Estimating Gene-Specific Phenotpes with gespeR

Fabian Schmich

April 16, 2015

Contents

1 Introduction

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 use:

> Schmich F (2015). *gespeR: Gene-Specific Phenotype EstimatoR*. R package version 1.0.0, `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. First, we load the package:

```
library(gespeR)
```

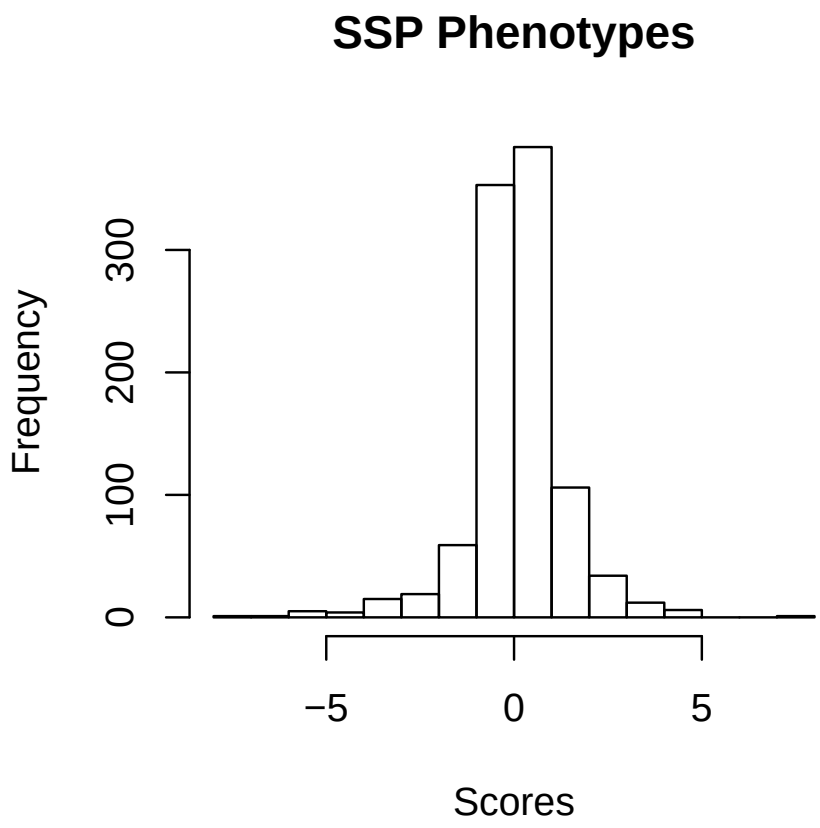
Now the phenotypes and target relations can be initialised using the `Phenotypes` and `TargetRelations` commands. First, we load the four phenotypes vectors:

```
  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
##
##            IDs     Scores
## 1 siRNAID_0001 -0.9302872
## 2 siRNAID_0002 -1.1282038
## 3 siRNAID_0003 -1.0526504
## 4 siRNAID_0004  0.8079272
## 5 siRNAID_0005 -1.4153335
## ...
```

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

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



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] "678.5 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] "159.1 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",
```

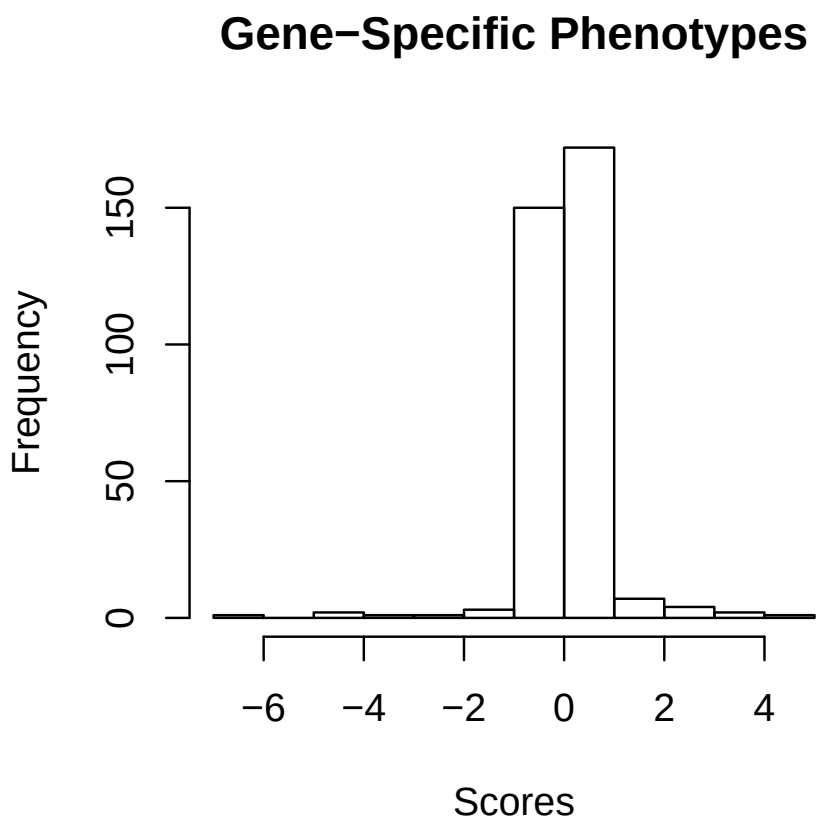
```
          alpha = 0.5,
          ncores = 1)
  })
```

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
##
##            IDs     Scores
## 1 siRNAID_0001 -0.9302872
## 2 siRNAID_0002 -1.1282038
## 3 siRNAID_0003 -1.0526504
## 4 siRNAID_0004  0.8079272
## 5 siRNAID_0005 -1.4153335
## ...

  gsp(res.cv[[1]])

## 344 GSP Phenotypes
##
##           IDs      Scores
## 1 geneID_0018 -0.06448336
## 2 geneID_0020  0.03674769
## 3 geneID_0021 -0.08929625
## 4 geneID_0024 -0.01985580
## 5 geneID_0027  0.13249567
## ...

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

## geneID_0018 geneID_0020 geneID_0021 geneID_0024 geneID_0027 geneID_0028
## -0.06448336  0.03674769 -0.08929625 -0.01985580  0.13249567  0.67765341
```

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

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

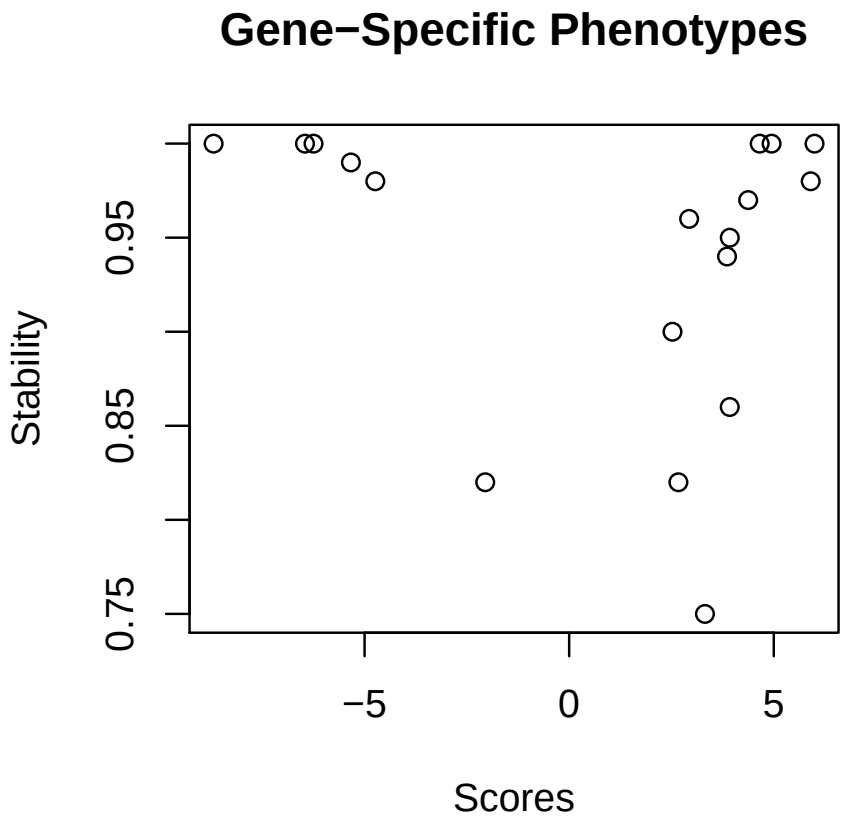


Another way to fit `gespeR` models is to use stability selection:

```
res.stab <- lapply(1:length(phenos), function(i) {
  gespeR(phenotypes = phenos[[i]],
    target.relations = tr[[i]],
    mode = "stability",
    nbootstrap = 100,
    fraction = 0.67,
    threshold = 0.75,
    EV = 1,
    weakness = 0.8,
    ncores = 1)
})
```

Again, these models can be visualised using the `plot` method. This time, the phenotypic scores are plotted against their stability value:

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

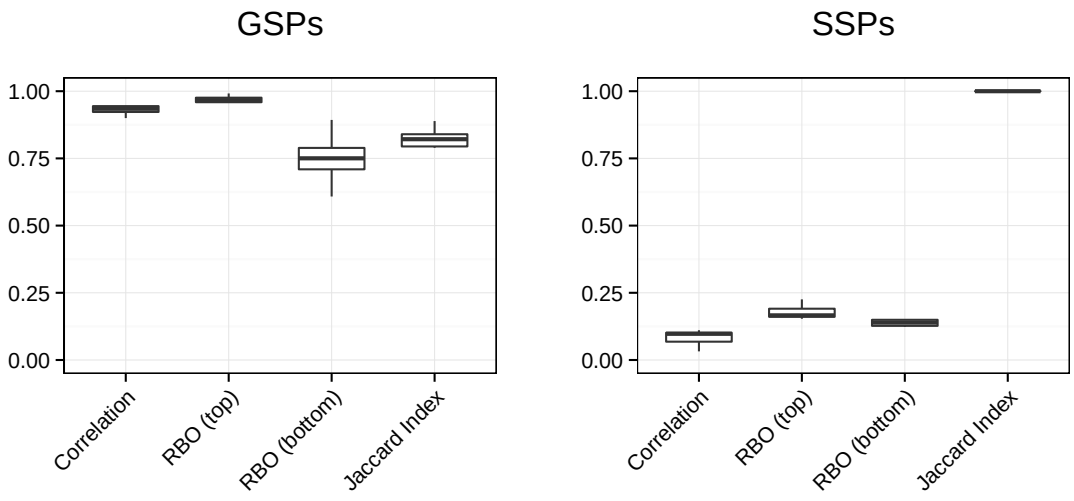


The `concordance` method can be used to compute the concordance between ranked lists of phenotypes. Here we compute concordance between all pairs of GSPs, as well as between all pairs of SSPs, from all four data sets:

```
conc.gsp <- concordance(lapply(res.stab, gsp))
conc.ssp <- concordance(lapply(res.stab, 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 3.2.0 RC (2015-04-08 r68161), `x86_64-apple-darwin13.4.0`
- Locale: `C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8`
- Base packages: base, datasets, grDevices, graphics, methods, stats, utils
- Other packages: gespeR 1.0.0, ggplot2 1.0.1, knitr 1.9
- Loaded via a namespace (and not attached): AnnotationDbi 1.30.0, Biobase 2.28.0, BiocGenerics 0.14.0, BiocInstaller 1.18.1, Category 2.34.0, DBI 0.3.1, DEoptimR 1.0-2, GSEABase 1.30.0, GenomeInfoDb 1.4.0, IRanges 2.2.0, MASS 7.3-40, Matrix 1.2-0, RBGL 1.44.0, RColorBrewer 1.1-2, RCurl 1.95-4.5, RSQLite 1.0.0, Rcpp 0.11.5, S4Vectors 0.6.0, XML 3.98-1.1, affy 1.46.0, affyio 1.36.0, annotate 1.46.0, biomaRt 2.24.0, bitops 1.0-6, cellHTS2 2.32.0, cluster 2.0.1, codetools 0.2-11, colorspace 1.2-6, compiler 3.2.0, digest 0.6.8, doParallel 1.0.8, evaluate 0.6, foreach 1.4.2, formatR 1.1, genefilter 1.50.0, glmnet 2.0-2, graph 1.46.0, grid 3.2.0, gtable 0.1.2, highr 0.4.1, iterators 1.0.7, labeling 0.3, lattice 0.20-31, limma 3.24.0, munsell 0.4.2, mvtnorm 1.0-2, parallel 3.2.0, pcaPP 1.9-60, plyr 1.8.1, prada 1.44.0, preprocessCore 1.30.0, proto 0.3-10, reshape2 1.4.1, robustbase 0.92-3, rrcov 1.3-8, scales 0.2.4, splines 3.2.0, stats4 3.2.0, stringr 0.6.2, survival 2.38-1, tools 3.2.0, vsn 3.36.0, xtable 1.7-4, zlibbioc 1.14.0