# Hadoop and Spark Tutorial for Statisticians

Feng Li

November 30, 2015

## Contents

<table>
<thead>
<tr>
<th>1 Install Hadoop</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Pre-requests</td>
<td>3</td>
</tr>
<tr>
<td>1.1.1 SSH</td>
<td>3</td>
</tr>
<tr>
<td>1.1.2 JDK</td>
<td>3</td>
</tr>
<tr>
<td>1.1.3 Get Hadoop</td>
<td>3</td>
</tr>
<tr>
<td>1.2 Configuring Hadoop</td>
<td>3</td>
</tr>
<tr>
<td>1.2.1 Core configuration files</td>
<td>3</td>
</tr>
<tr>
<td>1.2.2 Important environment variables</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2 Start and stop Hadoop</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 Format HDFS</td>
<td>4</td>
</tr>
<tr>
<td>2.2 Start/Stop HDFS</td>
<td>4</td>
</tr>
<tr>
<td>2.3 Start/Stop MapReduce</td>
<td>4</td>
</tr>
<tr>
<td>2.4 Basic Hadoop shell commands</td>
<td>4</td>
</tr>
<tr>
<td>2.4.1 Create a directory in HDFS</td>
<td>4</td>
</tr>
<tr>
<td>2.4.2 Upload a local file to HDFS</td>
<td>4</td>
</tr>
<tr>
<td>2.4.3 Check files in HDFS</td>
<td>4</td>
</tr>
<tr>
<td>2.4.4 Hadoop task managements</td>
<td>6</td>
</tr>
<tr>
<td>2.4.5 Getting help from Hadoop</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3 Hadoop Streaming</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 A very simple word count example</td>
<td>7</td>
</tr>
<tr>
<td>3.2 Hadoop Streaming with R</td>
<td>7</td>
</tr>
<tr>
<td>3.2.1 Write an R script that accepts standard input and output</td>
<td>7</td>
</tr>
<tr>
<td>3.2.2 Your script has to be executable</td>
<td>9</td>
</tr>
<tr>
<td>3.2.3 Quick test your file and mapper function</td>
<td>9</td>
</tr>
<tr>
<td>3.2.4 Upload the data file to HDFS</td>
<td>9</td>
</tr>
</tbody>
</table>
1 Install Hadoop

1.1 Pre-requests

1.1.1 SSH

fli@carbon:~$ sudo apt-get install openssh-server
fli@carbon:~$ ssh-keygen -t rsa
fli@carbon:~$ cat ~/.ssh/id_rsa.pub >> authorized_keys

1.1.2 JDK

fli@carbon:~$ sudo apt-get install openjdk-7-jdk
fli@carbon:~$ java -version

1.1.3 Get Hadoop

Visit [Hadoop homepage](http://hadoop.apache.org) to download the latest version of Hadoop for Linux.

1.2 Configuring Hadoop

1.2.1 Core configuration files

The configuration files for Hadoop is at `etc/hadoop`. You have to set the at least the four core configuration files in order to start Hadoop properly.

mapred-site.xml
hdfs-site.xml
core-site.xml
hadoop-env.sh

1.2.2 Important environment variables

You have to set the following environment variables by either editing your Hadoop `etc/hadoop/hadoop-env.sh` file or editing your `~/.bashrc` file

export HADOOP_HOME="/hadoop # This is your Hadoop installation directory
export JAVA_HOME=/usr/lib/jvm/default-java/ # location to Java
export HADOOP_CONF_DIR=$HADOOP_HOME/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib"

- Single node mode
- Pseudo mode
• Cluster mode

2 Start and stop Hadoop

2.1 Format HDFS

fli@carbon:~/hadoop/bin$ hdfs namenode -format

2.2 Start/Stop HDFS

fli@carbon:~/hadoop/sbin$ start-dfs.sh

Namenode information then is accessible from \textcolor{blue}{http://localhost:50070}. However \texttt{sbin/stop-dfs.sh} will stop HDFS.

2.3 Start/Stop MapReduce

fli@carbon:~/hadoop/sbin$ start-yarn.sh

Hadoop administration page then is accessible from \textcolor{blue}{http://localhost:8088/}. However \texttt{sbin/stop-yarn.sh} will stop MapReduce.

2.4 Basic Hadoop shell commands

2.4.1 Create a directory in HDFS

fli@carbon:~/hadoop/bin$ hadoop fs -mkdir /test

2.4.2 Upload a local file to HDFS

fli@carbon:~/hadoop/bin$ hadoop fs -put ~/StudentNameList.xls /test

2.4.3 Check files in HDFS

fli@carbon:~/hadoop/bin$ hadoop fs -ls /test

Type \texttt{hadoop fs} to check other basic HDFS data operation commands

fli@carbon:~/hadoop/bin$ hadoop fs

Usage: hadoop fs [generic options]
[-appendToFile <localsrc> ... <dst>]
[-cat [-ignoreCrc] <src> ...]
[-checksum <src> ...]
[-chgrp [-R] GROUP PATH...]
[-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...]
[-chown [-R] [OWNER][[:GROUP]] PATH...]
[-copyToLocal [-f] [-p] <localsrc> ... <dstdst>]
[-count [-q] <path> ...]
[-cp [-f] [-p | -p[<topaxl]] <src> ... <dstdst>]
[-createSnapshot <snapshotDir> <snapshotName>]
[-deleteSnapshot <snapshotDir> <snapshotName>]
[-df [-h] <path> ...]
[-du [-s] [-h] <path> ...]
[-expunge]
[-get [-p] [-ignoreCrc] [-crc] <src> ... <dstdst>]
[-getfacl [-R] <path>]
[-getfattr [-R] {-n name | -d} [-e en] <path>]
[-getmerge [-nl] <src> <localdst>]
[-help [cmd ...]]
[-ls [-d] [-h] [-R] <path> ...]
[-mkdir [-p] <path> ...]
[-moveFromLocal <localsrc> ... <dstdst>]
[-moveToLocal <src> <localdst>]
[-mv <src> ... <dstdst>]
[-put [-f] [-p] <localsrc> ... <dstdst>]
[-renameSnapshot <snapshotDir> <oldName> <newName>]
[-rmdir [--ignore-fail-on-non-empty] <dir> ...]
[-setfacl [-R] [-b|-k] {-m|-x <acl_spec>} <path>]
[-setfattr {-n name [-v value] | -x name} <path>]
[-setrep [-R] [-w] <rep> <path> ...]
[-stat [format] <path> ...]
[-tail [-f] <file>]
[-test [-defsz] <path>]
[-text [-ignoreCrc] <src> ...]
[-touchz <path> ...]
[-usage [cmd ...]]

Generic options supported are

-conf <configuration file> specify an application configuration file
-D <property=value> use value for given property
-fs <local|namenode:port> specify a namenode
-jt <local|jobtracker:port> specify a job tracker
-files <comma separated list of files> specify comma separated files to be copied to
-libjars <comma separated list of jars> specify comma separated jar files to include
-archives <comma separated list of archives> specify comma separated archives to be

The general command line syntax is
bin/hadoop command [genericOptions] [commandOptions]

2.4.4 Hadoop task managements

fli@carbon:/hadoop/bin$ mapred job
Usage: CLI <command> <args>
[-submit <job-file>]
[-status <job-id>]
[-counter <job-id> <group-name> <counter-name>]
[-kill <job-id>]
[-set-priority <job-id> <priority>]. Valid values for priorities are: VERY_HIGH HIGH NORMAL LOW VERY_LOW
[-events <job-id> <from-event-#> <#-of-events>]
[-history <jobHistoryFile>]
[-list [all]]
[-list-active-trackers]
[-list-blacklisted-trackers]
[-list-attempt-ids <job-id> <task-type> <task-state>]. Valid values for <task-type> are
[-kill-task <task-attempt-id>]
[-fail-task <task-attempt-id>]
[-logs <job-id> <task-attempt-id>]

Generic options supported are
-conf <configuration file> specify an application configuration file
-D <property=property> use value for given property
-fs <local|namenode:port> specify a namenode
-jt <local|jobtracker:port> specify a job tracker
-files <comma separated list of files> specify comma separated files to be copied to
-libjars <comma separated list of jars> specify comma separated jar files to include
-archives <comma separated list of archives> specify comma separated archives to be

The general command line syntax is
bin/hadoop command [genericOptions] [commandOptions]
2.4.5 Getting help from Hadoop

Use your web browser to open the file `hadoop/share/doc/hadoop/index.html` which will guide you to the document entry for current Hadoop version.

3 Hadoop Streaming

3.1 A very simple word count example

```bash
fli@carbon:~$ hadoop/bin/hadoop jar 
  ~/hadoop/share/hadoop/tools/lib/hadoop-streaming-2.5.2.jar 
  -input /stocks.txt 
  -output wcoutput 
  -mapper "/bin/cat" 
  -reducer "/usr/bin/wc" 
```

3.2 Hadoop Streaming with R

3.2.1 Write an R script that accepts standard input and output.

See such example `stock_day_avg.R`

```R
#! /usr/bin/env Rscript

sink("/dev/null")

input <- file("stdin", "r")
while(length(currentLine <- readLines(input, n=1, warn=FALSE)) > 0)
{
  fields <- unlist(strsplit(currentLine, ","))
  lowHigh <- c(as.double(fields[3]), as.double(fields[6]))
  stock_mean <- mean(lowHigh)
  sink()
  cat(fields[1], fields[2], stock_mean, "\n", sep="\t")
  sink("/dev/null")
}

close(input)
```

And you input data file `stocks.txt` looks like the following format. The complete dataset can be downloaded from [http://finance.yahoo.com/](http://finance.yahoo.com/)
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Date</th>
<th>Open</th>
<th>High</th>
<th>Low</th>
<th>Close</th>
<th>Volume</th>
<th>Adj Close</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAPL</td>
<td>2009-01-02</td>
<td>85.88</td>
<td>91.04</td>
<td>85.16</td>
<td>90.75</td>
<td>26643400</td>
<td>90.75</td>
</tr>
<tr>
<td>AAPL</td>
<td>2008-01-02</td>
<td>199.27</td>
<td>200.26</td>
<td>192.55</td>
<td>194.84</td>
<td>38542100</td>
<td>194.84</td>
</tr>
<tr>
<td>AAPL</td>
<td>2007-01-03</td>
<td>86.29</td>
<td>86.58</td>
<td>81.90</td>
<td>83.80</td>
<td>44225700</td>
<td>83.80</td>
</tr>
<tr>
<td>AAPL</td>
<td>2006-01-03</td>
<td>72.38</td>
<td>74.75</td>
<td>72.25</td>
<td>74.75</td>
<td>28829800</td>
<td>74.75</td>
</tr>
<tr>
<td>AAPL</td>
<td>2005-01-03</td>
<td>64.78</td>
<td>65.11</td>
<td>62.60</td>
<td>63.29</td>
<td>5165800</td>
<td>10.64</td>
</tr>
<tr>
<td>AAPL</td>
<td>2004-01-02</td>
<td>21.55</td>
<td>21.75</td>
<td>21.18</td>
<td>21.28</td>
<td>5165800</td>
<td>10.64</td>
</tr>
<tr>
<td>AAPL</td>
<td>2002-01-02</td>
<td>22.05</td>
<td>23.30</td>
<td>21.96</td>
<td>23.30</td>
<td>18910600</td>
<td>11.65</td>
</tr>
<tr>
<td>AAPL</td>
<td>2001-01-02</td>
<td>14.88</td>
<td>15.25</td>
<td>14.56</td>
<td>14.88</td>
<td>16161800</td>
<td>7.44</td>
</tr>
<tr>
<td>AAPL</td>
<td>2000-01-03</td>
<td>104.87</td>
<td>112.50</td>
<td>101.69</td>
<td>111.94</td>
<td>19144000</td>
<td>27.99</td>
</tr>
<tr>
<td>CSCO</td>
<td>2009-01-02</td>
<td>16.41</td>
<td>17.00</td>
<td>16.25</td>
<td>16.96</td>
<td>40980600</td>
<td>16.96</td>
</tr>
<tr>
<td>CSCO</td>
<td>2008-01-02</td>
<td>27.00</td>
<td>27.98</td>
<td>27.33</td>
<td>27.73</td>
<td>64226000</td>
<td>27.73</td>
</tr>
<tr>
<td>CSCO</td>
<td>2007-01-03</td>
<td>17.21</td>
<td>17.49</td>
<td>17.18</td>
<td>17.45</td>
<td>55426000</td>
<td>17.45</td>
</tr>
<tr>
<td>CSCO</td>
<td>2006-01-03</td>
<td>19.42</td>
<td>19.61</td>
<td>19.27</td>
<td>19.32</td>
<td>56725600</td>
<td>19.32</td>
</tr>
<tr>
<td>CSCO</td>
<td>2004-01-02</td>
<td>13.11</td>
<td>13.69</td>
<td>13.09</td>
<td>13.64</td>
<td>61335700</td>
<td>13.64</td>
</tr>
<tr>
<td>CSCO</td>
<td>2002-01-02</td>
<td>38.13</td>
<td>38.50</td>
<td>32.63</td>
<td>33.31</td>
<td>17384600</td>
<td>33.31</td>
</tr>
<tr>
<td>CSCO</td>
<td>2001-01-03</td>
<td>109.94</td>
<td>110.25</td>
<td>103.56</td>
<td>108.06</td>
<td>53076000</td>
<td>54.03</td>
</tr>
<tr>
<td>GOOG</td>
<td>2009-01-02</td>
<td>308.60</td>
<td>321.82</td>
<td>305.50</td>
<td>312.32</td>
<td>29980600</td>
<td>29980600</td>
</tr>
<tr>
<td>GOOG</td>
<td>2008-01-02</td>
<td>692.87</td>
<td>697.37</td>
<td>677.73</td>
<td>685.19</td>
<td>43069000</td>
<td>685.19</td>
</tr>
<tr>
<td>GOOG</td>
<td>2007-01-03</td>
<td>466.00</td>
<td>476.66</td>
<td>461.11</td>
<td>467.59</td>
<td>7706500</td>
<td>467.59</td>
</tr>
<tr>
<td>GOOG</td>
<td>2006-01-03</td>
<td>422.52</td>
<td>435.67</td>
<td>418.22</td>
<td>435.23</td>
<td>13121200</td>
<td>435.23</td>
</tr>
<tr>
<td>GOOG</td>
<td>2005-01-03</td>
<td>197.40</td>
<td>203.64</td>
<td>195.46</td>
<td>202.71</td>
<td>15844200</td>
<td>202.71</td>
</tr>
<tr>
<td>MSFT</td>
<td>2008-01-02</td>
<td>35.79</td>
<td>35.96</td>
<td>35.00</td>
<td>35.22</td>
<td>63004200</td>
<td>33.79</td>
</tr>
<tr>
<td>MSFT</td>
<td>2007-01-03</td>
<td>29.91</td>
<td>30.25</td>
<td>29.40</td>
<td>29.86</td>
<td>76935100</td>
<td>28.26</td>
</tr>
<tr>
<td>MSFT</td>
<td>2006-01-03</td>
<td>26.25</td>
<td>27.00</td>
<td>26.10</td>
<td>26.84</td>
<td>79973000</td>
<td>25.04</td>
</tr>
<tr>
<td>MSFT</td>
<td>2005-01-03</td>
<td>26.80</td>
<td>26.95</td>
<td>26.65</td>
<td>26.74</td>
<td>65002900</td>
<td>24.65</td>
</tr>
<tr>
<td>MSFT</td>
<td>2004-01-02</td>
<td>27.58</td>
<td>27.77</td>
<td>27.33</td>
<td>27.45</td>
<td>44487700</td>
<td>22.64</td>
</tr>
<tr>
<td>MSFT</td>
<td>2003-01-03</td>
<td>52.30</td>
<td>53.75</td>
<td>51.71</td>
<td>53.72</td>
<td>67025200</td>
<td>21.95</td>
</tr>
<tr>
<td>MSFT</td>
<td>2002-01-02</td>
<td>66.65</td>
<td>67.11</td>
<td>65.51</td>
<td>67.04</td>
<td>48124000</td>
<td>27.40</td>
</tr>
<tr>
<td>MSFT</td>
<td>2001-01-02</td>
<td>44.13</td>
<td>45.00</td>
<td>42.88</td>
<td>43.38</td>
<td>82413200</td>
<td>17.73</td>
</tr>
<tr>
<td>YHOO</td>
<td>2008-01-02</td>
<td>23.80</td>
<td>24.15</td>
<td>23.60</td>
<td>23.72</td>
<td>25671700</td>
<td>23.72</td>
</tr>
<tr>
<td>YHOO</td>
<td>2006-01-03</td>
<td>39.69</td>
<td>41.22</td>
<td>38.79</td>
<td>40.91</td>
<td>24227700</td>
<td>40.91</td>
</tr>
<tr>
<td>YHOO</td>
<td>2005-01-03</td>
<td>38.36</td>
<td>38.90</td>
<td>37.65</td>
<td>38.18</td>
<td>25482800</td>
<td>38.18</td>
</tr>
</tbody>
</table>
3.2.2 Your script has to be executable

```bash
fli@carbon:~$ chmod +x stock_day_avg.R
```

And very importantly, you have to have your R installed on every worker node and the necessary R packages should be installed as well.

3.2.3 Quick test your file and mapper function

```bash
fli@carbon:~$ cat stocks.txt | stock_day_avg.R
```

3.2.4 Upload the data file to HDFS

```bash
fli@carbon:~$ hadoop/bin/hadoop fs -put stocks.txt /
```

3.2.5 Submitting tasks

```bash
fli@carbon:~$ hadoop/bin/hadoop \
   jar ~/hadoop/share/hadoop/tools/lib/hadoop-streaming-2.5.2.jar \
   -input /stocks.txt \
   -output output \
   -mapper "stock_day_avg.R"
```

3.2.6 View your result

You can either view your result from the web interface or use the following HDFS command

```bash
fli@carbon:~$ hadoop/bin/hdfs dfs -cat /user/fli/output/part-00000
```

3.3 Hadoop Streaming Documentation

The complete Hadoop Streaming Documentation can be found from Hadoop Installation directory share/doc/hadoop/hadoop-mapreduce-client/hadoop-mapreduce-client-core
4 Hadoop with Java API

We have the following Java WordCount version MapReduce program that counts the number of occurrences of each word in a given input set. This works with a local-standalone, pseudo-distributed or fully-distributed Hadoop installation.

```java
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {

    public static class TokenizerMapper
        extends Mapper<Object, Text, Text, IntWritable>{

            private final static IntWritable one = new IntWritable(1);
            private Text word = new Text();

            public void map(Object key, Text value, Context context
                ) throws IOException, InterruptedException {
                StringTokenizer itr = new StringTokenizer(value.toString());
                while (itr.hasMoreTokens()) {
                    word.set(itr.nextToken());
                    context.write(word, one);
                }
            }
        }

    public static class IntSumReducer
        extends Reducer<Text,IntWritable,Text,IntWritable> {

```
private IntWritable result = new IntWritable();

public void reduce(Text key, Iterable<IntWritable> values,
   Context context
   ) throws IOException, InterruptedException {
  int sum = 0;
  for (IntWritable val : values) {
    sum += val.get();
  }
  result.set(sum);
  context.write(key, result);
}

public static void main(String[] args) throws Exception {
  Configuration conf = new Configuration();
  Job job = Job.getInstance(conf, "word count");
  job.setJarByClass(WordCount.class);
  job.setMapperClass(TokenizerMapper.class);
  job.setCombinerClass(IntSumReducer.class);
  job.setReducerClass(IntSumReducer.class);
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(IntWritable.class);
  FileInputFormat.addInputPath(job, new Path(args[0]));
  FileOutputFormat.setOutputPath(job, new Path(args[1]));
  System.exit(job.waitForCompletion(true) ? 0 : 1);
}

Before we compile our java program. Make sure the following environment variables are set properly.

export JAVA_HOME=/usr/lib/jvm/default-java/
export PATH=$JAVA_HOME/bin:$PATH
export HADOOP_CLASSPATH=$JAVA_HOME/lib/tools.jar

You can check them from the terminal as

echo $HADOOP_CLASSPATH

Now we can compile WordCount.java and create a jar file
Then you will find a `wc.jar` at the same directory with `WordCount.java`. Now let’s upload some files to HDFS. We make an input directory named `input` that contains all our files to be counted. We would like to write all the output to `output` directory.

Please note that in above commands we have omitted the absolute path. So `WordCount/input` really means `/user/fli/WordCount/input` in HDFS.

We are going to submit our WordCount program to Hadoop.

Check the command output message, you will see a line like `Job job_local1195814039_0001 completed successfully` and you can find the output at HDFS.

### 5 Statistical Modeling with Hadoop

#### 5.1 Linear Regression Models.

The core algorithm for linear regression modeling is to code up a mapreduce procedure for $X'Y$ and $X'X$. One can decompose this into many submatrix multiplications and sum them over in the end. See the lecture notes for details.

#### 5.2 Logistic Regression Models

You will need to code up your own algorithm for estimating the coefficients in the model. You can use the RHadoop API or Mahout.
5.2.1 RHadoop

RHadoop is a collection of five R packages that allow users to manage and analyze data with Hadoop. Examples and helps can be found from [https://github.com/RevolutionAnalytics/RHadoop/wiki](https://github.com/RevolutionAnalytics/RHadoop/wiki).

5.2.2 Mahout

See next section.

5.2.3 Via approximations.

See lecture notes.

6 Statistical Learning with Mahout

6.1 Quick Install Mahout

6.1.1 Use the binary release

Please visit [https://mahout.apache.org/](https://mahout.apache.org/) to download the latest binary version (currently 0.9 is the release version) of Mahout. But remember that this version does not work well with Hadoop 2.5.2.

6.1.2 Compile your mahout that matches your hadoop

Instead of using the binary version, one may need to compile mahout to match the system hadoop (version 2.x).

Make sure you have maven and git installed in your system

```
fli@carbon:~$ sudo apt-get install maven git
```

You need to clone the newest mahout from the repository with git

```
fli@carbon:~$ git clone --branch master git://github.com/apache/mahout.git mahout
```

Now compile and pack mahout with Hadoop 2.x. This take a while

```
fli@carbon:~$ cd mahout
fli@carbon:~/mahout$ mvn -Dhadoop2.version=2.5.2 clean compile
fli@carbon:~/mahout$ mvn -Dhadoop2.version=2.5.2 -DskipTests=true clean package
fli@carbon:~/mahout$ mvn -Dhadoop2.version=2.5.2 -DskipTests=true clean install
```
6.2 Set up the necessary environment variables

Make sure the following environment variables are set properly

```bash
export MAHOUT_HOME=$HOME/mahout/
export MAHOUT_CONF_DIR=$MAHOUT_HOME/conf/
```

To integrate Mahout with Hadoop, make sure your Hadoop is installed properly and the following environment variables are correctly specified.

```bash
export HADOOP_HOME=$HOME/hadoop/
export HADOOP_CLASSPATH=$JAVA_HOME/lib/tools.jar
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop/
```

Note: There is a special environment variable MAHOUT_LOCAL. If it is set to not empty value, Mahout will run locally.

After installation, you will find all possible algorithms in your version.

fli@carbon:~/mahout$ bin/mahout

MAHOUT_LOCAL is not set; adding HADOOP_CONF_DIR to classpath.
Running on hadoop, using /home/fli/hadoop/bin/hadoop and HADOOP_CONF_DIR=/home/fli/hadoop/etc/hadoop
MAHOUT-JOB: /home/fli/mahout/mahout-examples-0.9-job.jar

An example program must be given as the first argument.

Valid program names are:

- arff.vector: Generate Vectors from an ARFF file or directory
- baumwelch: Baum-Welch algorithm for unsupervised HMM training
- canopy: Canopy clustering
- cat: Print a file or resource as the logistic regression models would see it
- cleansvd: Cleanup and verification of SVD output
- cmdump: Dump confusion matrix in HTML or text formats
- concatmatrices: Concatenates 2 matrices of same cardinality into a single matrix
- cvb: LDA via Collapsed Variation Bayes (0th deriv. approx)
- cvb0_local: LDA via Collapsed Variation Bayes, in memory locally.
- evaluateFactorization: Compute RMSE and MAE of a rating matrix factorization against
- fkmeans: Fuzzy K-means clustering
- hmmpredict: Generate random sequence of observations by given HMM
- itemsimilarity: Compute the item-item-similarities for item-based collaborative filtering
- kmeans: K-means clustering
- lucene.vector: Generate Vectors from a Lucene index
lucene2seq: Generate Text SequenceFiles from a Lucene index
matrixdump: Dump matrix in CSV format
matrixmult: Take the product of two matrices
parallelALS: ALS-WR factorization of a rating matrix
qualcluster: Runs clustering experiments and summarizes results in a CSV
recommendfactorized: Compute recommendations using the factorization of a rating matrix
recommenditembased: Compute recommendations using item-based collaborative filtering
regexconverter: Convert text files on a per line basis based on regular expressions
resplit: Splits a set of SequenceFiles into a number of equal splits
rowid: Map SequenceFile<Text,VectorWritable> to {SequenceFile<IntWritable,VectorWritable>, SequenceFile<IntWritable,Text>}
rowsimilarity: Compute the pairwise similarities of the rows of a matrix
runAdaptiveLogistic: Score new production data using a probably trained and validated model
runlogistic: Run a logistic regression model against CSV data
seq2encoded: Encoded Sparse Vector generation from Text sequence files
seq2sparse: Sparse Vector generation from Text sequence files
seqdirectory: Generate sequence files (of Text) from a directory
seqdumper: Generic Sequence File dumper
seqmailarchives: Creates SequenceFile from a directory containing gzipped mail archives
seqwiki: Wikipedia xml dump to sequence file
spectralkmeans: Spectral k-means clustering
split: Split Input data into test and train sets
splitDataset: split a rating dataset into training and probe parts
ssvd: Stochastic SVD
streamingkmeans: Streaming k-means clustering
svd: Lanczos Singular Value Decomposition
testnb: Test the Vector-based Bayes classifier
trainAdaptiveLogistic: Train an AdapтивeLogisticRegression model
trainlogistic: Train a logistic regression using stochastic gradient descent
trainnb: Train the Vector-based Bayes classifier
transpose: Take the transpose of a matrix
validateAdaptiveLogistic: Validate an AdapティブeLogisticRegression model against holdout data
tvecdist: Compute the distances between a set of Vectors (or Cluster or Canopy, they must fit in memory)
tvecdump: Dump vectors from a sequence file to text
tviterbi: Viterbi decoding of hidden states from given output states sequence

6.3 Run a Mahout Job

- Let Hadoop/HDFS up and run
- Upload data to HDFS
• Run the example

Assume you have uploaded a text data to HDFS’s user directory


You may run the command by calling Mahout directly which invokes Hadoop from the back,

```fli@carbon:~$ mahout/bin/mahout org.apache.mahout.clustering.syntheticcontrol.canopy.Job```

Or one can call Mahout from Hadoop

```fli@carbon:~$ hadoop/bin/hadoop jar \
$MAHOUT_HOME/examples/target/mahout-examples-1.0-SNAPSHOT-job.jar \
org.apache.mahout.clustering.syntheticcontrol.canopy.Job```

The output will be at your output directory under your HDFS user directory. For more information about this example, please visit [https://mahout.apache.org/users/clustering/canopy-clustering.html](https://mahout.apache.org/users/clustering/canopy-clustering.html)

### 6.4 Mahout build-in examples

There are a lot ready-to-use examples at `mahout/examples/bin` directory. Just run e.g.

```fli@carbon:~ mahout/examples/bin/classify-20newsgroups.sh```

### 6.5 Classification with random forests

We will run the random forests algorithm with Mahout 1.0 and Hadoop 2.5.2.

#### 6.5.1 Upload the data to HDFS’s directory

```fli@carbon:~$ ~/hadoop/bin/hadoop fs -put KDD* testdata
fli@carbon:~$ ~/hadoop/bin/hadoop fs -ls testdata
```

Found 2 items

- `rw-r--r--  1 fli supergroup  3365886 2014-12-14 17:32 testdata/KDDTest+.arff`
- `rw-r--r--  1 fli supergroup 18742306 2014-12-14 17:32 testdata/KDDTrain+.arff`
6.5.2 Generate the dataset description

fli@carbon:$ ~/hadoop/bin/hadoop jar \
   $MAHOUT_HOME/examples/target/mahout-examples-1.0-SNAPSHOT-job.jar \
   org.apache.mahout.classifier.df.tools.Describe \
   -p testdata/KDDTrain+.arff \
   -f testdata/KDDTrain+.info \
   -d N 3 C 2 N C 4 N C 8 N 2 C 19 N L

where the "N 3 C 2 N C 4 N C 8 N 2 C 19 N L" string describes all the
attributes of the data. In this cases, it means 1 numerical(N) attribute,
followed by 3 Categorical(C) attributes, ...L indicates the label.

A file named KDDTrain+.info will be generated and stored in testdata
directory. Check it with

fli@carbon:$ ~/hadoop/bin/hadoop fs -cat testdata/*.info

6.5.3 Build the model

We will try to build 100 trees (-t argument) using the partial implementation
(-p). Each tree is built using 5 random selected attribute per node (-sl
argument) and the example outputs the decision tree in the "nsl-forest"
directory (-o).

The number of partitions is controlled by the -Dmapred.max.split.size
argument that indicates to Hadoop the max. size of each partition, in this
case 1/10 of the size of the dataset. Thus 10 partitions will be used. IM-
PORTANT: using less partitions should give better classification results, but
needs a lot of memory.

fli@carbon:$ ~/hadoop/bin/hadoop jar \
   $MAHOUT_HOME/examples/target/mahout-examples-1.0-SNAPSHOT-job.jar \
   org.apache.mahout.classifier.df.mapreduce.BuildForest \n   -Dmapred.max.split.size=1874231 \n   -d testdata/KDDTrain+.arff \n   -ds testdata/KDDTrain+.info \n   -sl 5 -p -t 100 -o nsl-forest

A directory named nsl-forest will be generated that contains all the
model parameters.
6.5.4 Use the model to classify new data

Now we can compute the predictions of "KDDTest+.arff" dataset (-i argument) using the same data descriptor generated for the training dataset (-ds) and the decision forest built previously (-m). Optionally (if the test dataset contains the labels of the tuples) run the analyzer to compute the confusion matrix (-a), and you can also store the predictions in a text file or a directory of text files(-o). Passing the (-mr) parameter will use Hadoop to distribute the classification.

fli@carbon:~$ ~/hadoop/bin/hadoop jar
$MAHOUT_HOME/examples/target/mahout-examples-1.0-SNAPSHOT-job.jar
org.apache.mahout.classifier.df.mapreduce.TestForest \
-i testdata/KDDTest+.arff \
-ds testdata/KDDTrain+.info \
-m nsl-forest \
-a -mr \
-o predictions

which will return the following summary (as below) and the result will be stored in the predictions directory.

=======================================================
Summary
-------------------------------------------------------
Correctly Classified Instances : 17162 76.1267%
Incorrectly Classified Instances : 5382 23.8733%
Total Classified Instances : 22544
=======================================================
Confusion Matrix
-------------------------------------------------------
a b <--Classified as
8994 717 | 9711 a = normal
4665 8168 | 12833 b = anomaly
=======================================================
Statistics
-------------------------------------------------------
Kappa 0.536
Accuracy 76.1267%
Reliability 52.0883%
Reliability (standard deviation) 0.4738
Weighted precision 0.8069
Weighted recall 0.7613
Weighted F1 score 0.7597

If you have any question concerning with random forests, read Chapter 15 of [The Elements of Statistical Learning](#).

# 7 Introduction to Spark

## 7.1 Spark Shell

### 7.1.1 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 the Python version with exactly 4 cores by running the following in the Spark directory:

  ```
  ./bin/pyspark --master local[4]
  ```

  To find a complete list of options, run `pyspark --help`.

- Start the Scala version by running the following in the Spark directory:

  ```
  ./bin/spark-shell
  ```

- All examples based on this section will be based on Python. One may also check out the Scala version at [http://spark.apache.org/docs/latest/programming-guide.html](http://spark.apache.org/docs/latest/programming-guide.html)

- Spark’s primary abstraction is a distributed collection of items called a Resilient Distributed Dataset (RDD). RDDs can be created from Hadoop InputFormats (such as HDFS files) or by transforming other RDDs.

- To make a new RDD from the text of the README file in the Spark source directory:
RDDs have actions, which return values, and transformations, which return pointers to new RDDs.

```python
>>> textFile.count() # Number of items in this RDD
126

>>> textFile.first() # First item in this RDD
u'# Apache Spark'
```

- RDD actions and transformations can be used for more complex computations. Let's say we want to find the line with the most words:

```python
>>> textFile.map(lambda line: len(line.split())).reduce(lambda a, b: a if (a > b) else b)
15
```

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

```python
>>> linesWithSpark.cache()

>>> linesWithSpark.count()
15

>>> linesWithSpark.count()
15
```

### 7.2 Standalone Applications

- Assume we like to write a program that just counts the number of lines containing 'a' and the number containing 'b' in the Spark README.

#### 7.2.1 The Python version

```python
"""SimpleApp.py"
from pyspark import SparkContext

logFile = "YOUR_SPARK_HOME/README.md"  # some file on system
sc = SparkContext("local", "Simple App")
logData = sc.textFile(logFile).cache()
```
numAs = logData.filter(lambda s: 'a' in s).count()
numBs = logData.filter(lambda s: 'b' in s).count()

print "Lines with a: %i, lines with b: %i" % (numAs, numBs)

7.2.2 The Java version

/* SimpleApp.java */

import org.apache.spark.api.java.*;
import org.apache.spark.SparkConf;
import org.apache.spark.api.java.function.Function;

public class SimpleApp {
    public static void main(String[] args) {
        String logFile = "YOUR_SPARK_HOME/README.md"; // Should be some file on your system
        SparkConf conf = new SparkConf().setAppName("Simple Application");
        JavaSparkContext sc = new JavaSparkContext(conf);
        JavaRDD<String> logData = sc.textFile(logFile).cache();

        long numAs = logData.filter(new Function<String, Boolean>() {
            public Boolean call(String s) { return s.contains("a"); }
        }).count();

        long numBs = logData.filter(new Function<String, Boolean>() {
            public Boolean call(String s) { return s.contains("b"); }
        }).count();

        System.out.println("Lines with a: " + numAs + ", lines with b: " + numBs);
    }
}

7.2.3 The Scala version

/* SimpleApp.scala */

import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf
object SimpleApp {
  def main(args: Array[String]) {
    val logFile = "YOUR_SPARK_HOME/README.md" // Should be some file on your system
    val conf = new SparkConf().setAppName("Simple Application")
    val sc = new SparkContext(conf)
    val logData = sc.textFile(logFile, 2).cache()
    val numAs = logData.filter(line => line.contains("a")).count()
    val numBs = logData.filter(line => line.contains("b")).count()
    println("Lines with a: %s, Lines with b: %s".format(numAs, numBs))
  }
}

7.3 Submitting Applications to Spark

7.3.1 Bundling Your Application’s Dependencies

- If your code depends on other projects, you will need to package them
  alongside your application in order to distribute the code to a Spark
  cluster.

- To do this, to create an assembly jar containing your code and its
  dependencies. When creating assembly jars, list Spark and Hadoop
  as provided dependencies; these need not be bundled since they are
  provided by the cluster manager at runtime.

- For Python, you can use the --py-files argument of spark-submit
  to add .py, .zip or .egg files to be distributed with your application. If
  you depend on multiple Python files, pack them into a .zip or .egg.

- Once a user application is bundled, it can be launched using the
  bin/spark-submit script.

7.3.2 Run Your Application

- Run application locally on 8 cores

  ./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master local[8] \
  /path/to/examples.jar \ 
  100
• Run on a Spark standalone cluster

```bash
./bin/spark-submit \
   --class org.apache.spark.examples.SparkPi \
   --master spark://207.184.161.138:7077 \
   --executor-memory 20G \
   --total-executor-cores 100 \
   /path/to/examples.jar \n   1000
```

• Run on a Hadoop YARN cluster

```bash
export HADOOP_CONF_DIR=XXX
./bin/spark-submit \
   --class org.apache.spark.examples.SparkPi \
   --master yarn-cluster \ # can also be ‘yarn-client’ for client mode \
   --executor-memory 20G \
   --num-executors 50 \
   /path/to/examples.jar \n   1000
```

• Run a Python application on a cluster

```bash
./bin/spark-submit \
   --master spark://207.184.161.138:7077 \
   examples/src/main/python/pi.py \n   1000
```