-1654166949621} GitNotebookRepo.java[init]:81) - Git repo /zeppelin/notebook/.git does not exist, creating a new one
 INFO [2022-06-02 10:49:11,478] ({main} StandardDescriptorProcessor.java[visitServlet]:283) - NO JSP Support for /, did not find org.eclipse.jetty.jsp.JettyJspServlet
 INFO [2022-06-02 10:49:11,513] ({main} DefaultSessionIdManager.java[doStart]:365) - DefaultSessionIdManager workerName=node0
 INFO [2022-06-02 10:49:11,514] ({main} DefaultSessionIdManager.java[doStart]:370) - No SessionScavenger set, using defaults
 INFO [2022-06-02 10:49:11,530] ({main} HouseKeeper.java[startScavenging]:149) - node0 Scavenging every 600000ms
 INFO [2022-06-02 10:49:13,572] ({main} ContextHandler.java[doStart]:855) - Started o.e.j.w.WebAppContext@3fc2959f{zeppelin-web,/,jar:file:///zeppelin/zeppelin-web-0.9.0-preview1.war!/,AVAILABLE}{/zeppelin/zeppe
lin-web-0.9.0-preview1.war}
 INFO [2022-06-02 10:49:13,865] ({main} StandardDescriptorProcessor.java[visitServlet]:283) - NO JSP Support for /next, did not find org.eclipse.jetty.jsp.JettyJspServlet
 INFO [2022-06-02 10:49:14,939] ({main} ContextHandler.java[doStart]:855) - Started o.e.j.w.WebAppContext@233795b6{zeppelin-web-angular,/next,jar:file:///zeppelin/zeppelin-web-angular-0.9.0-preview1.war!/,AVAILA
BLE}{/zeppelin/zeppelin-web-angular-0.9.0-preview1.war}
 INFO [2022-06-02 10:49:15,095] ({main} AbstractConnector.java[doStart]:292) - Started ServerConnector@1877ab81{HTTP/1.1,[http/1.1]}{0.0.0.0:8080}
 INFO [2022-06-02 10:49:15,096] ({main} Server.java[doStart]:410) - Started @7535ms
 INFO [2022-06-02 10:49:20,097] ({main} ZeppelinServer.java[main]:265) - Done, zeppelin server started
```

3. Bashing into the created container:

We will keep the cmd session running after the previous command and will open another command prompt to continue with the bashing. Now, we can bash into the container using the following command.

docker exec -it mahoutContainer bash

```
C:\Users\salma>docker exec -it mahoutContainer bash
zeppelin@f5ba2661c651:~$
```

4. Starting zeppelin:

We can now start the zeppelin notebook using the following command in the bashed session.

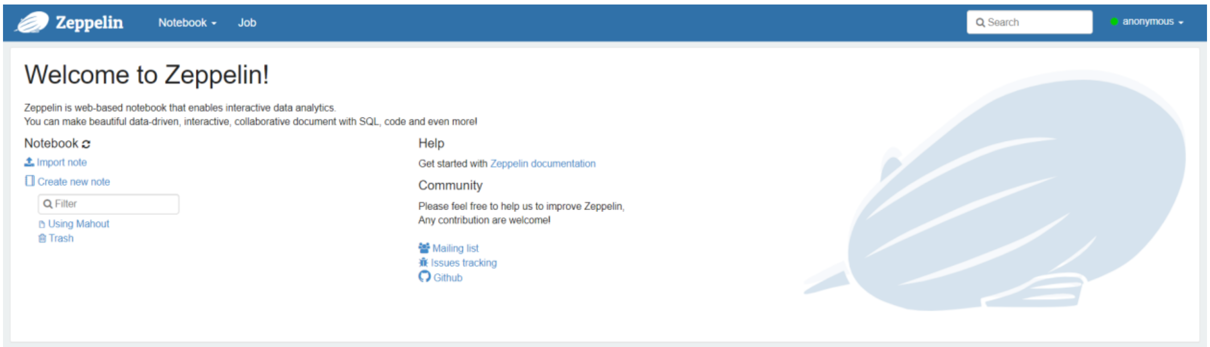
bin/zeppelin-daemon.sh start

```
zeppelin@f5ba2661c651:~$ bin/zeppelin-daemon.sh start
Zeppelin start                                             [  OK  ]
zeppelin@f5ba2661c651:~$
```

5. Accessing zeppelin using a web browser:

Now, we can access the zeppelin notebook, through a URL using a web browser.

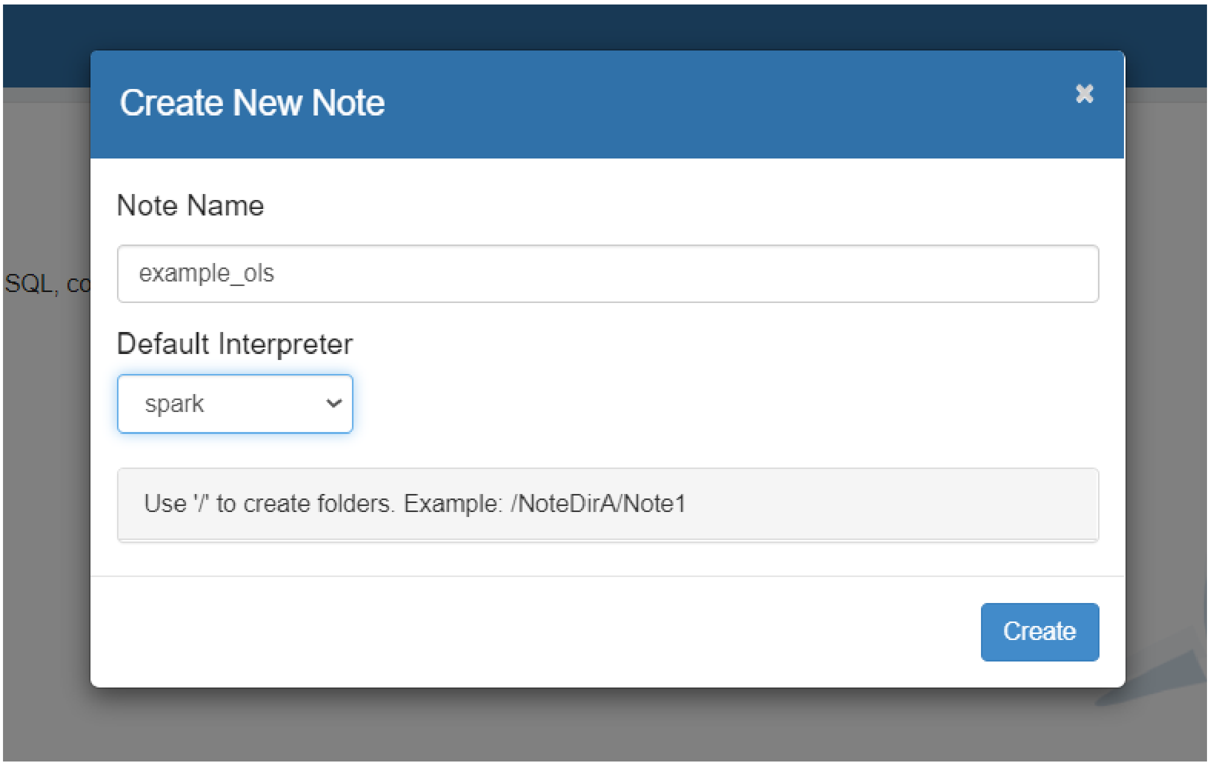
http://localhost:8080/



6. Training an example OLS model:

6.1. Create a new note:

Create a new note with the default interpreter 'spark'.



6.2. Import relevant modules:

```
import org.apache.mahout.math.algorithms.regression.OrdinaryLeastSquares
```
FINISHED

```
import org.apache.mahout.math.algorithms.regression.OrdinaryLeastSquares
```

Took 0 sec. Last updated by anonymous at June 02 2022, 4:49:00 PM.

```
import org.apache.mahout.math._
import org.apache.mahout.math.scalabindings._
import org.apache.mahout.math.drm._
import org.apache.mahout.math.scalabindings.RLikeOps._
import org.apache.mahout.math.drm.RLikeDrmOps._
import org.apache.mahout.sparkbindings._

implicit val sdc: org.apache.mahout.sparkbindings.SparkDistributedContext = sc2sdc(sc)
```
FINISHED

```
import org.apache.mahout.math._
import org.apache.mahout.math.scalabindings._
import org.apache.mahout.math.drm._
import org.apache.mahout.math.scalabindings.RLikeOps._
import org.apache.mahout.math.drm.RLikeDrmOps._
import org.apache.mahout.sparkbindings._
sdc: org.apache.mahout.sparkbindings.SparkDistributedContext = org.apache.mahout.sparkbindings.SparkDistributedContext@6ebc54c4
```

Took 1 sec. Last updated by anonymous at June 02 2022, 4:49:47 PM.

6.3. Create an example dataset:

```
val drmData = drmParallelize(dense(
  (2, 2, 10.5, 10, 29.509541),  // Apple Cinnamon Cheerios
  (1, 2, 12,   12, 18.042851),  // Cap'n'Crunch
  (1, 1, 12,   13, 22.736446),  // Cocoa Puffs
  (2, 1, 11,   13, 32.207582),  // Froot Loops
  (1, 2, 12,   11, 21.871292),  // Honey Graham Ohs
  (2, 1, 16,   8,  36.187559),  // Wheaties Honey Gold
  (6, 2, 17,   1,  50.764999),  // Cheerios
  (3, 2, 13,   7,  40.400208),  // Clusters
  (3, 3, 13,   4,  45.811716)), numPartitions = 2)
```

```
drmData: org.apache.mahout.math.drm.CheckpointedDrm[Int] = org.apache.mahout.sparkbindings.drm.CheckpointedDrmSpark@3c95521f
```

Took 2 sec. Last updated by anonymous at June 02 2022, 4:50:12 PM.

6.4. Looking at the features:

```
drmData.collect(::, 0 until 4)
```

```
res1: org.apache.mahout.math.Matrix =
{
 0 =>	{0:2.0,1:2.0,2:10.5,3:10.0}
 1 =>	{0:1.0,1:2.0,2:12.0,3:12.0}
 2 =>	{0:1.0,1:1.0,2:12.0,3:13.0}
 3 =>	{0:2.0,1:1.0,2:11.0,3:13.0}
 4 =>	{0:1.0,1:2.0,2:12.0,3:11.0}
 5 =>	{0:2.0,1:1.0,2:16.0,3:8.0}
 6 =>	{0:6.0,1:2.0,2:17.0,3:1.0}
 7 =>	{0:3.0,1:2.0,2:13.0,3:7.0}
 8 =>	{0:3.0,1:3.0,2:13.0,3:4.0}
}
```

Took 4 sec. Last updated by anonymous at June 02 2022, 4:50:34 PM.

6.5. Training OLS Model:

```
val drmX = drmData(::, 0 until 4)
val drmY = drmData(::, 4 until 5)

drmX: org.apache.mahout.math.drm.DrmLike[Int] = OpMapBlock(org.apache.mahout.sparkbindings.drm.CheckpointedDrmSpark@b90371d,<function1>,4,-1,true)
drmY: org.apache.mahout.math.drm.DrmLike[Int] = OpMapBlock(org.apache.mahout.sparkbindings.drm.CheckpointedDrmSpark@b90371d,<function1>,1,-1,true)
```

Took 0 sec. Last updated by anonymous at June 02 2022, 10:17:41 PM.

```
val model = new OrdinaryLeastSquares[Int]().fit(drmX, drmY, 'calcCommonStatistics → false)

model: org.apache.mahout.math.algorithms.regression.OrdinaryLeastSquaresModel[Int] = org.apache.mahout.math.algorithms.regression.OrdinaryLeastSquaresModel@4c7ae87f
```

Took 3 sec. Last updated by anonymous at June 02 2022, 10:18:02 PM.

```
println(model.summary)
```

Coef.	Estimate	Std. Error	t-score	Pr(Beta=0)
X0	-1.33627	+2.68781	-0.49716	+0.64516
X1	-13.15770	+5.39398	-2.43933	+0.07126
X2	-4.15265	+1.78491	-2.32654	+0.08056
X3	-5.67991	+1.88687	-3.01022	+0.03954
X4	+163.17933	+51.91530	+3.14318	+0.03474

Took 1 sec. Last updated by anonymous at June 02 2022, 10:18:26 PM.

Although the zeppelin was good for interactive programming and all, loading data on its container was not possible for an unknown reason. I will now use another image for loading my data and perform regression on it.

Second Approach:

1. Creating container from image michabirklbauer/mahout:

We will start by pulling the image and running its container, using the following command.

docker run -it michabirklbauer/mahout:latest

```
C:\Users\salma>docker run -it michabirklbauer/mahout:latest
Unable to find image 'michabirklbauer/mahout:latest' locally
latest: Pulling from michabirklbauer/mahout
0ecb575e629c: Pull complete
7467d1831b69: Pull complete
feab2c490a3c: Pull complete
f15a0f46f8c3: Pull complete
26cb1dfcbebb: Pull complete
5b224ce6d4ea: Pull complete
c932fe81bb40: Pull complete
0c39e3902a25: Pull complete
18fadcbee53b: Pull complete
ee3d54adf2d0: Pull complete
8e3d56b510ec: Pull complete
eee173c37d0f: Pull complete
2abd8bfd3e61: Pull complete
76866a3d54fd: Pull complete
bfb87913f780: Pull complete
2bfe793a1e2d: Pull complete
912fcec458b0: Pull complete
dec3180c7883: Pull complete
f643bf1aed1b: Pull complete
3c6258235011: Pull complete
536e64712f6c: Pull complete
e47d855c16f2: Pull complete
0aa6c9a8bbf4: Pull complete
8d6493e48cb8: Pull complete
a51c2f69bd7b: Pull complete
Digest: sha256:0234fa0dcc1f86eedff78bdc64dba947641cfcbf52ee8241507cf143d482e2f4
Status: Downloaded newer image for michabirklbauer/mahout:latest
Adding lib/ to CLASSPATH
22/06/02 13:22:35 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Spark context Web UI available at http://f2c89b7fe022:4040
Spark context available as 'sc' (master = local[*], app id = local-1654176164542).
Spark session available as 'spark'.


                  _                 _
_ __ ___   __ _| |__   ___  _   _| |_
 '_ ` _ \ / _` | '_ \ / _ \| | | | __|
 | | | | | (_| | | | | (_) | |_| | |_
_| |_| |_|\__,_|_| |_|\___/ \__,_|\__|  version 0.14
```

2. Copying dataset into the created container (in windows cmd):

```
C:\Users\salma>docker ps
CONTAINER ID   IMAGE                          COMMAND                  CREATED         STATUS         PORTS     NAMES
f2c89b7fe022   michabirklbauer/mahout:latest   "./mahout/bin/mahout…"   5 minutes ago   Up 4 minutes             vigilant_mcclintock

C:\Users\salma>docker cp trainX.csv vigilant_mcclintock:\apache

C:\Users\salma>docker cp testX.csv vigilant_mcclintock:\apache
```

3. Initiating spark session in the bashed container:

```
scala> val spark = org.apache.spark.sql.SparkSession.builder.master("local").appName("Spark CSV Reader").getOrCreate;
22/06/02 13:32:20 WARN SparkSession$Builder: Using an existing SparkSession; some spark core configurations may not take effect.
spark: org.apache.spark.sql.SparkSession = org.apache.spark.sql.SparkSession@f542e08
```

4. Creating a dataframe with data copied:

```
scala> val df = spark.read.format("csv").option("header", "true").option("mode", "DROPMALFORMED").load("trainX.csv")
df: org.apache.spark.sql.DataFrame = [full_sq: string, life_sq: string ... 272 more fields]
```

5. Repeat the process:

With this loaded data, we can repeat the steps taken above in the former approach to train our linear regression model and find out the target variable through predictions.

Section 3: Spark MLlib vs Apache Mahout

Conclusion:

Their framework is the most significant difference. Hadoop MapReduce is the framework for Mahout, whereas Spark is the framework for MLlib. MLlib is a Spark-based library of unconnected high-level algorithms. This is how Mahout used to work when Hadoop MapReduce was the only option.

Flexibility in regression analysis:

MLlib has turned out to be a clear winner in regression analysis, it provides a vast range of models from OLS to KNNs and from random forest to gradient boosting. While using Mahout we can only train OLS and ridge regressor models.

For classification and recommendation models, mahout has better options available.

Performance:

If your ML algorithm is mapped to the single MR job, the main difference will be only startup overhead, which is dozens of seconds for Hadoop MR, and let's say one second for Spark. So, in the case of model training, it is not that important.

If your method is mapped to many jobs, things will be different. In this situation, the difference in overhead for every iteration will be the same, which might be a game changer.

Let's say we require 100 iterations, each of which will take 5 seconds of cluster CPU.

On Spark, it will take (100*5 + 100*1) = 600 seconds.

On MR(Mahout), it will take (100*5 + 100*30) = 3500 seconds.

At the same time, Hadoop MR is a far more developed framework than Spark, and if you have a lot of data and stability is a must, Mahout is a real contender.