Example workflow for *regsplice* package

Lukas M. Weber

1 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 fast and make use of the lasso to improve power compared to standard generalized linear models. The methodology and comparisons to previous approaches 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.

1.3 SummarizedExperiment class

The *regsplice* package uses the `SummarizedExperiment` S4 class to enable easier integration into Bioconductor workflows. The `SummarizedExperiment` class stores matrices of data values (such as RNA-seq read counts or microarray expression intensities) along with associated meta-data for rows (genes or exons) and columns (biological samples).

The main benefit of using `SummarizedExperiment` objects is that subsetting operations work intuitively, keeping data and meta-data in sync. The `SummarizedExperiment` class replaces the earlier `ExpressionSet` class. For more information, see the `SummarizedExperiment` Bioconductor vignette, available here.

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 / matrices, or as a `SummarizedExperiment` object. If you provide filenames or matrices, you also need to provide gene and exon indices, so that *regsplice* can create the `SummarizedExperiment` object automatically.

. . .

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.

```
file_counts <- system.file("extdata/counts.txt", package = "regsplice")

data <- read.table(file_counts, header = TRUE, sep = "\t", stringsAsFactors = FALSE)

dim(data)
## [1] 3191    7
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

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

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

2.2.2 Create SummarizedExperiment object

Create a `SummarizedExperiment` object containing the data (counts), as well as meta-data for rows (genes and exons) and columns (biological samples).

This can be done using the `prepare_data()` function, which requires a matrix or data frame of counts, and character vectors containing the gene and exon identifiers. Optionally, the `exon` argument can be left empty, in which case the exons within each gene will be numbered sequentially.

The row and column meta-data can be accessed with `S4Vectors::mcols()` and `SummarizedExperiment::colData()`.

```
suppressPackageStartupMessages(library(regsplice))
suppressPackageStartupMessages(library(SummarizedExperiment))

se <- prepare_data(counts = counts, gene = gene, exon = exon, condition = condition)

str(se, max.level = 1)
## Formal class 'SummarizedExperiment0' [package "SummarizedExperiment"] with 5 slots
```

```
mcols(se)
## DataFrame with 3191 rows and 2 columns
##                 gene        exon
##          <character> <character>
## 1    ENSG00000000003         001
## 2    ENSG00000000003         002
## 3    ENSG00000000003         003
## 4    ENSG00000000003         004
## 5    ENSG00000000003         005
## ...              ...         ...
## 3187 ENSG00000005513         002
## 3188 ENSG00000005513         003
## 3189 ENSG00000005513         004
## 3190 ENSG00000005513         005
## 3191 ENSG00000005513         006

colData(se)
## DataFrame with 6 rows and 1 column
##     condition
##   <character>
## 1   untreated
## 2   untreated
## 3   untreated
## 4     treated
## 5     treated
## 6     treated
```

2.2.3 Create design matrices

The `create_design_matrix()` function is used to create design matrices for each gene.

2.2.4 Fit models

regularized model and null model for each gene

2.2.5 Calculate likelihood ratio tests

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

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.