Package 'SmartSVA'

January 20, 2025

Type Package

Title Fast and Robust Surrogate Variable Analysis

Version 0.1.3

Author Jun Chen <Chen.Jun2@mayo.edu>, Ehsan Behnam <behnamgh@usc.edu>

Maintainer Jun Chen <Chen.Jun2@mayo.edu>

Description

Introduces a fast and efficient Surrogate Variable Analysis algorithm that captures variation of unknown sources (batch effects) for high-dimensional data sets. The algorithm is built on the 'irwsva.build' function of the 'sva' package and proposes a revision on it that achieves an order of magnitude faster running time while trading no accuracy loss in return.

License GPL-3

Depends R (>= 3.1.0), sva, isva, RSpectra

Imports Rcpp, stats, utils

LinkingTo Rcpp, RcppEigen

Encoding UTF-8

LazyData true

NeedsCompilation yes

Repository CRAN

Date/Publication 2017-05-28 07:22:56 UTC

Contents

`smartsva.cpp` *Fast and robust surrogate variable analysis.*

Description

The traditional SVA uses a fixed number of iterations. In highly-confounded scenarios (i.e., biological effects are confounded with batch effects), a large number of iterations are needed to achieve optimal results and better false positive control. SmartSVA thus imposes an explicit convergence criterion to improve the robustness of SVA. Moreover, we use three strategies to accelerate the computation: (1) more efficient initialization by using `alpha`; (2) QR decomposition to reduce the computational complexity; (3) Further acceleration using RcppEigen. Based on empirical studies, SmartSVA achieves is 10-50 times faster than traditional SVA on large data sets if the same convergence criterion is imposed.

Usage

```
smartsva.cpp(dat, mod, mod0 = NULL, n.sv, B = 100, alpha = 0.25,
  epsilon = 0.001, VERBOSE = F)
```

Arguments

`dat`	the measurement matrix, where rows are features and columns are samples.
`mod`	the model matrix being used to fit the data.
`mod0`	the null model matrix.
`n.sv`	the number of surrogate variables to estimate. The use of random matrix theory is recommended to estimate n.sv. See the example for more details.
`B`	the maximum iteration number.
`alpha`	determines the initial point for optimization which affects the convergence rate. Small values reduce the number of iterations needed to reach convergence; however, the solution could be trapped into a local optimum if the value is too small. The default value 0.25 works well across a wide range of tested scenarios. If computation is not a big concern, setting `alpha = 1` will ensure optimality, producing the same result as traditional SVA if the same convergence criterion is used.
`epsilon`	the convergence threshold. The Spearman's correlation between posterior probabilities of consecutive iterations of the algorithm is compared to epsilon. Empirical evaluation on several data sets revealed epsilon=0.005 gives very reasonable results. However, we suggest epsilon=1e-3 as a conservative threshold.
`VERBOSE`	a logical variable. If TRUE, prints some details about iterative progress of the algorithm.

Value

Returns a list containing the surrogate variables and some meta data about the convergence criterion.

References

Jun Chen, Ehsan Behnam, Jinyan Huang, Miriam F. Moffatt, Daniel J. Schaid, Liming Liang, Xihong Lin. Fast and robust adjustment of cell mixtures in epigenome-wide association studies with SmartSVA. BMC Genomics, 2017 18:413

Examples

```
## Methylation M values (CpG by Sample)
Y <- matrix(rnorm(20*1000), 1000, 20)
df <- data.frame(pred=gl(2, 10))

## Determine the number of SVs
Y.r <- t(resid(lm(t(Y) ~ pred, data=df)))

## Add one extra dimension to compensate potential loss of 1 degree of freedom
##  in confounded scenarios (very important)
n.sv <- EstDimRMT(Y.r, FALSE)$dim + 1
mod <- model.matrix( ~ pred, df)
sv.obj <- smartsva.cpp(Y, mod, mod0=NULL, n.sv=n.sv)

## Speed comparison to traditional SVA
## Not run:
## Methylation M values (CpG by Sample, 27K by 1,000)
require(sva)
require(SmartSVA)
Y <- matrix(rnorm(1000*27000), 27000, 1000)
df <- data.frame(pred=gl(2, 500))

## Determine the number of SVs
Y.r <- t(resid(lm(t(Y) ~ pred, data=df)))
n.sv <- 50
mod <- model.matrix( ~ pred, df)
system.time(sv.obj1 <- smartsva.cpp(Y, mod, mod0=NULL, B=5, alpha = 1, VERBOSE=TRUE, n.sv=n.sv))
system.time(sv.obj2 <- sva(Y, mod, mod0=NULL, B=5,  n.sv=n.sv))

## Check if the solutions are the same
head(sv.obj1$sv)
head(sv.obj2$sv)

## End(Not run)
```

Index