SMOKE TEST DOCUMENT

Spark Smoke Test Cases

Date Prepared: August 2019

Document Information

Project Name	**Spark Smoke Test Document**		
Project Owner		**Document Version No**	1.0
Quality Review Method	By email/HP SharePoint		
Prepared By		**Preparation Date**	August 2019
Reviewed By	Refer to version history	**Review Date**	

Table of Contents

1 NOTE

Set SPARK_HOME to $PATH or “cd /usr/lib/spark/spark-2.3.1-bin-hadoop2.7/” and run the following examples

All the Spark examples related to Python, Scala, Java & R are under this location “/usr/lib/spark/spark-2.3.1-bin-hadoop2.7/examples/src/main”

Replace <Spark-master-IP> with the actual IP address

Reference link: https://spark.apache.org/docs/latest/submitting-applications.html

2 SAMPLE TEST CASE FOR SPARK-SUBMIT

- Run application locally on 8 cores

```
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master local[8] \
  /usr/lib/spark/spark-2.3.1-bin- hadoop2.7/examples/jars/spark-examples_2.11-2.3.1.jar \
  100
```

- Run below command on a Spark standalone cluster in client deploy mode

```
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master spark://<Spark-master-IP>:7077 \
  --executor-memory 20G \
  --total-executor-cores 100 \
 /usr/lib/spark/spark-2.3.1-bin-hadoop2.7/examples/jars/spark-examples_2.11-2.3.1.jar \
  1000
```

- Run below command on a Spark standalone cluster in cluster deploy mode with supervise

```
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master spark://<Spark-master-IP>:7077 \
  --deploy-mode cluster \
  --supervise \
```

- Run a python application on a Spark standalone cluster

```
./bin/spark-submit \
 --class org.apache.spark.examples.SparkPi \
 --master spark://<Spark-master-IP>:7077 \
 --deploy-mode cluster \
 --supervise \
```

3 SAMPLE TEST CASE FOR SPARK-SHELL

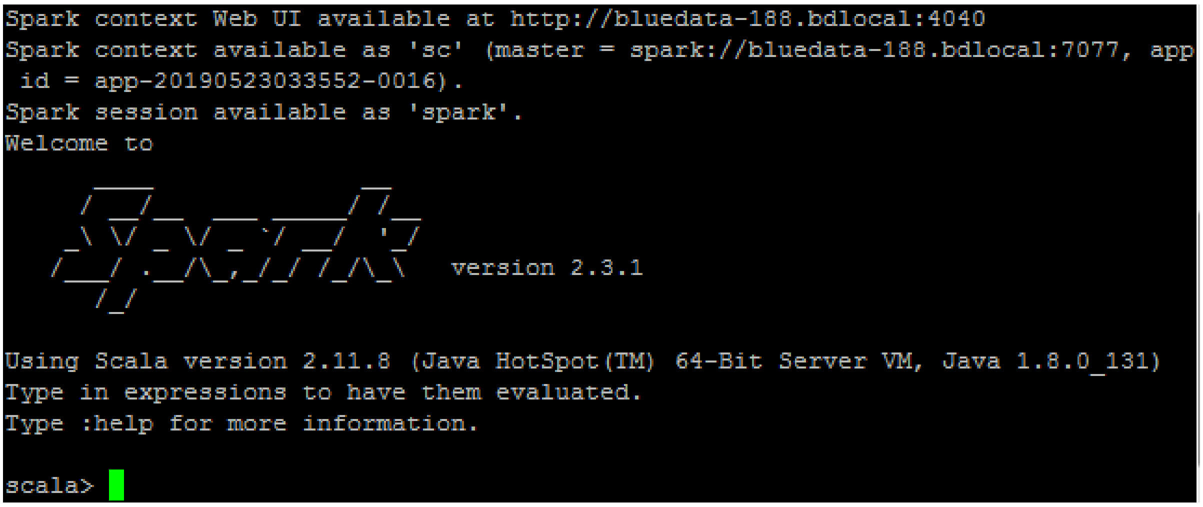
We will first introduce the API through Spark’s interactive shell (in Python or Scala) and then show how to write applications in Java, Scala and Python.

3.1 Interactive analysis with the Spark Shell

Spark’s shell provides a simple way to learn the API, as well as a powerful tool to analyze data interactively. It is available in either Scala (which runs on the Java VM and is thus a good way to use existing Java libraries) or Python.

Start it by running the following in the Spark directory

./bin/spark-shell



```
Spark context Web UI available at http://bluedata-188.bdlocal:4040
Spark context available as 'sc' (master = spark://bluedata-188.bdlocal:7077, app
 id = app-20190523033552-0016).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.3.1
      /_/

Using Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_131)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

- Make a new Dataset from the text of the README file in the Spark source directory:

  scala> val textFile = spark.read.textFile("README.md")

  textFile: org.apache.spark.sql.Dataset[String] = [value: string]

- Get values from Dataset directly, by calling some actions, or transform the Dataset to get new one

```
scala> textFile.count() // Number of items in this Dataset

res0: Long = 126 // May be different from yours as README.md

scala> textFile.first() // First item in this Dataset

res1: String = # Apache Spark
```

- Transform this Dataset into a new one

```
scala> val linesWithSpark = textFile.filter(line =>line.contains("Spark"))

linesWithSpark: org.apache.spark.sql.Dataset[String] =[value: string]
```

- Chain together transformations and actions

```
scala> textFile.filter(line    =>line.contains("Spark")).count() // How many lines contain
"Spark"?

res3: Long = 15
```

3.2 Caching operations on Spark Shell

Spark also supports pulling data sets into a cluster-wide in-memory cache. This is very useful when data is accessed repeatedly.

```
scala> linesWithSpark.cache()

res7: linesWithSpark.type = [value: string]

scala> linesWithSpark.count()
```

res8: Long = 15

scala> linesWithSpark.count()

res9: Long = 15

Reference Link: https://spark.apache.org/docs/latest/quick-start.html#basics

3.3 Example for Scala Word Count program:

Following are the commands that we shall use for Word Count Example in Spark Shell

- Using Spark context variable, sc to read a text file

  scala> sc.textFile("usr/lib/spark/spark-2.3.1-bin-hadoop2.7/word.txt”)

- Split each line using space ” ” as separator

  scala> flatMap(line => line.split(" "))

- Map each word to a tuple (word, 1), 1 being the number of occurrences of word

  scala> map(word => (word,1))

- Reduce all the words based on Key

  scala> var counts = map.reduceByKey(_ + _);

- Save counts to local file

```
scala> counts.saveAsTextFile("usr/lib/spark/spark-2.3.1-bin-hadoop2.7/result.txt”)
```

4 TEST CASES FOR JUPYTERHUB

- Create a Linux user on the master controller node or login as AD user.
- Login to Jupyterhub.

Note: All the Spark examples related to Python, Scala, Java & R are under this location "/usr/lib/spark/spark-2.3.1-bin-hadoop2.7/examples/src/main/"

4.1 Spark Scala testing

Start a toree scala kernel -> Wait till kernel creates a spark shell. Run following Pearson's correlation. You can run up to 4 Spark shells with current configurations. If your shell doesn't start, you may have used up all the cores. Kill unused Kernels to release resources

Code: Running Pearson's correlation using mllib

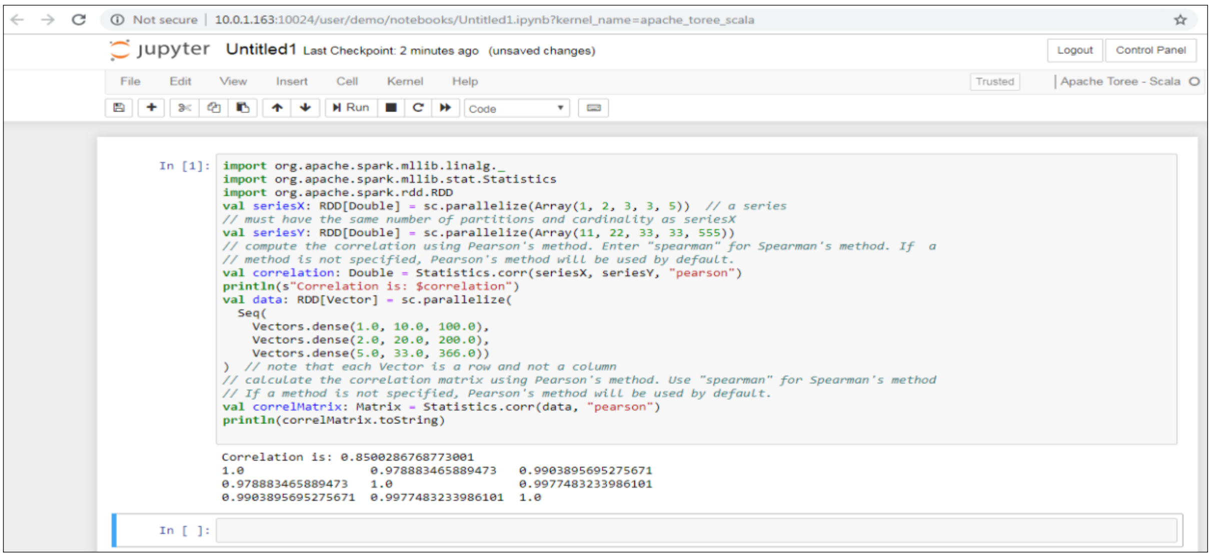
Note: You can copy the sample code below from this link: https://spark.apache.org/docs/latest/mllib-statistics.html

```
import org.apache.spark.mllib.linalg._

import org.apache.spark.mllib.stat.Statistics

import org.apache.spark.rdd.RDD

val seriesX: RDD[Double] = sc.parallelize(Array(1, 2, 3, 3, 5))  // a series

// must have the same number of partitions and cardinality as seriesX

val seriesY: RDD[Double] = sc.parallelize(Array(11, 22, 33, 33, 555))

// compute the correlation using Pearson's method. Enter "spearman" for Spearman's method. If  a

// method is not specified, Pearson's method will be used by default.

val correlation: Double = Statistics.corr(seriesX, seriesY, "pearson")
```

```
println(s"Correlation is: $correlation")

val data: RDD[Vector] = sc.parallelize(

  Seq(

    Vectors.dense(1.0, 10.0, 100.0),

    Vectors.dense(2.0, 20.0, 200.0),

    Vectors.dense(5.0, 33.0, 366.0))

)  // note that each Vector is a row and not a column

// calculate the correlation matrix using Pearson's method. Use "spearman" for Spearman's method

// If a method is not specified, Pearson's method will be used by default.

val correlMatrix: Matrix = Statistics.corr(data, "pearson")

println(correlMatrix.toString)
```

Input: Input is generated within the code. No external input is provided.

Output: Sample output is as given below.


Not secure | 10.0.1.163:10024/user/demo/notebooks/Untitled1.ipynb?kernel_name=apache_toree_scala
jupyter Untitled1 Last Checkpoint: 2 minutes ago (unsaved changes)
Logout
Control Panel
File Edit View Insert Cell Kernel Help
Trusted
Apache Toree - Scala
Run
Code

In [1]:

```
import org.apache.spark.mllib.linalg._
import org.apache.spark.mllib.stat.Statistics
import org.apache.spark.rdd.RDD
val seriesX: RDD[Double] = sc.parallelize(Array(1, 2, 3, 3, 5))  // a series
// must have the same number of partitions and cardinality as seriesX
val seriesY: RDD[Double] = sc.parallelize(Array(11, 22, 33, 33, 555))
// compute the correlation using Pearson's method. Enter "spearman" for Spearman's method. If  a
// method is not specified, Pearson's method will be used by default.
val correlation: Double = Statistics.corr(seriesX, seriesY, "pearson")
println(s"Correlation is: $correlation")
val data: RDD[Vector] = sc.parallelize(
  Seq(
    Vectors.dense(1.0, 10.0, 100.0),
    Vectors.dense(2.0, 20.0, 200.0),
    Vectors.dense(5.0, 33.0, 366.0))
)  // note that each Vector is a row and not a column
// calculate the correlation matrix using Pearson's method. Use "spearman" for Spearman's method
// If a method is not specified, Pearson's method will be used by default.
val correlMatrix: Matrix = Statistics.corr(data, "pearson")
println(correlMatrix.toString)
```

```
Correlation is: 0.8500286768773001
1.0                 0.978883465889473   0.9903895695275671
0.978883465889473   1.0                 0.9977483233986101
0.9903895695275671  0.9977483233986101  1.0
```

In [ ]:

4.2 PySpark testing

Start a toree pySpark kernel -> Wait till kernel creates a spark shell. You can run up to 4 Spark shells with current configurations. If your shell doesn’t start, you may have used up all the cores. Kill unused Kernels to release resources.

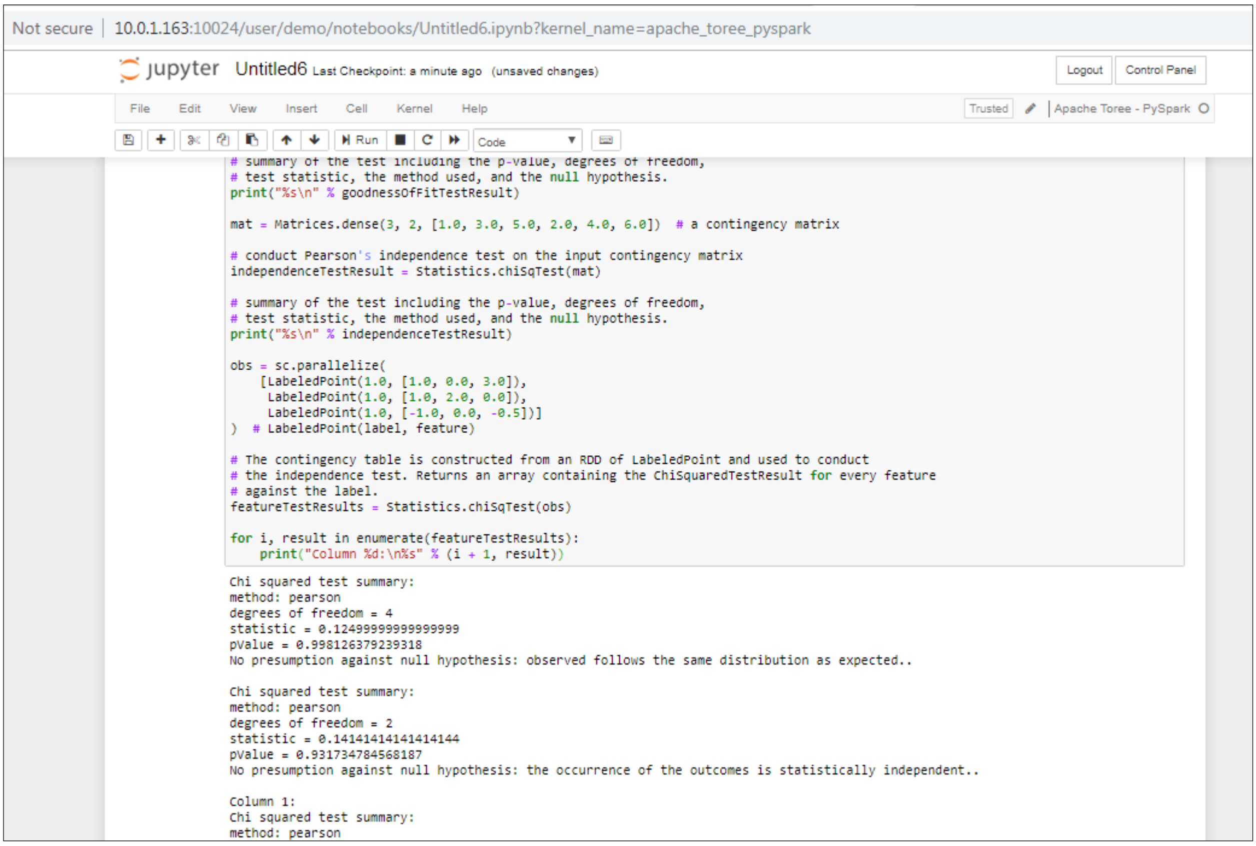
Code:

```
from pyspark.mllib.linalg import Matrices, Vectors

from pyspark.mllib.regression import LabeledPoint

from pyspark.mllib.stat import Statistics


vec = Vectors.dense(0.1, 0.15, 0.2, 0.3, 0.25)  # a vector composed of the frequencies of events


# compute the goodness of fit. If a second vector to test against

# is not supplied as a parameter, the test runs against a uniform distribution.

goodnessOfFitTestResult = Statistics.chiSqTest(vec)


# summary of the test including the p-value, degrees of freedom,

# test statistic, the method used, and the null hypothesis.

print("%s\n" % goodnessOfFitTestResult)


mat = Matrices.dense(3, 2, [1.0, 3.0, 5.0, 2.0, 4.0, 6.0])  # a contingency matrix
```

```
# conduct Pearson's independence test on the input contingency matrix

independenceTestResult = Statistics.chiSqTest(mat)



# summary of the test including the p-value, degrees of freedom,

# test statistic, the method used, and the null hypothesis.

print("%s\n" % independenceTestResult)



obs = sc.parallelize(

    [LabeledPoint(1.0, [1.0, 0.0, 3.0]),

     LabeledPoint(1.0, [1.0, 2.0, 0.0]),

     LabeledPoint(1.0, [-1.0, 0.0, -0.5])]

)  # LabeledPoint(label, feature)



# The contingency table is constructed from an RDD of LabeledPoint and used to conduct

# the independence test. Returns an array containing the ChiSquaredTestResult for every feature

# against the label.

featureTestResults = Statistics.chiSqTest(obs)



for i, result in enumerate(featureTestResults):

    print("Column %d:\n%s" % (i + 1, result))
```

Input: No input files used. Data is generated in the code.

Output: Sample output is as given below.


Not secure | 10.0.1.163:10024/user/demo/notebooks/Untitled6.ipynb?kernel_name=apache_toree_pyspark
jupyter Untitled6 Last Checkpoint: a minute ago (unsaved changes)
Logout
Control Panel
File Edit View Insert Cell Kernel Help
Trusted
Apache Toree - PySpark
Run
Code

```
# summary of the test including the p-value, degrees of freedom,
# test statistic, the method used, and the null hypothesis.
print("%s\n" % goodnessOfFitTestResult)

mat = Matrices.dense(3, 2, [1.0, 3.0, 5.0, 2.0, 4.0, 6.0])  # a contingency matrix

# conduct Pearson's independence test on the input contingency matrix
independenceTestResult = Statistics.chiSqTest(mat)

# summary of the test including the p-value, degrees of freedom,
# test statistic, the method used, and the null hypothesis.
print("%s\n" % independenceTestResult)

obs = sc.parallelize(
    [LabeledPoint(1.0, [1.0, 0.0, 3.0]),
     LabeledPoint(1.0, [1.0, 2.0, 0.0]),
     LabeledPoint(1.0, [-1.0, 0.0, -0.5])]
)  # LabeledPoint(label, feature)

# The contingency table is constructed from an RDD of LabeledPoint and used to conduct
# the independence test. Returns an array containing the ChiSquaredTestResult for every feature
# against the label.
featureTestResults = Statistics.chiSqTest(obs)

for i, result in enumerate(featureTestResults):
    print("Column %d:\n%s" % (i + 1, result))
```

```
Chi squared test summary:
method: pearson
degrees of freedom = 4
statistic = 0.12499999999999999
pValue = 0.998126379239318
No presumption against null hypothesis: observed follows the same distribution as expected..

Chi squared test summary:
method: pearson
degrees of freedom = 2
statistic = 0.14141414141414144
pValue = 0.931734784568187
No presumption against null hypothesis: the occurrence of the outcomes is statistically independent..

Column 1:
Chi squared test summary:
method: pearson
```

4.3 Execute Spark Sumbit job on JupterHub

Start a toree pySpark kernel -> Wait till kernel creates a spark shell. You can run up to 4 Spark shells with current configurations. If your shell doesn’t start, you may have used up all the cores. Kill unused Kernels to release resources.

Code:

```
##sh

./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master local[8] \
  /usr/lib/spark/spark-2.3.1-bin-hadoop2.7/examples/jars/spark-examples_2.11-2.3.1.jar \
 100
```

Output:

Check Spark master GUI that job is running under Running Applications section

5 SAMPLE TEST CASES FOR SPARK WITH NOTEBOOKS

5.1 PySpark testing

Start a toree pySpark kernel -> Wait till kernel creates a spark shell. You can run up to 4 Spark shells with current configurations. If your shell doesn’t start, you may have used up all the cores. Kill unused Kernels to release resources.

Code:

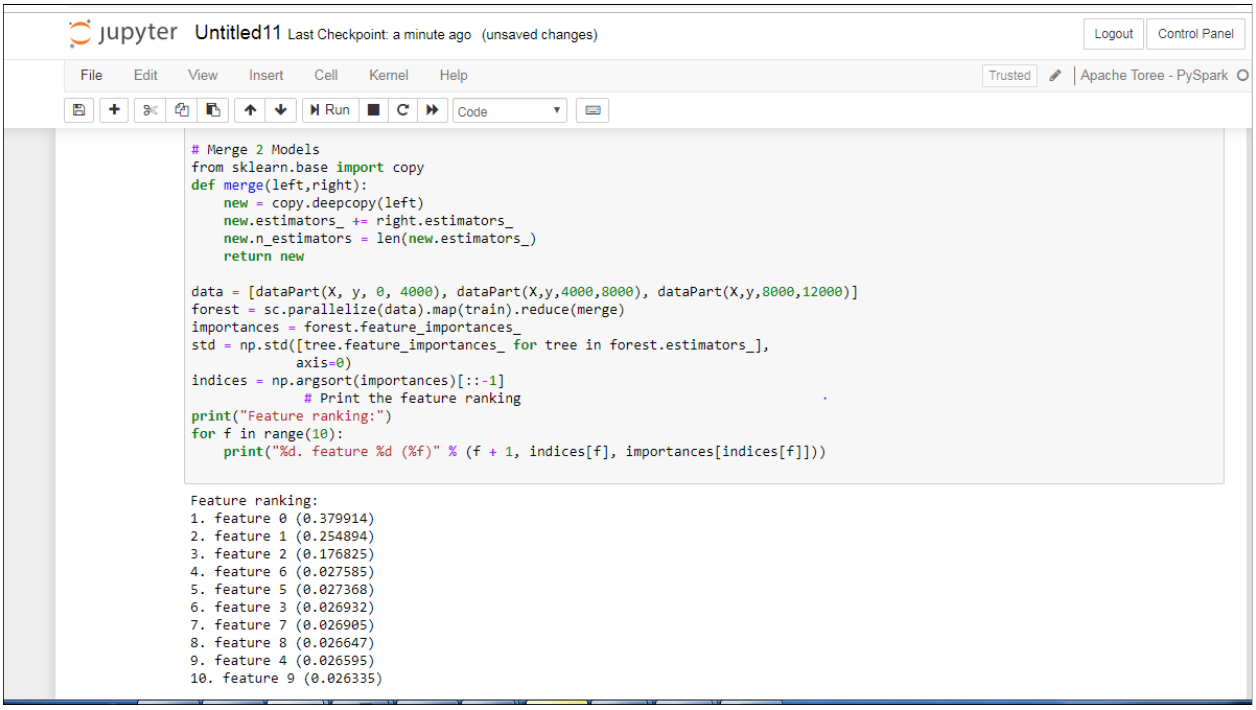
```
from pyspark import SparkConf, SparkContext

from sklearn.datasets import make_classification

from sklearn.ensemble import ExtraTreesClassifier

import pandas as pd

import numpy as np

# Build a classification task using 3 informative features

X, y = make_classification(n_samples=12000,

                    n_features=10,

                    n_informative=3,

                    n_redundant=0,

                    n_repeated=0,

                    n_classes=2,

                    random_state=0,

                    shuffle=False)
```

```
# Partition data

def dataPart(X, y, start, stop): return dict(X=X[start:stop, :], y=y[start:stop])

def train(data):

    X = data['X']

    y = data['y']

    return ExtraTreesClassifier(n_estimators=100,random_state=0).fit(X,y)

# Merge 2 Models

from sklearn.base import copy

def merge(left,right):

    new = copy.deepcopy(left)

    new.estimators_ += right.estimators_

    new.n_estimators = len(new.estimators_)

    return new

data = [dataPart(X, y, 0, 4000), dataPart(X,y,4000,8000), dataPart(X,y,8000,12000)]

forest = sc.parallelize(data).map(train).reduce(merge)

importances = forest.feature_importances_

std = np.std([tree.feature_importances_ for tree in forest.estimators_],

             axis=0)

indices = np.argsort(importances)[::-1]

              # Print the feature ranking

print("Feature ranking:")
```

for f in range(10):

print("%d. feature %d (%f)" % (f + 1, indices[f], importances[indices[f]]))

Output: Sample output is as given below.



```
# Merge 2 Models
from sklearn.base import copy
def merge(left,right):
    new = copy.deepcopy(left)
    new.estimators_ += right.estimators_
    new.n_estimators = len(new.estimators_)
    return new

data = [dataPart(X, y, 0, 4000), dataPart(X,y,4000,8000), dataPart(X,y,8000,12000)]
forest = sc.parallelize(data).map(train).reduce(merge)
importances = forest.feature_importances_
std = np.std([tree.feature_importances_ for tree in forest.estimators_],
             axis=0)
indices = np.argsort(importances)[::-1]
              # Print the feature ranking
print("Feature ranking:")
for f in range(10):
    print("%d. feature %d (%f)" % (f + 1, indices[f], importances[indices[f]]))
```

```
Feature ranking:
1. feature 0 (0.379914)
2. feature 1 (0.254894)
3. feature 2 (0.176825)
4. feature 6 (0.027585)
5. feature 5 (0.027368)
6. feature 3 (0.026932)
7. feature 7 (0.026905)
8. feature 8 (0.026647)
9. feature 4 (0.026595)
10. feature 9 (0.026335)
```

5.2 Spark Scala testing

Start a toree scala kernel -> Wait till kernel creates a spark shell. Run following Pearson’s correlation. You can run up to 4 Spark shells with current configurations. If your shell doesn’t start, you may have used up all the cores. Kill unused Kernels to release resources

Code:

import org.apache.spark.mllib.linalg._

import org.apache.spark.mllib.stat.Statistics

```
import org.apache.spark.rdd.RDD

val seriesX: RDD[Double] = sc.parallelize(Array(1, 2, 3, 3, 5))  // a series

// must have the same number of partitions and cardinality as seriesX

val seriesY: RDD[Double] = sc.parallelize(Array(11, 22, 33, 33, 555))

// compute the correlation using Pearson's method. Enter "spearman" for Spearman's method. If a

// method is not specified, Pearson's method will be used by default.

val correlation: Double = Statistics.corr(seriesX, seriesY, "pearson")

println(s"Correlation is: $correlation")

                val data: RDD[Vector] = sc.parallelize(

  Seq(

    Vectors.dense(1.0, 10.0, 100.0),

    Vectors.dense(2.0, 20.0, 200.0),

    Vectors.dense(5.0, 33.0, 366.0))

)  // note that each Vector is a row and not a column

              // calculate the correlation matrix using Pearson's method. Use "spearman" for Spearman's
method

// If a method is not specified, Pearson's method will be used by default.

val correlMatrix: Matrix = Statistics.corr(data, "pearson")

println(correlMatrix.toString)
```

Spark R-studio test on jupyter notebook

Open R-studio GUI and execute the following scrip library(data.table)

dt <- data.table(1:3)

print(dt)

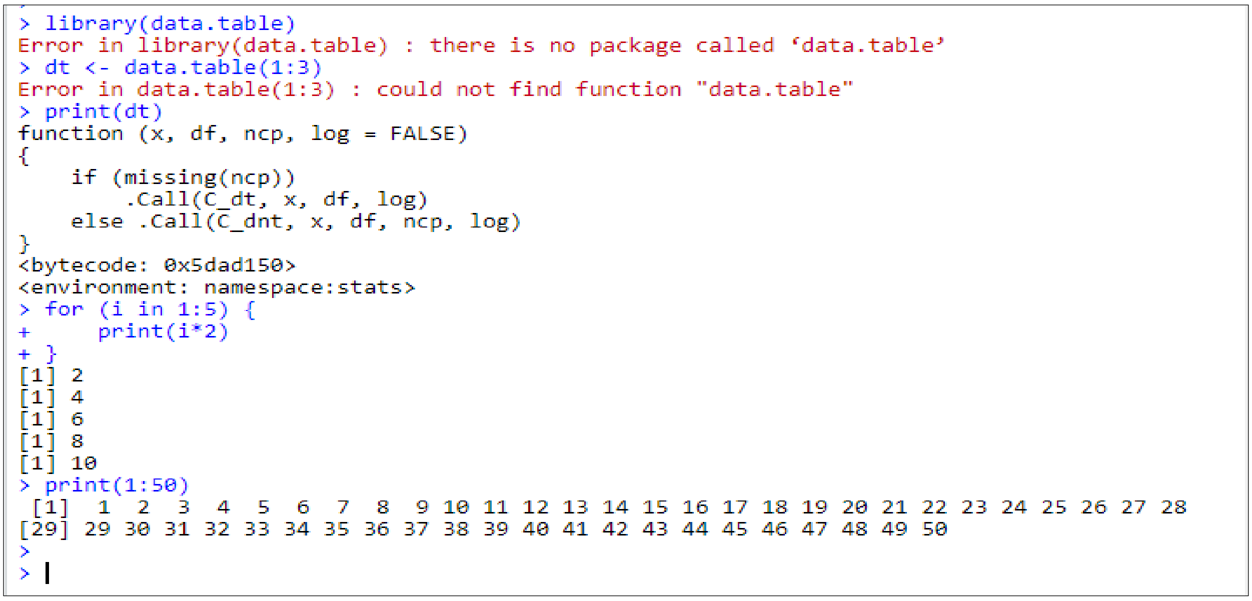
for (i in 1:5) {

  print(i*2)

}

print(1:50)

Output: Sample output is as given below.



```
> library(data.table)
Error in library(data.table) : there is no package called ‘data.table’
> dt <- data.table(1:3)
Error in data.table(1:3) : could not find function "data.table"
> print(dt)
function (x, df, ncp, log = FALSE)
{
    if (missing(ncp))
        .Call(C_dt, x, df, log)
    else .Call(C_dnt, x, df, ncp, log)
}
<bytecode: 0x5dad150>
<environment: namespace:stats>
> for (i in 1:5) {
+     print(i*2)
+ }
[1] 2
[1] 4
[1] 6
[1] 8
[1] 10
> print(1:50)
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
[29] 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
>
> |
```

6 TEST CASES FOR SPARK WITH RSTUDIO

- Create a Linux user on the master controller node or login as AD user.
- Login to R-studio

6.1 Base-R testing on RStudio GUI

data(iris) # Load the dataset iris

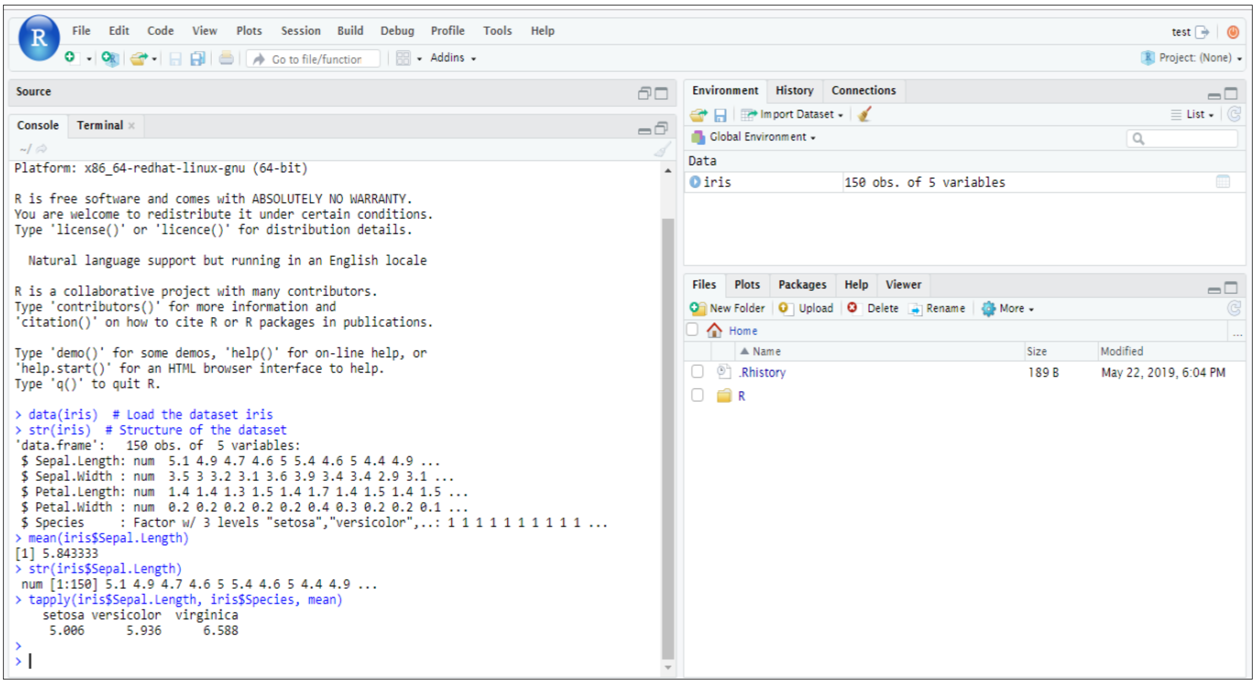
str(iris) # Structure of the dataset

mean(iris$Sepal.Length)

str(iris$Sepal.Length)

tapply(iris$Sepal.Length, iris$Species, mean)

Output: Sample output is as given below.



```
Platform: x86_64-redhat-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> data(iris)  # Load the dataset iris
> str(iris)  # Structure of the dataset
'data.frame':	150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
> mean(iris$Sepal.Length)
[1] 5.843333
> str(iris$Sepal.Length)
 num [1:150] 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
> tapply(iris$Sepal.Length, iris$Species, mean)
    setosa versicolor  virginica 
     5.006      5.936      6.588 
> 
> |
```

Data

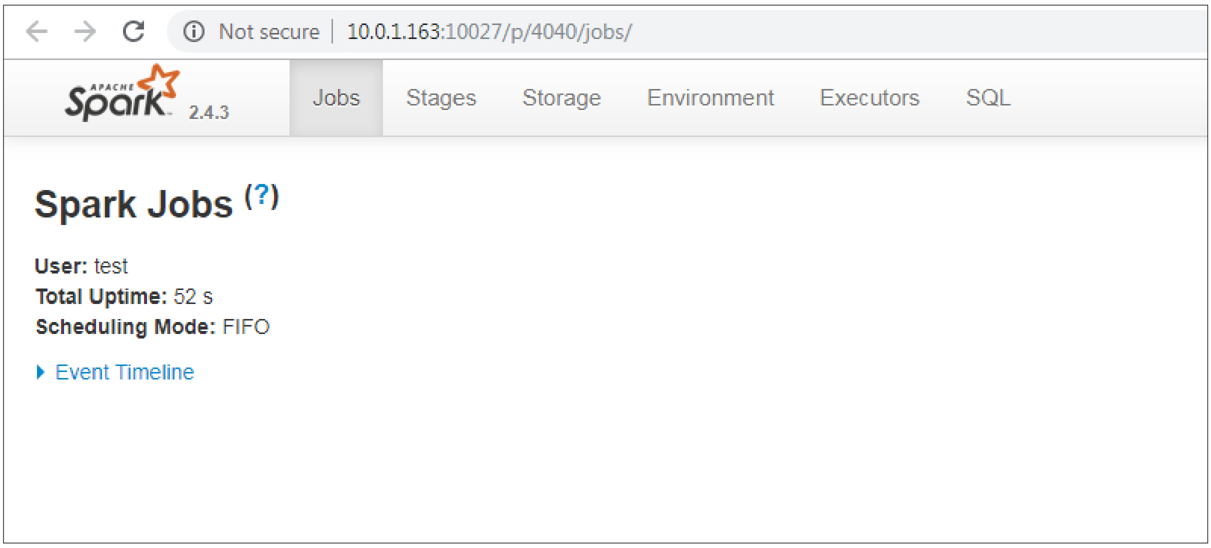
iris	150 obs. of 5 variables

Name	Size	Modified
.Rhistory	189 B	May 22, 2019, 6:04 PM
R		

6.2 Sparklyr testing on RStudio GUI

```
>install.packages("sparklyr")

>sparklyr::spark_install()

>library(sparklyr)

>sc <- spark_connect(master = 'local')
```

Output: Sample output is as given below.



Spark Jobs

User: test
Total Uptime: 52 s
Scheduling Mode: FIFO

6.3 Simple test on RStudio GUI

data(iris) # Load the dataset iris

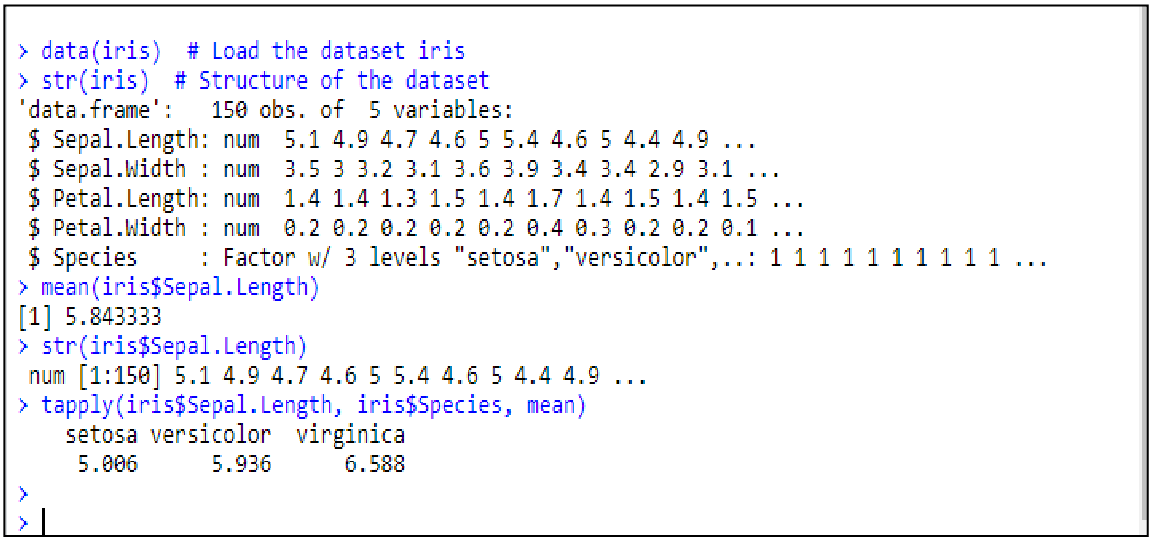
str(iris) # Structure of the dataset

mean(iris$Sepal.Length)

str(iris$Sepal.Length)

tapply(iris$Sepal.Length, iris$Species, mean)

Output: Sample output is as given below



```
> data(iris)  # Load the dataset iris
> str(iris)  # Structure of the dataset
'data.frame':   150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
> mean(iris$Sepal.Length)
[1] 5.843333
> str(iris$Sepal.Length)
 num [1:150] 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
> tapply(iris$Sepal.Length, iris$Species, mean)
    setosa versicolor  virginica 
     5.006      5.936      6.588 
> 
> 
```

6.4 MLLib usage test on RSudio GUI

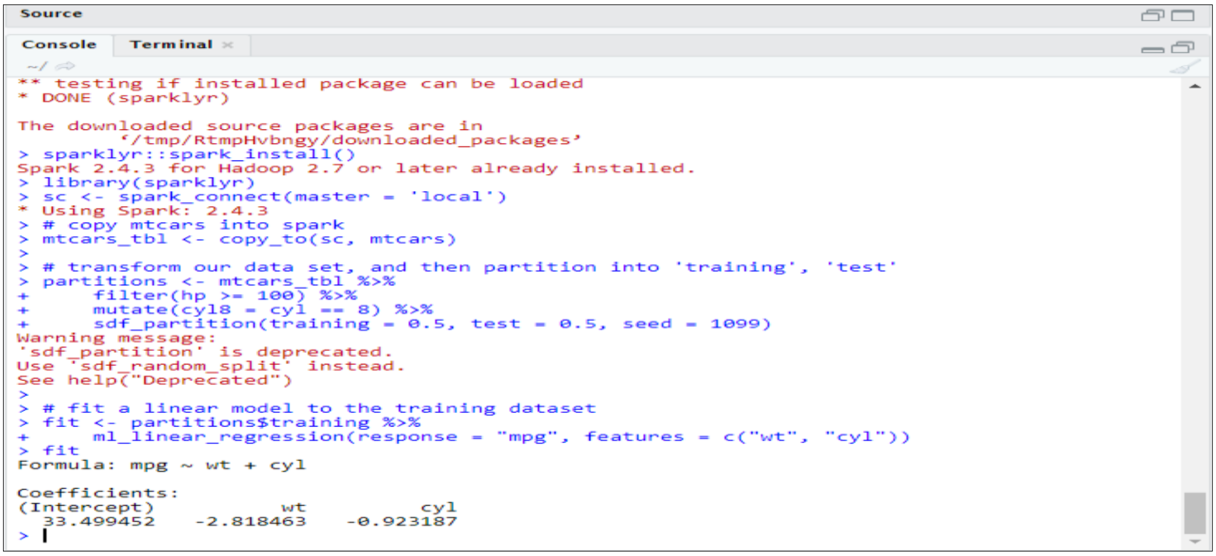
>install.packages("sparklyr")

>sparklyr::spark_install()

>library(sparklyr)

```
>sc <- spark_connect(master = 'local')

> library(dplyr)

# copy mtcars into spark
> mtcars_tbl <- copy_to(sc, mtcars)

 #  ** May show an error regarding problem with database. Seems to work OK after that
>src_tbls(sc)
# transform our data set, and then partition into 'training', 'test'
> partitions <- mtcars_tbl %>%
  filter(hp >= 100) %>%
  mutate(cyl8 = cyl == 8) %>%
  sdf_partition(training = 0.5, test = 0.5, seed = 1099)

# fit a linear model to the training dataset
> fit <- partitions$training %>%
  ml_linear_regression(response = "mpg", features = c("wt", "cyl"))

> summary(fit)
```

Output: Sample output is as given below



```
** testing if installed package can be loaded
* DONE (sparklyr)

The downloaded source packages are in
        '/tmp/RtmpHvbngy/downloaded_packages'
> sparklyr::spark_install()
Spark 2.4.3 for Hadoop 2.7 or later already installed.
> library(sparklyr)
> sc <- spark_connect(master = 'local')
* Using Spark: 2.4.3
> # copy mtcars into spark
> mtcars_tbl <- copy_to(sc, mtcars)
>
> # transform our data set, and then partition into 'training', 'test'
> partitions <- mtcars_tbl %>%
+     filter(hp >= 100) %>%
+     mutate(cyl8 = cyl == 8) %>%
+     sdf_partition(training = 0.5, test = 0.5, seed = 1099)
Warning message:
'sdf_partition' is deprecated.
Use 'sdf_random_split' instead.
See help("Deprecated")
>
> # fit a linear model to the training dataset
> fit <- partitions$training %>%
+     ml_linear_regression(response = "mpg", features = c("wt", "cyl"))
> fit
Formula: mpg ~ wt + cyl

Coefficients:
(Intercept)          wt         cyl
  33.499452   -2.818463   -0.923187
> 
```

7 TEST CASES FOR SPARK WITH SQL

Spark SQL allows relational queries expressed in SQL or Scala to be executed using Spark. At the core of this component is a new type of RDD, SchemaRDD. SchemaRDDs are composed of Row objects, along with a schema that describes the data types of each column in the row. A SchemaRDD is similar to a table in a traditional relational database. The SchemaRDD can be created from an existing RDD, a Parquet file, a JSON dataset.

Follow these tests for testing spark-sql for your cluster:

7.1 Testing with user defined functions

- Creating a dataset “hello world”

```
 val dataset = Seq((0, "hello"),(1, "world")).toDF("id","text")
```

- Defining a function ‘upper’ which converts a string into upper case

```
val upper: String => String =_.toUpperCase
```

- We now import the ‘udf’ package into Spark

```
import org.apache.spark.sql.functions.udf
```

- Defining our UDF, ‘upperUDF’ and importing our function ‘upper’

```
val upperUDF = udf(upper)
```

- Displaying the results of our User Defined Function in a new column ‘upper’

```
dataset.withColumn("upper", upperUDF('text)).show
```

7.2 Starting a Spark Session and displaying DataFrame of people.json

For the querying examples, we will be using files, ’people.txt’ and ’people.json’. These file stored at‘/usr/lib/spark/spark-2.3.1-bin-hadoop2.7/examples/src/main/resources/’

- We first import a Spark Session into Apache Spark

```
import org.apache.spark.sql.SparkSession
```

- Creating a Spark Session ‘spark’ using the ‘builder()’ function

```
val spark = SparkSession.builder().appName("Spark SQL basic
example").config("spark.some.config.option", "some-value").getOrCreate()
```

- Importing the Implicts class into our ‘spark’ Session.

```
import spark.implicits._
```

- We now create a DataFrame ‘df’ and import data from the ’employee.json’ file.

```
val df = spark.read.json("examples/src/main/resources/people.json ")
```

- Displaying the DataFrame ‘df’. The result is a table of 5 rows of ages and names from our ’employee.json’ file.

```
df.show()
```

7.3 Creating a Dataset

- Creating a class ‘Employee’ to store name and age of an employee

```
case class Employee(name: String, age: Long)
```

- Assigning a Dataset ‘caseClassDS’ to store the record of Andrew

```
val caseClassDS = Seq(Employee("Andrew", 55)).toDS()
```

- Displaying the Dataset ‘caseClassDS’

```
caseClassDS.show()
```

- Creating a primitive Dataset to demonstrate mapping of DataFrames into Datasets

```
val primitiveDS = Seq(1, 2, 3).toDS
```

- Assigning the above sequence into an array

```
primitiveDS.map(_ + 1).collect()
```