DRIMSeq: Dirichlet-multinomial framework for differential splicing and sQTL analyses in RNA-seq.

Malgorzata Nowicka[*], Mark Robinson

October 21, 2015

This vignette describes version 0.3.1 of the *DRIMSeq* package.

Contents

1 Overview of Dirichlet-multinomial model 2

2 Hints for DRIMSeq pipeline 2

3 Differential splicing analysis work-flow 2
- 3.1 Preparing table of counts . . . 3
- 3.2 Example data . . . 3
- 3.3 Differential splicing analysis with DRIMSeq package . . . 3
  - 3.3.1 Loading pasilla data into R . . . 3
  - 3.3.2 Filtering . . . 6
  - 3.3.3 Dispersion estimation . . . 7
  - 3.3.4 Proportions estimation . . . 9
  - 3.3.5 Testing for differential splicing . . . 10

4 sQTL analysis work-flow 12
- 4.1 Example data . . . 12
- 4.2 Genotypes preprocessing . . . 13
- 4.3 sQTL analysis with DRIMSeq package . . . 13

APPENDIX 14

A Session information 14

B References 14

[*]gosia.nowicka@uzh.ch

1 Overview of Dirichlet-multinomial model

In the *DRIMSeq* package we implemented a Dirichlet-multinomial framework that can be used for modeling various multivariate count data with the interest in finding the instances where the ratios of observed features are different between the experimental conditions. Such model can be applied, for example, in differential splicing or sQTL analysis where the multivariate features are transcripts or exons of a gene. Depending on the type of counts that are used in the analysis, you can test for differential splicing at the level of transcirpt or exon ratio changes. *fixme: Another applications*. The implementation of Dirichlet-multinomial model in *DRIMSeq* package is customised for differential splicing and sQTL analyses, but the data object *dmDSdata* can contain different types of counts, and thus, other types of multivariate differential analysis between groups can be performed.

The statistical details of *DRIMSeq* are presented in our paper *fixme: citation to DRIMSeq paper*. In short, the method consists of three statistical steps. First, we use the profile likelihood to estimate the dispersion, i.e., the variability of feature ratios between samples (replicates) within conditions. Dispersion is needed in order to find the significant changes in features ratios between conditions which should be sufficiently stronger than the changes/variability withing conditions. Second, we use the maximum likelihood to estimate the full model (estimated in every group/condition separetely) and null model (estimated from all data) proportions and its corresponding likelihoods. Finally, we use the likelihood ratio statistic to to test for the differences between feature proportions in different groups to identify the differentially spliced genes (differential splicing analysis) or the sQTLs (sQTL analysis).

2 Hints for DRIMSeq pipeline

In this vignette, we present how one could perform differential splicing analysis and sQTL analysis with *DRIMSeq* package. We use small subsets of data so that you can run the whole pipelines within few minutes in *R* on a single core computer.

Both pipelines consist of the initial steps where objects containing the data needed for the analysis are created and then filtered. Functions used for this purpose, such as `dmDSdata` or `dmSQTLdata` and `dmFilter`, have some parameters (like `counts`, `gene_id`, `min_samps_gene_expr`, etc.) for which no defalut values are predefined. These parameters must be specified by user in order to proceed with the pipeline.

Functions `dmDispersion`, `dmFit` and `dmTest`, which perform the actual statistical analysis described above, require that only one parameter `x` containinig the data is specified by user. These functions have many other parameters available for twicking, but they do have default values, which were chosen based on many real data analysis, assigned.

Some of the steps are quite time consumming, especially the dispersion estimation, where proportions of each gene are refitted for different dispersion parameters. Thus, we reccomend to, if possible, always increase the number of workers in `BPPARAM`.

3 Differential splicing analysis work-flow

3.1 Preparing table of counts

Different methods (reference to Charlotte's paper) require different steps like alignement

Exonic bin counts: HTSeq, featureCounts, Transcript expected counts: kallisto, RSEM, BitSeq

3.2 Example data

To demonstrate the usage of *DRIMSeq* in differentiall splicing analysis, we will use a *pasilla* data set produced by Brooks et al. [1]. The aim of their study was to identify exons that are regulated by *pasilla* protein, the Drosophila melanogaster ortholog of mammalian NOVA1 and NOVA2 which are one of the well studied splicing factors. In their RNA-seq experiment the libraries were prepared from 7 biologically independent samples: 4 control samples and 3 samples in which *pasilla* was knocked-down. The libraries were sequenced on Illumina Genome Analyzer II using single-end and paired-end sequencing and different read lengths. The RNA-seq data can be downloaded from the NCBI's Gene Expression Omnibus (GEO) under the accession number GSE18508. In the examples below, we use a subset of the *HTSeq* counts available in *pasilla* package, where you can find all the steps needed, for processing the GEO data, to get a table with exonic bin counts.

3.3 Differential splicing analysis with DRIMSeq package

In oder to do the analysis, we have to create a *dmDSdata* object, which contains the feature counts and information about grouping samples into conditions. With each step of the pipeline, additional elements are added to this object. So that, at the end of the analysis, the object contains results from all the steps, such as dispersion estimates, proportions estimates, likelihood ratio statistics, p-values, adjusted p-values. As new elements are added, the object also changes its name *dmDSdata* $->$ *dmDSdispersion* $->$ *dmDSfit* $->$ *dmDStest*, but every following one inherits the slots and the methods available for the previous one.

3.3.1 Loading pasilla data into R

The counts obtained from *HTSeq* are saved in separate text files for each sample. We want to put them together into one data frame.

```
library(pasilla)

data_dir  <- system.file("extdata", package="pasilla")
count_files <- list.files(data_dir, pattern="fb.txt$", full.names=TRUE)
count_files
## [1] "/Users/gosia/Library/R/3.2/library/pasilla/extdata/treated1fb.txt"
## [2] "/Users/gosia/Library/R/3.2/library/pasilla/extdata/treated2fb.txt"
## [3] "/Users/gosia/Library/R/3.2/library/pasilla/extdata/treated3fb.txt"
## [4] "/Users/gosia/Library/R/3.2/library/pasilla/extdata/untreated1fb.txt"
## [5] "/Users/gosia/Library/R/3.2/library/pasilla/extdata/untreated2fb.txt"
## [6] "/Users/gosia/Library/R/3.2/library/pasilla/extdata/untreated3fb.txt"
## [7] "/Users/gosia/Library/R/3.2/library/pasilla/extdata/untreated4fb.txt"
## Read the HTSeq files into R
htseq_list <- lapply(1:length(count_files), function(i){
  htseq <- read.table(count_files[i], header = FALSE, as.is = TRUE)
  colnames(htseq) <- c("group_id", gsub("fb.txt", "", strsplit(count_files[i],
    "extdata/")[[1]][2]))
  return(htseq)
})

## Merge them into one data frame
htseq_counts <- Reduce(function(...) merge(..., by = "group_id", all=TRUE,
  sort = FALSE), htseq_list)

## Remove the summary elements
tail(htseq_counts)
##             group_id treated1 treated2 treated3 untreated1 untreated2 untreated3
## 70462 FBgn0261575:001        2        0        0          0          1          0
## 70463 FBgn0261575:002       10        1        3          5          7          1
## 70464      _ambiguous     1317        0        0       1096       1144          0
## 70465          _empty 11173758 16686814 16057634    7756745   12150698   13990064
## 70466       _lowaqual 67756093        0        0   22017200   42760359          0
## 70467     _notaligned        0        0        0          0          0          0
##       untreated4
## 70462          0
## 70463          0
## 70464          0
## 70465   14248758
## 70466          0
## 70467          0
htseq_counts <- htseq_counts[!grepl(pattern = "_", htseq_counts$group_id), ]
```

The exon bin IDs produced by *HTSeq* consist of the gene IDs and the bin number separated by colon. To create the *dmDSdata* object, we need a vector of gene IDs and a vector of feature IDs, here bin IDs.

```
group_split <- limma::strsplit2(htseq_counts[, 1], ":")
```

Finally, load the *DRIMSeq* package.

```
suppressPackageStartupMessages(library(DRIMSeq))
```

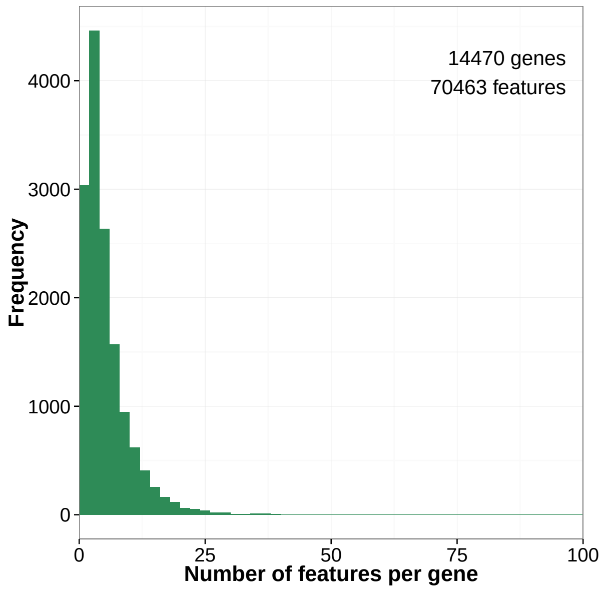
Create a `dmDSdata` object (saved as variable `d`), which contains counts and information about samples such as sample IDs and a variable `group` defining the experimental groups/conditions. When printing variable `d`, you can see its class, size and which accessor methods can be applied. For `dmDSdata` object, there are two methods which return data frames with counts and samples.

```
d <- dmDSdata(counts = htseq_counts[, -1], gene_id = group_split[, 1],
  feature_id = group_split[, 2], sample_id = colnames(htseq_counts)[-1],
  group = gsub("[1-4]", "", colnames(htseq_counts)[-1]))
d
## An object of class dmDSdata
## with 14470 genes and 7 samples
## * data accessors: counts(), samples()
```

```
head(counts(d), 4)
##       gene_id feature_id treated1 treated2 treated3 untreated1 untreated2 untreated3
## 1 FBgn0000003        001        0        0        1          0          0          0
## 2 FBgn0000008        001        0        0        0          0          0          0
## 3 FBgn0000008        002        0        0        0          0          0          1
## 4 FBgn0000008        003        0        1        0          1          1          1
##   untreated4
## 1          0
## 2          0
## 3          0
## 4          0
head(samples(d), 4)
##    sample_id     group
## 1   treated1   treated
## 2   treated2   treated
## 3   treated3   treated
## 4 untreated1 untreated
```

You can also make a data summary plot, which is a histogram of the number of features per gene. There are genes, that have nearly 100 of exonic bins.

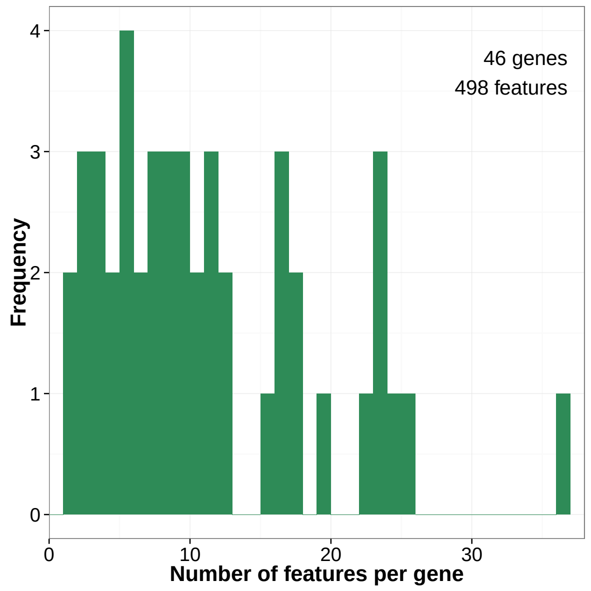
```
plotData(d)
```



To make the analysis runnable within this vignette, we want to keep only a small subset of genes, which is defined in the following file.

```
genes_subset = readLines(file.path(data_dir, "geneIDsinsubset.txt"))
d <- d[names(d) %in% genes_subset, ]
d
## An object of class dmDSdata
## with 46 genes and 7 samples
## * data accessors: counts(), samples()
```

```
plotData(d)
```



After subsetting, d contains counts for 46 genes.

3.3.2 Filtering

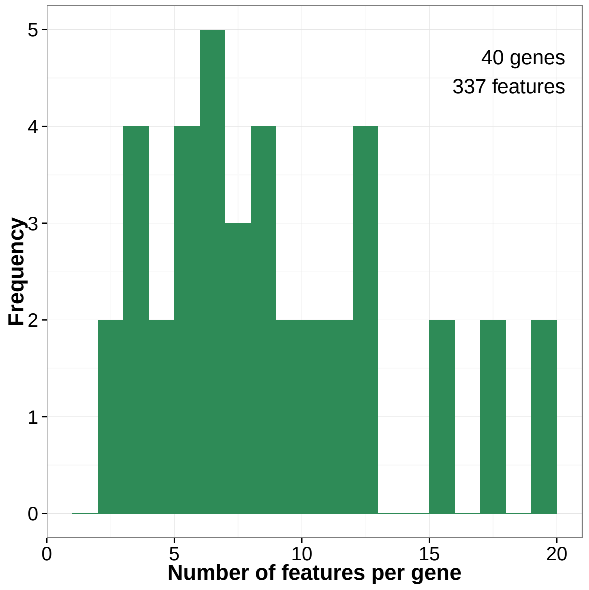
Filtering of genes and features with low expression improves the performance of Dirichlet-multinomial model. Genes may have many features that are expressed very lowly (so that even if there was any differential expression, it would not be biologically significant) or not expressed at all. Removing such features from the analysis has two effects. First, it reduces the number of parameters (proportions) to estimate. Second, improves the estimation of dispersion *fixme: Do simulation*. However, too stringent filtering may remove genes and features where the differential changes happen, and, in consequence, lead to loss of power.

Thus, finding an optimal filter is a challenge. In differential splicing analysis, we reccomend to adjust the filtering parameters according to the number of replicates per condition. At the feature level, we suggest using `min_samps_feature_prop` equal to the minimal number of replicates in any of the conditions. For example, in *pasilla* assay with 3 knock-down and 4 control samples, we would set this parameters to 3. Like this, we allow a situation where a feature (here, an exonic bin) is expressed in one condition but may not be expressed at all in another one. The level of feature expression is controlled by `min_feature_prop`. Our default value is set up to 0.01, which means that only the features with ratio of at least 1% in 3 samples are kept.

Filtering at the gene level ensures that the observed feature ratios have some minimal reliability. Although, Dirichlet-multinomial model works on feature counts, and not on feature ratios, which means that it gives more confidence to the ratios based on 100 versus 500 reads than 1 versus 5, a minimal filtering for the gene expression removes the genes with mostly zero counts and reduces the number of tests in multiple test correction. The default value that we propose is `min_samps_gene_expr = 3` and `min_gene_expr = 1`, which means that only genes that are expressed at the level of 1 cpm in at least 3 samples are kept for the downstream analysis.

```
# Check what is the minimal number of replicates per condition
table(samples(d)$group)
```

```
##
##   treated untreated
##         3         4
d <- dmFilter(d, min_samps_gene_expr = 3, min_gene_expr = 1,
  min_samps_feature_prop = 3, min_feature_prop = 0.01, max_features = Inf)
plotData(d)
```



3.3.3 Dispersion estimation

Ideally, we would like to get accurate dispersion estimates for every gene, which is problematic when analysing small sample size data sets because dispersion estimates become inaccurate when the sample size decreases especially for lowly expressed genes. As an alternative, we could assume that all the genes have the same dispersion and based on all the data we could calculate a common dispersion, which is less inaccurate. However, such assumption is quite unrealistic. Moderated dispersion is a trade-off between gene-wise and common dispersion. The moderated estimates are a weighted combination of common and individual dispersion and we reccomend such approach when analysing small sample size data sets.

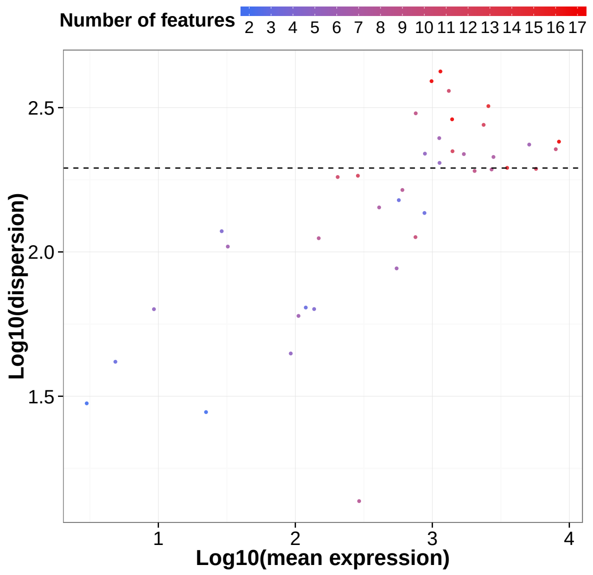
At this step three values may be calculated: mean expression of genes, common dispersion and gene-wise dispersions. In the default setting all of them are computed and common dispersion is used as an initial value in the grid appraoch to estimate moderated gene-wise dispersions, which are shrunk toward the common dispersion.

This step of our pipeline is the most time consuming, thus consider increasing the number of workers in `BPPARAM`.

```
d <- dmDispersion(d, BPPARAM = BiocParallel::MulticoreParam(workers = 1))
## * Calculating mean gene expression..
## Took  9.373  seconds.
## * Estimating common dispersion..
## Took  361.81  seconds.
## !  Using common_dispersion = 195.27 as disp_init !
## * Estimating genewise dispersion..
## Took  16.775  seconds.
d
## An object of class dmDSdispersion
## with 40 genes and 7 samples
## * data accessors: counts(), samples()
##   mean_expression(), common_dispersion(), genewise_dispersion()
head(mean_expression(d))
##       gene_id mean_expression
## 1 FBgn0000256       1318.2857
## 2 FBgn0000578       3515.0000
## 3 FBgn0002921       7962.2857
## 4 FBgn0003089          2.0000
## 5 FBgn0010226        104.4286
## 6 FBgn0010280       2367.1429
common_dispersion(d)
## [1] 195.2712
head(genewise_dispersion(d))
##       gene_id genewise_dispersion
## 1 FBgn0000256           361.38987
## 2 FBgn0000578           195.58597
## 3 FBgn0002921           226.94313
## 4 FBgn0003089            29.87752
## 5 FBgn0010226            60.02510
## 6 FBgn0010280           275.62462
```

To inspect the behaviour of dispersion estimates, you can plot them against the mean gene expression. Here, the effect of shrinking is not so well visible because our data set is very small. To see how it works on real data sets, go to our paper *fixme: citation to DRIMSeq paper*.

```
plotDispersion(d)
```



3.3.4 Proportions estimation

In this step, we estimate the full model proportions, meaning, that transcript or exon proportions are estimated for each condition separetely. You can access this estimates and the corresponding statisctics, such as log-likelihoods, with `proportions` and `statistics` functions, respectively.

```
d <- dmFit(d, BPPARAM = BiocParallel::MulticoreParam(workers = 1))
## * Fitting full model..
## Took  9.49  seconds.
d
## An object of class dmDSfit
## with 40 genes and 7 samples
## * data accessors: counts(), samples()
##   mean_expression(), common_dispersion(), genewise_dispersion()
##   proportions(), statistics()
head(proportions(d), 3)
##       gene_id feature_id    treated  untreated
## 1 FBgn0000256        001 0.04754941 0.04804901
## 2 FBgn0000256        002 0.09192056 0.07878007
## 3 FBgn0000256        003 0.26135466 0.26411140
head(statistics(d), 3)
##       gene_id lik_treated lik_untreated
## 1 FBgn0000256   -6866.356     -12197.61
## 2 FBgn0000578  -22743.567     -37572.26
## 3 FBgn0002921  -42450.168     -48057.55
```

3.3.5 Testing for differential splicing

Calling the `dmTest` function results in two things. First, null model proportions, i.e., feature ratios based on pooled (no grouping into conditions) counts, are estimated. Second, likelihood ratio statistic is used to test for the difference between feature proportions in different groups to identify the differentially spliced genes.

By default, a parameter `compared_groups` in `dmTest` equals to `1:nlevels(samples(x)$group)`, which means that we test for differences in splicing between any of the groups specified in `samples(d)$group`. In *pasilla* example, there are only two conditions, and there is only one comparison that can be done. In the case where the gouping variable has more that two levels, you could be intereseted in the pair-wise comparisons, which you can specify with `compared_groups` parameter.

Now, if you call `proportions` or `statistics` function, results of null estimation are added to the previous data frames.

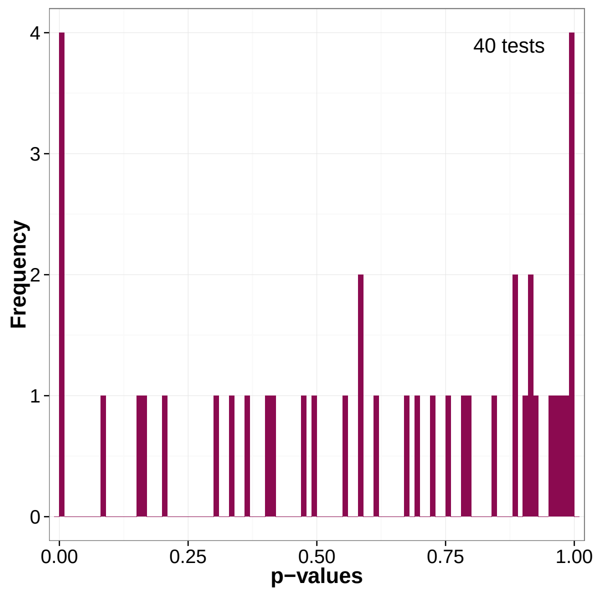
```
d <- dmTest(d, BPPARAM = BiocParallel::MulticoreParam(workers = 1))

## Running comparison  between groups:  treated, untreated

## * Fitting null model..
## Took  9.337  seconds.
## * Calculating likelihood ratio statistics..
## Took  0.001582146  seconds.
d
## An object of class dmDStest
## with 40 genes and 7 samples
## * data accessors: counts(), samples()
##   mean_expression(), common_dispersion(), genewise_dispersion()
##   proportions(), statistics()
##   results()
head(proportions(d), 3)
##       gene_id feature_id    treated  untreated       null
## 1 FBgn0000256        001 0.04754941 0.04804901 0.04813549
## 2 FBgn0000256        002 0.09192056 0.07878007 0.08457247
## 3 FBgn0000256        003 0.26135466 0.26411140 0.26459246
head(statistics(d), 3)
##       gene_id lik_treated lik_untreated  lik_null df
## 1 FBgn0000256   -6866.356     -12197.61 -19079.64 10
## 2 FBgn0000578  -22743.567     -37572.26 -60318.54 14
## 3 FBgn0002921  -42450.168     -48057.55 -90509.34  9
```

To obtain the results of likelihood ratio test, you have to call the function `results`, which returns a data frame with likelihood ratio statistics, degrees of freedom, p-values and Benjamini and Hochberg adjusted p-values for each gene.

```
head(results(d), 3)
##       gene_id        lr df       pvalue  adj_pvalue
## 1 FBgn0000256 31.349649 10 0.0005135684 0.006847578
## 2 FBgn0000578  5.425912 14 0.9789583045 0.999025040
## 3 FBgn0002921  3.239646  9 0.9540310716 0.999025040
```

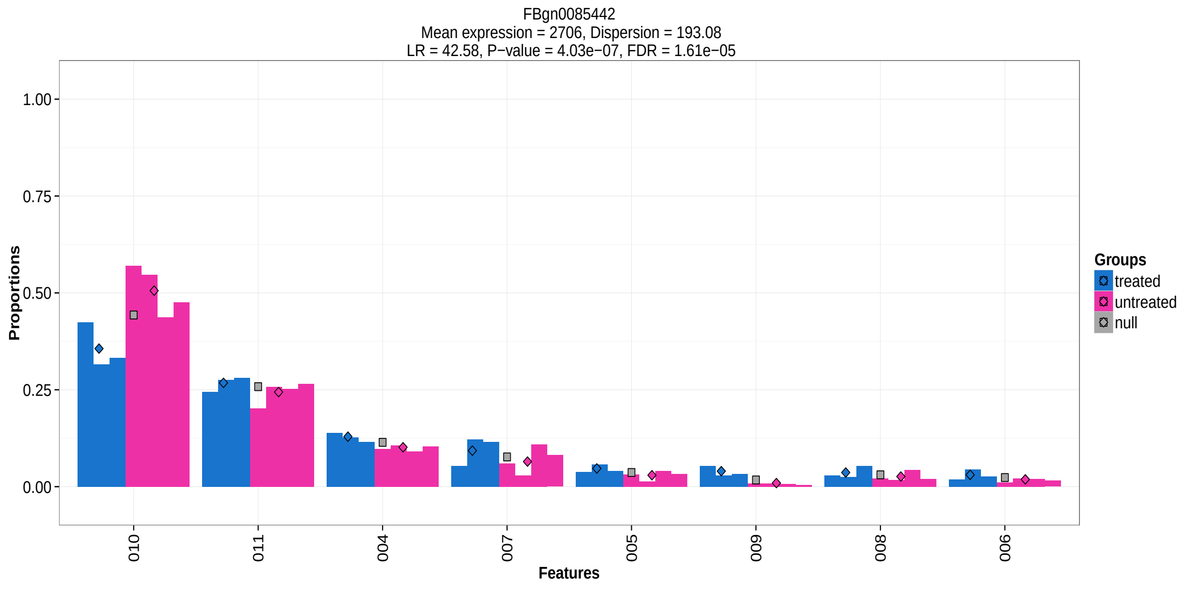
You can plot the histogram of p-values.

```
plotTest(d)
```

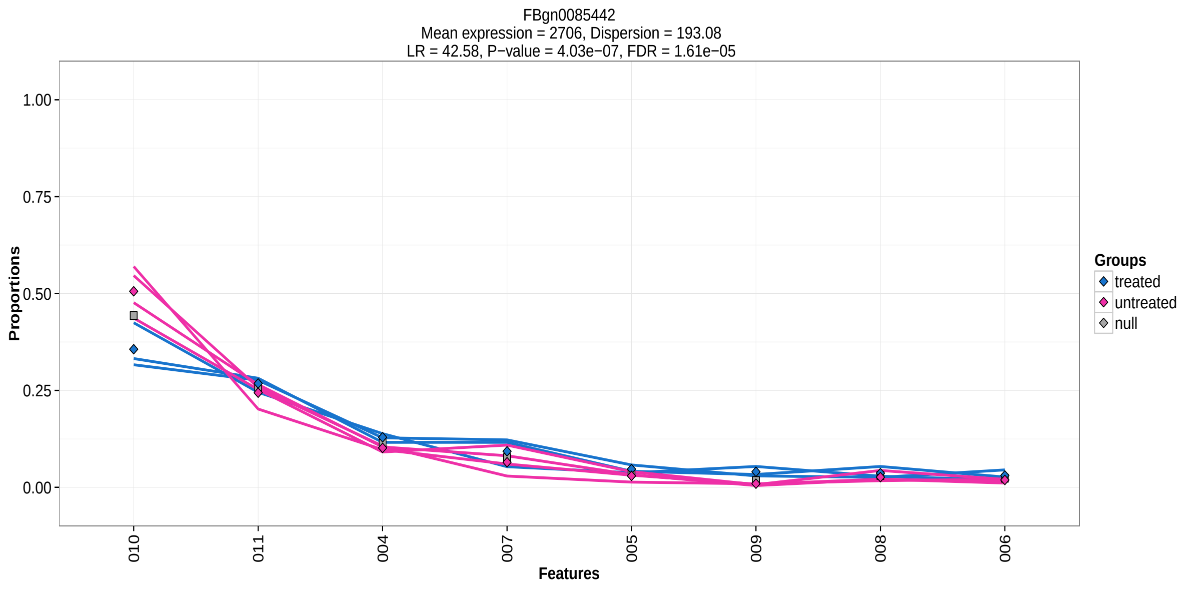


For genes of interest, you can make plots (bar plots, line plots, box plots, ribbon plots) of observed and estimated with Dirichlet-multinomial model feature ratios. Estimated proportions are marked with diamond shapes. Here, we plot the results for the top significant gene.

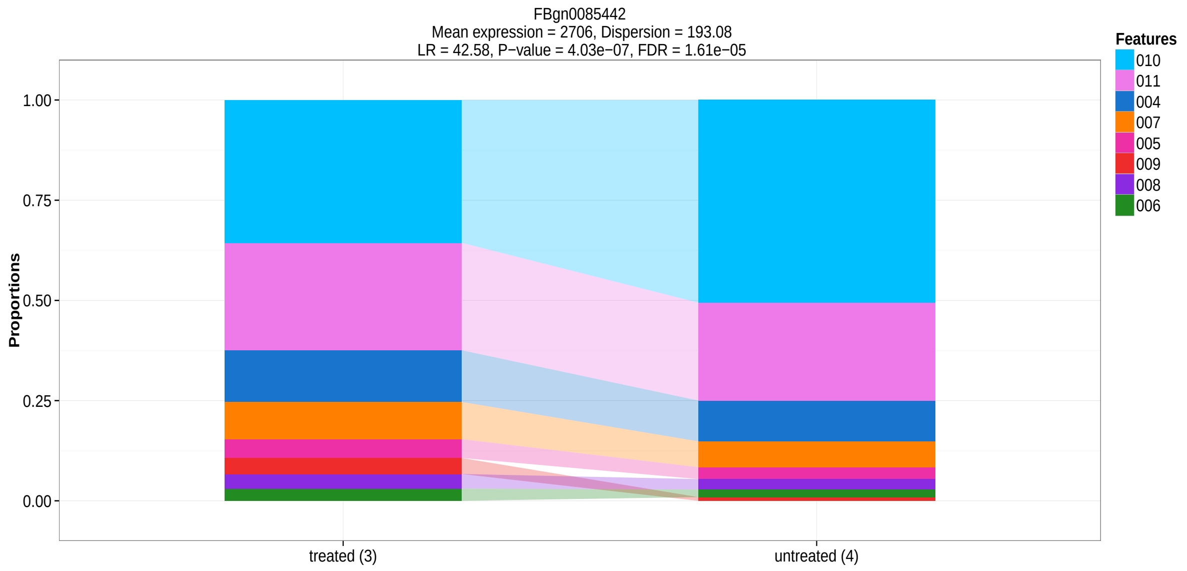
```
res <- results(d)
res <- res[order(res$pvalue, decreasing = FALSE), ]
gene_id <- res$gene_id[1]

plotFit(d, gene_id = gene_id)
## Plot gene 1: FBgn0085442
```



```
plotFit(d, gene_id = gene_id, plot_type = "lineplot")
## Plot gene 1: FBgn0085442
```



```
plotFit(d, gene_id = gene_id, plot_type = "ribbonplot")
## Plot gene 1: FBgn0085442
```



4 sQTL analysis work-flow

4.1 Example data

Data downloaded from

4.2 Genotypes preprocessing

Preprocessing steps on CSV files with genotypes

4.3 sQTL analysis with DRIMSeq package

Assuming you have counts and bi-allelic genotypes...

APPENDIX

A Session information

```
sessionInfo()
## R version 3.2.2 (2015-08-14)
## Platform: x86_64-apple-darwin13.4.0 (64-bit)
## Running under: OS X 10.10.5 (Yosemite)
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] parallel  stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] pasilla_0.8.0       DRIMSeq_0.3.1       BiocGenerics_0.14.0 knitr_1.11
## [5] colorout_1.1-1
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.1          XVector_0.8.0        magrittr_1.5         edgeR_3.10.4
##  [5] MASS_7.3-44          GenomicRanges_1.20.8 IRanges_2.2.8        munsell_0.4.2
##  [9] BiocParallel_1.2.22  colorspace_1.2-6     stringr_1.0.0        highr_0.5.1
## [13] GenomeInfoDb_1.4.3   plyr_1.8.3           tools_3.2.2          grid_3.2.2
## [17] gtable_0.1.2         lambda.r_1.1.7       futile.logger_1.4.1  digest_0.6.8
## [21] reshape2_1.4.1       ggplot2_1.0.1        formatR_1.2.1        codetools_0.2-14
## [25] S4Vectors_0.6.6      futile.options_1.0.0 evaluate_0.8         labeling_0.3
## [29] limma_3.24.15        stringi_0.5-5        scales_0.3.0         stats4_3.2.2
## [33] BiocStyle_1.6.0      proto_0.3-10
```

B References

References

[1] 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*, vol. 21, no. 2, pp. 193–202, 2011.