Brundle Example using Spike-in Control

Andrew Holding

8/8/2017

Brundle Examples

This markdown provides an example of a workflow using Brundle applied to a minimal dataset as included in the BrundleData package.

The packages are found on GitHub as AndrewHolding/Brundle & AndrewHolding/BrundleData. They can be installed with the following code.

```
install.packages("devtools")
library(devtools)

install_github("AndrewHolding/Brundle")
install_github("AndrewHolding/BrundleData")
```

To run this example you will also need to download and install the modified version of DiffBind included with the manuscript.

```
URL <- paste0("https://raw.githubusercontent.com/",
              "andrewholding/flypeaks/master/Diffbind/DiffBind_2.5.6.tar.gz")
download.file(URL, destfile = "./DiffBind_2.5.6.tar.gz", method="curl")
install.packages("DiffBind_2.5.6.tar.gz", repos = NULL, type="source")
```

Once installed, we do not need to install them again and can load them as normal. The Brundle package will load DiffBind automatically.

```
library(Brundle)
library(BrundleData)
```

The initial steps of the Brundle Pipeline are to set the variables. Here we are using the data from the BrundleData package which contains two sample sheets formatted as required by DiffBind. They refer to the two sets of data (BAM files) one for the reads aligned to the human genome from each sample and the other refers to the reads aligned to the drosphila. Each sample sheet also contain refences the BED files with peak cordinates for the the correct genome.

A fully working series of shell scripts to align reads to a merged drosphila/human genome, peak call, and output the data in the format used by Brundle can be found in the preproccessing folder of the github repository along with sample read data in FastQ format.

The peaks in the drosophila BAM files are to provide our control peaks, while the ER binding provides our experimental peak changes. In this example, we have treated MCF7 cells with Fulvestrant to alter ER binding.

```
#Set up the initial varible
jg.controlMinOverlap      <- 5

jg.controlSampleSheet <-
    system.file("extdata","samplesheet/samplesheet_SLX8047_dm.csv", package =
                "BrundleData")

jg.experimentSampleSheet <-
    system.file("extdata", "samplesheet/samplesheet_SLX8047_hs.csv", package =
```

```
                    "BrundleData")

jg.treatedCondition       =  "Fulvestrant"
jg.untreatedCondition     =  "none"
```

Once configured we load the data from the samples sheets as normal with DiffBind. This provides us with two DiffBind objects: one experimental and one control.

```
setwd(system.file("extdata",package="BrundleData"))

dbaExperiment <- jg.getDba(jg.experimentSampleSheet, bRemoveDuplicates=TRUE)
dbaControl    <- jg.getDba(jg.controlSampleSheet, bRemoveDuplicates=TRUE)
```

We then use Brundle to extract the data from the DiffBind object to generate a peakset. This provides us with the read count at each peak location for each sample.

```
jg.experimentPeakset <- jg.dbaGetPeakset(dbaExperiment)
jg.controlPeakset    <- jg.dbaGetPeakset(dbaControl)
```

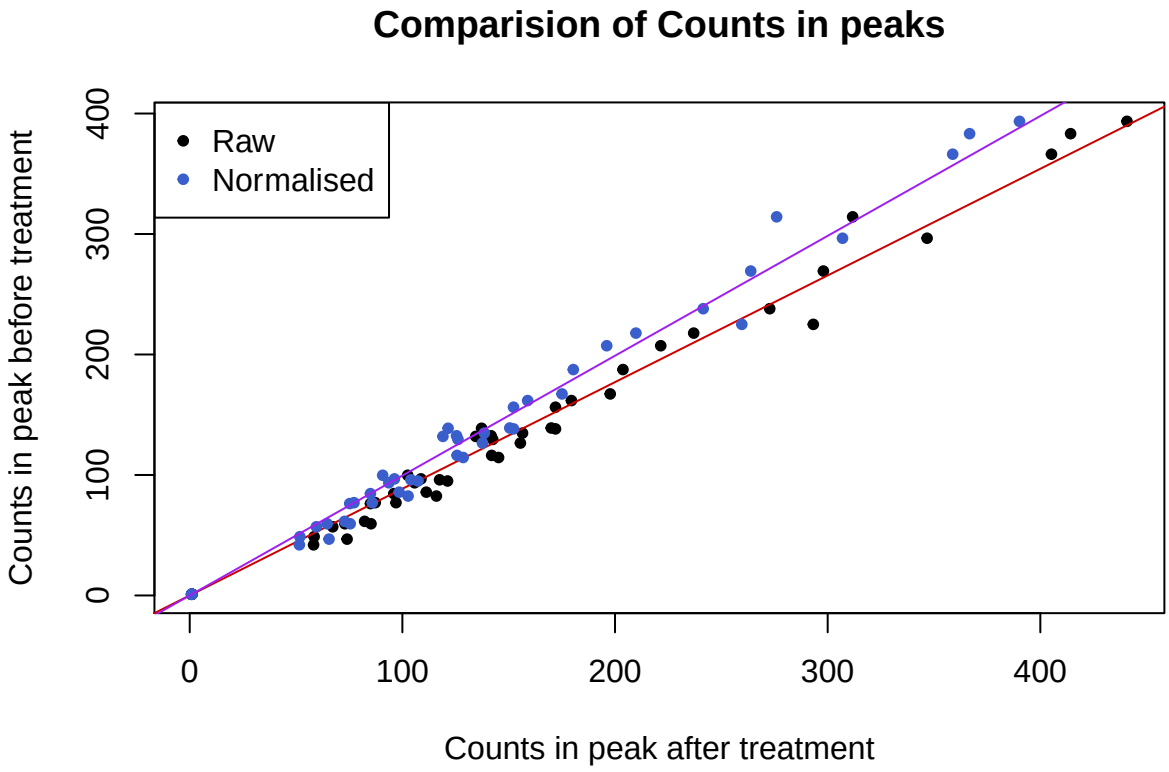
To normalise the data, we need to count the control and treated samples separately. This uses the original information we provided at the start of the script to split the control samples into two matrices. For convenience, we also record the names of the samples relating to each condition.

```
#Get counts for the treated control samples.
jg.controlCountsTreated<-jg.getControlCounts(jg.controlPeakset,
                                             jg.controlSampleSheet,
                                             jg.treatedCondition )
#Repeat for the untreated/control samples
jg.controlCountsUntreated<-jg.getControlCounts(jg.controlPeakset,
                                               jg.controlSampleSheet,
                                               jg.untreatedCondition)

#Get the sample names for replicates that represent the two conditions.
jg.untreatedNames <- names(jg.controlCountsUntreated)
jg.treatedNames   <- names(jg.controlCountsTreated)
```

Next we generate a normalization coefficient from the data. Typically this is visualised with the included plot function but the step is not required; it can be calculated directly from the data.

```
jg.plotNormalization(jg.controlCountsTreated,
                     jg.controlCountsUntreated)
```




```
## rowMeans(jg.controlCountsTreated)
##                          0.8853258
```

```
#Calculate the normalisation coefficent
jg.coefficient<-jg.getNormalizationCoefficient(jg.controlCountsTreated,
                                               jg.controlCountsUntreated)
```

To reinsert the data into DiffBind, we calculate a correction factor. This is essential as DiffBind will try to normalise our data, this correction factor ensures that our normalisation coefficent is applied correctly.

```
setwd(system.file("extdata",package="BrundleData"))
jg.correctionFactor<-jg.getCorrectionFactor(jg.experimentSampleSheet,
                                            jg.treatedNames,
                                            jg.untreatedNames)
```

We then apply the normalisation coefficent and correction factor to the treated samples.

```
jg.experimentPeaksetNormalised<-jg.applyNormalisation(jg.experimentPeakset,
                                                      jg.coefficient,
                                                      jg.correctionFactor,
                                                      jg.treatedNames)
```

For convience we return the data to DiffBind (using a modified version from https://github.com/andrewholding/flypeaks/tree/master/Diffbind) and use DiffBind to analyse the data. We could then go on to generate a DiffBind report. As this is the analysis of chromosome 22 (and control chromosome 4) only, we get only a small number of sites; nonetheless, the proceedure documented here will work for much larger datasets.

```
jg.dba <- DiffBind:::pv.resetCounts(dbaExperiment,
                                    jg.experimentPeaksetNormalised)

dba.analysis<-dba.analyze(jg.dba)
```

```
## converting counts to integer mode

## gene-wise dispersion estimates

## mean-dispersion relationship

## Warning in lfproc(x, y, weights = weights, cens = cens, base = base, geth =
## geth, : Estimated rdf < 1.0; not estimating variance

## final dispersion estimates
```

```
dba.plotMA(dba.analysis,bSmooth=FALSE,bFlip = TRUE)
```