

bulk-RNAseq with edgeR-Differentially Gene Expression (DEG) Analysis Notes

Dinçer Kılıç

2024-15-01

Following this **guide**.

1.4 Quick start

edgeR offers many variants on analyses. The glm approach is more popular than the classic approach as it offers great flexibilities. There are two testing methods under the glm framework: likelihood ratio tests and quasi-likelihood F-tests. The quasi-likelihood method is highly recommended for differential expression analyses of bulk RNA-seq data as it gives stricter error rate control by accounting for the uncertainty in dispersion estimation. The likelihood ratio test can be useful in some special cases such as single cell RNA-seq and datasets with no replicates. The details of these methods are described in Chapter 2.

A typical edgeR analysis might look like the following. Here we assume there are four RNASeq libraries in two groups, and the counts are stored in a tab-delimited text file, with gene symbols in a column called Symbol.

```
x <- read.delim("TableOfCounts.txt",row.names="Symbol")
group <- factor(c(1,1,2,2))
y <- DGEList(counts=x,group=group)
keep <- filterByExpr(y)
y <- y[keep,,keep.lib.sizes=FALSE]
y <- normLibSizes(y)
design <- model.matrix(~group)
y <- estimateDisp(y,design)
```

To perform quasi-likelihood F-tests:

```
fit <- glmQLFit(y,design)
qlf <- glmQLFTest(fit,coef=2)
topTags(qlf)
```

To perform likelihood ratio tests:

```
fit <- glmFit(y,design)
lrt <- glmLRT(fit,coef=2)
topTags(lrt)
```

2.2 Aligning reads to a genome

The first step in an RNA-seq analysis is usually to align the raw sequence reads to a reference genome, although there are many variations on this process.

We find the Subread-featureCounts pipeline [25, 26] to be very fast and effective for this purpose, but the STAR-featureCounts, STAR-hits and Bowtie-TopHat-hits pipelines are also popular. Subread and featureCounts are particularly convenient because they are implemented in the Bioconductor R package Rsubread [27].

2.3 Producing a table of read counts

edgeR works on a table of read counts, with rows corresponding to genes and columns to independent libraries. The counts represent the total number of reads aligning to each gene (or other genomic locus).

Such counts can be produced from aligned reads by a variety of short read software tools. We find the featureCounts function of the Rsubread package [26, 27] to be particularly effective and convenient, but other tools are available such as findOverlaps in the GenomicRanges package or the Python software htseq-counts.

Reads can be counted in a number of ways. When conducting gene-level analyses, the counts could be for reads mapping anywhere in the genomic span of the gene or the counts could be for exons only. We usually count reads that overlap any exon for the given gene, including the UTR as part of the first exon [26].

For data from pooled shRNA-seq or CRISPR-Cas9 genetic screens, the processAmplicons function [11] can be used to obtain counts directly from FASTQ files. Note that edgeR is designed to work with actual read counts. We do not recommend that predicted transcript abundances are input to edgeR in place of actual counts.

2.4 Reading the counts from a file

If the table of counts has been written to a file, then the first step in any analysis will usually be to read these counts into an R session.

If the count data is contained in a single tab-delimited or comma-separated text file with multiple columns, one for each sample, then the simplest method is usually to read the file into R using one of the standard R read functions such as read.delim.

If the counts for different samples are stored in separate files, then the files have to be read separately and collated together. The edgeR function readDGE is provided to do this. Files need to contain two columns, one for the counts and one for a gene identifier.

2.5 Pseudoalignment and quasi-mapping

The kallisto and Salmon software tools align sequence reads to the transcriptome instead of the reference genome and produce estimated counts per transcript. Output from either tool can be input to edgeR via the tximport package, which produces gene-level estimated counts and an associated edgeR offset matrix. Alternatively, kallisto or Salmon output can be read directly into edgeR using the catchSalmon and catchKallisto functions if the intention is to conduct a transcript-level analysis.

2.6 The DGEList data class

edgeR stores data in a simple list-based data object called a DGEList. This type of object is easy to use because it can be manipulated like any list in R. The function readDGE makes a DGEList object directly. If the table of counts is already available as a matrix or a data.frame, x say, then a DGEList object can be made by

```
y <- DGEList(counts=x)
```

A grouping factor can be added at the same time:

```
group <- c(1,1,2,2)
y <- DGEList(counts=x, group=group)
```

The main components of an `DGEList` object are a matrix counts containing the integer counts, a `data.frame` samples containing information about the samples or libraries, and an optional `data.frame` genes containing annotation for the genes or genomic features. The `data.frame` samples contains a column `lib.size` for the library size or sequencing depth for each sample. If not specified by the user, the library sizes will be computed from the column sums of the counts. For classic `edgeR` the `data.frame` samples must also contain a column `group`, identifying the group membership of each sample.

2.7 Filtering

Genes with very low counts across all libraries provide little evidence for differential expression. In the biological point of view, a gene must be expressed at some minimal level before it is likely to be translated into a protein or to be biologically important. In addition, the pronounced discreteness of these counts interferes with some of the statistical approximations that are used later in the pipeline. These genes should be filtered out prior to further analysis.

As a rule of thumb, genes are dropped if they can't possibly be expressed in all the samples for any of the conditions. Users can set their own definition of genes being expressed. Usually a gene is required to have a count of 5-10 in a library to be considered expressed in that library. Users should also filter with count-per-million (CPM) rather than filtering on the counts directly, as the latter does not account for differences in library sizes between samples.

Here is a simple example. Suppose the sample information of a `DGEList` object `y` is shown as follows:

```
y$samples
      group lib.size norm.factors
Sample1  1   10880519      1
Sample2  1    9314747      1
Sample3  1   11959792      1
Sample4  2    7460595      1
Sample5  2    6714958      1
```

We filter out lowly expressed genes using the following commands:

```
keep <- filterByExpr(y)
y <- y[keep, , keep.lib.sizes=FALSE]
```

The `filterByExpr` function keeps rows that have worthwhile counts in a minimum number of samples (two samples in this case because the smallest group size is two). The function accesses the group factor contained in `y` in order to compute the minimum group size, but the filtering is performed independently of which sample belongs to which group so that no bias is introduced. It is recommended to recalculate the library sizes of the `DGEList` object after the filtering, although the downstream analysis is robust to whether this is done or not.

The group factor or the experimental design matrix can also be given directly to the `filterByExpr` function by

```
keep <- filterByExpr(y, group=group)
```

if not already set in the `DGEList` object. More generally, `filterByExpr` can be used with any design matrix:

```
keep <- filterByExpr(y, design)
```

The filtering should be based on the grouping factors or treatment factors that will be involved in the differential expression test tested for, rather than on blocking variables that are not of scientific interest in

themselves. For example, consider a paired comparison experiment in which the same treatment regimes applied to each of a number of subjects or patients:

```
design <- model.matrix(~ Patient + Treatment)
```

In this design, Patient is included in the design matrix to correct for baseline differences between the Patients, but we will not be testing for differential expression between the Patients. The filtering should therefore be based solely Treatment rather than on Patient, i.e

```
keep <- filterByExpr(y, group=Treatment)
```

rather than

```
keep <- filterByExpr(y, design)
```

2.8 Normalization

2.8.1 Normalization is only necessary for sample-specific effects

edgeR is concerned with differential expression analysis rather than with the quantification of expression levels. It is concerned with relative changes in expression levels between conditions, but not directly with estimating absolute expression levels. This greatly simplifies the technical influences that need to be taken into account, because any technical factor that is unrelated to the experimental conditions should cancel out of any differential expression analysis. For example, read counts can generally be expected to be proportional to length as well as to expression for any transcript, but edgeR does not generally need to adjust for gene length because gene length has the same relative influence on the read counts for each RNA sample. For this reason, normalization issues arise only to the extent that technical factors have sample-specific effects.

2.8.2 Sequencing depth

The most obvious technical factor that affects the read counts, other than gene expression levels, is the sequencing depth of each RNA sample. edgeR adjusts any differential expression analysis for varying sequencing depths as represented by differing library sizes. This is part of the basic modeling procedure and flows automatically into fold-change or p-value calculations. It is always present, and doesn't require any user intervention.

2.8.3 Effective library sizes

The second most important technical influence on differential expression is one that is less obvious. RNA-seq provides a measure of the relative abundance of each gene in each RNA sample, but does not provide any measure of the total RNA output on a per-cell basis. In other words, RNA-seq measure relative expression rather than absolute expression. This becomes important for differential expression analyses when a small number of genes are very highly expressed in some samples but not in others. If a small proportion of highly expressed genes consume a substantial proportion of the total library size for a particular sample, this will cause the remaining genes to be under-sampled for that sample. Unless this effect is adjusted for, the remaining genes may falsely appear to be down-regulated in that sample [41].

The `normLibSizes` function normalizes the library sizes in such a way to minimize the log-fold changes between the samples for most genes. The default method for computing these scale factors uses a trimmed mean of M-values (TMM) between each pair of samples [41]. We call the product of the original library size and the scaling factor the effective library size, i.e., the normalized library size. The effective library size replaces the original library size in all downstream analyses.

TMM is recommended for most RNA-Seq data where the majority (more than half) of the genes are believed not differentially expressed between any pair of the samples. If `y` is a `DGEList` object, then the following

commands perform the TMM normalization and display the normalization factors.

```
y <- normLibSizes(y)
y$samples

      group lib.size norm.factors
Sample1 1    10880519 1.17
Sample2 1    9314747 0.86
Sample3 1    11959792 1.32
Sample4 2    7460595 0.91
Sample5 2    6714958 0.83
```

The set of all normalization factors for a DGEList multiply to unity, ensuring that the geometric mean of the effective library sizes is the same as the geometric mean of the original library sizes. A normalization factor below one indicates that a small number of high count genes are monopolizing the sequencing, causing the counts for other genes to be lower than would be usual given the library size. As a result, the library size will be scaled down, analogous to scaling the counts upwards in that library. Conversely, a factor above one scales up the library size, analogous to downscaling the counts.

2.9 Negative binomial models

Watch [this](#).

2.11.3 Testing for DE genes

As a brief example, consider a situation in which are three treatment groups, each with two replicates, and the researcher wants to make pairwise comparisons between them. A QL model representing the study design can be fitted to the data with commands such as:

```
group <- factor(c(1,1,2,2,3,3))
design <- model.matrix(~group)
fit <- glmQLFit(y, design)
```

The fit has three parameters. The first is the baseline level of group 1. The second and third are the 2 vs 1 and 3 vs 1 differences.

To compare 2 vs 1:

```
qlf.2vs1 <- glmQLFTest(fit, coef=2)
topTags(qlf.2vs1)
```

To compare 3 vs 1:

```
qlf.3vs1 <- glmQLFTest(fit, coef=3)
```

To compare 3 vs 2:

```
qlf.3vs2 <- glmQLFTest(fit, contrast=c(0,-1,1))
```

To find genes different between any of the three groups:

```
qlf <- glmQLFTest(fit, coef=2:3)
topTags(qlf)
```

Alternatively, one can perform likelihood ratio test to test for differential expression. The testing can be done by using the functions `glmFit()` and `glmLRT()`. To apply the likelihood ratio test to the above example and compare 2 vs 1:

```
fit <- glmFit(y, design)
lrt.2vs1 <- glmLRT(fit, coef=2)
topTags(lrt.2vs1)
```