Estimating Gene-Specific Phenotpes with `gespeR`

Fabian Schmich

May 19, 2021

Contents

1 The gespeR Model

This package provides algorithms for deconvoluting off-target confounded phenotypes from RNA interference screens. The packages uses (predicted) siRNA-to-gene target relations in a regularised linear regression model, in order to infer individual gene-specific phenotype (GSP) contributions. The observed siRNA-specific phenotypes (SSPs) for reagent $i = 1, \ldots, n$ as the weighted linear sum of GSPs of all targeted genes $j = 1, \ldots, p$

$$Y_i = x_{i1}\beta_j + \ldots + x_{ip}\beta_p + \epsilon_i, \tag{1}$$

where x_{ij} represents the strength of knockdown of reagent i on gene j, β_j corresponds to the GSP of gene j and ϵ_i is the error term for SSP i. The linear reegression model is fit using elastic net regularization:

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} \left\{ \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2 + \lambda \sum_{j=1}^{p} \left(\alpha\beta_j^2 + (1-\alpha)\left|\beta_j\right| \right) \right\}. \tag{2}$$

Here λ determines the amount of regualrisation and α is the mixing parameter between the ridge and lasso penalty with $0 \leq \alpha \leq 1$. The elastic net penalty selects variables like the lasso and shrinks together the coefficients of correlated predictors like ridge. This allows for a sparse solution of nonzero GSPs, while retaining simultaneous selection of genes with similar RNAi reagent binding patterns in their respective 3' UTRs. For more information and for citing the `gespeR` package please refer to:

> Schmich F (2021). *gespeR: Gene-Specific Phenotype EstimatoR*. R package version 1.25.0, `http://www.cbg.ethz.ch/software/gespeR`.

2 Working Example

In this example, we load first load simulated phenotypic readout and siRNA-to-gene target relations. The toy data consists of four screens (A, B, C, D) of 1,000 siRNAs and a limited gene universe of 1,500 genes. Detailed description of how the data was simulated can be accessed using `?simData`. First, we load the package:

```
  library(gespeR)
```

Now the phenotypes and target relations can be initialised using the `Phenotypes` and `TargetRelations` commands. First, we load the four phenotypes vectors:

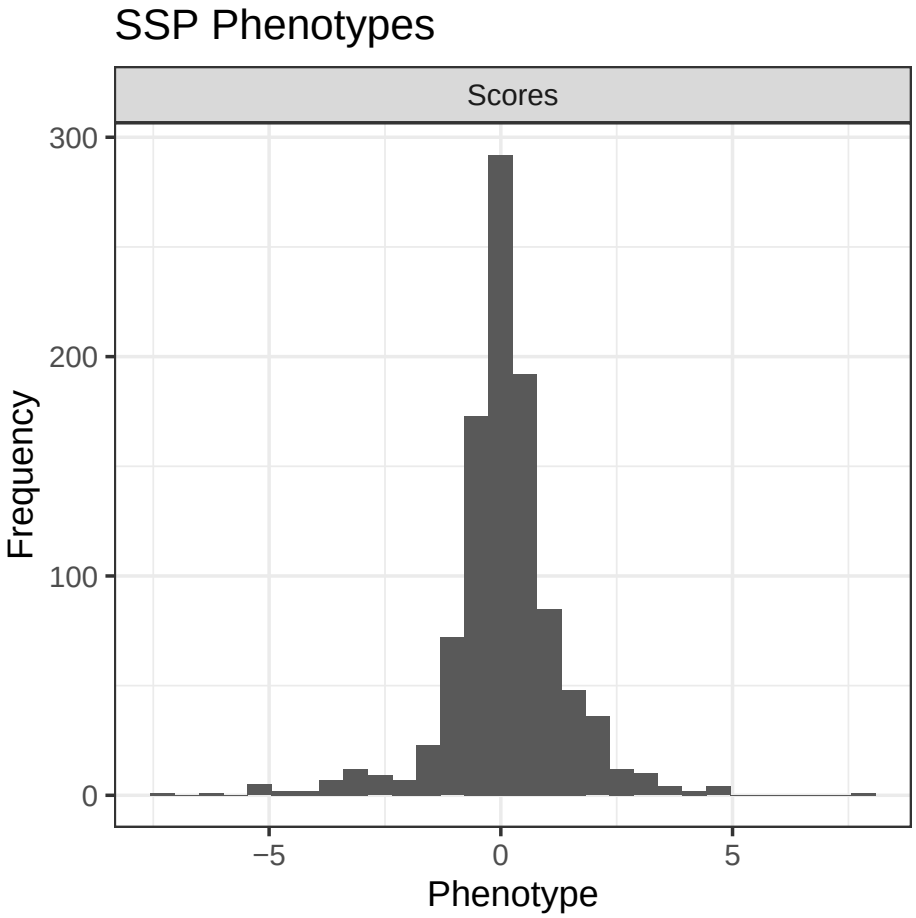
```
  phenos <- lapply(LETTERS[1:4], function(x) {
    sprintf("Phenotypes_screen_%s.txt", x)
  })
  phenos <- lapply(phenos, function(x) {
    Phenotypes(system.file("extdata", x, package="gespeR"),
      type = "SSP",
      col.id = 1,
      col.score = 2)
  })
  show(phenos[[1]])
```

```
## 1000 SSP Phenotypes

## Warning:  `tbl_df()` was deprecated in dplyr 1.0.0.
## Please use `tibble::as_tibble()` instead.

## # A tibble: 1,000 x 2
##    ID            Scores
##    <chr>          <dbl>
##  1 siRNAID_0001 -0.930
##  2 siRNAID_0002 -1.13
##  3 siRNAID_0003 -1.05
##  4 siRNAID_0004  0.808
##  5 siRNAID_0005 -1.42
##  6 siRNAID_0006  1.64
##  7 siRNAID_0007 -0.157
##  8 siRNAID_0008  0.748
##  9 siRNAID_0009 -0.959
## 10 siRNAID_0010 -0.0440
## # ... with 990 more rows
```

A visual representation of the phenotypes can be obtained with the `plot` method:

```
  plot(phenos[[1]])
```

```
## `stat_bin()` using `bins = 30`.  Pick better value with `binwidth`.
```



Now, we load the target relations for all four screens using the constructor of the `TargetRelations` class:

```
  tr <- lapply(LETTERS[1:4], function(x) {
    sprintf("TR_screen_%s.rds", x)
  })
  tr <- lapply(tr, function(x) {
    TargetRelations(system.file("extdata", x, package="gespeR"))
  })
  show(tr[[2]])
```

```
## 1000 x 1500 siRNA-to-gene relations.
## 10 x 5 sparse Matrix of class "dgCMatrix"
##                colnames
## rownames        geneID_0001 geneID_0002 geneID_0003 geneID_0004 geneID_0005
##   siRNAID_0001            .           .           .           .           .
##   siRNAID_0002            .           .           .           .           .
##   siRNAID_0003            .   0.4385757           .           .           .
##   siRNAID_0004            .           .           .           .           .
##   siRNAID_0005            .           .           .           .           .
##   siRNAID_0006            .           .           .           .           .
##   siRNAID_0007            .           .           .           .           .
##   siRNAID_0008            .           .           .           .           .
##   siRNAID_0009            .           .           .           .           .
##   siRNAID_0010            .           .           .           .           .
## ...
```

For large data sets, e.g. genome–wide screens, target relations objects can become very big and the user may not want to keep all values in the RAM. For this purpose, we can use the `unloadValues` method. In this example, we write the values to a temp-file, i.e. not the original source file, which may be required, when we do not want to overwrite exisiting data, after, for instance, subsetting the target relations object.

```
  # Size of object with loaded values
  format(object.size(tr[[1]]), units = "Kb")

## [1] "717.7 Kb"

  tempfile <- paste(tempfile(pattern = "file", tmpdir = tempdir()), ".rds", sep="")
  tr[[1]] <- unloadValues(tr[[1]], writeValues = TRUE, path = tempfile)

  # Size of object after unloading
  format(object.size(tr[[1]]), units = "Kb")

## [1] "178.3 Kb"

  # Reload values
  tr[[1]] <- loadValues(tr[[1]])
```

In order to obtain deconvoluted gene-specific phenotypes (GSPs), we fit four models on the four separate data sets using cross validation by setting `mode = "cv"`. We set the elastic net mixing parameter to 0.5 and use only one core in this example:

```
  res.cv <- lapply(1:length(phenos), function(i) {
    gespeR(phenotypes = phenos[[i]],
           target.relations = tr[[i]],
           mode = "cv",
           alpha = 0.5,
           ncores = 1)
  })
```

The `ssp` and `gsp` methods can be used to obtain SSP and GSP scores from a `gespeR` object:

```
  ssp(res.cv[[1]])

## 1000 SSP Phenotypes
##
## # A tibble: 1,000 x 2
##    ID            Scores
##    <chr>          <dbl>
##  1 siRNAID_0001 -0.930
##  2 siRNAID_0002 -1.13
##  3 siRNAID_0003 -1.05
##  4 siRNAID_0004  0.808
##  5 siRNAID_0005 -1.42
##  6 siRNAID_0006  1.64
##  7 siRNAID_0007 -0.157
##  8 siRNAID_0008  0.748
##  9 siRNAID_0009 -0.959
## 10 siRNAID_0010 -0.0440
## # ... with 990 more rows

  gsp(res.cv[[1]])
```

```
## 1500 GSP Phenotypes
##
## # A tibble: 1,500 x 2
##    ID            Scores
##    <chr>          <dbl>
##  1 geneID_0001 NA
##  2 geneID_0002 NA
##  3 geneID_0003 NA
##  4 geneID_0004 NA
##  5 geneID_0005  0.00993
##  6 geneID_0006 NA
##  7 geneID_0007 NA
##  8 geneID_0008 NA
##  9 geneID_0009 NA
## 10 geneID_0010 NA
## # ... with 1,490 more rows
```

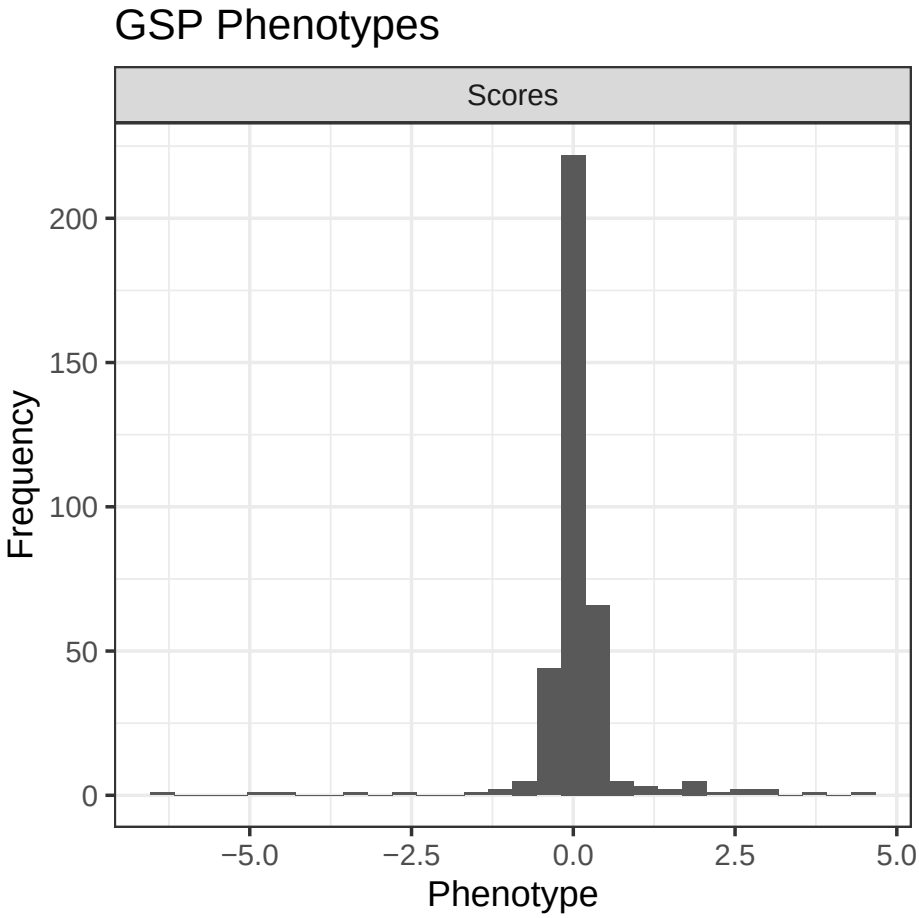
```
  head(scores(res.cv[[1]]))
```

```
## # A tibble: 6 x 2
##   ID            Scores
##   <chr>          <dbl>
## 1 geneID_0001 NA
## 2 geneID_0002 NA
## 3 geneID_0003 NA
## 4 geneID_0004 NA
## 5 geneID_0005  0.00993
## 6 geneID_0006 NA
```

The fitted models can also be visualised using the `plot` method:

```
  plot(res.cv[[1]])
```

```
## `stat_bin()` using `bins = 30`.  Pick better value with `binwidth`.
## Warning:  Removed 1133 rows containing non-finite values (stat_bin).
```

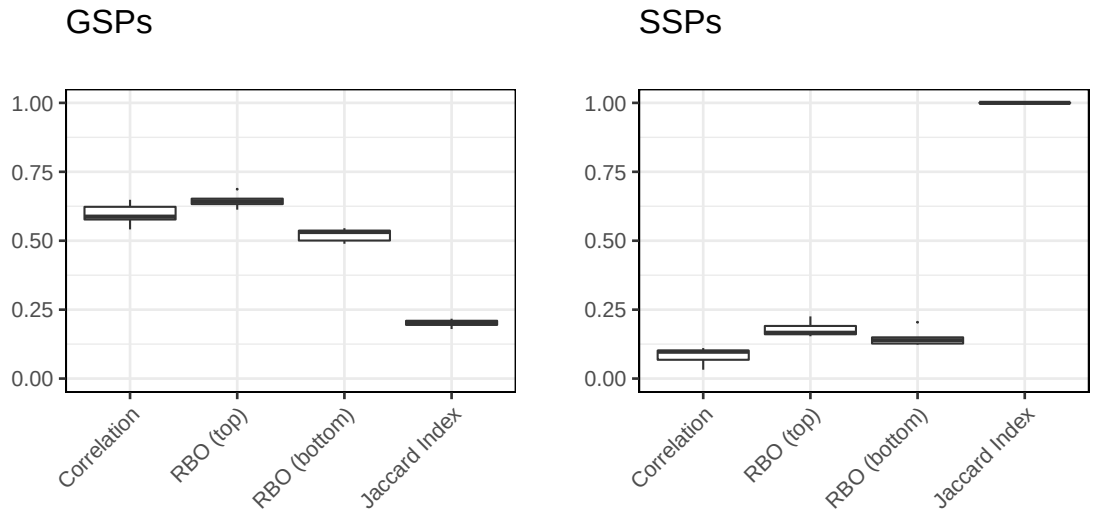


The `concordance` method can be used to compute the concordance between ranked lists of phenotypes. Here we compute concordance between all pairs of GSPs, as well as between all pairs of SSPs, from all four data sets:

```
conc.gsp <- concordance(lapply(res.cv, gsp))
conc.ssp <- concordance(lapply(res.cv, ssp))
```

We can visualise the `concordance` objects using the `plot` method:

```
plot(conc.gsp) + ggtitle("GSPs\n")
plot(conc.ssp) + ggtitle("SSPs\n")
```



3 sessionInfo()

- R version 4.1.0 beta (2021-05-03 r80259), `x86_64-pc-linux-gnu`
- Locale: `LC_CTYPE=en_US.UTF-8`, `LC_NUMERIC=C`, `LC_TIME=en_GB`, `LC_COLLATE=C`, `LC_MONETARY=en_US.UTF-8`, `LC_MESSAGES=en_US.UTF-8`, `LC_PAPER=en_US.UTF-8`, `LC_NAME=C`, `LC_ADDRESS=C`, `LC_TELEPHONE=C`, `LC_MEASUREMENT=en_US.UTF-8`, `LC_IDENTIFICATION=C`
- Running under: `Ubuntu 20.04.2 LTS`
- Matrix products: default
- BLAS: `/home/biocbuild/bbs-3.14-bioc/R/lib/libRblas.so`
- LAPACK: `/home/biocbuild/bbs-3.14-bioc/R/lib/libRlapack.so`
- Base packages: base, datasets, grDevices, graphics, methods, stats, utils
- Other packages: gespeR 1.25.0, ggplot2 3.3.3, knitr 1.33
- Loaded via a namespace (and not attached): AnnotationDbi 1.55.0, Biobase 2.53.0, BiocFileCache 2.1.0, BiocGenerics 0.39.0, BiocManager 1.30.15, Biostrings 2.61.0, Category 2.59.0, DBI 1.1.1, GSEABase 1.55.0, GenomeInfoDb 1.29.0, GenomeInfoDbData 1.2.6, IRanges 2.27.0, KEGGREST 1.33.0, Matrix 1.3-3, R6 2.5.0, RBGL 1.69.0, RColorBrewer 1.1-2, RCurl 1.98-1.3, RSQLite 2.2.7, Rcpp 1.0.6, S4Vectors 0.31.0, XML 3.99-0.6, XVector 0.33.0, affy 1.71.0, affyio 1.63.0, annotate 1.71.0, assertthat 0.2.1, biomaRt 2.49.0, bit 4.0.4, bit64 4.0.5, bitops 1.0-7, blob 1.2.1, cachem 1.0.5, cellHTS2 2.57.0, cli 2.5.0, codetools 0.2-18, colorspace 2.0-1, compiler 4.1.0, crayon 1.4.1, curl 4.3.1, dbplyr 2.1.1, digest 0.6.27, doParallel 1.0.16, dplyr 1.0.6, ellipsis 0.3.2, evaluate 0.14, fansi 0.4.2, farver 2.1.0, fastmap 1.1.0, filelock 1.0.2, foreach 1.5.1, genefilter 1.75.0, generics 0.1.0, glmnet 4.1-1, glue 1.4.2, graph 1.71.0, grid 4.1.0, gtable 0.3.0, highr 0.9, hms 1.1.0, httr 1.4.2, hwriter 1.3.2, iterators 1.0.13, labeling 0.4.2, lattice 0.20-44, lifecycle 1.0.0, limma 3.49.0, locfit 1.5-9.4, magrittr 2.0.1, memoise 2.0.0, munsell 0.5.0, parallel 4.1.0, pillar 1.6.1, pkgconfig 2.0.3, plyr 1.8.6, png 0.1-7, preprocessCore 1.55.0, prettyunits 1.1.1, progress 1.2.2, ps 1.6.0, purrr 0.3.4, rappdirs 0.3.3, reshape2 1.4.4, rlang 0.4.11, rstudioapi 0.13, scales 1.1.1, shape 1.4.6, splines 4.1.0, splots 1.59.0, stats4 4.1.0, stringi 1.6.2, stringr 1.4.0, survival 3.2-11, tibble 3.1.2, tidyselect 1.1.1, tools 4.1.0, utf8 1.2.1, vctrs 0.3.8, vsn 3.61.0, withr 2.4.2, xfun 0.23, xtable 1.8-4, zlibbioc 1.39.0