Example workflow for *regsplice* package

Lukas M. Weber

26 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 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)
##                   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
## 4 ENSG00000000003:004      86      76      77      98      72     112
## 5 ENSG00000000003:005      97      83      87     113      76     126
## 6 ENSG00000000003:006     133     107     116     155      97     170

dim(data)
## [1] 3191    7

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

head(gene, 6)
## [1] "ENSG00000000003" "ENSG00000000003" "ENSG00000000003" "ENSG00000000003"
## [5] "ENSG00000000003" "ENSG00000000003"

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

condition
## [1] "untreated" "untreated" "untreated" "treated"   "treated"   "treated"
```

2.2 Complete workflow with wrapper function

The `regsplice()` wrapper function runs the complete workflow in one command.

The results consist of p-values quantifying the evidence for differential exon usage for each gene. Multiple testing adjusted p-values are also provided, as well as the likelihood ratio test statistics and degrees of freedom for each test.

The required inputs for the wrapper function are `counts` (matrix or data frame of RNA-seq counts), `gene` (vector of gene IDs), and `condition` (vector of experimental conditions for each sample).

have used `n_cores_reg = 1` due to Travis CI (or not required here?)

```
library(regsplice)

res <- regsplice(counts, gene, condition)
## removed 936 exon(s) with zero counts
## removed 1 remaining single-exon gene(s)
## removed 240 low-count exon(s)
## removed 4 remaining single-exon gene(s)

str(res)
## List of 4
##  $ p_vals  : num [1:81] 1 0 1 1 0 ...
##  $ p_adj   : num [1:81] 1 0 1 1 0 ...
##  $ LR_stats: num [1:81] NA 107224 NA NA 4995 ...
##  $ df_tests: int [1:81] NA 16 NA NA 16 NA 32 11 1 NA ...
```

2.3 Individual steps

Alternatively, you can run each of the individual functions in the *regsplice* workflow in sequence, which provides additional flexibility and insight into the statistical methodology. Each step is described below.

2.3.1 Prepare data

The first step consists of pre-processing the input data, and preparing it into the required format.

The function `split_genes()` splits the RNA-seq count table into a list of sub-tables, one for each gene. The gene lengths (i.e. the number of exons in each gene) are calculated from the numbers of repeated entries in the vector of gene IDs. The function `filter_zeros()` removes any genes with zero total counts, and `filter_single_exons()` removes genes containing only a single exon.

... rewrite this using new `prepare_data()` function

```
library(regsplice)

Y <- prepare_data(counts, gene)
## removed 936 exon(s) with zero counts
## removed 1 remaining single-exon gene(s)

length(Y)
## [1] 87
```

2.3.2 Filter low-count exons

Filtering step to remove low-count exons

includes default values, but these may not be optimal depending on the experimental design, so should be adjusted

```
Y <- filter_exons(Y)
## removed 240 low-count exon(s)
## removed 4 remaining single-exon gene(s)

length(Y)
## [1] 81
```

2.3.3 Optional: 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, so you do not need to run it directly. Here, we demonstrate how it works for a single gene, and show an example design matrix.

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
```

2.3.4 Fit models

rearrange order throughout: `fit_reg`, `fit_null`, `fit_GLM`

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 regularized and null 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. 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.

using single core (`n_cores = 1` argument) for Travis CI

suppress warning messages which are due to small number of observations for some genes

```
# fit regularized models
fit_reg  <- fit_models_reg(Y, condition, n_cores = 1)
## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations
## per fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations
## per fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations
## per fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations
## per fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations
## per fold

## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations
## per fold
```

```
## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations
## per fold

# fit null models
fit_null <- fit_models_null(Y, condition)

# fit GLMs (not required if 'when_null_selected = "ones"' below)
fit_GLM  <- fit_models_GLM(Y, condition)
```

2.3.5 Calculate likelihood ratio tests

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

As mentioned above, 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 among these genes to be distinguished; or 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 `fit_GLM()` in the previous step, and set `fitted_models_GLM = NULL` (the default) in the `LR_tests()` function.

```
# calculate likelihood ratio tests
res_no_wrapper <- LR_tests(fit_reg = fit_reg,
                           fit_null = fit_null,
                           when_null_selected = "ones")
```

2.3.6 Plot results

Plot results: p-values and multiple testing adjusted p-values

to do: create plotting function (`plot_results.R`)

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

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

3 Additional steps for microarray data

Additional steps are required if you are using microarray expression data instead of RNA-seq counts.

use limma/voom to convert data

to do

```
# microarrays example code
```

4 Weights

Observation-level (exon-level) weights

from limma/voom

to do

5 Options

to do: examples of varying the other options, e.g. `alpha` and `lambda_choice` parameters; possibly move into previous sections