

# Estimating Gene-Specific Phenotypes with **gespeR**

Fabian Schmich

October 29, 2019

## Contents

<b>1</b>	<b>The <code>gespeR</code> Model</b>	<b>1</b>
<b>2</b>	<b>Working Example</b>	<b>1</b>
<b>3</b>	<b><code>sessionInfo()</code></b>	<b>7</b>

## 1 The `gespeR` Model

This package provides algorithms for deconvoluting off-target confounded phenotypes from RNA interference screens. The package 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  $i = 1, \dots, n$  as the weighted linear sum of GSPs of all targeted genes  $j = 1, \dots, p$

$$Y_i = x_{i1}\beta_1 + \dots + x_{ip}\beta_p + \epsilon_i, \quad (1)$$

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

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

Here  $\lambda$  determines the amount of regularization and  $\alpha$  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 (2019). *gespeR: Gene-Specific Phenotype Estimator*. R package version 1.18.0, <http://www.cbgl.ethz.ch/software/gespeR>.

## 2 Working Example

In this example, we 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)
```

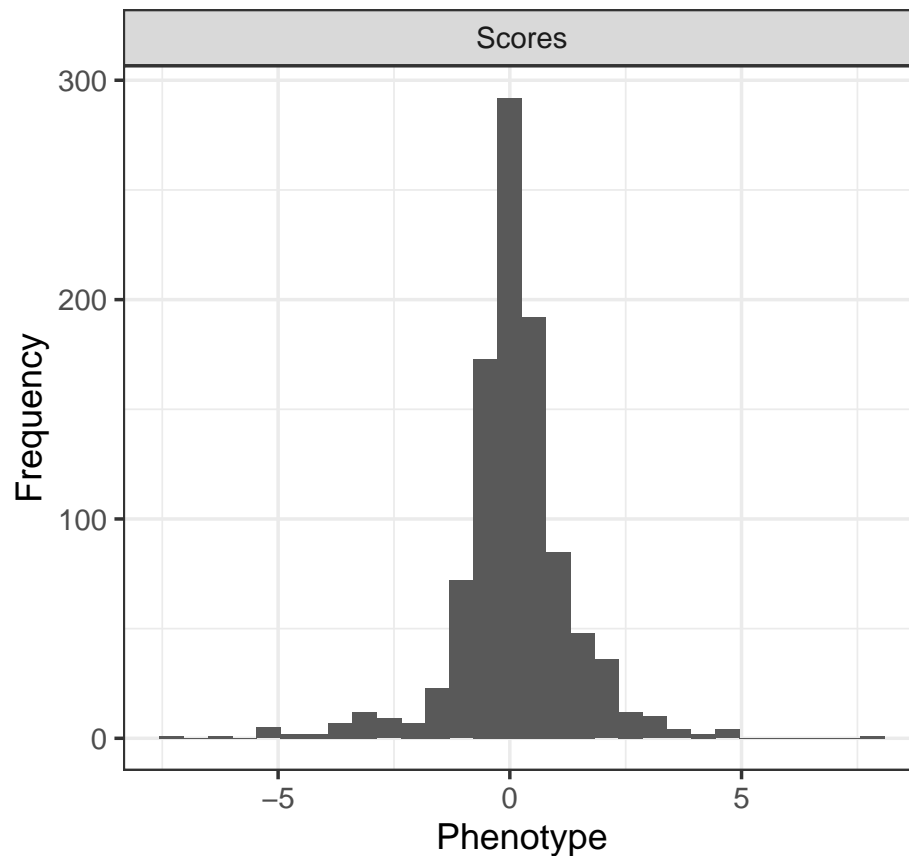
Now the phenotypes and target relations can be initialised using the `Phenotypes` and `TargetRelations` commands. First, we load the four phenotypes vectors:

```
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  
##  
## # A tibble: 1,000 x 2  
##   ID          Scores  
##   <fct>         <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'.
```

## SSP Phenotypes



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="gesperR"))
})
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 existing 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 = tmpdir()), ".rds", sep="")
tr[[1]] <- unloadValues(tr[[1]], writeValues = TRUE, path = tempfile)

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

## [1] "178.9 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
##   <fct>         <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
##   <fct>        <dbl>
## 1 geneID_0001      NA
## 2 geneID_0002      NA
## 3 geneID_0003      NA
## 4 geneID_0004      NA
## 5 geneID_0005      NA
## 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

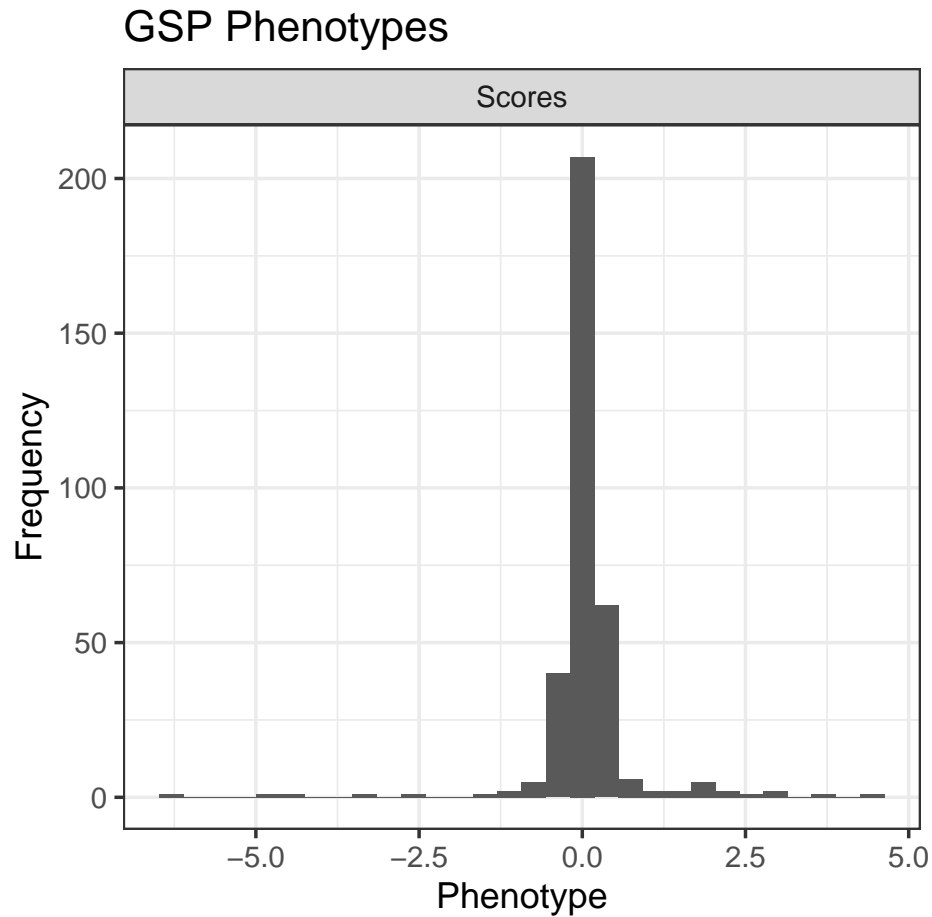
head(scores(res.cv[[1]]))

## # A tibble: 6 x 2
##   ID          Scores
##   <fct>        <dbl>
## 1 geneID_0001      NA
## 2 geneID_0002      NA
## 3 geneID_0003      NA
## 4 geneID_0004      NA
## 5 geneID_0005      NA
## 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 1156 rows containing non-finite values (stat_bin).
```

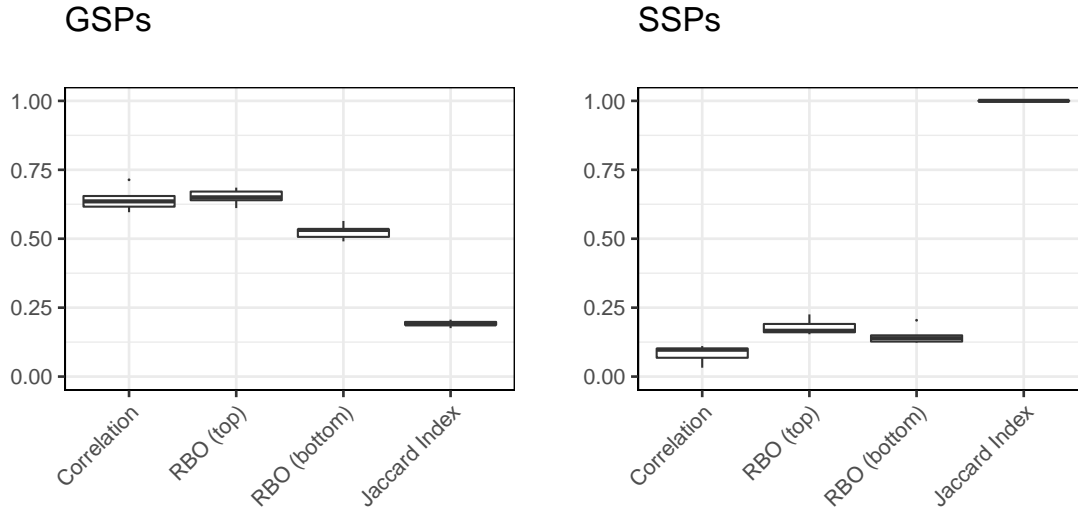


The `concordance` method can be used to compute the concordance between ranked lists of phenotypes. Here we compute concordance between all pairs of GSPs, as well as between all pairs of SSPs, from all four data sets:

```
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 3.6.1 (2019-07-05), x86\_64-w64-mingw32
- Locale: LC\_COLLATE=C, LC\_CTYPE=English\_United States.1252, LC\_MONETARY=English\_United States.1252, LC\_NUMERIC=C, LC\_TIME=English\_United States.1252
- Running under: Windows Server 2012 R2 x64 (build 9600)
- Matrix products: default
- Base packages: base, datasets, grDevices, graphics, methods, stats, utils
- Other packages: gesper 1.18.0, ggplot2 3.2.1, knitr 1.25
- Loaded via a namespace (and not attached): AnnotationDbi 1.48.0, Biobase 2.46.0, BiocFileCache 1.10.0, BiocGenerics 0.32.0, BiocManager 1.30.9, Category 2.52.0, DBI 1.0.0, DEoptimR 1.0-8, GSEABase 1.48.0, IRanges 2.20.0, MASS 7.3-51.4, Matrix 1.2-17, R6 2.4.0, RBGL 1.62.0, RColorBrewer 1.1-2, RCurl 1.95-4.12, RSQLite 2.1.2, Rcpp 1.0.2, S4Vectors 0.24.0, XML 3.98-1.20, affy 1.64.0, affyio 1.56.0, annotate 1.64.0, askpass 1.1, assertthat 0.2.1, backports 1.1.5, biomaRt 2.42.0, bit 1.1-14, bit64 0.9-7, bitops 1.0-6, blob 1.2.0, cellHTS2 2.50.0, cli 1.1.0, cluster 2.1.0, codetools 0.2-16, colorspace 1.4-1, compiler 3.6.1, crayon 1.3.4, curl 4.2, dbplyr 1.4.2, digest 0.6.22, doParallel 1.0.15, dplyr 0.8.3, evaluate 0.14, fansi 0.4.0, foreach 1.4.7, genefilter 1.68.0, glmnet 2.0-18, glue 1.3.1, graph 1.64.0, grid 3.6.1, gtable 0.3.0, highr 0.8, hms 0.5.1, httr 1.4.1, hwriter 1.3.2, iterators 1.0.12, labeling 0.3, lattice 0.20-38, lazyeval 0.2.2, limma 3.42.0, locfit 1.5-9.1, magrittr 1.5, memoise 1.1.0, munsell 0.5.0, mvtnorm 1.0-11, openssl 1.4.1, parallel 3.6.1, pcaPP 1.9-73, pillar 1.4.2, pkgconfig 2.0.3, plyr 1.8.4, prada 1.62.0, preprocessCore 1.48.0, prettyunits 1.0.2, progress 1.2.2, purrr 0.3.3, rappdirs 0.3.1, reshape2 1.4.3, rlang 0.4.1, robustbase 0.93-5, rrcov 1.4-7, scales 1.0.0, splines 3.6.1, splots 1.52.0, stats4 3.6.1, stringi 1.4.3, stringr 1.4.0, survival 2.44-1.1, tibble 2.1.3, tidyselect 0.2.5, tools 3.6.1, utf8 1.1.4, vctrs 0.2.0, vsn 3.54.0, withr 2.1.2, xfun 0.10, xtable 1.8-4, zeallot 0.1.0, zlibbioc 1.32.0