Hadoop Examples

Aggregate Wordcount
reads the text input files, breaks each line into words and counts them. The output is a locally sorted list of words and the count of how often they occurred.

```
bin/hadoop jar hadoop-examples-*.jar aggregatewordcount in-dir
out-dir numOfReducers textinputformat
```

Aggregate Word Histogram
Computes the histogram of the words in the input texts.

```
bin/hadoop jar hadoop-examples-*.jar aggregatewordhistin-dir out-
dir numOfReducers textinputformat
```

Join
This is the trivial map/reduce program that does absolutely nothing other than use the framework to fragment and sort the input values.

```
bin/hadoop jar build/hadoop-examples.jar join [-m maps] [-r
reduces] [-inFormat input format class] [-outFormat output format
class] [-outKey output key class] [-outValue output value class]
[-joinOp <inner|outer|override>] [in-dir]* in-dir out-dir
```

Sort
This is the trivial map/reduce program that does absolutely nothing other than use the framework to fragment and sort the input values.

```
bin/hadoop jar build/hadoop-examples.jar sort [-m maps] [-r
reduces] [-inFormat input format class] [-outFormat output format
class] [-outKey output key class] [-outValue output value class]
[-totalOrder pcnt num samples max splits] in-dir out-dir
```

Secondary Sort
This is an example Hadoop Map/Reduce application. It reads the text input files that must contain two integers per a line. The output is sorted by the first and second number and grouped on the first number.

```
bin/hadoop jar build/hadoop-examples.jar secondarysort in-dir out-
dir
```

Mahout

Classification

Testing Naive Bayes on 20Newsgroup Dataset

Run the 20 newsgroup example by executing the script as below
```
$ ./examples/bin/build-20news-bayes.sh
```
After MAHOUT-857 is committed (available when 0.6 is released), the command will be:
```
$ ./examples/bin/classify-20newsgroups.sh
```
This later version allows you to also try out running Stochastic Gradient Descent (SGD) on the same data.

```
$> $MAHOUT_HOME/bin/mahout trainclassifier \
  -i 20news-input \
  -o newsmodel \
  -type cbayes \
  -ng 2 \
  -source hdfs

$> $MAHOUT_HOME/bin/mahout testclassifier \
  -m newsmodel \
  -d 20news-input \
  -type cbayes \
  -ng 2 \
  -source hdfs \
  -method mapreduce
```

Bayesian Classification - Generic

Put the data:
```
$HADOOP_HOME/bin/hadoop fs -put <PATH TO DATA> testdata
```

Run the Job:
```
$HADOOP_HOME/bin/hadoop jar $MAHOUT_HOME/core/target/mahout-core-
<MAHOUT VERSION>.job
org.apache.mahout.classifier.bayes.mapreduce.bayes.BayesDriver
<OPTIONS>
```

Get the data out of HDFS and have a look. Use bin/hadoop fs -lsr output to view all outputs.

Brieman

Create a File Descriptor for the Dataset
```
$HADOOP_HOME/bin/hadoop jar $MAHOUT_HOME/core/target/mahout-core-
<VERSION>-job.jar org.apache.mahout.classifier.df.tools.Describe
-p testdata/glass.data -f testdata/glass.info -d I 9 N L
```

```
$HADOOP_HOME/hadoop jar $MAHOUT_HOME/examples/target/mahout-
examples-<VERSION>-job.jar
org.apache.mahout.classifier.df.BreimanExample -d
testdata/glass.data -ds testdata/glass.info -i 10 -t 100
```

Clustering

K Means

Data should be in the folder testdata
Single Node :-

```
./bin/mahout kmeans -i testdata -o output -c clusters -dm
org.apache.mahout.common.distance.CosineDistanceMeasure -x 5 -ow
-cd 1 -k 25
```

Cluster :-
Prepare hadoop and put data in HDFS

```
$HADOOP_HOME/bin/start-all.sh
$HADOOP_HOME/bin/hadoop fs -put <PATH TO DATA> testdata

export HADOOP_HOME=<Hadoop Home Directory>
export HADOOP_CONF_DIR=$HADOOP_HOME/conf
./bin/mahout kmeans -i testdata -o output -c clusters -dm
org.apache.mahout.common.distance.CosineDistanceMeasure -x 5 -ow
-cd 1 -k 25
```

clusters contain The input centroids, as Vectors. Must be a SequenceFile of Writable, Cluster/Canopy. If k is also specified, then a random set of vectors will be selected and written out to this path first x defines max iterations. ow allows overwrite, cd defines convergence delta

Canopy Algorithm

Prepare hadoop and put data in dhfs

```
./bin/mahout canopy -i testdata -o output -dm
org.apache.mahout.common.distance.CosineDistanceMeasure -ow -t1 5
-t2 2

bin/mahout canopy \
    -i <input vectors directory> \
    -o <output working directory> \
    -dm <DistanceMeasure> \
    -t1 <T1 threshold> \
    -t2 <T2 threshold> \
    -t3 <optional reducer T1 threshold> \
    -t4 <optional reducer T2 threshold> \
    -cf <optional cluster filter size (default: 0)> \
    -ow <overwrite output directory if present>
    -cl <run input vector clustering after computing Canopies>
    -xm <execution method: sequential or mapreduce>
```

Recommendation System using Examples

In the examples directory type:

```
mvn -q exec:java
-Dexec.mainClass="org.apache.mahout.cf.taste.example.bookcrossing.
BookCrossingRecommenderEvaluatorRunner" -Dexec.args="<OPTIONS>"
mvn -q exec:java
-Dexec.mainClass="org.apache.mahout.cf.taste.example.netflix.Netfl
ixRecommenderEvaluatorRunner" -Dexec.args="<OPTIONS>"
mvn -q exec:java
-Dexec.mainClass="org.apache.mahout.cf.taste.example.netflix.Trans
poseToByUser" -Dexec.args="<OPTIONS>"
mvn -q exec:java
-Dexec.mainClass="org.apache.mahout.cf.taste.example.jester.Jester
RecommenderEvaluatorRunner" -Dexec.args="<OPTIONS>"
mvn -q exec:java
-Dexec.mainClass="org.apache.mahout.cf.taste.example.grouplens.Gro
upLensRecommenderEvaluatorRunner" -Dexec.args="<OPTIONS>"

Here, the command line options need only be:
-i [input file]
```

R

Running R Programs

Each application on R, like Clustering or Classification, will be stored as separate scripts in the system.

Syntax for running R programs from the command-line. Requires in first line of my_script.R the following statement:

```
     #!/usr/bin/env Rscript
        $ Rscript my_script.R # or just ./myscript.R after making
file executable with 'chmod +x my_script.R'
```

Alternatively, one can use the following syntax to run R programs in BATCH mode from the command-line. All commands starting with a '$' sign need to be executed from a Unix or Linux shell.

```
    $ R CMD BATCH [options] my_script.R [outfile]
```

The output file lists the commands from the script file and their outputs. If no outfile is specified, the name used is that of 'infile' and '.Rout' is appended to outfile. To stop all the usual R command line information from being written to the outfile, add this as first line to my_script.R file: 'options(echo=FALSE)'. If the command is run like this 'R CMD BATCH --no-save my_script.R', then nothing will be saved in the .Rdata file which can get often very large. More on this can be found on the help pages: '$ R CMD BATCH --help' or '> ? BATCH'.

Examples

```
Rscript -e 'date()' -e 'format(Sys.time(), "%a %b %d %X %Y")'

## example #! script for a Unix-alike

#! /path/to/Rscript --vanilla --default-packages=utils
args <- commandArgs(TRUE)
res <- try(install.packages(args))
if(inherits(res, "try-error")) q(status=1) else q()
```

R programs can also be called using EXEC command.
For example the following line calls R and loads a script:

```
exec('/bin/R --vanilla < /home/docs/R/plotTemplate.R');
```

If you want a standard procedure you can go for JRI.
http://rforge.net/JRI/
JRI is a Java/R Interface, which allows to run R inside Java applications as a single thread. Basically it loads R dynamic library into Java and provides a Java API to R functionality. It supports both simple calls to R functions and a full running REPL.

In a sense JRI is the inverse of rJava and both can be combined (i.e. you can run R code inside JRI that calls back to the JVM via rJava). TheJGR project makes the full use of both JRI and rJava to provide a full Java GUI for R.

Data Preprocessing

```
y <- matrix(rnorm(50), 10, 5, dimnames=list(paste("g", 1:10,
sep=""), paste("t", 1:5, sep="")))
```

Creates a sample dataset

Data centering and scaling

```
scale(t(y)); yscaled <- t(scale(t(y)));
apply(yscaled, 1, sd)
```

The function scale() centers and/or scales numeric matrices column-wise. When used with its default settings, the function returns columns that have a mean close to zero and a standard deviation of one. For row-wise scaling the data set needs to be transposed first using the t() function. Another transposing step is necessary to return the data set in its original orientation (same row/column assignment). Centering and scaling are common data transformation steps for many clustering techniques.

Hierarchical Clustering

Clustering with hclust

```
y <- matrix(rnorm(50), 10, 5, dimnames=list(paste("g", 1:10,
sep=""), paste("t", 1:5, sep=""))) # Creates a sample data set.
```

```
c <- cor(t(y), method="spearman"); d <- as.dist(1-c); hr <-
hclust(d, method = "complete", members=NULL)
```

This is a short example of clustering the rows of the generated sample matrix 'y' with hclust. Seven different clustering methods can be selected with the 'method' argument: ward, single, complete, average, mcquitty, median and centroid. The object returned by hclust is a list of class hclust which describes the tree generated by the clustering process with the following components: merge, height, order, labels, method, call and dist.method.

```
par(mfrow = c(2, 2)); plot(hr, hang = 0.1); plot(hr, hang = -1)
```

The generated tree can be plotted with the plot() function. When the hang argument is set to '-1' then all leaves end on one line and their labels hang down from 0.

`plot(as.dendrogram(hr), edgePar=list(col=3, lwd=4), horiz=T)` # To plot trees horizontally, the hclust tree has to be transformed into a dendrogram object.
`unclass(hr)` # Prints the full content of the hclust object.
`str(as.dendrogram(hr))` # Prints dendrogram structure as text.
`hr$labels[hr$order]` # Prints the row labels in the order they appear in the tree.

```
par(mfrow=c(2,1)); hrd1 <- as.dendrogram(hr); plot(hrd1); hrd2 <-
reorder(hrd1, sample(1:10)); plot(hrd2); labels(hrd1);
labels(hrd2)
```

Example to reorder a dendrogram and print out its labels.
library(ctc); hc2Newick(hr) # The 'hc2Newick' function of the BioC ctc library can convert a hclust object into the Newick tree file format for export into external programs.

Other Clustering Algorithms

K-Means

```
km <- kmeans(t(scale(t(y))), 3); km$cluster
```

K-means clustering is a partitioning method where the number of clusters is pre-defined. The basic R implementation requires as input the data matrix and uses Euclidean distance. In contrast to pam(), the implementation does not allow to provide a distance matrix. In the given example the row-wise centered data matrix is provided.

```
mycol <- sample(rainbow(256)); mycol <-
mycol[as.vector(km$cluster)]; heatmap(y, Rowv=as.dendrogram(hr),
Colv=as.dendrogram(hc), col=my.colorFct(), scale="row",
ColSideColors=heat.colors(length(hc$labels)), RowSideColors=mycol)
```

This example shows how the obtained K-means clusters can be compared with the hierarchical clustering results by highlighting them in the heatmap color bar.

PAM (partitioning around medoids)

```
library(cluster) # Loads the required cluster library.
y <- matrix(rnorm(50), 10, 5, dimnames=list(paste("g", 1:10,
sep=""), paste("t", 1:5, sep=""))) # Creates a sample data set.
pamy <- pam(y, 4); pamy$clustering
```

Clusters data into 4 clusters using default Euclidean as distance method.

```
mydist <- as.dist(1-cor(t(y), method="pearson"))
```

Generates distance matrix using Pearson correlation as distance method.

```
pamy <- pam(mydist, 4); pamy$clustering
```

Same a above, but uses provided distance matrix.

```
pam(mydist, 4, cluster.only=TRUE)
```

Same as before, but with faster computation.

```
plot(pamy) # Generates cluster plot.
```

Clara (clustering large applications: PAM method for large data sets)

```
clarax <- clara(t(scale(t(y))), 4); clarax$clustering
```

Clusters above data into 4 clusters using default Euclidean as distance method. If the data set consists of gene expression values, then row-wise scaling of y is recommend in combination with Euclidean distances.

Giraph

Benchmarks

Page Rank in Giraff

```
hadoop jar giraph-0.1-jar-with-dependencies.jar
org.apache.giraph.benchmark.PageRankBenchmark -h
usage: org.apache.giraph.benchmark.PageRankBenchmark [-e <arg>] [-
h] [-s
   <arg>] [-v] [-V <arg>] [-w <arg>]
 -e,--edgesPerVertex <arg>      Edges per vertex
 -h,--help                      Help
 -s,--supersteps <arg>          Supersteps to execute before
finishing
 -v,--verbose                   Verbose
 -V,--aggregateVertices <arg>   Aggregate vertices
 -w,--workers <arg>             Number of workers
```

Example

```
$ hadoop jar giraph-0.1-jar-with-dependencies.jar
org.apache.giraph.benchmark.PageRankBenchmark -e 1 -s 3 -v -V
50000000 -w 30
```

Shortest Paths Benchmark

```
hduser@Adie:/home/aditya$ /usr/local/hadoop/bin/hadoop jar giraph-
0.2-SNAPSHOT-for-hadoop-0.20.203.0-jar-with-dependencies.jar
org.apache.giraph.benchmark.ShortestPathsBenchmark
usage: org.apache.giraph.benchmark.ShortestPathsBenchmark [-c
<arg>] [-e
       <arg>] [-h] [-nc] [-v] [-V <arg>] [-w <arg>]
 -c,--vertexClass <arg>         Vertex class (0 for HashMapVertex,
1 for
                                EdgeListVertex)
 -e,--edgesPerVertex <arg>      Edges per vertex
 -h,--help                      Help
 -nc,--noCombiner               Don't use a combiner
 -v,--verbose                   Verbose
 -V,--aggregateVertices <arg>   Aggregate vertices
 -w,--workers <arg>             Number of workers
```

Random Message Benchmark

Random Message Benchmark for evaluating the messaging performance.

```
$ /usr/local/hadoop/bin/hadoop jar giraph-0.2-SNAPSHOT-for-hadoop-
0.20.203.0-jar-with-dependencies.jar
org.apache.giraph.benchmark.RandomMessageBenchmark
usage: org.apache.giraph.benchmark.RandomMessageBenchmark [-b
<arg>] [-e
```

```
       <arg>] [-f <arg>] [-h] [-n <arg>] [-s <arg>] [-v] [-V
<arg>] [-w
       <arg>]
 -b,--bytes <arg>                Message bytes per memssage
 -e,--edgesPerVertex <arg>       Edges per vertex
 -f,--flusher <arg>              Number of flush threads
 -h,--help                       Help
 -n,--number <arg>               Number of messages per edge
 -s,--supersteps <arg>           Supersteps to execute before
finishing
 -v,--verbose                    Verbose
 -V,--aggregateVertices <arg>    Aggregate vertices
 -w,--workers <arg>              Number of workers
```

Computations

Connected Components

Implementation of the HCC algorithm that identifies connected components and assigns each vertex its "component identifier" (the smallest vertex id in the component) The idea behind the algorithm is very simple: propagate the smallest vertex id along the edges to all vertices of a connected component. The number of supersteps necessary is equal to the length of the maximum diameter of all components + 1

Allow user to add arguments

```
$ /usr/local/hadoop/bin/hadoop jar giraph-0.2-SNAPSHOT-for-hadoop-
0.20.203.0-jar-with-dependencies.jar
org.apache.giraph.examples.ConnectedComponents
```

Simple PageRank Vertex

Demonstrates the basic Pregel PageRank implementation.

```
$ /usr/local/hadoop/bin/hadoop jar giraph-0.2-SNAPSHOT-for-hadoop-
0.20.203.0-jar-with-dependencies.jar
org.apache.giraph.examples.SimplePageRankVertex
```

Verify Message

An example that simply uses its id, value, and edges to compute new data every iteration to verify that messages are sent and received at the appropriate location and superstep.

```
$ /usr/local/hadoop/bin/hadoop jar giraph-0.2-SNAPSHOT-for-hadoop-
0.20.203.0-jar-with-dependencies.jar
org.apache.giraph.examples.VerifyMessage
```