MBatch 02 Running Tests
Tod Casasent
2017-11-02-0830

1 Introduction

These instructions are aimed at people familiar with R. Additional details about each R test will be covered in the "Using MBatch" portion of the MBatch documentation series.

2 Target Operating System and Installation

These instructions were tested on Debian 9.1 installed on Oracle VirtualBox. These instructions with appropriate modifications should work as a basis for testing MBatch on other distributions.

If MBatch is installed from GitHub using the devtools install, the tests directory is installed automatically. If using R CMD INSTALL, use "--install-tests" to include tests.

3 Setting up Tests

MBatch uses a default directory of /bea_testing for tests. Testing uses three subdirectories.

The first subdirectory is /bea_testing/MATRIX_DATA. Download the zip file located at http://tcgadata.mdanderson.org/std_archives/MATRIX_DATA.zip and copy the contents of the MATRIX_DATA folder inside the ZIP into /bea_testing/MATRIX_DATA. This is the data used by the check tests.

The second subdirectory is /bea_testing/COMPARE. Download the zip file located at http://tcgadata.mdanderson.org/std_archives/COMPARE.zip and copy the contents of the COMPARE folder inside the ZIP into /bea_testing/COMPARE. This is the data used to confirm the results of the check tests.

Finally, create /bea_testing/output. This is where the check tests write their results.

4 Running Tests

Then run the following code to execute the "tests". Later sections will describe each of the tests. Most tests compare the output from the test with known good output. Some tests, documented as such, rely on errors being thrown to detect problems. These tests should take about an hour to run on a standard machine.

```
# Within R
library(MBatch)
# Set these environment variable to override file locations if needed
#Sys.setenv(MBATCH_TEST_OUTPUT="/bea_testing/output")
#Sys.setenv(MBATCH_TEST_INPUT="/bea_testing/MATRIX_DATA")
#Sys.setenv(MBATCH_TEST_COMPARE="/bea_testing/COMPARE")

baseDir <- file.path(system.file(package = "MBatch"), "tests")
message(baseDir)
testFiles <- list.files(path=baseDir)
print(testFiles)
results <- c()
for(myFile in testFiles)
{
	message("**************************************************************")
	message("**************************************************************")
	message("**** ", file.path(baseDir, myFile))
	message("**************************************************************")
	message("**************************************************************")
	test <- source(file.path(baseDir, myFile))
	if (isTRUE(test$value))
	{
		results <- c(results, paste("Test succeeded for ", myFile, sep=""))
	}
	else
	{
		results <- c(results, paste("Test failed for ", myFile, sep=""))
	}
}
print(results)
```

The "print(results)" line should display results similar to this:

```
 [1] "Test succeeded for AN_Adjusted.R"
 [2] "Test succeeded for AN_Unadjusted.R"
 [3] "Test succeeded for Boxplot_AllSamplesData_Structures.R"
 [4] "Test succeeded for Boxplot_AllSamplesRLE_Structures.R"
 [5] "Test succeeded for Boxplot_Group_Structures.R"
 [6] "Test succeeded for CDP_Files.R"
 [7] "Test succeeded for CDP_Plot.R"
 [8] "Test succeeded for CDP_Structures.R"
 [9] "Test succeeded for EB_withNonParametricPriors.R"
[10] "Test succeeded for EB_withParametricPriors.R"
[11] "Test succeeded for EBNPlus_CombineBatches.R"
[12] "Test succeeded for EBNPlus_Correction_Files.R"
[13] "Test succeeded for EBNPlus_Correction_Structures.R"
[14] "Test succeeded for HierarchicalClustering_Structures.R"
[15] "Test succeeded for MP_ByBatch.R"
[16] "Test succeeded for MP_Overall.R"
[17] "Test succeeded for PCA_DualBatch_Structures.R"
[18] "Test succeeded for PCA_Regular_Structures.R"
[19] "Test succeeded for RBN_Pseudoreplicates.R"
[20] "Test succeeded for RBN_Replicates.R"
[21] "Test succeeded for SupervisedClustering_Batches_Structures.R"
[22] "Test succeeded for SupervisedClustering_Pairs_Structures.R"
```

5 General Test Overview

The test code follows a set pattern. First variables are set to contain directory paths, then the we assign input and output files along with any arguments, such as the random number seed. Here, we see this portion of the code for the EBNPlus_Correction_Structures,R test.

```
#directory paths
inputDir <- getTestInputDir()
outputDir <- getTestOutputDir()
compareDir <- getTestCompareDir()

# input and output files
theDataFile1=file.path(inputDir, "brca_rnaseq2_matrix_data.tsv")
theDataFile2=file.path(inputDir, "brca_agi4502_matrix_data.tsv")
theOutputDir=file.path(outputDir, "ebnplus")
theCompareFile=file.path(compareDir, "EBNPlus_Correction_Structures.tsv")
theBatchId1="RNASeqV2"
theBatchId2="Agilent4502"
theRandomSeed=314
```

Some tests will then have utility or helper functions defined. For the EBNPlus_Correction_Structures,R test, here is one helper function, that trims the Entrez Id off the Hugo Gene Symbol from Standardized Data. (Standardized Data and data formats are covered in other entries in the MBatch documentation series.)

```
# trim genes to get just gene symbols from standardized data
trimGenes <- function(theGenes)
{
 foo <- as.vector(unlist(
  sapply(theGenes, function(theGene)
  {
   # keep the same if it starts with ?
   if (TRUE==grepl("^[?]+", theGene))
   {
    return(theGene)
   }
   else
   {
    # split on the | and take the first argument
    # this makes no change if no pipe
    return(strsplit(theGene, "|", fixed=TRUE)[[1]][1])
   }
  })
 ))
 foo
}
```

The next section is set off in an "if statement" that checks to see if the input directory is not null.

```
if (!is.null(inputDir))
{
 # Other testing code goes here
} else {
 message("No test data. Skip test.")
 TRUE
}
```

Within the "if statement" lies the code which sets up test conditions, reads the data, runs the selected code, and compares the output. The below portion of code sets up the test conditions, like the paths, directories and R "options". Here we set R to treat warnings as errors, and to reset the error level back to normal once the test is done. We also want to see the call stack for warnings and errors, to assist in debugging. Finally, any existing output directory is removed and recreated, so output is fresh each time.

```
warnLevel<-getOption("warn")
on.exit(options(warn=warnLevel))
# warnings are errors
options(warn=3)
# if there is a warning, show the calls leading up to it
options(showWarnCalls=TRUE)
# if there is an error, show the calls leading up to it
options(showErrorCalls=TRUE)

outdir <- file.path(theOutputDir, "EBNPlus_Correction_Structures")
unlink(outdir, recursive=TRUE)
dir.create(outdir, showWarnings=FALSE, recursive=TRUE)
```

Next, generally, the data is loaded and prepared, if necessary, for the actual MBatch code call. Here, we read two matrix files (which are to be combined using the EBNPlus correction algorithm), clean up their gene lists, and remove duplicates from the gene list (a requirement of the algorithm). The "readAsGenericMatrix" sorts the row and column names. Additional details about algorithms and requirements are covered in other entries in the MBatch documentation series.

```
# read the files in. This can be done however you want
theDataMatrix1 <- readAsGenericMatrix(theDataFile1)
theDataMatrix2 <- readAsGenericMatrix(theDataFile2)
# this is the reduce genes to just gene symbols, handling those from standardized data
rownames(theDataMatrix1) <- trimGenes(rownames(theDataMatrix1))
rownames(theDataMatrix2) <- trimGenes(rownames(theDataMatrix2))
# remove any duplicates (this is a requirement for EBNplus)
theDataMatrix1 <-
removeDuplicatesFromColumns(removeDuplicatesFromRows(theDataMatrix1))
theDataMatrix2 <-
removeDuplicatesFromColumns(removeDuplicatesFromRows(theDataMatrix2))
```

Next, the MBatch function is called. In this case, the corrected data matrix is returned. In some cases, the file path with the corrected data is returned, which is then read in the next section.

```
 corrrectedMatrix <- EBNPlus_Correction_Structures(theDataMatrix1, theDataMatrix2,
theBatchId1, theBatchId2,
                                theEBNP_BatchWithZero="1",
                                theEBNP_FixDataSet=as.numeric(NA),
                                theEBNP_CorrectForZero=TRUE,
                                theEBNP_ParametricPriorsFlag=TRUE,
                                theSeed=theRandomSeed,
                                theEBNP_PriorPlotsFile=file.path(outdir, "priorplots.PNG"))
```

In the final section, any files needed for the comparison are read (in this case, the data to compare to the just generated corrected data), and the comparison is done. Note that "compareTwoMatrices" compares the row and column names, requiring the names to be sorted (see earlier comment about "readAsGenericMatrix"). The compare function also uses "all.equal" to compare each value cell-by-cell, to compensate for CPU floating point rounding differences between computers. The "inefficient" cell-by-cell comparison is done to make error checking simpler and allow for printing of cell location and values when a comparison fails. Comparison of corrected data cells and compare data cells that are both "NA" or "NaN" are skipped. The results of the compare (TRUE or FALSE) are returned.

```
compareMatrix <- readAsGenericMatrix(theCompareFile)
compared <- compareTwoMatrices(correctedMatrix, compareMatrix)
print(compared)
return(compared)
```