Brundle Example

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/BrundleDate")
```

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

```
URL <- "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 both refer to the same data (BAM files), but one provides BED files for the CTCF peaks only while the other provides BED files for only the ER regions. These were generated with the script in the appendix at the end of the mark down.

The CTCF regions 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.

```
#Set up the initial varible
jg.controlMinOverlap      <- 5
jg.controlSampleSheet     <-
    system.file("extdata", "samplesheet_SLX14438_hs_CTCF_DBA.csv", package =
                    "Brundle")
jg.experimentSampleSheet  <-
    system.file("extdata", "samplesheet_SLX14438_hs_ER_DBA.csv", package =
                    "Brundle")
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)
```

```
## 1a MCF7 ERCTCF none  1 macs
## 1b MCF7 ERCTCF Fulvestrant  1 macs
## 2a MCF7 ERCTCF none  2 macs
## 2b MCF7 ERCTCF Fulvestrant  2 macs
## 3b MCF7 ERCTCF Fulvestrant  3 macs
## 3a MCF7 ERCTCF none  3 macs
```

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

```
## 1a MCF7 ERCTCF none  1 bed
## 1b MCF7 ERCTCF Fulvestrant  1 bed
## 2a MCF7 ERCTCF none  2 bed
## 2b MCF7 ERCTCF Fulvestrant  2 bed
## 3b MCF7 ERCTCF Fulvestrant  3 bed
## 3a MCF7 ERCTCF none  3 bed
```

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.

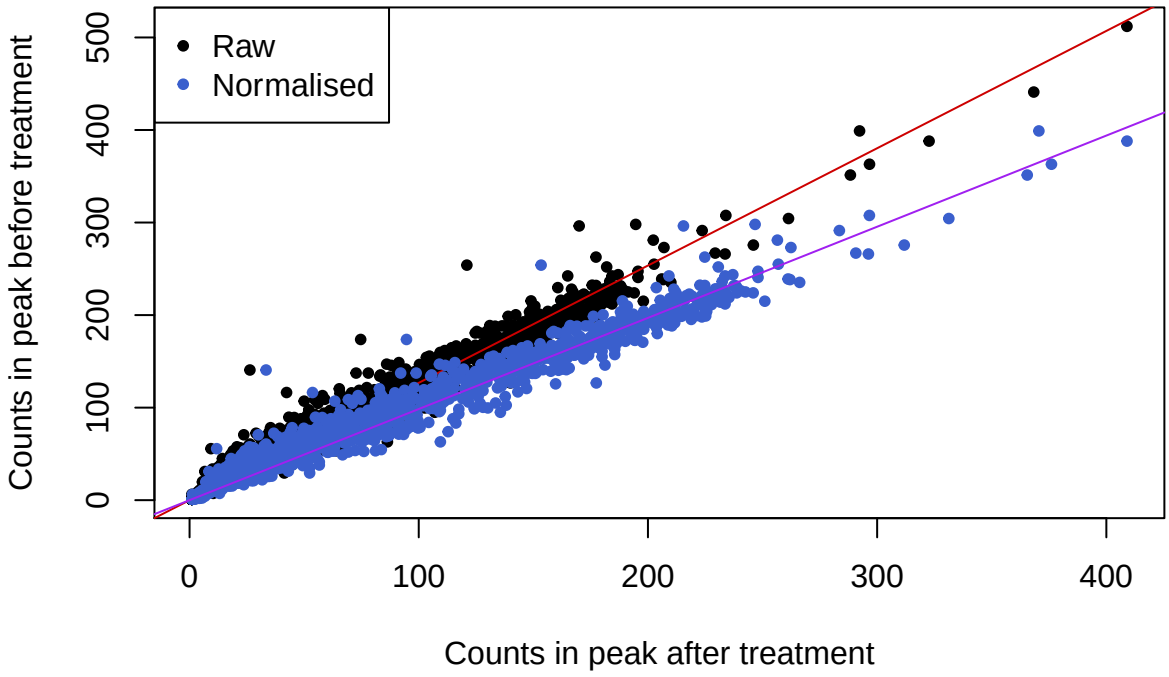
```
setwd(system.file("extdata",package="BrundleData"))

#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)
##                             1.267618
```

```
#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.

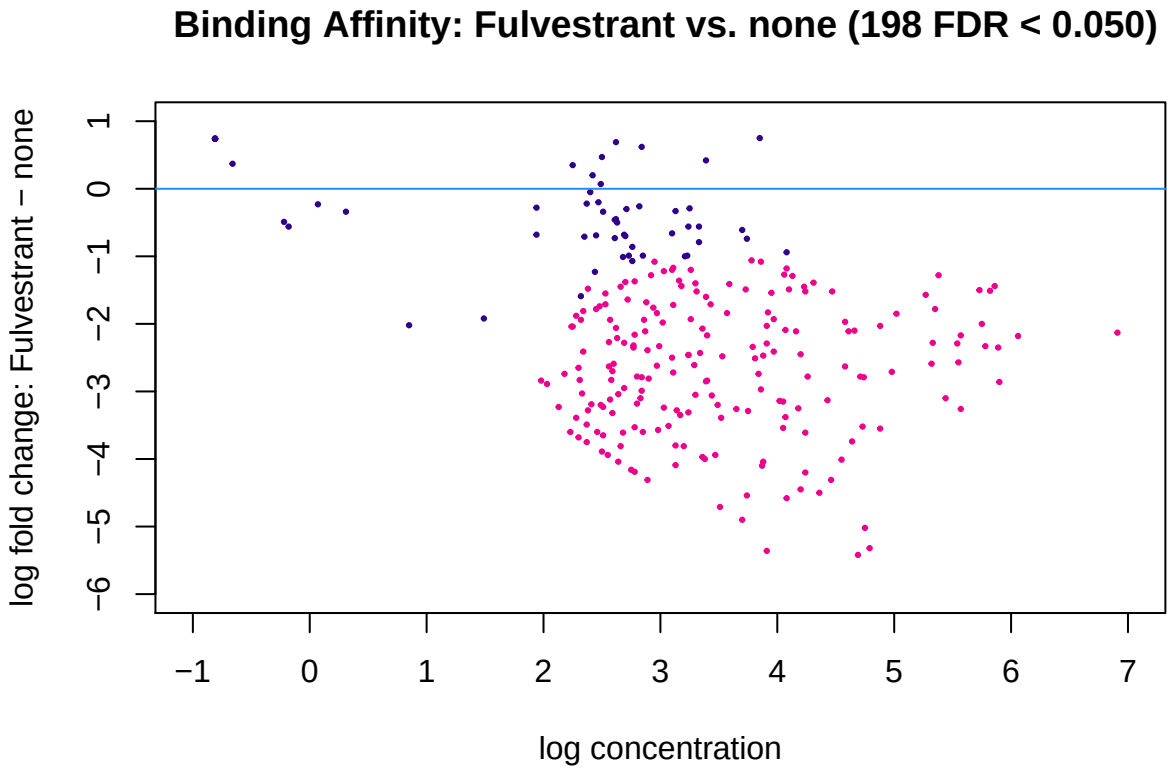
```
jg.experimentPeaksetNormalised<-jg.applyNormalisation(jg.experimentPeakset,
                                                      jg.coefficient,
                                                      jg.correctionFactor,
                                                      jg.treatedNames)
```

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 only, we get only a small number of differentially bound 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
## final dispersion estimates
dba.plotMA(dba.analysis,bSmooth=FALSE,bFlip = TRUE)
```



Appendix

Peaks were called for each BAM file as normal using macs2 then seperated in ER and CTCF peaks using the following script.

```
#This script was run in a directory containing the output
#from macs2 (ie. peak files) for all of BAM files.

#Merge the peaks called by macs2 on the aligned reads for two
#CTCF ChIP-seq experiments. One is untreated and one is treated
#with fulvestrant.

cat  SLX-14438.D701_D504.HKN27BBXX.s_8.r_1.fq.gz.bam_peaks.xls    \
   SLX-14438.D702_D503.HKN27BBXX.s_8.r_1.fq.gz.bam_peaks.xls > CTCF_merged.bed

#Remove comments from start of macs2 peak file because we have two copies.
#One from each of the merged files.
grep -v \# CTCF_merged.bed  | grep -v start > CTCF_merged_filtered.bed

#Sort the bed
```

```
sort -k1,1 -k2,2n CTCF_merged_filtered.bed > CTCF_merged_sorted_filtered.bed

#Make a union of any overlapping peak. This removes duplicates from peaks
#called in both experiemnts.
bedtools merge -i CTCF_merged_sorted_filtered.bed  >CTCF_union.bed

#Now we have a CTCF set of concensus peaks from our two experiements
#we use them to filter our peak files for each of the other samples.

mkdir CTCF

for f in *.narrowPeak
do
    echo $f
    bedtools intersect -a $f -b CTCF_union.bed  > CTCF/$f
done

mkdir ER
for f in *.narrowPeak
do
    echo $f
    bedtools subtract -b CTCF_union.bed -a $f -A > ER/$f
done
```