wsiHD - Weak Signal Inference Under High Dimensionality

August 5, 2021

Introduction

wtsHD is an R package that implements an analytic framework to regulate false negative errors under measures tailored towards modern applications with high-dimensional data. The method controls the false negative proportion (FNP) at a user-specified level and regulates the amount of unnecessary false positives to achieve the FNP level. The framework comprises three steps, the estimation of the bounding sequences, the estimation of the signal proportion, and the estimation of the FNP. The implementation of each step is described in the following section. An illustrative example based on the dataset provided with the package is also provided.

Functions

cSeq()

This function estimates the bounding sequences using the empirical distributions

$$V_{0.5,a} = \max_{1\le j\le p} \frac{|j/p - \overline{\Phi}(w_{(j)})|}{\sqrt{\overline{\Phi}(w_{(j)})}} \qquad \text{and} \qquad V_{1,a} = \max_{1\le j\le p} \frac{|j/p - \overline{\Phi}(w_{(j)})|}{\overline{\Phi}(w_{(j)})},$$

where p is the total number of variables (signal + noise), $\overline{\Phi}(t) = 1 - \Phi(t)$, $\Phi(t)$ is the cumulative distribution function, and $w_{(j)}, j = 1, \ldots, p$ are statistics generated under the null distribution. The bounding sequences $c_{\alpha,0.5}$ and $c_{\alpha,1}$ are taken as the $(1-\alpha)$-th quantiles of $V_{0.5,a}$ and $V_{1,a}$, respectively.

The function call takes the following form:

```
cSeq(pval_null, alpha = 0.1)
```

where input `pval_null` is a $\{p \times n\}$ maxtrix-*like* object containing n sets of p p-values obtained from samples of the null distribution and `alpha` is a numeric object in $(0, 1]$ specifying the quantile with default value $\alpha = 0.1$.

Note that each set of samples is provided as a column vector; typically $p \gg n$.

The function is an S4 method with implementations defined for the following signatures:

- `pval_null` = "`matrix`"

  Input `pval_null` is a standard R "`numeric matrix`," and only methods defined in base R and the **stats** package are used to obtain estimates;

- `pval_null` = "`big.matrix`"

Input `pval_null` is an object of class "`big.matrix`" as defined by package **bigmemory**, and **RcppArmadillo** methods are used to obtain estimates; and

- `pval_null` = "`ff_matrix`"

  Input `pval_null` is an object of class "`ff_matrix`" as defined by package **ff**, and methods defined in packages **ff** and **ffbase** are used to obtain estimates.

The **bigmemory** and **ff** packages offer file-based access to data, which can accommodate larger matrices. Though highly optimized, these methods can have longer run times.

signalProp()

This function estimates the signal proportion as follows. First, the estimators using each bounding sequence are calculated as

$$\hat{\pi}_{0.5} = \max_{1\le j\le p} \frac{j/p - \overline{\Phi}(z_{(j)}) - c_{\alpha,0.5}\sqrt{\overline{\Phi}(z_{(j)})}}{1-\overline{\Phi}(z_{(j)})} \qquad \text{and} \qquad \hat{\pi}_{1} = \max_{1\le j\le p} \frac{j/p - \overline{\Phi}(z_{(j)}) - c_{\alpha,1}\overline{\Phi}(z_{(j)})}{1-\overline{\Phi}(z_{(j)})}$$

where $z_{(j)}$ are the test statistics and $c_{\alpha,0.5}$ and $c_{\alpha,1}$ are the estimated bounding sequences defined previously. The estimated signal proportion is taken as $\hat{\pi} = \max\{\hat{\pi}_{0.5}, \hat{\pi}_1\}$.

This estimator is implemented through function *signalProp()* and can be called to estimate the signal proportions given estimators for the bounding sequences

```
signalProb(pval, ..., c05, c1)
```

or to estimate both the bounding sequences and the signal proportions

```
signalProb(pval, pval_null, ..., alpha = 0.1)
```

where input `pval` is a vector-*like* object of p-values. Inputs `c05` and `c1` are the estimated bounding sequences $c_{\alpha,0.5}$ and $c_{\alpha,1}$, respectively. Inputs `pval_null` and α are as defined for *cSeq()*.

Recall that formal arguments after an ellipsis (...) must be provided as named inputs.

The function is an S4 method with implementations as follows:

If estimated bounding sequences are provided as numeric objects through named inputs `c05` and `c1`, and input `pval_null` is either not provided ("missing") or is set to NULL,
the following S4 signatures are defined

- `pval` = "`numeric`"

  If input `pval` is a standard R "`numeric vector`," only methods defined in base R and the **stats** package are used to obtain estimates.

- `pval` = "`big.matrix`"

  If input `pval` is an object of class "`big.matrix`" as defined by package **bigmemory** with a single row or a single column, **RcppArmadillo** methods are used to obtain estimates.

- `pval` = "`ff_vector`"

  If input `pval` is an object of class "`ff_vector`" as defined by package **ff**, methods defined in packages **ff** and **ffbase** are used to obtain estimates.

The second input configuration is equivalent to calling `cSeq(pval_null, alpha)` and `signalProp(pval, c05, c1)` sequentially. Thus, the signatures defined for `pval_null` for *cSeq()* in the preceding subsection and those for `pval` above apply. Note that the class of the objects used for `pval` and `pval_null` do not have to be related, e.g. `pval` = "`numeric`" and `pval_null` = "`big.matrix`" is an accepted combination.

Further there are signatures that we do not explicitly mention here that have been included to accommodate vector-*like* classes. For example, if `pval` = "`matrix`" with a single column or a single row, methods have been implemented to identify these cases and process the input data into vector form. Similarly for objects of class "`ff_array`."

fnpOpt()

The *fnpOpt()* function estimates

$$\widehat{FNP}(z_{(j)}) = \max\{1 - j/\hat{s} + (p - \hat{s})\overline{\Phi}(z_{(j)})/\hat{s}, 0\},$$

where $z_{(j)}, j = 1, \ldots, p$, are the test statistics, $\hat{s} = \lceil p * \hat{\pi} \rceil$, and $\hat{\pi}$ is the estimated signal proportion defined previously.

This functions can be called to estimate the FNP given an estimated number of signals

```
fnpOpt(pval, ..., beta, sHat)
```

or to perform all three steps of the framework, i.e., estimate the bounding sequence, signal proportion, and FNP

```
fnpOpt(pval, pval_null, ..., beta, alpha = 0.1)
```

where input `pval` is a vector-*like* object of p-values and `beta` is the tolerance threshold. Input `sHat` is the estimated number of signal variables. Inputs `pval_null` and `alpha` are as defined for *cSeq()*.

Again we mention that formal arguments after an ellipsis (...) must be provided as named inputs.

Similar to the description given for *signalProb()*, this function is an S4 method with implementations as follows:

If `sHat` and `beta` are provided, and input `pval_null` is either not provided ("missing") or is set to NULL, the following S4 signatures are defined

- `pval` = "`numeric`"

  If `pval` is a standard R "`numeric vector`," only methods defined in base R and the **stats** package are used to obtain the estimates.

- `pval` = "`big.matrix`"

  If `pval` is an object of class "`big.matrix`" as defined by package **bigmemory** with a single row or a single column, **RcppArmadillo** methods are used to obtain the estimates.

- `pval` = "`ff_vector`"

  If `pval` is an object of class "`ff_vector`" as defined by package **ff**, methods defined in packages **ff** and **ffbase** are used to obtain the estimates.

The full framework call structure simply uses the inputs to call `cSeq(pval_null, alpha)`, `signalProp(pval, c05, c1)`, and `fnpOpt(pval, beta, sHat)` sequentially. Thus, the signatures previously discussed for these functions apply.

As for *signalProb()* there are signatures that we do not explicitly mention here that have been included to accommodate vector-*like* classes.

print()

A convenience function to provide results in a tidy format.

Examples

Data

We use the dataset provided with the package, wsiData, to illustrate a typical analysis. This dataset is a publicly available high-throughput genomic dataset (Buhlmann et al. (2014)) providing gene expression levels and the rate of riboflavin production with *Bacillus subtilis* for 71 individuals. The data comprises 4088 gene expression levels and the logarithm of the riboflavin production rate ($q_RRIBFLC).

The data can be loaded in the usual way

```
data(wsiData)
```

```
dim(wsiData)
```

```
## [1]   71 4089
```

Consider the summary statistics of only the outcome

```
summary(wsiData$q_RIBFLC)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  -9.966  -7.688  -6.948  -7.159  -6.449  -5.673
```

We see that the range of the logarithm of the riboflavin production rate, y, is -9.9658 $\leq y \leq$ -5.673. The range of the gene expression levels, ℓ, is 3.3228 $\leq \ell \leq$ 14.3896.

```
summary(c(data.matrix(wsiData[,-1L])))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   3.323   6.946   7.665   7.669   8.352  14.390
```

P-values

We first obtain the test statistics using marginal regression

```
dm <- data.matrix(frame = wsiData)
pval <- apply(X = dm[,-1L],
              MARGIN = 2L,
              FUN = function(x,y) {
                summary(object = lm(formula = y~x))$coef[2L,4L]
              },
```

```
              y = dm[,1L])
p <- length(x = pval)
```

Next, we obtain 1000 sets of samples from the null distribution and their corresponding p-values

```
n <- 1000L
sig <- stats::cor(x = dm[,-1L])
zz <- MASS::mvrnorm(n = n, mu = rep(x = 0.0, times = p), Sigma = sig)
pval_null <- t(x = {1.0 - stats::pnorm(q = abs(x = zz))}*2.0)
```

where we have transposed the p-value matrix to put it into the expected input format.

Though our data is not of sufficient dimension to warrant the use of more memory efficient storage and access, we will define variables of class "`big.matrix`" and "`ff_matrix`" for illustration purposes.

```
pval_nullBM <- as.big.matrix(pval_null, type = "double")
pval_nullFF <- ff(vmode = "double", dim = c(p,n), pval_null)
```

cSeq()

The first step of the framework is to estimate the bounding sequences. We will set $\alpha = 0.2$.

Using the standard R "`matrix`" object, the call takes the form

```
cs <- cSeq(pval_null = pval_null, alpha = 0.2)
```

```
## estimating bounding sequence using 1000 samples, each containing 4088 variables
```

A message is generated indicating the number of samples (n) and variables (p). An S3 object of class "`wsiHD`" comprising a list object is returned with element $c05 and $c1.

```
cs
```

```
## Bounding Sequences
##   c05:  0.3165
##    c1:  0.9809
```

For the "`big.matrix`" and "`ff_matrix`" objects,

```
cSeq(pval_null = pval_nullBM, alpha = 0.2)
```

```
## estimating bounding sequence using 1000 samples, each containing 4088 variables
```

```
## Bounding Sequences
##   c05:  0.3166
##    c1:  0.9809
```

```
cSeq(pval_null = pval_nullFF, alpha = 0.2)
```

```
## estimating bounding sequence using 1000 samples, each containing 4088 variables
```

```
## Bounding Sequences
##   c05:  0.3164
##    c1:  0.9809
```

yield similar return objects. Notice, however, that the estimates are not identical. Any differences are due to the algorithm used to obtain the $(1 - \alpha)$-th quantile. In base R, there are nine algorithms available to obtain quantiles. We have opted to use the default algorithm in this implementation. However, this default algorithm is not the same as that implemented by Armadillo (the underpinnings for the "`big.matrix`" implementation),

which uses type = 5, nor that of **ff**, which uses type = 1. Thus, estimates of the bounding sequences might differ slightly for inputs of different classes but with equivalent p-values. For large p, any differences will be very small.

signalProp()

The second step of the framework is to use the estimated bounding sequences to obtain the estimated signal proportion.

```
piHat <- signalProp(pval = pval, c05 = cs$c05, c1 = cs$c1)
```

An S3 object of class "`wsiHD`" comprising a list object is returned containing $\hat{\pi}$ ($piHat), $\hat{\pi}_{0.5}$ ($piHat05), and $\hat{\pi}_1$ ($piHat1).

```
piHat
```

```
## Signal Proportions
##      piHat:  0.0978
##    piHat05:  0.0791
##     piHat1:  0.0978
```

These results can be equivalently obtained using a slightly different input structure. Namely,

```
signalProp(pval = pval, pval_null = pval_null, alpha = 0.2)
```

```
## estimating bounding sequence using 1000 samples, each containing 4088 variables
```

```
## Bounding Sequences
##   c05:  0.3165
##    c1:  0.9809
## Signal Proportions
##      piHat:  0.0978
##    piHat05:  0.0791
##     piHat1:  0.0978
```

Here we see that the bounding sequences were estimated internally and are returned through the value object.

Again, we see that using the alternative input classes leads to slightly different results due to the underlying quantile algorithms.

```
signalProp(pval = pval, pval_null = pval_nullBM, alpha = 0.2)
```

```
## estimating bounding sequence using 1000 samples, each containing 4088 variables
```

```
## Bounding Sequences
##   c05:  0.3166
##    c1:  0.9809
## Signal Proportions
##      piHat:  0.0978
##    piHat05:  0.0791
##     piHat1:  0.0978
```

```
signalProp(pval = pval, pval_null = pval_nullFF, alpha = 0.2)
```

```
## estimating bounding sequence using 1000 samples, each containing 4088 variables
```

```
## Bounding Sequences
##   c05:  0.3164
```

```
##    c1:  0.9809
## Signal Proportions
##      piHat:  0.0978
##    piHat05:  0.0792
##     piHat1:  0.0978
```

fnpOpt()

The final step of the framework is to use the estimated signal proportion to obtain the estimated number of signal variables and then estimate the FNP. We will take $\beta = 0.1$ in this example.

```
sHat <- ceiling(x = p*piHat$piHat)
fnp <- fnpOpt(pval = pval, beta = 0.1, sHat = sHat)
```

An S3 object of class "`wsiHD`" comprising a list object is returned containing $ind, the rank satisfying the threshold condition; $pvalue, the maximum p-value for the variables that satisfy the threshold condition; and $FNP, the estimated FNP for all variables. Note that the class of the $FNP element will depend on the class input `pval`.

```
fnp
```

```
## Number of Signals:  398
## max p-value:  0.01049627
## Summary of FNP:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.00000 0.00000 0.00000 0.05199 0.00000 0.99750
```

Though the full p-dimensional vector of $\widehat{FNP}$ is returned, the print function only displays the summary statistics.

Similar in spirit to the *signalProp()* function, these results can be equivalently obtained using a slightly different input structure. Namely,

```
fnpOpt(pval = pval, pval_null = pval_null, beta = 0.1, alpha = 0.2)
```

```
## estimating bounding sequence using 1000 samples, each containing 4088 variables
```

```
## Bounding Sequences
##   c05:  0.3165
##    c1:  0.9809
## Signal Proportions
##      piHat:  0.0978
##    piHat05:  0.0791
##     piHat1:  0.0978
## Number of Signals:  398
## max p-value:  0.01049627
## Summary of FNP:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.00000 0.00000 0.00000 0.05199 0.00000 0.99750
```

Here we see that the bounding sequences were estimated internally as well as the estimated signal proportions. These are returned through the value object.

And using the alternative input classes

```
fnpOpt(pval = pval, pval_null = pval_nullBM, beta = 0.1, alpha = 0.2)
```

```
## estimating bounding sequence using 1000 samples, each containing 4088 variables
```

```
## Bounding Sequences
##   c05:  0.3166
##    c1:  0.9809
## Signal Proportions
##      piHat:  0.0978
##    piHat05:  0.0791
##     piHat1:  0.0978
## Number of Signals:  398
## max p-value:  0.01049627
## Summary of FNP:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.00000 0.00000 0.00000 0.05199 0.00000 0.99750
```

```
fnpOpt(pval = pval, pval_null = pval_nullFF, beta = 0.1, alpha = 0.2)
```

```
## estimating bounding sequence using 1000 samples, each containing 4088 variables
```

```
## Bounding Sequences
##   c05:  0.3164
##    c1:  0.9809
## Signal Proportions
##      piHat:  0.0978
##    piHat05:  0.0792
##     piHat1:  0.0978
## Number of Signals:  398
## max p-value:  0.01049627
## Summary of FNP:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.00000 0.00000 0.00000 0.05199 0.00000 0.99750
```

References

Jeng, X. J. and Hu, Y. (2021). Weak signal inference under dependence and sparsity, submitted.

Jeng, X. J. (2021). Estimating the proportion of signal variables under arbitrary covariance dependence. <arXiv:2102.09053>.

Buhlmann, P., Kalisch, M. and Meier, L. (2014). High-Dimensional statistics with a view toward applications in Biology. *Annual Review of Statistics and Its Application*, 1, 255–278.