HTSFilter: Independent data-based filtering for replicated high-throughput sequencing experiments

Andrea Rau, Mélina Gallopin, Gilles Celeux, Florence Jaffrézic

Modified: October 17, 2012. Compiled: October 17, 2012

Abstract

This vignette illustrates the use of the *HTSFilter* package to filter replicated data from high-throughput sequencing experiments (e.g., RNA sequencing data) for four different data classes: matrix, *CountDataSet* (the S4 class associated with the *DESeq* package), *DGEList* (the S3 class associated with the *edgeR* package), and *SeqExpressionSet* (the S4 class associated with the *EDASeq* package).

Contents

1 Introduction

High-throughput sequencing (HTS) data, such as RNA-sequencing (RNA-seq) data, are increasingly used to conduct differential analyses, in which gene-by-gene statistical tests

are performed in order to identify genes whose expression levels show systematic covariation with a particular condition, such as a treatment or phenotype of interest. Because hypothesis tests are performed for gene-by-gene differential analyses, the obtained p-values must be adjusted to correct for multiple testing. However, procedures to adjust p-values to control the number of detected false positives often lead to a loss of power to detect truly differentially expressed (DE) genes due to the large number of hypothesis tests performed. To reduce the impact of such procedures, independent data filters are often used to identify and remove genes that appear to generate an uninformative signal [2]; this in turn moderates the correction needed to adjust for multiple testing. For independent filtering methods for microarray data, see for example the *genefilter* Bioconductor package [5].

The *HTSFilter* package implements a novel independent filtering procedure based on the calculation of a similarity index among biological replicates for read counts arising from replicated high-throughput sequencing data; see [8] for additional details. This technique provides an intuitive data-driven way to filter high-throughput sequencing data and to effectively remove genes with low, constant expression levels without incorrectly removing those that would otherwise have been identified as DE. The three fundamental assumptions of the filter implemented in the *HTSFilter* package are as follows:

1. Biological replicates are present for each experimental condition,

2. Data can be appropriately normalized (scaled) to correct for systematic inter-sample biases, and

3. Within a given condition, after controlling for inter-sample biases, each gene exhibits similar expression levels among replicates.

For more information about between-sample normalization strategies, see [4]; in particular, strategies for normalizing data with differences in library size and composition may be found in [1] and [10], and strategies for normalizing data exhibiting sample-specific biases due to GC content may be found in [9] and [6]. Within the *HTSFilter* package, the Trimmed Means of M-values (TMM) [10] and DESeq [1] normalization strategies may be used prior to calculating an appropriate data-based filter. If an alternative normalization strategy is needed or desired, the normalization may be applied prior to filtering the data with `normalization="none"` in the `HTSFilter` function; see Section 6.2 for an example.

The *HTSFilter* package is able to accommodate unnormalized or normalized replicated count data in the form of a matrix (in which each row corresponds to a biological feature and each column to a biological sample), a *CountDataSet* (the S4 class associated with the *DESeq* package), a *DGEList* (the S3 class associated with the *edgeR* package), and a *SeqExpressionSet* (the S4 class associated with the *EDASeq* package), as illustrated in the following sections.

2 Input data

For the purposes of this vignette, we make use of the gene-level counts from RNA-seq data in *Drosophila melanogastor* [3] contained in the Bioconductor package *pasilla* [7]. Briefly, the experiment aimed to study the effect of siRNA knock-down of the gene pasilla, which is thought to be involved in splicing regulation; three biological replicates are available for the knockdown and four for the untreated control. Within both conditions, samples were obtained using both single-end and paired-end sequencing. For illustrative purposes, we have ignored the sequencing protocol labels and considered only the condition labels ("treated" and "untreated"). See [3] and the documentation for the *pasilla* package [7] for additional detail.

To begin, we load the *HTSFilter* and *pasilla* packages, and attach the gene-level count data contained in `pasillaGenes`:

```
> library(HTSFilter)
> library(pasilla)
> data("pasillaGenes")

> pData(pasillaGenes)

             sizeFactor condition        type
treated1fb           NA   treated single-read
treated2fb           NA   treated  paired-end
treated3fb           NA   treated  paired-end
untreated1fb         NA untreated single-read
untreated2fb         NA untreated single-read
untreated3fb         NA untreated  paired-end
untreated4fb         NA untreated  paired-end

> dim(pasillaGenes)

Features  Samples
   14470        7
```

The unfiltered data contain 14470 genes in seven samples.

3 Matrix class

To filter high-throughput sequencing data in the form of a matrix, we first coerce the `pasillaGenes` data into matrix format, and create a vector identifying the condition labels for each of the samples. We then filter the data using the `HTSFilter` function, specifying that the number of tested thresholds be only 25 (`s.len=25`) rather than the default value of 100 to reduce computation time for this example. Note that as it is unspecified, the default normalization method is used for filtering the data, namely the Trimmed Mean of M-values (TMM) method of Robinson and Oshlack.

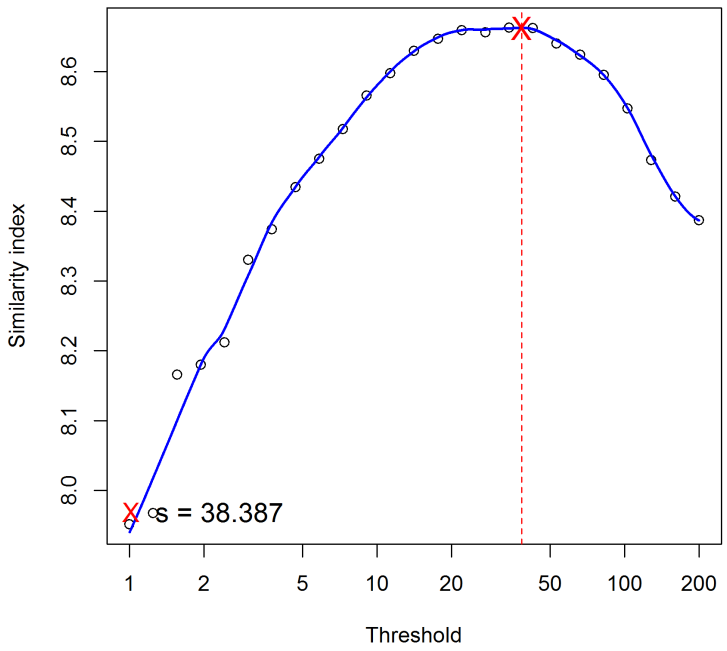

Figure 1: Global Jaccard index for the `pasillaGenes` data calculated for a variety of threshold values after TMM normalization [10], with a loess curve (blue line) superposed and data-based threshold values (red cross and red dotted line) equal to 38.387.

```
> mat <- counts(pasillaGenes)
> conds <- pData(pasillaGenes)$condition
> filter <- HTSFilter(mat, conds, s.len=25)
> mat <- filter$filteredData
> dim(mat)

[1] 7124    7

> dim(filter$removedData)

[1] 7346    7

> class(mat)

[1] "matrix"
```

The proposed filter thus removes 7346 genes from further analyses, leaving 7124 genes. We note that an important part of the filter proposed in the *HTSFilter* package is

a check of the behavior of the global similarity index calculated over a range of threshold values, and in particular, to verify that a maximum is reached for the global similarity index over the range of tested threshold values (see Figure 1). For this example, we find a data-based threshold equal to 38.387; genes with normalized values less than this threshold in all samples are filtered from subsequent analyses. Finally, we note that the filtered data are of the same class as the original data, in this case an object of class *matrix*.

4 S4 class *CountDataSet* (*DESeq* package)

To filter high-throughput sequencing data in the form of a *CountDataSet* (the class used within the *DESeq* pipeline for differential analysis), we note that `pasillaGenes` is already an object of the appropriate class. Once again, we specify that the number of tested thresholds be only 25 (`s.len=25`) rather than the default value of 100 to reduce computation time for this example. In addition, we specify that the normalization strategy to be used is `normalization="DESeq"`. Note that the filtered data are of the same class as the original data, *CountDataSet*.

```
> cds <- pasillaGenes
> filter <- HTSFilter(cds, conds, s.len=25, normalization = "DESeq")
> cds <- filter$filteredData
> dim(cds)

Features  Samples
    7156        7

> dim(filter$removedData)

[1] 7314    7

> class(cds)

[1] "CountDataSet"
attr(,"package")
[1] "DESeq"
```

As the normalization strategy used here was slightly different, the proposed filter now removes 7314 genes from further analyses, leaving 7156 genes. Again we verify the behavior of the global similarity index calculated over a range of threshold values (see Figure 2). For this example, we find a data-based threshold equal to 35.047; genes with normalized values less than this threshold in all samples are filtered from subsequent analyses. Finally, we note that the filtered data are of the same class as the original data, in this case an object of class *CountDataSet*.

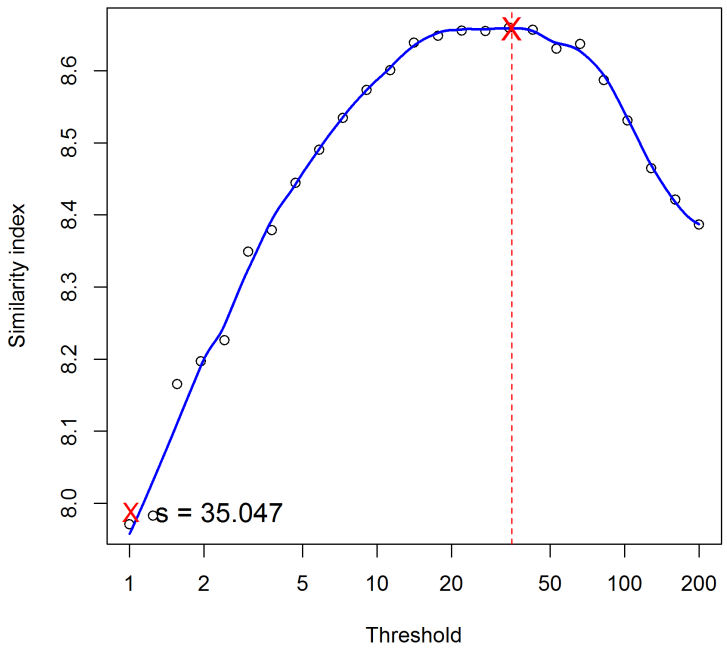

Figure 2: Global Jaccard index for the `pasillaGenes` data calculated for a variety of threshold values after DESeq normalization [1], with a loess curve (blue line) superposed and data-based threshold values (red cross and red dotted line) equal to 35.047.

Typically, the next step would be to perform a differential analysis using the *DESeq* pipeline, namely by estimating the size factors of the filtered data (`estimateSizeFactors`), estimating the per-gene and fitted dispersions (`estimateDispersions`), and performing an exact test (`nbinomTest`) to identify differentially expressed genes.

```
> ## Not run:
> ## cds <- estimateSizeFactors(cds)
> ## cds <- estimateDispersions(cds)
> ## res <- nbinomTest(cds, "treated", "untreated")
```

5 S3 class *DGEList* (*edgeR* package)

To filter high-throughput sequencing data in the form of a *DGEList* (the class used within the *edgeR* pipeline for differential analysis), we first coerce the data into the appropriate class with the function `DGEList`, where the `group` variable is set to contain a vector of condition labels for each of the samples. We then apply the filter using the `HTSFilter` function, again specifying that the number of tested thresholds be only 25

(s.len=25) rather than the default value of 100. Note that as it is unspecified, the default normalization method is used for filtering the data, namely the Trimmed Mean of M-values (TMM) method [10]. We suppress the plot of the global Jaccard index using plot = FALSE, as it is identical to that shown in Figure 1.

```
> dge <- DGEList(counts=counts(pasillaGenes), group=conds)
> filter <- HTSFilter(dge, s.len=25, plot=FALSE)
> dge <- filter$filteredData
> dim(dge)
```

```
[1] 7124    7
```

```
> dim(filter$removedData)
```

```
[1] 7346    7
```

```
> class(dge)
```

```
[1] "DGEList"
attr(,"package")
[1] "edgeR"
```

Note that the filtered data are of the same class as the original data, *DGEList*, and that library sizes have been re-estimated after filtering the data using the TMM method. Typically, the next step would be to perform a differential analysis using the *edgeR* pipeline, namely by estimating common (estimateCommonDisp) and per-gene dispersions (estimateTagwiseDisp), and performing an exact test (exactTest) to identify differentially expressed genes.

```
> ## Not run:
> ## dge <- estimateCommonDisp(dge)
> ## dge <- estimateTagwiseDisp(dge)
> ## res <- exactTest(dge)
```

6 S4 class *SeqExpressionSet* (*EDAseq* package)

6.1 Using default normalization methods

To filter high-throughput sequencing data in the form of a *SeqExpressionSet* (the class used within the *EDAseq* pipeline for exploratory analysis and normalization), we first create a new object of the appropriate class with the function newSeqExpressionSet. We then apply the filter using the HTSFilter function, again specifying that the number of tested thresholds be only 25 (s.len=25) rather than the default value of 100. Note that as it is unspecified, the default normalization method is used for filtering the data, namely the Trimmed Mean of M-values (TMM) method [10]. We suppress the plot of the global Jaccard index using plot=FALSE, as it is identical to that shown in Figure 1.

```
> ses <- newSeqExpressionSet(counts(pasillaGenes),
+         phenoData = phenoData(pasillaGenes))
> filter <- HTSFilter(ses, conds, s.len=25, plot=FALSE)
> ses <- filter$filteredData
> dim(ses)

Features  Samples
    7124        7

> dim(filter$removedData)

[1] 7346    7

> class(ses)

[1] "SeqExpressionSet"
attr(,"package")
[1] "EDASeq"
```

Note that the filtered data are of the same class as the original data, *SeqExpressionSet*; the various commands included in *EDASeq* may now be applied to this object. As an example, we produce a mean-variance plot of the counts across all samples after filtering the data (Figure 3).

```
> meanVarPlot(ses, log=T)
```

6.2 Alternative normalization using *EDAseq*

As a final example, we illustrate the use of the *HTSFilter* package with an alternative normalization strategy, namely the full quantile normalization method in the *EDASeq* package; such a step may useful when the TMM or DESeq normalization methods are not appropriate for a given dataset. Once again, we create a new object of the appropriate class with the function `newSeqExpressionSet` and normalize data using the `betweenLaneNormalization` function (with `which="full"`) in *EDASeq*.

```
> ses <- newSeqExpressionSet(counts(pasillaGenes),
+         phenoData=phenoData(pasillaGenes))
> ses.norm <- betweenLaneNormalization(ses, which="full")
```

Subsequently, `HTSFilter` is applied to the normalized data (again using `s.len=25`), and the normalization method is set to `normalization="none"`. We may then make use of the `on` vector in the results, which identifies filtered and unfiltered genes (respectively) with 0 and 1, to identify rows in the original data matrix to be retained.

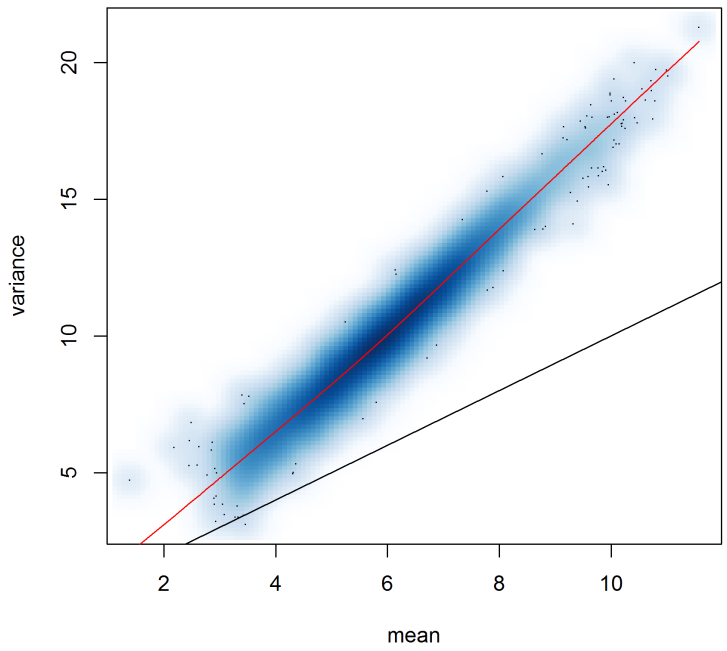

Figure 3: Mean-variance relationship across all seven samples of the `pasillaGenes` data after data filtering, where the black line corresponds to the Poisson distribution (equal mean and variance) and the red curve is a loess fit. Plot obtained using the `meanVarPlot` function in the *EDASeq* package.

```
> filter <- HTSFilter(ses.norm, conds, s.len=25, normalization="none",
+           plot=FALSE)
> head(filter$on)
```

```
FBgn0000003 FBgn0000008 FBgn0000014 FBgn0000015 FBgn0000017
          0           1           0           0           1
FBgn0000018
          1
```

```
> ses <- newSeqExpressionSet(counts(pasillaGenes)[which(filter$on == 1),],
+      phenoData=phenoData(pasillaGenes))
```

7 Session Info

```
> sessionInfo()
```

```
R version 2.15.1 (2012-06-22)
Platform: x86_64-pc-mingw32/x64 (64-bit)

locale:
[1] LC_COLLATE=English_United States.1252
[2] LC_CTYPE=English_United States.1252
[3] LC_MONETARY=English_United States.1252
[4] LC_NUMERIC=C
[5] LC_TIME=English_United States.1252

attached base packages:
[1] stats     graphics  grDevices utils     datasets
[6] methods   base

other attached packages:
 [1] pasilla_0.2.13           DEXSeq_1.2.1
 [3] HTSFilter_0.1.0          EDASeq_1.2.0
 [5] R.oo_1.9.9               aroma.light_1.24.0
 [7] R.methodsS3_1.4.2        ShortRead_1.14.4
 [9] latticeExtra_0.6-24      RColorBrewer_1.0-5
[11] Rsamtools_1.8.6          lattice_0.20-10
[13] Biostrings_2.24.1        GenomicRanges_1.8.13
[15] IRanges_1.14.4           DESeq_1.8.3
[17] locfit_1.5-8             edgeR_2.6.12
[19] limma_3.12.3             Biobase_2.16.0
[21] BiocGenerics_0.2.0       SweaveListingUtils_0.5.5
[23] startupmsg_0.7.2

loaded via a namespace (and not attached):
 [1] annotate_1.34.1      AnnotationDbi_1.18.4
 [3] biomaRt_2.12.0       bitops_1.0-4.1
 [5] DBI_0.2-5            genefilter_1.38.0
 [7] geneplotter_1.34.0   grid_2.15.1
 [9] hwriter_1.3          KernSmooth_2.23-8
[11] plyr_1.7.1           RCurl_1.95-0.1
[13] RSQLite_0.11.2       splines_2.15.1
[15] statmod_1.4.16       stats4_2.15.1
[17] stringr_0.6.1        survival_2.36-14
[19] tools_2.15.1         XML_3.95-0.1
[21] xtable_1.7-0         zlibbioc_1.2.0
```

References

[1] Simon Anders and Wolfgang Huber. Differential expression analysis for sequence count data. *Genome Biology*, 11(R106):1–28, 2010.

[2] Richard Bourgon, Robert Gentleman, and Wolfgang Huber. Independent filtering increases detection power for high-throughput experiments. *PNAS*, 107(21):9546–9551, 2010.

[3] A. N. Brooks, L. Yang, M. O. Duff, K. D. Hansen, J. W. Park, S. Dudoit, S. E. Brenner, and B. R. Graveley. Conservation of an RNA regulatory map between Drosophila and mammals. *Genome Research*, 21(2):193–202, 2011.

[4] Marie-Agnès Dillies, Andrea Rau, Julie Aubert, Christelle Hennequet-Antier, Marine Jeanmougin, Nicolas Servant, Céline Keime, Guillemette Marot, David Castel, Jordi Estelle, Gregory Guernec, Bernd Jagla, Luc Jouneau, Denis Laloë, Caroline Le Gall, Brigitte Schaëffer, Stéphane Le Crom, and Florence Jaffrézic. A comprehensive evaluation of normalization methods for Illumina high-throughput RNA sequencing data analysis. *Briefings in Bioinformatics*, (to appear), 2012.

[5] R. Gentleman, V. Carey, W. Huber, and F. Hahne. *genefilter: methods for filtering genes from microarray experiments*. R package version 1.38.0.

[6] Kasper D. Hansen, Rafael A. Irizarry, and Zhijin Wu. Removing technical variability in RNA-seq data using conditional quantile normalization. *Biostatistics*, (in press)(227), 2012.

[7] W. Huber and A. Reyes. *pasilla: Data package with per-exon and per-gene read counts of RNA-seq samples of Pasilla knock-down by Brooks et al., Genome Research 2011.*

[8] A. Rau, M. Gallopin, G. Celeux, and F. Jaffrézic. Independent data-based filtering for replicated high-throughput sequencing experiments. *(submitted)*, 2012.

[9] Davide Risso, Katja Schwartz, Gavin Sherlock, and Sandrine Dudoit. GC-content normalization for RNA-seq data. *BMC Bioinformatics*, 12(480), 2011.

[10] Mark D. Robinson and Alicia Oshlack. A scaling normalization method for differential expression analysis of RNA-seq data. *Genome Biology*, 11(R25), 2010.