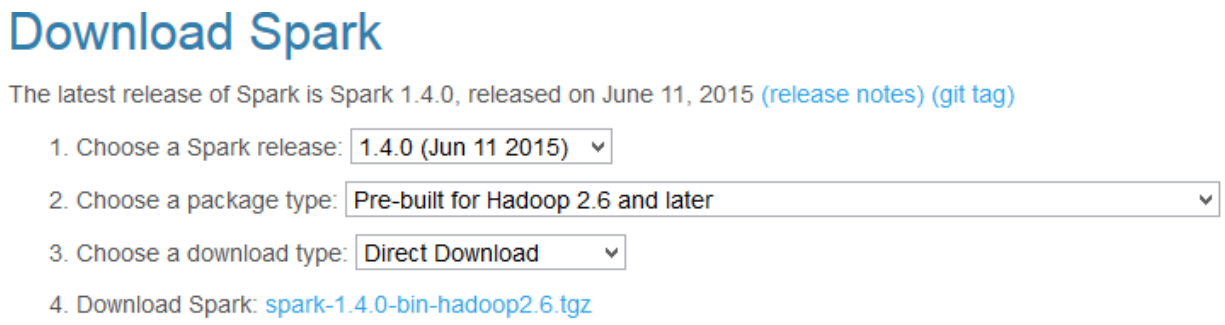
Assignment 1

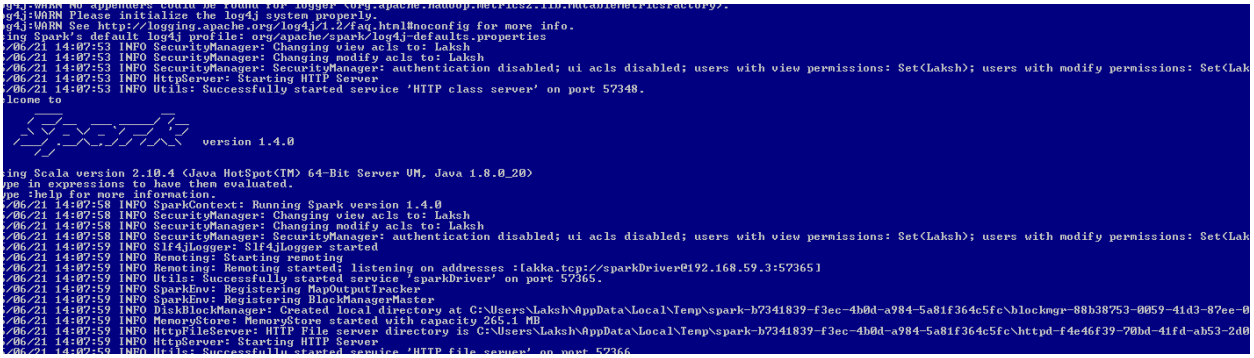
1. Basic Software Prerequisites:

1. Download the latest release of Spark (Spark 1.4)with Hadoop libraries:



2. Extract the zip in the C: folder
3. Download the **winutils.exe** file from the internet and place it in the **C:\spark-1.4.0-bin-hadoop2.6\bin**
4. Set the following variable in the Env path :
   SPARK_HOME = C:\spark-1.4.0-bin-hadoop2.6
   HADDOOP_HOME = %SPARK_HOME%
   PATH=%SPARK_HOME%/bin
5. Check if the SPARK is working properly :
   Go to CMD or Cygwin (if you installed) and run the following command it will open the Spark environment:

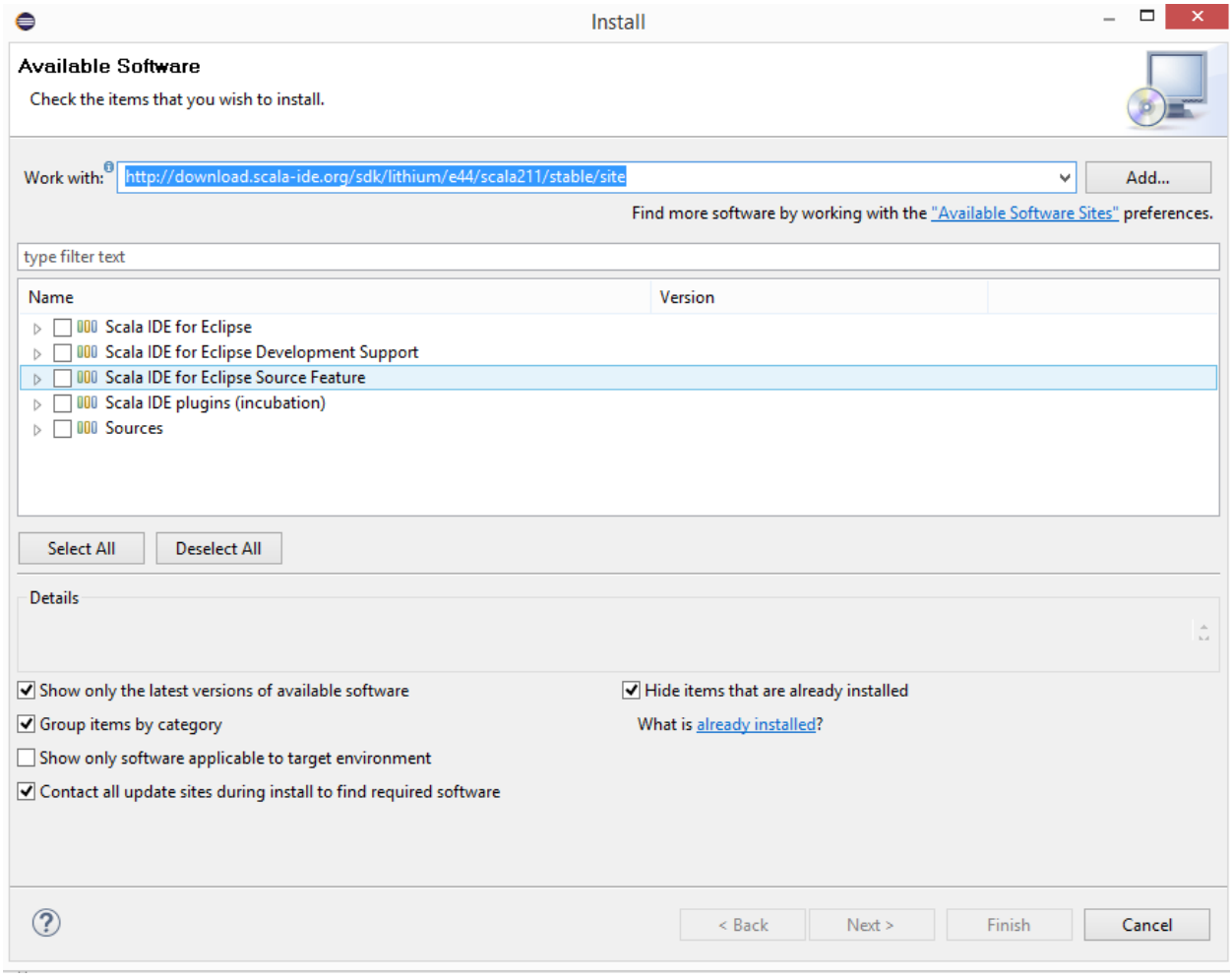
   C:\spark-1.4.0-bin-hadoop2.6\bin>spark-shell



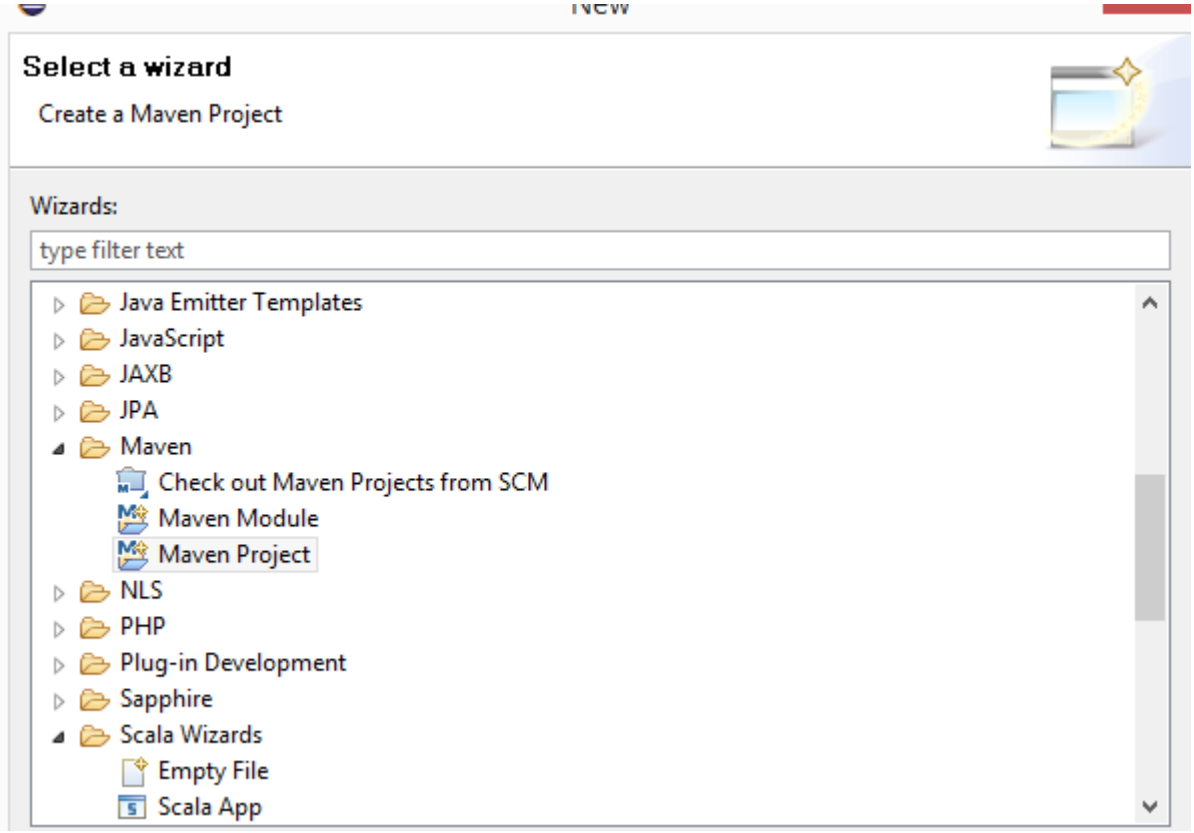
2. Scala:

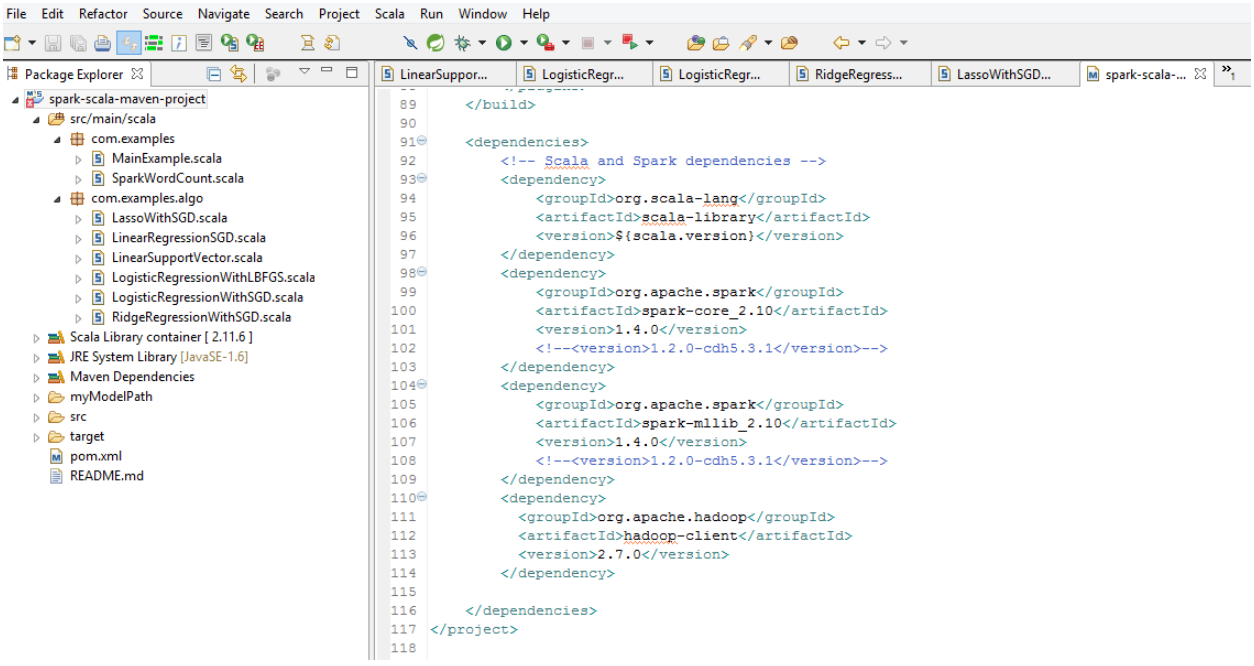
Software Prerequisites:

1. Download Java 8 and Install and set the JAVA_HOME path in the environment variable.
2. Install Maven Plugins in eclipse from Eclipse Market place.
3. Download Eclipse and Install Scala plugins from Eclipse Market Place:



4. Create New Maven Project from the Eclipse (See below Screenshot)


New
Select a wizard
Create a Maven Project
Wizards:
type filter text
Java Emitter Templates
JavaScript
JAXB
JPA
Maven
Check out Maven Projects from SCM
Maven Module
Maven Project
NLS
PHP
Plug-in Development
Sapphire
Scala Wizards
Empty File
Scala App


File Edit Refactor Source Navigate Search Project Scala Run Window Help
Package Explorer
spark-scala-maven-project
src/main/scala
com.examples
MainExample.scala
SparkWordCount.scala
com.examples.algo
LassoWithSGD.scala
LinearRegressionSGD.scala
LinearSupportVector.scala
LogisticRegressionWithLBFGS.scala
LogisticRegressionWithSGD.scala
RidgeRegressionWithSGD.scala
Scala Library container [2.11.6]
JRE System Library [JavaSE-1.6]
Maven Dependencies
myModelPath
src
target
pom.xml
README.md
LinearSuppor...
LogisticRegr...
LogisticRegr...
RidgeRegress...
LassoWithSGD...
spark-scala-...
</build>
<dependencies>
<!-- Scala and Spark dependencies -->
<dependency>
<groupId>org.scala-lang</groupId>
<artifactId>scala-library</artifactId>
<version>${scala.version}</version>
</dependency>
<dependency>
<groupId>org.apache.spark</groupId>
<artifactId>spark-core_2.10</artifactId>
<version>1.4.0</version>
<!--<version>1.2.0-cdh5.3.1</version>-->
</dependency>
<dependency>
<groupId>org.apache.spark</groupId>
<artifactId>spark-mllib_2.10</artifactId>
<version>1.4.0</version>
<!--<version>1.2.0-cdh5.3.1</version>-->
</dependency>
<dependency>
<groupId>org.apache.hadoop</groupId>
<artifactId>hadoop-client</artifactId>
<version>2.7.0</version>
</dependency>
</dependencies>
</project>

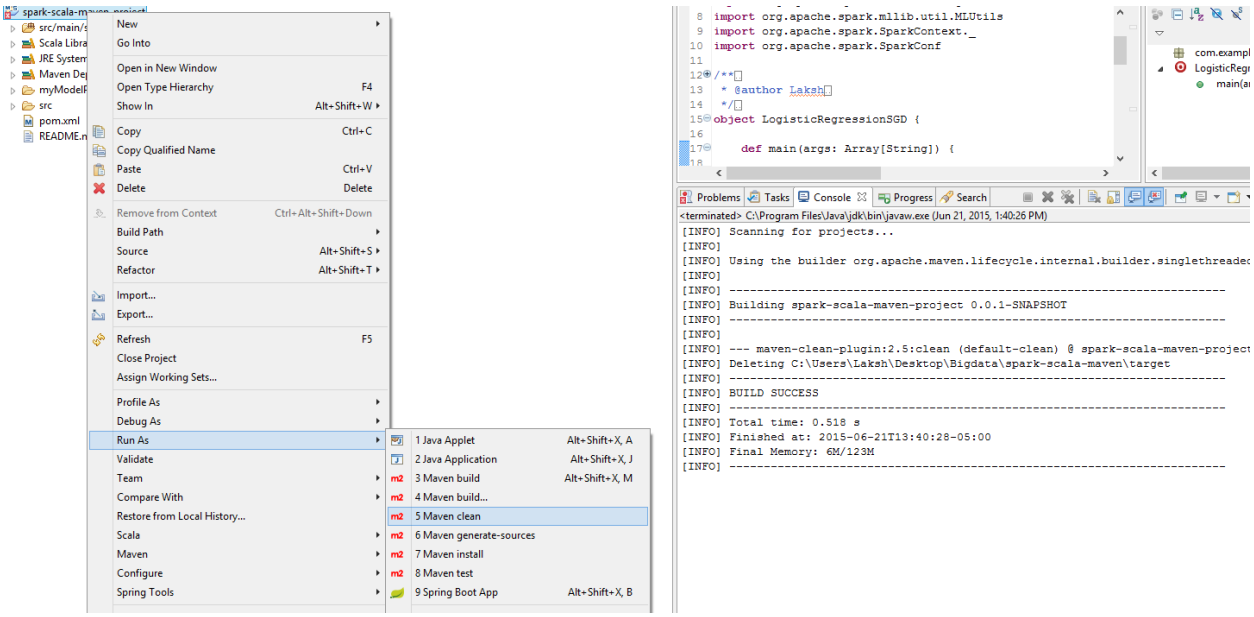
5. Install the below dependencies in the POM.xml file in the **spark-scala-maven-project**

```
      <project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
      <modelVersion>4.0.0</modelVersion>
      <groupId>spark-scala-maven-project</groupId>
      <artifactId>spark-scala-maven-project</artifactId>
      <version>0.0.1-SNAPSHOT</version>
      <name>${project.artifactId}</name>
      <description>This is a boilerplate maven project to start using Spark
in Scala</description>
      <inceptionYear>2010</inceptionYear>

      <properties>
            <maven.compiler.source>1.6</maven.compiler.source>
            <maven.compiler.target>1.6</maven.compiler.target>
            <encoding>UTF-8</encoding>
            <scala.tools.version>2.10</scala.tools.version>
            <!-- Put the Scala version of the cluster -->
            <scala.version>2.10.4</scala.version>
      </properties>

      <!-- repository to add org.apache.spark -->
      <repositories>
            <repository>
                  <id>cloudera-repo-releases</id>

      <url>https://repository.cloudera.com/artifactory/repo/</url>
            </repository>
      </repositories>

      <build>
            <sourceDirectory>src/main/scala</sourceDirectory>
            <testSourceDirectory>src/test/scala</testSourceDirectory>
            <plugins>
                  <plugin>
                        <!-- see http://davidb.github.com/scala-maven-plugin
-->
                        <groupId>net.alchim31.maven</groupId>
                        <artifactId>scala-maven-plugin</artifactId>
                        <version>3.1.3</version>
                        <executions>
                              <execution>
                                    <goals>
                                          <goal>compile</goal>
                                          <goal>testCompile</goal>
                                    </goals>
                                    <configuration>
                                          <args>
                                                <arg>-make:transitive</arg>
                                                <arg>-dependencyfile</arg>
```

```
		<arg>${project.build.directory}/.scala_dependencies</arg>
					</args>
				</configuration>
			</execution>
		</executions>
	</plugin>
	<plugin>
		<groupId>org.apache.maven.plugins</groupId>
		<artifactId>maven-surefire-plugin</artifactId>
		<version>2.13</version>
		<configuration>
			<useFile>false</useFile>
			<disableXmlReport>true</disableXmlReport>
			<!-- If you have classpath issue like
NoDefClassError,... -->
			<!--
useManifestOnlyJar>false</useManifestOnlyJar -->
			<includes>
				<include>**/*Test.*</include>
				<include>**/*Suite.*</include>
			</includes>
		</configuration>
	</plugin>

	<!-- "package" command plugin -->
	<plugin>
		<artifactId>maven-assembly-plugin</artifactId>
		<version>2.4.1</version>
		<configuration>
			<descriptorRefs>
				<descriptorRef>jar-with-
dependencies</descriptorRef>
			</descriptorRefs>
		</configuration>
		<executions>
			<execution>
				<id>make-assembly</id>
				<phase>package</phase>
				<goals>
					<goal>single</goal>
				</goals>
			</execution>
		</executions>
	</plugin>
	</plugins>
</build>

<dependencies>
	<!-- Scala and Spark dependencies -->
	<dependency>
		<groupId>org.scala-lang</groupId>
		<artifactId>scala-library</artifactId>
		<version>${scala.version}</version>
	</dependency>
	<dependency>
		<groupId>org.apache.spark</groupId>
```

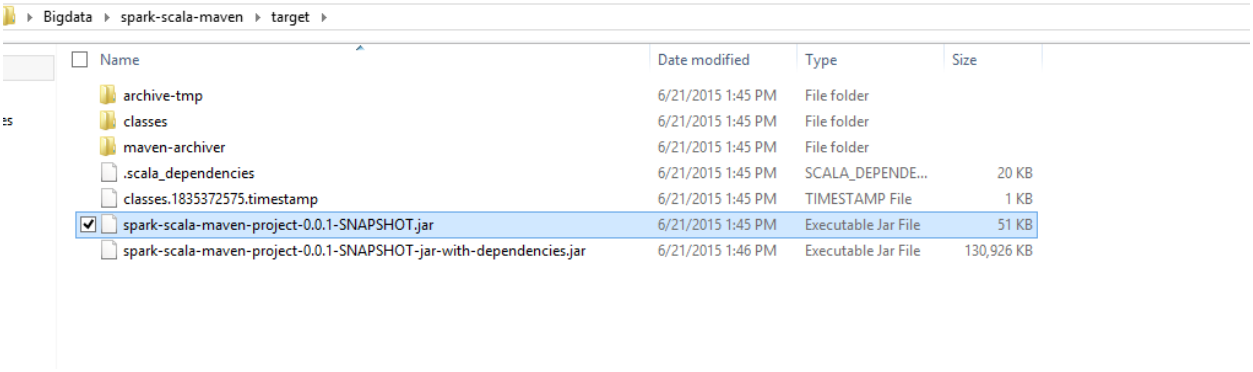
```
            <artifactId>spark-core_2.10</artifactId>
            <version>1.4.0</version>
            <!--<version>1.2.0-cdh5.3.1</version>-->
        </dependency>
        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-mllib_2.10</artifactId>
            <version>1.4.0</version>
            <!--<version>1.2.0-cdh5.3.1</version>-->
        </dependency>
        <dependency>
        <groupId>org.apache.hadoop</groupId>
        <artifactId>hadoop-client</artifactId>
        <version>2.7.0</version>
      </dependency>

    </dependencies>
</project>
```

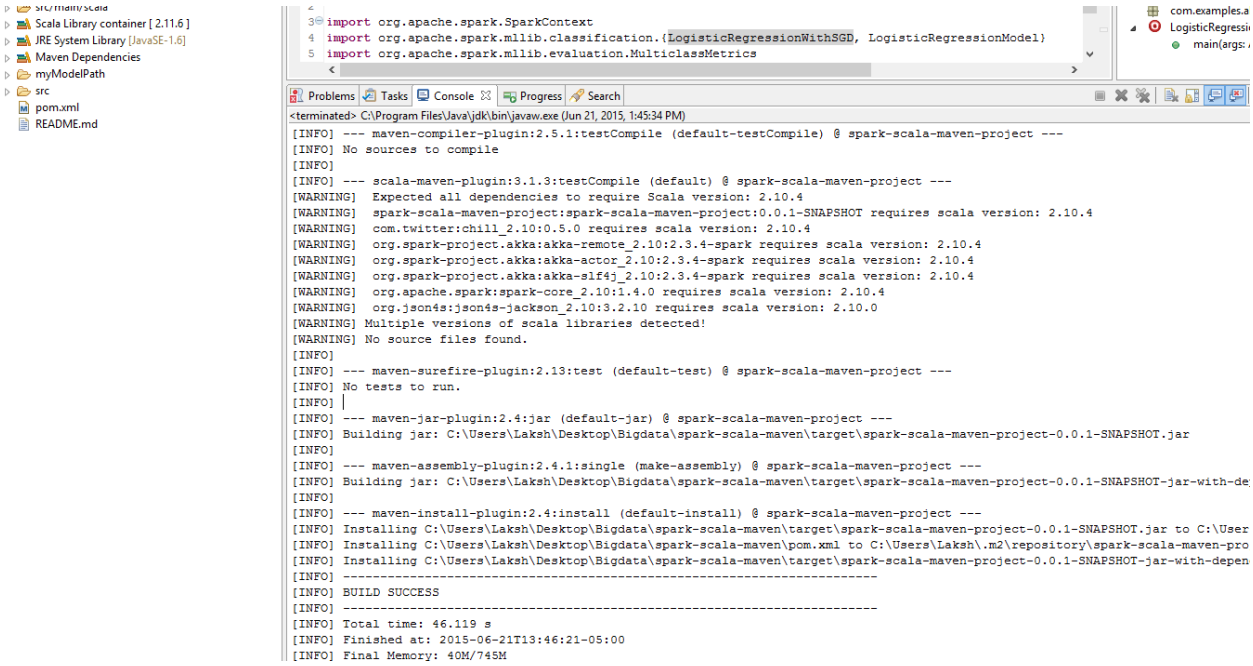
6. Right Click on the Project and Run ***Maven Clean***



7. Right click and Run Maven Install, It will create the jar in the target folder of the project location

:





Running SCALA Algorithms:

SPARK RUN TIME COMMAND

L2

C:/spark-1.4.0-bin-hadoop2.6/bin/spark-submit --class "com.examples.algo.LinearSupportVector" --master local[1] target/spark-scala-maven-project-0.0.1-SNAPSHOT.jar

AreaUnderCurve.scala:45, took 0.097492 s

Area under ROC = 0.6327268553062039

L1

C:/spark-1.4.0-bin-hadoop2.6/bin/spark-submit --class "com.examples.algo.LinearSupportVectorL1" --master local[1] target/spark-scala-maven-project-0.0.1-SNAPSHOT.jar

AreaUnderCurve.scala:45, took 0.120979 s

Area under ROC = 0.6346782988004352

L2

C:/spark-1.4.0-bin-hadoop2.6/bin/spark-submit --class "com.examples.algo.LogisticRegressionLBFGS" --master local[1] target/spark-scala-maven-project-0.0.1-SNAPSHOT.jar

countByValue at MulticlassMetrics.scala:44, took 0.318731 s

Precision = 0.9605809128630706

L1

C:/spark-1.4.0-bin-hadoop2.6/bin/spark-submit --class "com.examples.algo.LogisticRegressionLBFGSL1" --master local[1] target/spark-scala-maven-project-0.0.1-SNAPSHOT.jar

countByValue at MulticlassMetrics.scala:44, took 0.272651 s

Precision = 0.9605809128630706

L2

C:/spark-1.4.0-bin-hadoop2.6/bin/spark-submit --class "com.examples.algo.LogisticRegressionLBFGS" --master local[1] target/spark-scala-maven-project-0.0.1-SNAPSHOT.jar

countByValue at MulticlassMetrics.scala:44, took 0.288646 s

Precision = 0.9605809128630706

L1

C:/spark-1.4.0-bin-hadoop2.6/bin/spark-submit --class "com.examples.algo.LogisticRegressionLBFGSL1" --master local[1] target/spark-scala-maven-project-0.0.1-SNAPSHOT.jar

countByValue at MulticlassMetrics.scala:44, took 0.349361 s

Precision = 0.9605809128630706

L2

C:/spark-1.4.0-bin-hadoop2.6/bin/spark-submit --class "com.examples.algo.LogisticRegressionSGD" --master local[1] target/spark-scala-maven-project-0.0.1-SNAPSHOT.jar

countByValue at MulticlassMetrics.scala:44, took 0.351778 s

Precision = 0.9600622406639004

L1

C:/spark-1.4.0-bin-hadoop2.6/bin/spark-submit --class "com.examples.algo.LogisticRegressionSGDL1" --master local[1] target/spark-scala-maven-project-0.0.1-SNAPSHOT.jar

countByValue at MulticlassMetrics.scala:44, took 0.350728 s

Precision = 0.9600631403650002

NO RegType

C:/spark-1.4.0-bin-hadoop2.6/bin/spark-submit --class "com.examples.algo.LinearRegressionSGD" --master local[1] target/spark-scala-maven-project-0.0.1-SNAPSHOT.jar

LinearRegressionSGD.scala:38, took 0.104092 s

training Mean Squared Error = 6.751135104498063E51

L2

C:/spark-1.4.0-bin-hadoop2.6/bin/spark-submit --class "com.examples.algo.RidgeRegressionSGD" --master local[1] target/spark-scala-maven-project-0.0.1-SNAPSHOT.jar

mean at RidgeRegressionWithSGD.scala:39, took 0.090872 s

training Mean Squared Error = 4.8042799175778876E51

L1

C:/spark-1.4.0-bin-hadoop2.6/bin/spark-submit --class "com.examples.algo.LassoSGD" --master local[1] target/spark-scala-maven-project-0.0.1-SNAPSHOT.jar

mean at LassoWithSGD.scala:39, took 0.062116 s

training Mean Squared Error = 1.4362289499005352E52

General Matrics for Scala

	Algorithms	L1	L2
Classification	SVMWithSGD	Area under ROC = 0.6346782988004352	Area under ROC = 0.6327268553062039
	LogisticRegressionWithLBFGS	Precision = 0.9605809128630706	Precision = 0.9605809128630706
	LogisticRegressionWithSGD	Precision = 0.9600631403650002	Precision = 0.9600622406639004
Regression	LinearRegressionWithSGD	(NA) training Mean Squared Error = 6.75113510449806385	
	RidgeRegressionWithSGD	NA	training Mean Squared Error = 4.8042799175778876951
	LassoWithSGD	training Mean Squared Error = 1.4362289499005352952	NA

3. PySpark:

1. Create the ipython profile using pyspark with below command:
   ipython profile create pyspark
2. It will create the below directory :
   C:\Users\Laksh\.ipython\profile_pyspark
3. Edit the file **/.ipython/profile_pyspark/ipython_notebook_config.py and add following :**
   c = get_config()
   c.NotebookApp.ip = '*'
   c.NotebookApp.open_browser = False
   c.NotebookApp.port = 8880 # or whatever you want; be aware of conflicts with CDH

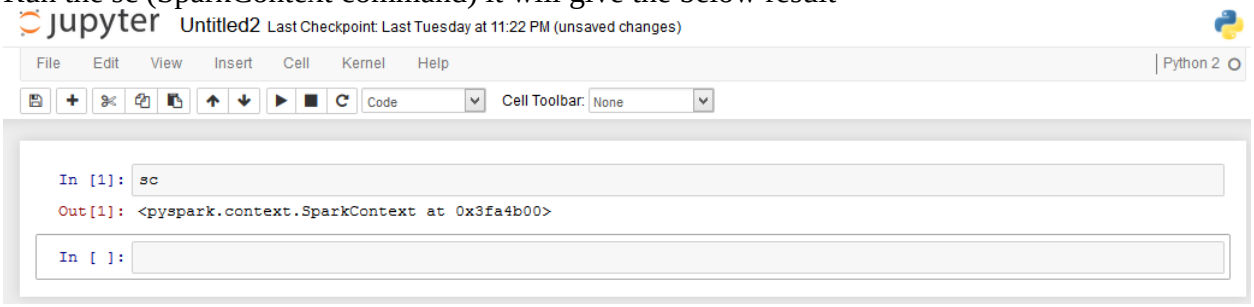
4. Create the file : C:\Users\Laksh\.ipython\profile_pyspark\startup\
   ipython_notebook_config.py and edit the file :
   import os
   import sys
   # Configure the environment

```
if 'SPARK_HOME' not in os.environ:
    os.environ['SPARK_HOME'] = 'C:/spark-1.4.0-bin-hadoop2.6'

# Create a variable for our root path
SPARK_HOME = os.environ['SPARK_HOME']

# Add the PySpark/py4j to the Python Path
sys.path.insert(0, os.path.join(SPARK_HOME, "python", "build"))
sys.path.insert(0, os.path.join(SPARK_HOME, "python"))
sys.path.insert(0, os.path.join(SPARK_HOME, 'python/lib/py4j-0.8.1-src.zip'))
execfile(os.path.join(SPARK_HOME, 'python/pyspark/shell.py'))
```

5. Now the ipython notebook is ready and got command promt and open the notebook with below commands:
   ipython notebook --profile=pyspark

6. Go to browser : **http://localhost:8880/tree**
   Click new and ipython notebook , it will open the another tab with ipython notebook.

7. Run the sc (SparkContext command) it will give the below result



8. Run the code.

Running Algorithms in Pyspark:

Running L1 with **SVMWithSGD:**
Various Parameter : iterations=100, regType='l1' ,step=.001

In [4]:
```
from pyspark.mllib.classification import SVMWithSGD, SVMModel
from pyspark.mllib.regression import LabeledPoint

# Load and parse the data
def parsePoint(line):
    values = [float(x) for x in line.split(',')]
    return LabeledPoint(values[11], values[0:10])

data = sc.textFile("C:/spark-1.4.0-bin-hadoop2.6/data/mllib/winequality-white.csv")
parsedData = data.map(parsePoint)

# Build the model with l1
model = SVMWithSGD.train(parsedData, iterations=100, regType='l1' ,step=.001 )

# Build the model with l2
# model = SVMWithSGD.train(parsedData, iterations=100, regType='l2' ,step=.001 )

# Evaluating the model on training data
labelsAndPreds = parsedData.map(lambda p: (p.label, model.predict(p.features)))
trainErr = labelsAndPreds.filter(lambda (v, p): v != p).count() / float(parsedData.count())
print("Training Error = " + str(trainErr))
```

```
Training Error = 0.0367496937526
```

Running L2 with **SVMWithSGD:**

Various Parameter: iterations=100, regType='l2' ,step=.001

In [1]:
```
from pyspark.mllib.classification import SVMWithSGD, SVMModel
from pyspark.mllib.regression import LabeledPoint

# Load and parse the data
def parsePoint(line):
    values = [float(x) for x in line.split(',')]
    return LabeledPoint(values[11], values[0:10])

data = sc.textFile("C:/spark-1.4.0-bin-hadoop2.6/data/mllib/winequality-white.csv")
parsedData = data.map(parsePoint)

# Build the model with l1
# model = SVMWithSGD.train(parsedData, iterations=100, regType='l1' ,step=.001 )

# Build the model with l2
model = SVMWithSGD.train(parsedData, iterations=100, regType='l2' ,step=.001 )

# Evaluating the model on training data
labelsAndPreds = parsedData.map(lambda p: (p.label, model.predict(p.features)))
trainErr = labelsAndPreds.filter(lambda (v, p): v != p).count() / float(parsedData.count())
print("Training Error = " + str(trainErr))
```

```
Training Error = 0.0367496937526
```

Running L1 with LogisticRegressionWithLBFGS:

Various Parameter : iterations=100, regType='l1'

```
In [6]: from pyspark.mllib.classification import LogisticRegressionWithLBFGS
        from pyspark.mllib.regression import LabeledPoint
        from numpy import array

        # Load and parse the data
        def parsePoint(line):
            values = [float(x) for x in line.split(',')]
            return LabeledPoint(values[11], values[1:10])

        data = sc.textFile("C:/spark-1.4.0-bin-hadoop2.6/data/mllib/winequality-white.csv")
        parsedData = data.map(parsePoint)

        # Build the model wit l1
         model = LogisticRegressionWithLBFGS.train(parsedData, iterations=100, regType='l1')

        # Build the model wit l2
        # model = LogisticRegressionWithLBFGS.train(parsedData, iterations=100, regType='l2')

        # Evaluating the model on training data
        labelsAndPreds = parsedData.map(lambda p: (p.label, model.predict(p.features)))
        trainErr = labelsAndPreds.filter(lambda (v, p): v != p).count() / float(parsedData.count())
        print("Training Error = " + str(trainErr))

        Training Error = 0.0367496937526
```

Running L1 with LogisticRegressionWithLBFGS:
Various Parameter : iterations=100, regType='l2'

```
In [6]: from pyspark.mllib.classification import LogisticRegressionWithLBFGS
        from pyspark.mllib.regression import LabeledPoint
        from numpy import array

        # Load and parse the data
        def parsePoint(line):
            values = [float(x) for x in line.split(',')]
            return LabeledPoint(values[11], values[1:10])

        data = sc.textFile("C:/spark-1.4.0-bin-hadoop2.6/data/mllib/winequality-white.csv")
        parsedData = data.map(parsePoint)

        # Build the model wit l1
        # model = LogisticRegressionWithLBFGS.train(parsedData, iterations=100, regType='l1')

        # Build the model wit l2
         model = LogisticRegressionWithLBFGS.train(parsedData, iterations=100, regType='l2')

        # Evaluating the model on training data
        labelsAndPreds = parsedData.map(lambda p: (p.label, model.predict(p.features)))
        trainErr = labelsAndPreds.filter(lambda (v, p): v != p).count() / float(parsedData.count())
        print("Training Error = " + str(trainErr))

        Training Error = 0.0367496937526
```

Running L1 with LogisticRegressionWithSGD
Various Parameter : iterations=100, regType='l1'

```
In [4]: from pyspark.mllib.classification import LogisticRegressionWithSGD, SVMModel
        from pyspark.mllib.regression import LabeledPoint

        # Load and parse the data
        def parsePoint(line):
            values = [float(x) for x in line.split(',')]
            return LabeledPoint(values[11], values[0:10])

        data = sc.textFile("C:/spark-1.4.0-bin-hadoop2.6/data/mllib/winequality-white.csv")
        parsedData = data.map(parsePoint)

        # Build the model with l1
         model = LogisticRegressionWithSGD.train(parsedData, iterations=100, regType='l1')

        # Build the model with l2
        # model = LogisticRegressionWithSGD.train(parsedData, iterations=1000, regType='l2')

        # Evaluating the model on training data
        labelsAndPreds = parsedData.map(lambda p: (p.label, model.predict(p.features)))
        trainErr = labelsAndPreds.filter(lambda (v, p): v != p).count() / float(parsedData.count())
        print("Training Error = " + str(trainErr))
```

Running L2 with LogisticRegressionWithSGD
Various Parameter: iterations=100, regType='l2'

```
In [4]: from pyspark.mllib.classification import LogisticRegressionWithSGD, SVMModel
        from pyspark.mllib.regression import LabeledPoint

        # Load and parse the data
        def parsePoint(line):
            values = [float(x) for x in line.split(',')]
            return LabeledPoint(values[11], values[0:10])

        data = sc.textFile("C:/spark-1.4.0-bin-hadoop2.6/data/mllib/winequality-white.csv")
        parsedData = data.map(parsePoint)

        # Build the model with l1
        # model = LogisticRegressionWithSGD.train(parsedData, iterations=100, regType='l1')

        # Build the model with l2
        model = LogisticRegressionWithSGD.train(parsedData, iterations=1000, regType='l2')

        # Evaluating the model on training data
        labelsAndPreds = parsedData.map(lambda p: (p.label, model.predict(p.features)))
        trainErr = labelsAndPreds.filter(lambda (v, p): v != p).count() / float(parsedData.count())
        print("Training Error = " + str(trainErr))
```

Running LinearRegressionWithSGD:

Various Parameter: iterations=100, stepSize='.001'

```
In [7]: from pyspark.mllib.regression import LabeledPoint, LinearRegressionWithSGD
        from numpy import array

        # Load and parse the data
        def parsePoint(line):
            values = [float(x) for x in line.split(',')]
            return LabeledPoint(values[11], values[0:10])

        data = sc.textFile("C:/spark-1.4.0-bin-hadoop2.6/data/mllib/winequality-white-WB.csv")
        parsedData = data.map(parsePoint)

        # Build the model (no reg type)
        model = LinearRegressionWithSGD.train(parsedData,.001)

        # Evaluate the model on Mean data
        valuesAndPreds = parsedData.map(lambda p: (p.label, model.predict(p.features)))
        MSE = valuesAndPreds.map(lambda (v, p): (v - p)**2).reduce(lambda x, y: x + y) / valuesAndPreds.count()
        print("Mean Squared Error = " + str(MSE))

        Mean Squared Error = 35.3340138832
```

Running RidgeRegressionWithSGD:
Various Parameter: iterations=100, stepSize='.001'

In [8]:
```
from pyspark.mllib.regression import LabeledPoint, RidgeRegressionWithSGD
from numpy import array

# Load and parse the data
def parsePoint(line):
    values = [float(x) for x in line.split(',')]
    return LabeledPoint(values[11], values[0:10])

data = sc.textFile("C:/spark-1.4.0-bin-hadoop2.6/data/mllib/winequality-white-WB.csv")
parsedData = data.map(parsePoint)

# Build the model (with l2)
model = RidgeRegressionWithSGD.train(parsedData,.001)

# Evaluate the model on Mean data
valuesAndPreds = parsedData.map(lambda p: (p.label, model.predict(p.features)))
MSE = valuesAndPreds.map(lambda (v, p): (v - p)**2).reduce(lambda x, y: x + y) / valuesAndPreds.count()
print("Mean Squared Error = " + str(MSE))
```

```
Mean Squared Error = 35.3340138832
```

Running LassoWithSGD
Various Parameter: iterations=100, stepSize='.001'

In [9]:
```
from pyspark.mllib.regression import LabeledPoint, LassoWithSGD
from numpy import array

# Load and parse the data
def parsePoint(line):
    values = [float(x) for x in line.split(',')]
    return LabeledPoint(values[11], values[0:10])

data = sc.textFile("C:/spark-1.4.0-bin-hadoop2.6/data/mllib/winequality-white-WB.csv")
parsedData = data.map(parsePoint)

# Build the model (l1 reg type)
model = LassoWithSGD.train(parsedData,.001)

# Evaluate the model on training data
valuesAndPreds = parsedData.map(lambda p: (p.label, model.predict(p.features)))
MSE = valuesAndPreds.map(lambda (v, p): (v - p)**2).reduce(lambda x, y: x + y) / valuesAndPreds.count()
print("Mean Squared Error = " + str(MSE))
```

```
Mean Squared Error = 35.3340138832
```

General Matrics for PySpark

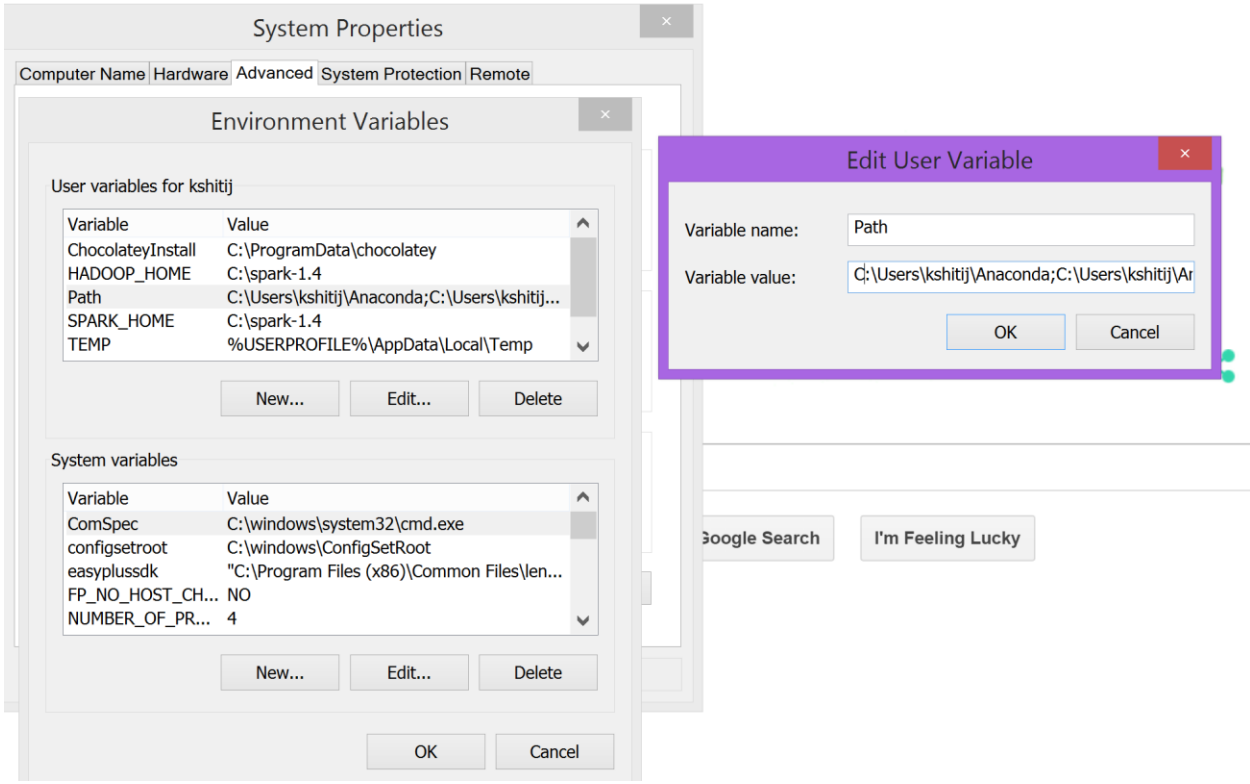
	Algorithms	L1	L2
Classification	SVMWithSGD	Training Error = 0.0367496937526	Training Error = 0.0367496937526
	LogisticRegressionWithLBFGS	Training Error = 0.0367496937526	Training Error = 0.0367496937526

	LogisticRegressionWithSGD	Training Error = 0.0367496937526	Training Error = 0.0367496937526
Regression	LinearRegressionWithSGD	(NA) Mean Squared Error = 35.3340138832	
	RidgeRegressionWithSGD	NA	Mean Squared Error = 35.3340138832
	LassoWithSGD	Mean Squared Error = 35.3340138832	NA

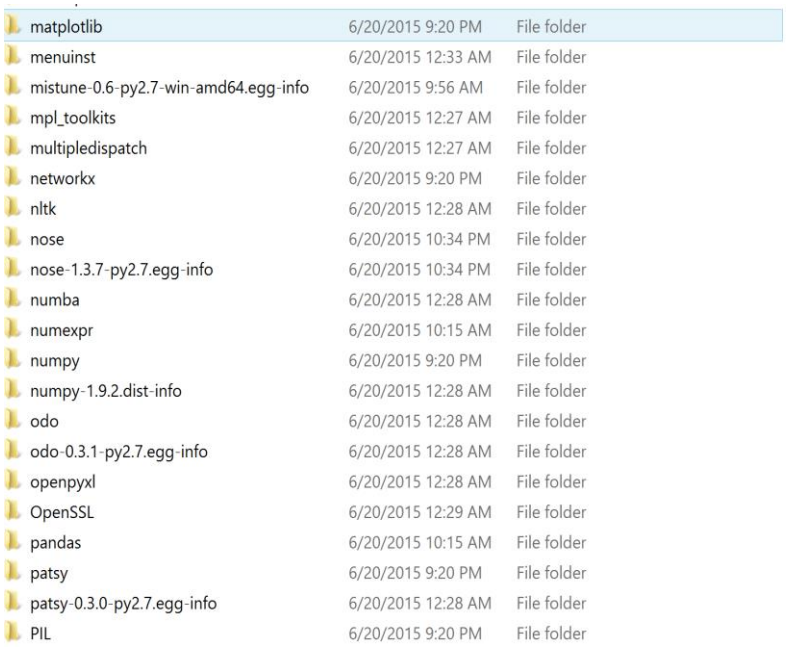
4. In Plain Python

1. Setting up anaconda and python

a) Installed anaconda for windows 8.1 using http://continuum.io/downloads for 64-Bit
b) Installed version for anaconda is 2.2.0 and for python should be 2.7.9. Higher version can also be installed as per the requirements.
c) set Anaconda in the environment path variable

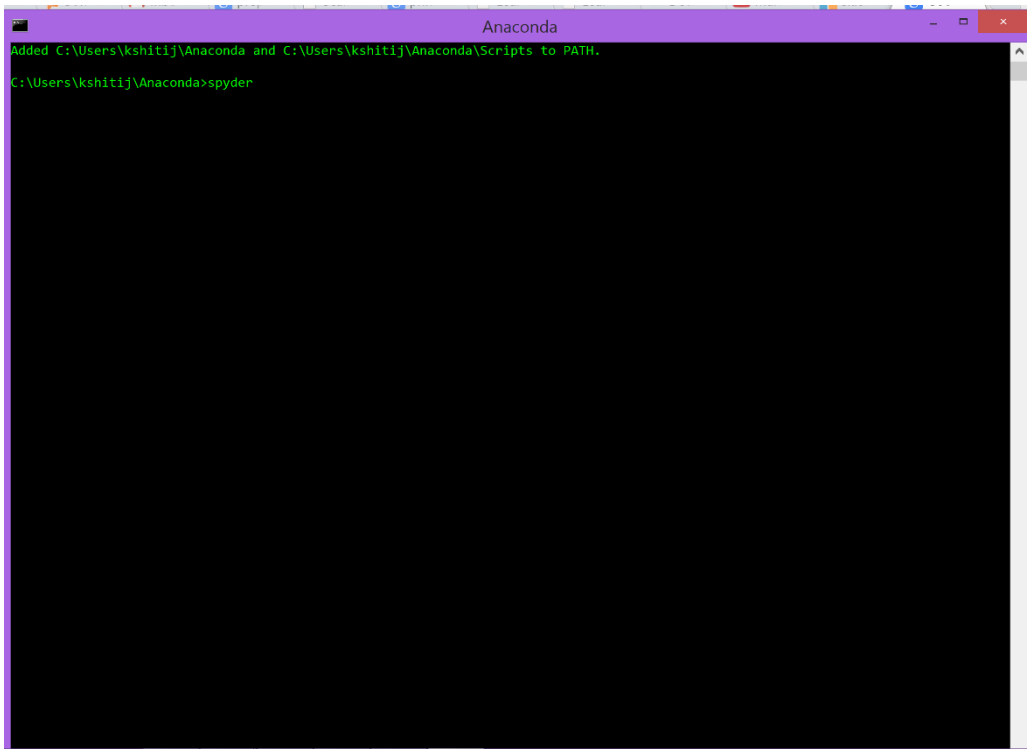


d) Updated numpy and scipy libraries using commands on anaconda command prompt
    a. conda update scipy and conda update numpy

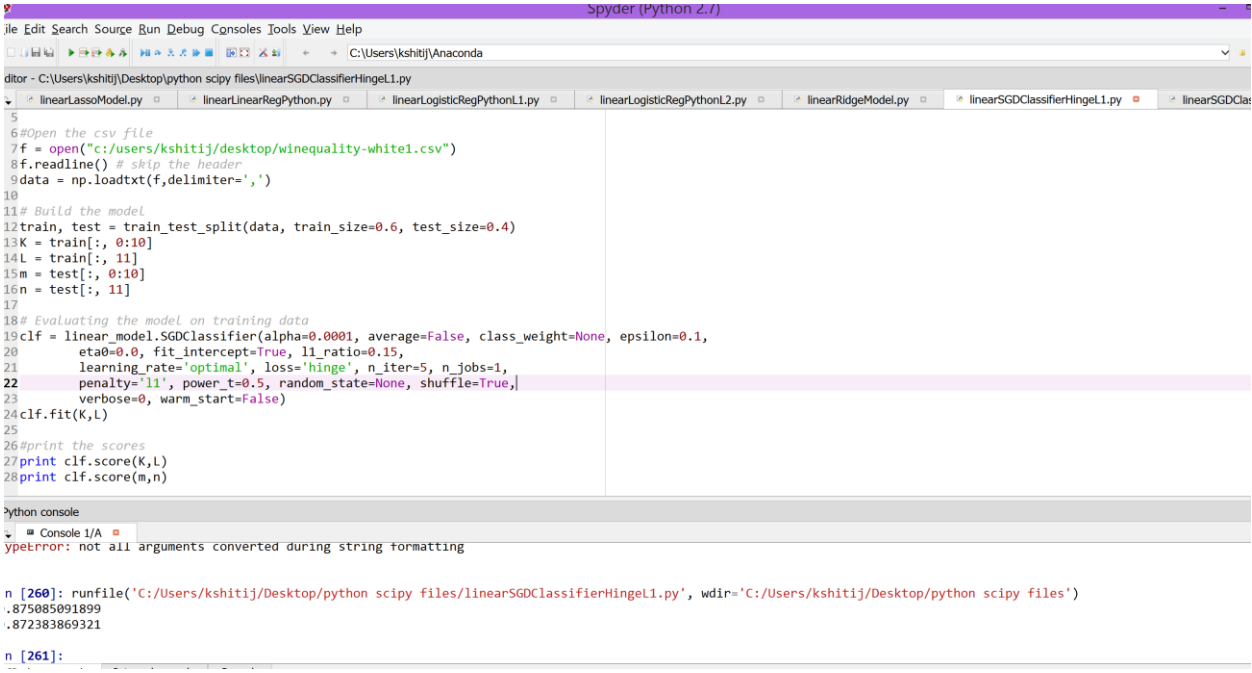


matplotlib	6/20/2015 9:20 PM	File folder
menuinst	6/20/2015 12:33 AM	File folder
mistune-0.6-py2.7-win-amd64.egg-info	6/20/2015 9:56 AM	File folder
mpl_toolkits	6/20/2015 12:27 AM	File folder
multipledispatch	6/20/2015 12:27 AM	File folder
networkx	6/20/2015 9:20 PM	File folder
nltk	6/20/2015 12:28 AM	File folder
nose	6/20/2015 10:34 PM	File folder
nose-1.3.7-py2.7.egg-info	6/20/2015 10:34 PM	File folder
numba	6/20/2015 12:28 AM	File folder
numexpr	6/20/2015 10:15 AM	File folder
numpy	6/20/2015 9:20 PM	File folder
numpy-1.9.2.dist-info	6/20/2015 12:28 AM	File folder
odo	6/20/2015 12:28 AM	File folder
odo-0.3.1-py2.7.egg-info	6/20/2015 12:28 AM	File folder
openpyxl	6/20/2015 12:28 AM	File folder
OpenSSL	6/20/2015 12:29 AM	File folder
pandas	6/20/2015 10:15 AM	File folder
patsy	6/20/2015 9:20 PM	File folder
patsy-0.3.0-py2.7.egg-info	6/20/2015 12:28 AM	File folder
PIL	6/20/2015 9:20 PM	File folder

e) Run Python IDE SPYDER using anaconda command prompt



f) SPYDER IDE



2. Writing algorithms
    a. Classification Models
        i. Linear SGD Classifier – Hinge Loss - Used L1 regularizes and Hinge Loss function

```
import csv
import numpy as np
from sklearn import linear_model
from sklearn.cross_validation import train_test_split

#Open the csv file
f = open("c:/users/kshitij/desktop/winequality-white1.csv")
f.readline() # skip the header
data = np.loadtxt(f,delimiter=',')

# Build the model
train, test = train_test_split(data, train_size=0.6, test_size=0.4)
K = train[:, 0:10]
L = train[:, 11]
m = test[:, 0:10]
n = test[:, 11]

# Evaluating the model on training data  - using regularization level as L1 , loss function used hinge
clf = linear_model.SGDClassifier(alpha=0.0001, average=False, epsilon=0.1,
        loss='hinge', n_iter=5,penalty='l1', power_t=0.5, shuffle=True,
        verbose=0)
clf.fit(K,L)

#print the scores
print clf.score(K,L)
print clf.score(m,n)
```

ii) Linear SGD Classifier – Hinge Loss - Used L1 regularizes and Hinge Loss function

```
import numpy as np
from sklearn import linear_model
from sklearn.cross_validation import train_test_split

#Open the csv file
f = open("c:/users/kshitij/desktop/winequality-white1.csv")
f.readline() # skip the header
data = np.loadtxt(f,delimiter=',')

# Build the model
train, test = train_test_split(data, train_size=0.6, test_size=0.4)
K = train[:, 0:10]
L = train[:, 11]
m = test[:, 0:10]
n = test[:, 11]

# Evaluating the model on training data  - using regularization level as L2 and hinge loss
clf = linear_model.SGDClassifier(alpha=0.0001, average=False, epsilon=0.1,
        loss='hinge', n_iter=5,penalty='l2', power_t=0.5, shuffle=True,
        verbose=0)
clf.fit(K,L)

#print the scores
print clf.score(K,L)
print clf.score(m,n)
```

iii) Linear SGD Classifier – Log Loss - Used L1 regularizes and Log Loss function

```
import numpy as np
from sklearn import linear_model
from sklearn.cross_validation import train_test_split

#Open the csv file
f = open("c:/users/kshitij/desktop/winequality-white1.csv")
f.readline() # skip the header
data = np.loadtxt(f,delimiter=',')

# Build the model
train, test = train_test_split(data, train_size=0.6, test_size=0.4)
K = train[:, 0:10]
L = train[:, 11]
m = test[:, 0:10]
n = test[:, 11]

# Evaluating the model on training data  - using regularization level as L1 , loss function used log
clf = linear_model.SGDClassifier(alpha=0.0001, average=False, epsilon=0.1,
         loss='log', n_iter=5,penalty='l1', power_t=0.5, shuffle=True,
         verbose=0)
clf.fit(K,L)

#print the scores
print clf.score(K,L)
print clf.score(m,n)
```

iv) Linear SGD Classifier – Log Loss - Used L2 regularizes and Log Loss function

```
import numpy as np
from sklearn import linear_model
from sklearn.cross_validation import train_test_split

#Open the csv file
f = open("c:/users/kshitij/desktop/winequality-white1.csv")
f.readline() # skip the header
data = np.loadtxt(f,delimiter=',')

# Build the model
train, test = train_test_split(data, train_size=0.6, test_size=0.4)
K = train[:, 0:10]
L = train[:, 11]
m = test[:, 0:10]
n = test[:, 11]

# Evaluating the model on training data  - using regularization level as L2 , loss function used log
clf = linear_model.SGDClassifier(alpha=0.0001, average=False, epsilon=0.1,
         loss='log', n_iter=5,penalty='l2', power_t=0.5, shuffle=True,
         verbose=0)
clf.fit(K,L)

#print the scores
print clf.score(K,L)
print clf.score(m,n)
```

v) Logistic Regression Classification – L1 regularization

```
import numpy as np
from sklearn import linear_model
from sklearn.cross_validation import train_test_split
# reading the file
f = open("c:/users/kshitij/desktop/winequality-white1.csv")
f.readline() # skip the header
data = np.loadtxt(f,delimiter=',')

# splitting up the data set into training and test
train, test = train_test_split(data, train_size=0.6, test_size=0.4)
K = train[:, 0:10]
L = train[:, 11]
m = test[:, 0:10]
n = test[:, 11]

# setting up the model arguments
clf = linear_model.LogisticRegression(penalty='l1', fit_intercept=True,
                                      random_state=None, max_iter=100)
clf.fit(K,L)
#printing the values
print clf.score(K,L)
# Evaluating the model on training data
print clf.score(m,n)
```

vi) Logistic Regression Classification – L2 regularization

```
import numpy as np
from sklearn import linear_model
from sklearn.cross_validation import train_test_split
# reading the file
f = open("c:/users/kshitij/desktop/winequality-white1.csv")
f.readline() # skip the header
data = np.loadtxt(f,delimiter=',')

# splitting up the data set into training and test
train, test = train_test_split(data, train_size=0.6, test_size=0.4)
K = train[:, 0:10]
L = train[:, 11]
m = test[:, 0:10]
n = test[:, 11]

# setting up the model arguments
clf = linear_model.LogisticRegression(penalty='l2', fit_intercept=True,
                                      random_state=None, max_iter=100)
clf.fit(K,L)
#printing the values
print clf.score(K,L)
# Evaluating the model on training data
print clf.score(m,n)
```

Regression Models

vii) Linear Regression Model

```
import numpy as np
from sklearn import linear_model
from sklearn.cross_validation import train_test_split

f = open("c:/users/kshitij/desktop/winequality-white1.csv")
f.readline() # skip the header
data = np.loadtxt(f,delimiter=',')

#splitting the data into training and test
train, test = train_test_split(data, train_size=0.6, test_size=0.4)
K = train[:, 0:10]
L = train[:, 11]
m = test[:, 0:10]
n = test[:, 11]

# Build the model
clf = linear_model.LinearRegression(fit_intercept=True, normalize=False, copy_X=True, n_jobs=1)
# Evaluating the model on test data
clf.fit(K,L)
#printing the results
print clf.score(K,L)
print clf.score(m,n)
```

vii) SGD Regressor Model – L1 Regularization

```
from sklearn import linear_model
from sklearn.cross_validation import train_test_split
from sklearn.preprocessing import StandardScaler

#fetching thd data file
f = open("c:/users/kshitij/desktop/winequality-white1.csv")
f.readline() # skip the header
data = np.loadtxt(f,delimiter=',')

#setting the test and training parameters
train, test = train_test_split(data, train_size=0.6, test_size=0.4)
X  = train[:,0:10]
#normalising the data set
scaler = StandardScaler().fit(X)
K = scaler.transform(X)
L = train[:, 11]
m = test[:, 0:10]
m = scaler.transform(m)
n = test[:, 11]

# Build the model
clf = linear_model.SGDRegressor(loss='squared_loss', penalty='l1', alpha=0.00001,
                    fit_intercept=True, n_iter=1000, shuffle=True)
# Evaluating the model on training data
clf.fit(K,L)
print clf.score(K,L)
print clf.score(m,n)
```

viii) SGD Regressor Model – L2 Regularization

```
from sklearn import linear_model
from sklearn.cross_validation import train_test_split
from sklearn.preprocessing import StandardScaler

#fetching thd data file
f = open("c:/users/kshitij/desktop/winequality-white1.csv")
f.readline() # skip the header
data = np.loadtxt(f,delimiter=',')

#setting the test and training parameters
train, test = train_test_split(data, train_size=0.6, test_size=0.4)
X  = train[:,0:10]
#normalising the data set
scaler = StandardScaler().fit(X)
K = scaler.transform(X)
L = train[:, 11]
m = test[:, 0:10]
m = scaler.transform(m)
n = test[:, 11]

# Build the model
clf = linear_model.SGDRegressor(loss='squared_loss', penalty='l2', alpha=0.00001,
                    fit_intercept=True, n_iter=1000, shuffle=True)
# Evaluating the model on training data
clf.fit(K,L)
print clf.score(K,L)
print clf.score(m,n)
```

ix) Ridge Model

```
import numpy as np
from sklearn import linear_model
from sklearn.cross_validation import train_test_split
from sklearn.preprocessing import StandardScaler

#open the csv file
f = open("c:/users/kshitij/desktop/winequality-white1.csv")
f.readline() # skip the header
data = np.loadtxt(f,delimiter=',')

# splitting the file into test and trainig data and normalising the data
train, test = train_test_split(data, train_size=0.6, test_size=0.4)
X  = train[:,0:10]
scaler = StandardScaler().fit(X)
K = scaler.transform(X)
L = train[:, 11]
m = test[:, 0:10]
m = scaler.transform(m)
n = test[:, 11]
# Build the model parameters
clf = linear_model.Ridge(alpha=1.0, fit_intercept=True, normalize=False,
                         copy_X=True, max_iter=100)
# Evaluating the model on training data
clf.fit(K,L)

print clf.score(K,L)
print clf.score(m,n)
```

x) Lasso Model

```
import numpy as np
from sklearn import linear_model
from sklearn.cross_validation import train_test_split
from sklearn.preprocessing import StandardScaler

    #open the csv file
f = open("c:/users/kshitij/desktop/winequality-white1.csv")
f.readline() # skip the header
data = np.loadtxt(f,delimiter=',')

train, test = train_test_split(data, train_size=0.6, test_size=0.4)
X  = train[:,0:10]
scaler = StandardScaler().fit(X)
K = scaler.transform(X)
L = train[:, 11]
m = test[:, 0:10]
m = scaler.transform(m)
n = test[:, 11]

# Build the model
clf = linear_model.Lasso(alpha=.010, fit_intercept=True,
                         max_iter=1000)
clf.fit(K,L)
print clf.score(K,L)
# Evaluating the model on training data
print clf.score(m,n)
```

1. What performance metrics did you implement and use to evaluate Classification algorithms?

Parameters

Default : stepSize: 1.0, numIterations: 100, regParm: 0.01, miniBatchFraction: 1.0
When L1 regularization is not required, L-BFGS version is strongly recommended since it converges faster and more accurately compared to SGD.
Change in the number of iteration doesn't impact the result much as the data is very small. However decrease in the step-size increase the accuracy of the result.

NOTE: if stepSize = 1 ; All Algorithms will give result NAN

Algorithms Type	Parameters	Pyspark	Scala
SVMWithSGD	stepSize: .001, numIterations: 1000, regParm: 0.01,	Training Error = 0.0367496937526	Area under ROC = 0.6346782988004352

	miniBatchFraction: 1.0 regType = “L1”		
	stepSize: .001, numIterations: 100, regParm: 0.01, miniBatchFraction: 1.0 regType = “L1”	Training Error = 0.0367496937526	Area under ROC = 0.5946580918004352
	stepSize: .001, numIterations: 1000, regParm: 0.01, miniBatchFraction: 1.0 regType = “L2”	Training Error = 0.0367496937526	Area under ROC = 0.6327268553062039
	stepSize: .01, numIterations: 100, regParm: 0.01, miniBatchFraction: 1.0 regType = “L2”	Training Error = 0.0367496937526	Area under ROC = 0.5946580918004352
LogisticRegressionWithLBFGS	stepSize: .001, numIterations: 1000, regParm: 0.01, miniBatchFraction: 1.0 regType = “L1”	Training Error = 0.0367496937526	Precision = 0.9605809128630706
	stepSize: .01, numIterations: 100, regParm: 0.01, miniBatchFraction: 1.0 regType = “L1”	Training Error = 0.0367496937526	Precision = 0.9605809128630706
	stepSize: .001, numIterations: 1000, regParm: 0.01, miniBatchFraction: 1.0 regType = “L2”	Training Error = 0.0367496937526	Precision = 0.9605809128630706
	stepSize: .01, numIterations: 100,	Training Error = 0.0367496937526	Precision = 0.9605809128630706

	regParm: 0.01, miniBatchFraction: 1.0 regType = “L2”		
LogisticRegressionWithSGD	stepSize: .001, numIterations: 1000, regParm: 0.01, miniBatchFraction: 1.0 regType = “L1”	Training Error = 0.0367496937526	Precision = 0.9600631403650002
	stepSize: .01, numIterations: 100, regParm: 0.01, miniBatchFraction: 1.0 regType = “L1”	Training Error = 0.0367496937526	Precision = 0.9600631403650002
	stepSize: .001, numIterations: 1000, regParm: 0.01, miniBatchFraction: 1.0 regType = “L2”	Training Error = 0.0367496937526	Precision = 0.9600622406639004
	stepSize: .01, numIterations: 100, regParm: 0.01, miniBatchFraction: 1.0 regType = “L2”	Training Error = 0.0367496937526	Precision = 0.9600622406639004

Performance metrics (In Python)– Performance metrics – We tried to change various factors such as alpha, number of iterations and saw a significant change in the score value on changing alpha values

a) Linear SGD Classifier – Hinge Loss - Used L1 regularizes and Hinge Loss function

alpha	score
1	.9592343456
.5	.9675455422

.10	.9689773636

Confusion metrics –

[[1872 26]

[61 0]]

b) Linear SGD Classifier – Hinge Loss - Used L2 regularizes and Hinge Loss function

alpha	score
1	.9484565433
.5	.9657678655
.10	.9673478644

confusion metrics below

[[1867 13]

[76 3]]

c) Linear SGD Classifier – Hinge Loss - Used L1 regularizes and Log Loss function

alpha	score
1	.9557673899
.5	.9584738902
.10	.9679877653

d) Linear SGD Classifier – Hinge Loss - Used L2 regularizes and Log Loss function

alpha	Score
1	.9597867555
.5	.9639876567
.10	.9689878877

e) Logistic Regression Classification – L1 regularization

Tried different number of iteration, but does not seems to put any effect any effect in the score. Alpha was not a parameter for this algorithm.

Score = .9578656455

f) Logistic Regression Classification – L2 regularization

Tried different number of iteration, but does not seems to put any effect any effect in the score. Alpha was not a parameter for this algorithm.

Score = .969876765

2. What performance metrics did you implement and use to evaluate Regression algorithms?

Default : stepSize: 1.0, numIterations: 100, regParm: 0.01, miniBatchFraction: 1.0
When L1 regularization is not required, L-BFGS version is strongly recommended since it converges faster and more accurately compared to SGD.
Change in the number of iteration doesn't impact the result much as the data is very small. However decrease in the step-size increase the accuracy of the result.

NOTE: if stepSize = 1 ; All Algorithms will give result NAN

Algorithms Type	Various Parameter	PySpark	Scala
LinearRegressionWithSGD	stepSize: .01, numIterations: 100	Mean Squared Error = 35.3340138832	training Mean Squared Error = 6.751135104498063E5

	stepSize: .001, numIterations: 1000	Mean Squared Error = 35.3340138832	training Mean Squared Error = 6.751135104498063E5
RidgeRegressionWithSGD	stepSize: .001, numIterations: 100 regType =”L2”	Mean Squared Error = 35.3340138832	training Mean Squared Error = 4.8042799175778876351
	stepSize: .001, numIterations: 1000 regType =”L2”	Mean Squared Error = 35.3340138832	training Mean Squared Error = 4.8042799175778876351
LassoWithSGD	stepSize: .01, numIterations: 100 regType =”L1”	Mean Squared Error = 35.3340138832	training Mean Squared Error = 1.4362289499005352052
	stepSize: .001, numIterations: 1000 regType =”L1”	Mean Squared Error = 35.3340138832	training Mean Squared Error = 1.4362289499005352052

In Plain Python

a) Linear Regression
Tried normalizing the data to get more accurate value but does not put any effect on the accuracy.

Score = .03898675437

b) SGD Regression L1 Regularization – value drastically changes on changing the alpha value

alpha	Score
.5	-.002187874
.1	-.001023323
.0001	.0452376345

c) SGD Regression L2 Regularization – value drastically changes on changing the alpha value

alpha	Score
.5	-1.017956763
.1	-.0017467834
.0001	.0368796544

d) Lasso Model

alpha	Score
1	-.0011236545
.5	-.0005456722
.0001	.0340986787

e) Ridge Model

alpha	Score
1	.0427865775
.5	.0454434676
.0001	.0460987888

Answer 4

No doubt, Scipy has rich libraries than Apache Spark but lacks a lot in reveling a lot about the data, for example when we use the libraries for scipy we can found accuracy using score function which tells about how accurate the algorithm is working on predicting the data using the training data calculations.

With Apache Spark we have different calculation parameters which predicts the accuracy of the system such as Area under ROC, training errors etc.

On performance factors - Apache spark libraries are better for a lot of processing so when u have to dig the data deep in and figure out lot of computations Apache spark is better and comparing with scipy libraries they are not very useful for high processing work.

So, would prefer to use Apache Spark

Easy Use – Spark is built on scala so it is much easier to use its libraries and easy to understand as compared to scipy.

Classification

Classification	Scipy	Apache Spark
SGD ClassifierSVM	0.961715160796	0.963250062474
SGD Classifier Lograthmic Regression	0.969372128637	0.969372128637
Logistic Regression	0.960183767228	0.969372128637

Regression

Regression Algos	Scipy	Apache Spark
Linear Regression	0.0452406207707	.94132634007
SGD Regression	0.0405330237698	N/A
Ridge Model	0.0381319185139	.94132634007
Lasso Model	0.0253828576146	.94132634007

As you can see from the above interpretation for classification Apache Spark seems to be slightly better in Classification. Whereas for the regression models Apache Spark is far better in the results.