Example workflow for *regsplice* package

Lukas M. Weber

8 May 2016

Package version: regsplice 0.2

Contents

1 Introduction

The *regsplice* package implements statistical methods for the detection of differential exon usage (differential splicing) in RNA sequencing (RNA-seq) and microarray data sets.

The *regsplice* methods are based on the use of the lasso (L1-regularization) to improve the power of standard generalized linear models, with fast runtimes compared to other leading approaches. The statistical methodology and comparisons to other methods are described in our paper:

> Title of paper and link to bioRxiv preprint here.

1.1 Example workflow

This vignette demonstrates an example workflow for the *regsplice* package using a small simulated RNA-seq data set.

There are two options for running *regsplice*: you can run a complete workflow in one step using the wrapper function `regsplice()`; or you can run the individual functions for each step in sequence, which provides additional insight into the methodology. Both options are demonstrated below.

1.2 Data set

The data set used for the workflow consists of exon-level read counts for a subset of 100 genes from a simulated human RNA-seq data set, consisting of 6 biological samples, with 3 samples in each of 2 conditions.

The original data set is from the paper:

> Soneson et al. (2016), *Isoform prefiltering improves performance of count-based methods for analysis of differential transcript usage*, Genome Biology, available here.

Original data files from this paper, containing the simulated RNA-seq reads (FASTQ and BAM files), are available from ArrayExpress at accession code E-MTAB-3766.

Exon bin counts were generated with the Python counting scripts provided with the *DEXSeq* package, using the option to exclude exons from overlapping genes instead of aggregating them into multi-gene complexes (see Soneson et al. 2016, Supplementary Material).

For this workflow, we have selected a subset of the first 100 genes from this simulated data set. The exon-level read counts and the true differential splicing status labels for these 100 genes are saved as tab-delimited `.txt` files in the `extdata/` directory in the *regsplice* package source code.

2 Workflow

2.1 Complete workflow with wrapper function

..... to do:

How to run a complete workflow in a single step using the `regsplice()` wrapper function

You can provide the data either as filenames or matrices / data frames. Also need gene/exon IDs etc.

...

Below, we show how to run the functions for the individual steps in the *regsplice* workflow in sequence, which provides additional flexibility and insight into the methodology.

2.2 Individual steps

2.2.1 Load data

Load the demo data file, which contains simulated RNA-seq read counts for 100 genes across 6 biological samples; and create the meta-data for the biological samples.

```
file_counts <- system.file("extdata/counts.txt", package = "regsplice")
data <- read.table(file_counts, header = TRUE, sep = "\t", stringsAsFactors = FALSE)

head(data, 3)
##                   exon sample1 sample2 sample3 sample4 sample5 sample6
## 1 ENSG00000000003:001     576     506     526     643     482     826
## 2 ENSG00000000003:002     141     122     126     157     121     191
## 3 ENSG00000000003:003     123     102     106     133      99     156
dim(data)
## [1] 3191    7

# extract counts and gene/exon identifiers
counts <- data[, 2:7]
gene <- sapply(strsplit(data$exon, ":"), function(s) s[[1]])
exon <- sapply(strsplit(data$exon, ":"), function(s) s[[2]])

head(gene, 3)
## [1] "ENSG00000000003" "ENSG00000000003" "ENSG00000000003"
head(exon, 3)
## [1] "001" "002" "003"

# create meta-data for biological samples
condition <- rep(c("untreated", "treated"), each = 3)
condition
## [1] "untreated" "untreated" "untreated" "treated"   "treated"   "treated"
```

2.2.2 Prepare data

Prepare the data into the correct format for the subsequent steps. The function `split_genes()` splits the RNA-seq count table into a list of sub-tables, one for each gene. The length of each gene (i.e. the number of exons) is taken from the number of repeated entries in the vector of gene identifiers. The function `filter_genes()` then removes any genes with zero total counts.

```
library(regsplice)

Y <- split_genes(counts = counts, gene = gene)
length(Y)
## [1] 100

Y <- filter_genes(Y)
length(Y)
## [1] 88
```

2.2.3 Create design matrices

The function `create_design_matrix()` creates the model design matrix for each gene. This function is called automatically by the model fitting functions in the later steps. Here, we demonstrate how it works for a single gene.

The design matrix includes main effect terms for each exon and each sample, and interaction terms between the exons and the biological conditions.

Note that the design matrix does not include main effect terms for the biological conditions, since these are absorbed into the main effect terms for the samples. In addition, the design matrix does not include an intercept column, since it is simpler to let the model fitting functions add an intercept term later.

```
# gene with 3 exons; 4 biological samples, 2 samples in each of 2 conditions
design_example <- create_design_matrix(condition = rep(c(0, 1), each = 2), n_exons = 3)
design_example
##    Exon2 Exon3 Samp2 Samp3 Samp4 Exon2:Cond1 Exon3:Cond1
## 1      0     0     0     0     0           0           0
## 2      1     0     0     0     0           0           0
## 3      0     1     0     0     0           0           0
## 4      0     0     1     0     0           0           0
## 5      1     0     1     0     0           0           0
## 6      0     1     1     0     0           0           0
## 7      0     0     0     1     0           0           0
## 8      1     0     0     1     0           1           0
## 9      0     1     0     1     0           0           1
## 10     0     0     0     0     1           0           0
## 11     1     0     0     0     1           1           0
## 12     0     1     0     0     1           0           1
```

2.2.4 Fit models

There are three model fitting functions: `fit_reg()`, `fit_GLM()`, and `fit_null()`. These fit the regularized (lasso) models containing an optimal subset of exon:condition interaction terms; the full GLMs containing interaction terms for every exon; and the null models with zero interaction terms.

The lasso model penalizes the interaction terms only, so that the main effect terms for exons and samples are always included. This ensures that the null model is nested within the lasso model, allowing likelihood ratio tests to be calculated.

The *regsplice* pipeline fits all three models for each gene. If the regularized (lasso) model contains at least one exon:condition interaction term, then this model is compared against the null model in the likelihood ratio test in the next step. However, if the lasso model contains zero interaction terms, then it is not possible to calculate a likelihood ratio test, since the fitted and null models are identical. In this case, the user has the option to either set a p-value of 1; or calculate a likelihood ratio test using the full GLM containing all interaction terms, with reduced power (see next section).

The model fitting functions are parallelized, with the `n_cores` argument controlling the number of cores. For `fit_reg()`, the default is 8 cores, or the maximum available if less than 8. For `fit_GLM()` and `fit_null()`, the default is one core, since these functions are already extremely fast; if they take longer than a few seconds for your data set, it may be beneficial to try increasing the number of cores.

The `seed` argument can be used to set a random number generation seed for reproducible results, if required.

```
# fit regularized models
fitted_models_reg <- fit_reg(Y = Y, condition = condition)

# fit GLMs
fitted_models_GLM <- fit_GLM(Y = Y, condition = condition)
```

```
# fit null models
fitted_models_null <- fit_null(Y = Y, condition = condition)
```

2.2.5 Calculate likelihood ratio tests

After the models have been fitted, the function `LR_tests()` calculates likelihood ratio (LR) tests for each gene.

If the regularized (lasso) model contains at least one exon:condition interaction term, the LR test compares the lasso model against the null model. However, if the lasso model contains zero interaction terms, then the lasso and null models are identical, so the LR test cannot be calculated. The `when_null_selected` argument lets the user choose what to do in these cases: either set p-values equal to 1 (`when_null_selected = "ones"`); or calculate a LR test using the full GLM containing all exon:condition interaction terms (`when_null_selected = "GLM"`), which reduces power due to the larger number of terms, but allows the evidence for differential exon usage among these genes to be distinguished. You can also return `NA`s for these genes ('when_null_selected = "NA").

The default option is `when_null_selected = "ones"`. This simply calls all these genes non-significant, which in most cases is sufficient since we are more interested in genes with strong evidence for differential exon usage. However, if it is important to rank the low-evidence genes in your data set, then use the `when_null_selected = GLM` option.

If `when_null_selected = "ones"`, the full GLM fitted models are not required, so you can skip the `fit_GLM()` step above and set `fitted_models_GLM = NULL` (the default) in the `LR_tests()` function below.

```
# calculate likelihood ratio tests
res <- LR_tests(fitted_models_reg = fitted_models_reg,
                fitted_models_GLM = NULL,
                fitted_models_null = fitted_models_null,
                when_null_selected = "ones")
```

2.2.6 Plot results

Plot results: p-values and adjusted p-values from previous step

replace with visualization functions in a new file visualize-results.R

```
# p-values
#plot(res$p_vals[order(res$p_vals)], type = "b")

# p-values adjusted for multiple testing
#plot(res$p_adj[order(res$p_adj)], type = "b")
```

3 Additional steps for microarray data

If you are using microarray data, you also need to use limma/voom to convert data...

give example code

Vignette info from template:

Figures

The figure sizes have been customised so that you can easily put two images side-by-side.

```
#plot(1:10)
#plot(10:1)
```

You can enable figure captions by `fig_caption: yes` in YAML:

```
output:
  rmarkdown::html_vignette:
    fig_caption: yes
```

Then you can use the chunk option `fig.cap = "Your figure caption."` in **knitr**.

Math expressions

You can write math expressions, e.g. $Y = X\beta + \epsilon$, footnotes[1], and tables, e.g. using `knitr::kable()`.

[1] A footnote here.