Filtering low expressing genes in RNAseq time course experiment

Christopher Conley, Bertrand Perroud, & Richard Michelmore

August 5, 2014

1 Getting Started

First load the R library "noleaven", which contains the functions we will need. If you have not yet installed the package, in the terminal enter: R CMD INSTALL noleaven/. Assuming you have a recent version of R, the package should install.

```
library(noleaven)
```

2 Data simulation and the fundamental data structure

In this experiment, we will have 4 genes, 5 time points, and 4 replicates.

```
REPNUM <- 4 #four replicates
TPT <- 5 #five time points
N <- 1e4  #number of  genes
```

First we generate count data for each gene according to a negative binomial distribution and assemble it into a matrix of count values which we will filter. We need to index into the count matrix also using the featIdxList. The indexing generator assumes all genes have the same number of replciates.

```
set.seed(42)
geneRator <- function(i, nbn, tpt) {
  nbsize <- sample(seq_len(500), tpt)
  nbprob <- runif(tpt)
  res <- vapply(seq.int(nbsize),
        FUN = function(i)
          rnbinom(n=nbn, size = nbsize[i], prob = nbprob[i]),
        FUN.VALUE = vector(mode = "numeric", length = nbn))
  rownames(res) <- rep(paste("feature", i, sep = ""), nbn)
  colnames(res) <- paste(seq.int(tpt), "tpt", sep = "")
  res
}
```

The data structure representation has the feature names as the row names and the time points as the column names. This is important for preserving feature labels after filtering.

```
featCounts <- do.call(rbind,lapply(1:N, function(i) geneRator(i,REPNUM,TPT)))
featIdxList <- makeFeatIdxList(N, REPNUM)
head(featCounts)
```

```
##          1tpt 2tpt 3tpt 4tpt 5tpt
## feature1  408  186  952  198  128
## feature1  463  183  833  215  134
## feature1  498  157  857  215  137
## feature1  422  205 1002  213  145
## feature2   68  108  315    1  234
## feature2   76  122  333    4  246
```

```
head(featIdxList)
```

```
## [[1]]
## [1] 1 2 3 4
##
## [[2]]
## [1] 5 6 7 8
##
## [[3]]
## [1]  9 10 11 12
##
## [[4]]
## [1] 13 14 15 16
##
## [[5]]
## [1] 17 18 19 20
##
## [[6]]
## [1] 21 22 23 24
```

2.1 Accomodating your data to this structure

Its likely your data is not stored natively in the format described above. We have provided a convenience function to accomdate the native format to the required structure. Suppose for a given genotype, you have 3 time points and four replicates (equal reps for each time point). If that information is represented as a data frame with each row as a feature and the columns arranged as { feature name, (time1,rep1), . . . , (time1,rep4), (time2,rep1), . . . (time2,rep4), . . . }, then the function *wide2LongFormat* will transform the data appropriately. Here is a simple example.

```
filename <- "~/Downloads/GPC_all_raw_counts_08_21_2013.txt"
wideData <- read.table(file = filename,
    header = TRUE, sep = "\t")
#first two features
wideData[1:2,]
```

```
##                           Contig Start  End WW.H.1 WW.H.2 WW.H.3 WW.H.4 WW.12.1 WW.12.2
## 1 1AL_v2-ab-k71_contig_1000404  2845 3179      0      1      0      0       0       1
## 2 1AL_v2-ab-k71_contig_1004132  1818 3808     34    134     85    140      40     135
##   WW.12.3 WW.12.4 WW.22.1 WW.22.2 WW.22.3 WW.22.4
## 1       0       1       0       0       0       0
## 2      90     309      39     105     103     117
```

```
#remove 2nd/3rd column
wideData <- wideData[,-c(2,3)]
wideData$Contig <- as.character(wideData$Contig)
#subset data
subWide <- wideData[1:100,]
REPNUMWIDE <- 4
TPTNUMLONG <- 3
subLong <- wide2LongFormat(subWide, REPNUMWIDE,TPTNUMLONG)
subLong[1:8,]
```

```
##                               t1  t2  t3
## 1AL_v2-ab-k71_contig_1000404   0   0   0
## 1AL_v2-ab-k71_contig_1000404   1   1   0
## 1AL_v2-ab-k71_contig_1000404   0   0   0
## 1AL_v2-ab-k71_contig_1000404   0   1   0
## 1AL_v2-ab-k71_contig_1004132  34  40  39
## 1AL_v2-ab-k71_contig_1004132 134 135 105
## 1AL_v2-ab-k71_contig_1004132  85  90 103
## 1AL_v2-ab-k71_contig_1004132 140 309 117
```

3 Filtering Criterion

The filtering criteria requires that a gene has at least minCount read counts for at least minRep replicates and for at least minTpt time points. In the following example, we choose a high filter for illustrative purposes and return a logical vector indicating which of the corresponding features were kept.

```
#first check
minCount <- 300; minRep <- 4; minTpt <- 1;
result1 <- checkFeats(featIdxList, featCounts,
                      minCount, minRep, minTpt)
head(result1)
```

```
## [1]  TRUE  TRUE FALSE  TRUE  TRUE  TRUE
```

4 Determing filtering cutoffs

We visualise the proportion of features kept to choose adequate filtering cutoffs. You will need the following packages to visualise the results.

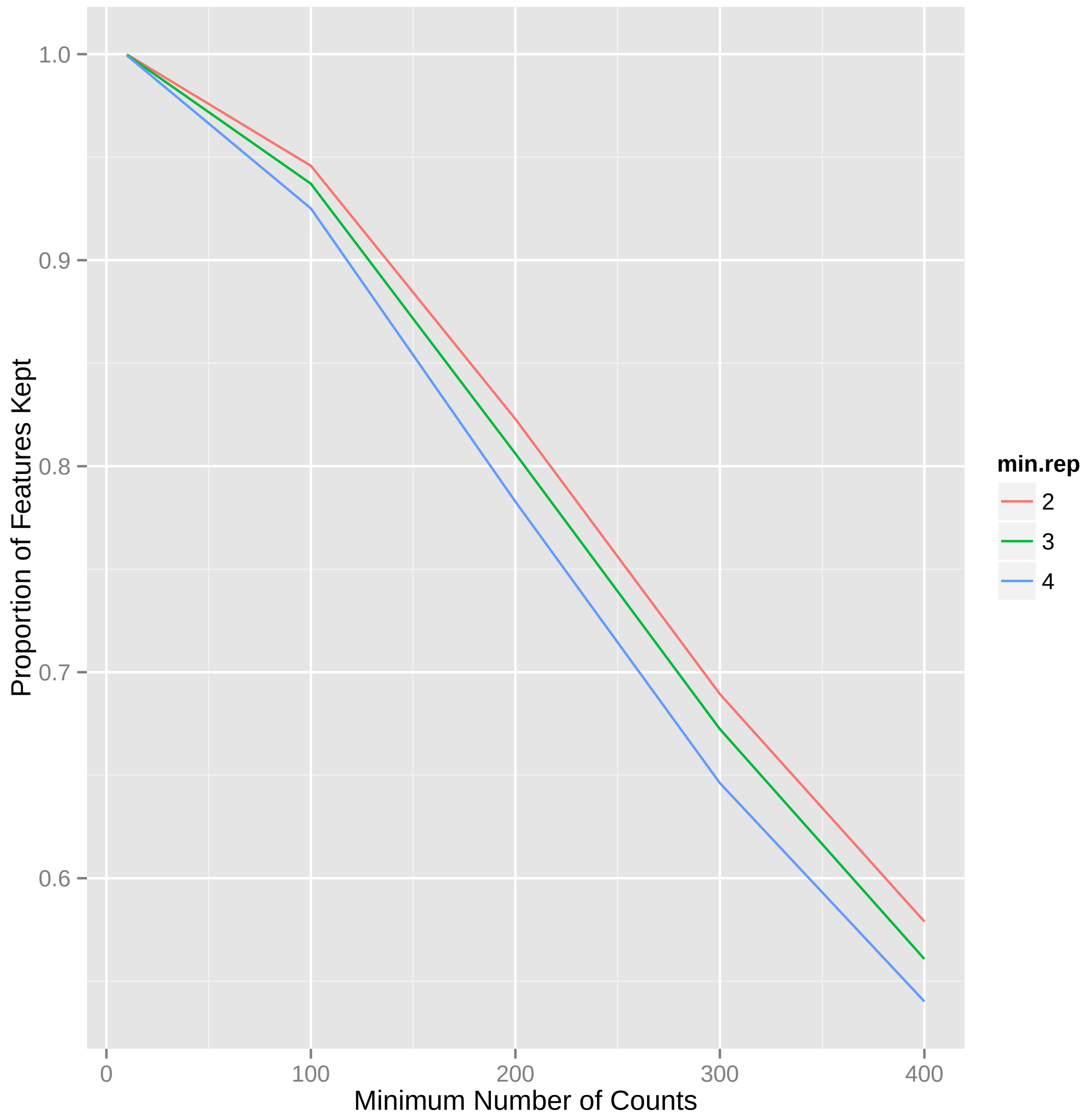
```
library(ggplot2)
library(reshape2)
```

If you do not have these R packages use the command *install.packages(c("ggplot2", "reshape2"))* to install. The following function produces a figure of proportion of features kept according to a range of minCount and minRep values specified by the user. It requires that minCount and minRep be vectors and that minTpt be a scalar (single value).

```
viewFilterReduction <- function(featIdxList, featCounts,
    minCount, minRep, minTpt) {
  require(ggplot2)
  require(reshape2)
  N <- length(featIdxList);
  result <- sapply(minCount, function(mc)  {
      sapply(minRep, function(mr) {
        featIdxKeep <- checkFeats(featIdxList, featCounts, mc, mr, minTpt);
        return(sum(featIdxKeep)/N);
        })
  })
  rownames(result) <- minRep
  colnames(result) <- minCount
  mpf <- melt(result)
  names(mpf) <- c("min.rep", "min.count", "percent.kept")
  mpf$min.rep <- as.factor(mpf$min.rep)
  return(ggplot(data=mpf, aes(x=min.count, y = percent.kept, colour  = min.rep)) +
    geom_line() + labs(x="Minimum Number of Counts", y = "Proportion of Features Kept"))
}
```

Here we specify a range of values for minCount and minRep to see their filtering effect on this simulated data set.

```
#determine the filtering criteria
minCount <- c(10, 100,200,300,400); minRep <- 2:4; minTpt <- 2;
viewFilterReduction(featIdxList, featCounts,
                    minCount, minRep, minTpt)
```



Once you determine suitable parameters, you can capture the filtered data with the function, filterFeats(...). In this instance, the minCount might be a bit higher than usual and is not intended as a default. The choice of how much should be filtered out is up to the investigators.

```
minCount <- 200; minRep <- 3; minTpt <- 1;
featCountsFiltered <- filterFeats(featIdxList, featCounts,
                                  minCount, minRep, minTpt)
```

5 Extending filtering to multiple experimental treatments

But the previous example assumed that there was only one experimental condition. If there are more than one experimental condition, we can take the union of features across all conditions that pass their respective criteria. As illustrated by the example below, each condition may have its own paramtric cut offs of minCount, minRep, and minTpt. The order in which the parameters are recieved correspond to the order of the condition list.

```
featCountsCondition1 <- do.call(rbind,lapply(1:N,
    function(i)
    geneRator(i,REPNUM,TPT)))
featCountsCondition2 <- do.call(rbind,lapply(1:N,
    function(i)
    geneRator(i,REPNUM,TPT)))
featCountsCondition3 <- do.call(rbind,lapply(1:N,
    function(i)
    geneRator(i,REPNUM,TPT)))

featCountsU <- list(featCountsCondition1,
  featCountsCondition2,
  featCountsCondition3)
minCountU <- c(200,300,400)
minRepU <- c(2,3,2)
minTptU <- c(1,2,2)
featCountsFilteredList <- unionFilterFeats(featIdxList,
    featCountsU, minCountU, minRepU, minTptU);
```

Here we now have the feature observations for each condition that passed the union of their respective criterion. That is, if it passed the criterion for one of the conditions, it is kept in all conditions.

```
head(featCountsFilteredList[[1]])

##          1tpt 2tpt 3tpt 4tpt 5tpt
## feature1  182  104 1321   90  158
## feature1  227   96 1363   90  147
## feature1  187   94 1349   88  164
## feature1  229   88 1331   84  157
## feature2  182 5201  617 2921    4
## feature2  183 5330  763 3226    4

head(featCountsFilteredList[[2]])

##          1tpt 2tpt 3tpt 4tpt  5tpt
## feature1   20   19    8   46 22744
## feature1   17   19    6   30 20438
## feature1   20   16   18   23 21438
## feature1   23   21   13   28 20238
## feature2 2613   93  205  384   144
## feature2 2573  116  158  301   152

head(featCountsFilteredList[[3]])

##          1tpt  2tpt 3tpt 4tpt 5tpt
## feature1   30   926   30   48  959
## feature1   35   843   33   52  862
## feature1   11   911   22   45  758
## feature1   34   911   37   47  768
## feature2  924 11161   51   38    8
## feature2 1052 10523   26   39   10
```

Note that the filtering paramters in this example might seem odd, but is intended to illustrate the flexibility one has in filtering out according to each experimental condition.

6 Session Info

```
sessionInfo()
```

```
## R version 3.0.1 (2013-05-16)
## Platform: x86_64-pc-linux-gnu (64-bit)
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C                 LC_TIME=en_US.UTF-8
##  [4] LC_COLLATE=en_US.UTF-8     LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=C                 LC_NAME=C                  LC_ADDRESS=C
## [10] LC_TELEPHONE=C             LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] reshape2_1.4  ggplot2_1.0.0 knitr_1.6     noleaven_1.0
##
## loaded via a namespace (and not attached):
##  [1] colorspace_1.2-4 digest_0.6.4     evaluate_0.5.5   formatR_0.10     grid_3.0.1
##  [6] gtable_0.1.2     highr_0.3        labeling_0.2     MASS_7.3-27      munsell_0.4.2
## [11] plyr_1.8.1       proto_0.3-10     Rcpp_0.11.2      scales_0.2.4     stringr_0.6.2
## [16] tools_3.0.1
```