Package 'SETPath'

February 19, 2015

Type Package

Title Spiked Eigenvalue Test for Pathway data

Version 1.0

Date 2014-12-26

Author Patrick Danaher

Maintainer Patrick Danaher <patrickjdanaher@gmail.com>

Description Tests gene expression data from a biological pathway for biologically meaningful differences in the eigenstructure between two classes. Specifically, it tests the null hypothesis that the two classes' leading eigenvalues and sums of eigenvalues are equal. A pathway's leading eigenvalue arguably represents the total variability due to variability in pathway activity, while the sum of all its eigenvalues represents the variability due to pathway activity and to other, unregulated causes. Implementation of the method described in Danaher (2015), ``Covariance-based analyses of biological pathways".

License GPL-2

NeedsCompilation no

Repository CRAN

Date/Publication 2015-02-02 22:37:42

R topics documented:

SETPath-package *Spiked Eigenvalue Test for Pathway data*

Description

Tests gene expression data from a biological pathway for biologically meaningful differences in the eigenstructure between two classes. Specifically, it tests the null hypothesis that the two classes' leading eigenvalues and sums of eigenvalues are equal. A pathway's leading eigenvalue arguably represents the total variability due to variability in pathway activity, while the sum of all its eigenvalues represents the variability due to pathway activity and to other, unregulated causes.

Details

Package:	SETPath
Type:	Package
Version:	1.0
Date:	2014-11-26
License:	GPL-2

The SETPath provides two functions for the implementation of the SETPath method. The function setpath() runs the test on data from a single pathway. The function setpath.wrapper() runs the test on a list of pathways within a dataset.

Author(s)

Patrick Danaher

Maintainer: Patrick Danaher <patrickjdanaher at gmail.com>

References

Patrick Danaher, Debashis Paul, and Pei Wang. "Covariance-based analyses of biological pathways." Biometrika (2015)

Examples

```
#load data:
data(setpath.data)
# identify desired gene list:
genes.in.pathway = pathwaygenes[[1]]
# run test using theoretical quantiles to derive a p-value:
setpath(d1[,genes.in.pathway],d2[,genes.in.pathway],M=1,transform=NULL,verbose=TRUE,minalpha=NULL,
normalize=TRUE,pvalue="chisq")
# now using a permutation test:
setpath(d1[,genes.in.pathway],d2[,genes.in.pathway],M=1,transform=NULL,verbose=TRUE,minalpha=NULL,
normalize=TRUE,pvalue="permutation",npermutations=1000)
# now using the "transform" argument to test the null hypothesis that variability unrelated to the
```

```
#  first principal component (i.e. the sum of the second through final eigenvalues) is the same
#  between classes:
setpath(d1[,genes.in.pathway],d2[,genes.in.pathway],M=1,transform=c(-1,1),verbose=TRUE,
minalpha=NULL,normalize=TRUE,pvalue="chisq",npermutations=1000)
# now using the "transform" argument to test the compound null hypothesis that the second and third
#  eigenvalues are the same between classes:
linear.transformation = matrix(c(0,1,0,0,0,0,1,0),4)
print(linear.transformation)
setpath(d1[,genes.in.pathway],d2[,genes.in.pathway],M=3,transform=linear.transformation,
verbose=TRUE,minalpha=NULL,normalize=TRUE,pvalue="chisq",npermutations=1000)

# use the function setpath.wrapper to analyze several pathways simultaneously
setpath.wrapper(d1,d2,pathwaygenes,pathwaynames,M=1,transform=NULL,minalpha=NULL,normalize=TRUE,
pvalue="chisq",npermutations=10000)
```

---

d1 — *Example data for the SETPath method - dataset from class 1*

Description

An example dataset from one class, with 25 samples and 80 genes.

Usage

```
data(setpath.data)
```

Format

The format is: d1 : num [1:25, 1:80] 0.701 -2.107 0.624 -0.885 0.344 ... ..- attr(*, "dimnames")=List of 2 .. ..$: chr [1:25] "control1" "control2" "control3" "control4" ... .$: chr [1:80] "gene1" "gene2" "gene3" "gene4" ...

---

d2 — *Example data for the SETPath method - dataset from class 2*

Description

An example dataset from one class, with 50 samples and 80 genes.

Usage

```
data(setpath.data)
```

Format

The format is: d2 : num [1:50, 1:80] 1.003 -1.323 0.906 0.708 1.131 ... ..- attr(*, "dimnames")=List of 2 .. ..$: chr [1:50] "case1" "case2" "case3" "case4" ... .$: chr [1:80] "gene1" "gene2" "gene3" "gene4" ...

pathwaygenes *Defines the gene memberships of the pathways in the example dataset*

Description

A list in which each entry is a character string giving the names of the genes belonging to a pathway

Usage

```
data(setpath.data)
```

Format

The format is: List of 4 ..$: chr [1:20] "gene63" "gene33" "gene21" "gene19" ... ..$: chr [1:50] "gene19" "gene25" "gene22" "gene30" ... ..$: chr [1:25] "gene72" "gene32" "gene62" "gene76" ... ..$: chr [1:30] "gene42" "gene7" "gene45" "gene39" ...

pathwaynames *Names of the pathways used in the example.*

Description

A character vector giving the names of each pathway in the example.

Usage

```
data(setpath.data)
```

Format

The format is: chr [1:4] "pathway 1" "pathway 2" "pathway 3" "pathway 4"

setpath *Runs the Spiked Eigenvalue Test for Pathway data (SETPath) on data from a single pathway*

Description

Compares a pathway's gene expression data from two classes, testing the null hypothesis that the first eigenvalue and the sum of the eigenvalues are equal between classes.

Usage

```
setpath(d1, d2, M = 1, transform = NULL, verbose = FALSE, minalpha = NULL,
normalize = TRUE, pvalue = "chisq", npermutations = 10000)
```

Arguments

d1	A n1*p matrix from the pathway's data in the first class
d2	A n2*p matrix from the pathway's data in the second class, with the same genes (column names) as d1
M	The null hypothesis specifies that the first M eigenvalues are equal between the two classes. SETPath was conceived as a test of just the first eigenvalue and the sum of the eigenvalues (M=1), but cases where multiple leading eigenvalues are of biological interest may indicate M>1.
transform	Default NULL. Otherwise, a matrix or vector specifying a linear transformation of the null hypothesis. The use of the transform argument modifies the standard SETPath test of equality of the first eigenvalue and the sum of eigenvalues between classes. Specifically, if the null hypothesis is that (alpha0.1, ..., alpha0.M,T0) = (alpha1.1, ..., alpha1.M,T1), the transform argument changes the null hypothesis to t(transform)%*%(alpha0.1, ..., alpha0.M,T0) = t(transform)%*%(alpha1.1, ..., alpha1.M,T1).
verbose	Indicates whether to return more than the p-value.
minalpha	Can be used to tweak the estimation of the leading eigenvalues under the null hypothesis. If NULL, eigenvalues are truncated below at 1 + 2*sqrt(gamma). Otherwise, eigenvalues are truncated below at 1 + sqrt(gamma) + minalpha.
normalize	SETPath assumes that the unspiked eigenvalues of each class's dataset are equal to 1. If the normalize argument is set to TRUE, the data will be rescaled by the average of the median of the non-zero eigenvalues of the two datasets. A further adjustment is performed when n<p.
pvalue	If pvalue=="chisq", the theoretical distribution of the test statistic will be used to compute a p-value. Alternatively, use pvalue=="permutation".
npermutations	If pvalue == "permutation", the number of permutations.

Value

pval	The p-value from SETPath

If verbose==TRUE, the additional values are output:

a0	Estimates of the leading M eigenvalues, assuming the null hypothesis is true.
correction	The correction factors applied to the between-class differences between the M leading eigenvalues. (Correction factors are needed to account for the sample size-dependent bias in empirical eigenvalues.)
covQ	An (M+1)*(M+1) matrix giving the estimated covariance of the between-class differences in the leading M eigenvalues and the sum of the eigenvalues.
m	M, the number of leading eigenvalues included in the null hypothesis.

Author(s)

Patrick Danaher

References

Patrick Danaher, Debashis Paul, and Pei Wang. "Covariance-based analyses of biological pathways." Biometrika (2015)

Examples

```
#load data:
data(setpath.data)
# identify desired gene list:
genes.in.pathway = pathwaygenes[[1]]
# run test using theoretical quantiles to derive a p-value:
setpath(d1[,genes.in.pathway],d2[,genes.in.pathway],M=1,transform=NULL,verbose=TRUE,minalpha=NULL,
normalize=TRUE,pvalue="chisq")
# now using a permutation test:
setpath(d1[,genes.in.pathway],d2[,genes.in.pathway],M=1,transform=NULL,verbose=TRUE,minalpha=NULL,
normalize=TRUE,pvalue="permutation",npermutations=1000)
# now using the "transform" argument to test the null hypothesis that variability unrelated to the
#  first principal component (i.e. the sum of the second through final eigenvalues) is the same
#  between classes:
setpath(d1[,genes.in.pathway],d2[,genes.in.pathway],M=1,transform=c(-1,1),verbose=TRUE,
minalpha=NULL,normalize=TRUE,pvalue="chisq",npermutations=1000)
# now using the "transform" argument to test the compound null hypothesis that the second and third
#  eigenvalues are the same between classes:
linear.transformation = matrix(c(0,1,0,0,0,0,1,0),4)
print(linear.transformation)
setpath(d1[,genes.in.pathway],d2[,genes.in.pathway],M=3,transform=linear.transformation,
verbose=TRUE,minalpha=NULL,normalize=TRUE,pvalue="chisq",npermutations=1000)



## The function is currently defined as
function (d1, d2, M = 1, transform = NULL, verbose = FALSE, minalpha = NULL,
    normalize = TRUE, pvalue = "chisq", npermutations = 10000)
```

```
{
    p = dim(d1)[2]
    n1 = dim(d1)[1]
    n2 = dim(d2)[1]
    if (normalize) {
      e1 = eigen(cov(d1),symmetric=TRUE,only.values=TRUE)$values
      e2 = eigen(cov(d2),symmetric=TRUE,only.values=TRUE)$values
      if(n1>p){medianeigen1 = median(e1)}
      if(n2>p){medianeigen2 = median(e2)}
      if(n1<=p){medianeigen1 = median(e1[e1>1e-12])*n1/p}
      if(n2<=p){medianeigen2 = median(e2[e2>1e-12])*n2/p}
      scaling.factor = mean(medianeigen1,medianeigen2)
      d1 = d1/sqrt(scaling.factor)
      d2 = d2/sqrt(scaling.factor)
    }
    p = dim(d1)[2]
    n1 = dim(d1)[1]
    n2 = dim(d2)[1]
    n = c(n1, n2)
    w = n/sum(n)
    d = list(d1, d2)
    g1 = p/n1
    g2 = p/n2
    g = c(g1, g2)
    covs = list()
    covs[[1]] = cov(d1)
    covs[[2]] = cov(d2)
    sighat = list(cov(d1), cov(d2))
    e1 = eigen(sighat[[1]], symmetric = TRUE, only.values = TRUE)$values
    e2 = eigen(sighat[[2]], symmetric = TRUE, only.values = TRUE)$values
    L = cbind(e1[1:M], e2[1:M])
    T1 = sum(e1)
    T2 = sum(e2)
    T = c(T1, T2)
    alphabar = matrix(NA, M, 2)
    a0 = QLcorrection = c()
    for (m in 1:M) {
        eigencorrect = unbias.eigens(L[m, ], g, w, minalpha)
        alphabar[m, ] = eigencorrect$a
        a0[m] = eigencorrect$a0
        QLcorrection[m] = eigencorrect$QLcorrection
    }
    thresh = (1 + sqrt(g))^2 + sqrt(2 * log(n)/n)
    mhat = c()
    mhat[1] = max(sum(e1 > thresh[1]), M)
    mhat[2] = max(sum(e2 > thresh[2]), M)
    spikes = list()
    for (k in 1:2) {
        spikes[[k]] = rep(NA, mhat[k])
    }
    for (k in 1:k) {
        for (m in 1:mhat[k]) {
            tempeigencorrect = unbias.eigens(c(e1[m], e2[m]),
```

```
            g, w, minalpha)
        spikes[[k]][m] = tempeigencorrect$a[k]
    }
}
covQ = matrix(0, M + 1, M + 1)
varT = c()
for (k in 1:2) {
    varT[k] = 2 * (sum(a0^2)/n[k] + (p - M)/n[k])
    if (mhat[k] > M) {
        varT[k] = varT[k] + 2/n[k] * (sum((spikes[[k]][(M +
            1):mhat[k]])^2) - (mhat[k] - M))
    }
}
covQ[M + 1, M + 1] = sum(varT)
for (m in 1:M) {
    rho = theta = varLs = c()
    c0 = (1/g[1] + 1/g[2])^2 * (a0[m] - 1)^2
    for (k in 1:2) {
        rho[k] = a0[m] * (1 + g[k]/(a0[m] - 1))
        deriv.f.k = 0.5 * (1 + (rho[k] - 1 - g[k])/sqrt((rho[k] -
            1 - g[k])^2 - 4 * g[k]))
        theta[k] = 1 + (g[1] - g[2])/c0 * deriv.f.k/g[k]
        varLs[k] = 2 * a0[m]/n[k] * theta[k]^2 * rho[k]/(1 +
            a0[m] * g[k]/((a0[m] - 1)^2 - g[k]))
    }
    covQ[m, m] = sum(varLs)
}
for (m in 1:M) {
    rho = theta = covLTs = c()
    c0 = (1/g[1] + 1/g[2])^2 * (a0[m] - 1)^2
    for (k in 1:2) {
        rho[k] = a0[m] * (1 + g[k]/(a0[m] - 1))
        deriv.f.k = 0.5 * (1 + (rho[k] - 1 - g[k])/sqrt((rho[k] -
            1 - g[k])^2 - 4 * g[k]))
        theta[k] = 1 + (g[1] - g[2])/c0 * deriv.f.k/g[k]
        covLTs[k] = 2 * a0[m]/n[k] * theta[k] * rho[k]/(1 +
            a0[m] * g[k]/((a0[m] - 1)^2 - g[k]))
    }
    covLT = sum(covLTs)
    covQ[m, M + 1] = covQ[M + 1, m] = covLT
}
Q = c(L[, 1] - L[, 2] - QLcorrection, T1 - T2)
A = transform
if (length(transform) == 0) {
    A = diag(M + 1)
}
A=as.matrix(A)
if(dim(A)[1]!=M+1){stop("Dimension of linear transformation does not match
the dimension of the null hypothesis.")}
stat = t(Q) %*% A %*% solve(t(A) %*% covQ %*% A) %*% t(A) %*%
    Q
out = list()
out$stat = stat
```

```
    if (pvalue == "chisq") {
        out$pval = 1 - pchisq(stat, dim(A)[2])
    }
    if (pvalue == "permutation") {
        d.combined = rbind(d1, d2)
        permstats = c()
        for (i in 1:npermutations) {
            prows1 = sample(1:dim(d.combined)[1], dim(d1)[1],
                replace = FALSE)
            prows2 = setdiff(1:dim(d.combined)[1], prows1)
            permstats[i] = setpath(d1 = d.combined[prows1, ], d2 = d.combined[prows2,
                ], M = M, transform = transform, verbose = FALSE,
                minalpha = minalpha, normalize = FALSE, pvalue = "chisq")$stat
        }
        out$pval = mean(as.vector(stat) < permstats)
    }
    if (verbose) {
        out$stats = rbind(L, c(T1, T2))
        out$a0 = a0
        out$correction = QLcorrection
        out$covQ = covQ
        out$m = m
    }
    return(out)
}
```

`setpath.data` *Example data for the SETPath method*

Description

An example dataset containing genes from 4 pathways.

Usage

```
data(setpath.data)
```

Format

The format is: List of 4 $ d1 : num [1:25, 1:80] 0.701 -2.107 0.624 -0.885 0.344 ... ..- attr(*, "dimnames")=List of 2 .. ..$: chr [1:25] "control1" "control2" "control3" "control4" ... .$: chr [1:80] "gene1" "gene2" "gene3" "gene4" ... $ d2 : num [1:50, 1:80] 1.003 -1.323 0.906 0.708 1.131 ... ..- attr(*, "dimnames")=List of 2 .. ..$: chr [1:50] "case1" "case2" "case3" "case4" ... .$: chr [1:80] "gene1" "gene2" "gene3" "gene4" ... $ pathwaynames: chr [1:4] "pathway 1" "pathway 2" "pathway 3" "pathway 4" $ pathwaygenes:List of 4 ..$: chr [1:20] "gene63" "gene33" "gene21" "gene19" ... ..$: chr [1:50] "gene19" "gene25" "gene22" "gene30" ... ..$: chr [1:25] "gene72" "gene32" "gene62" "gene76" ... ..$: chr [1:30] "gene42" "gene7" "gene45" "gene39" ...

`setpath.wrapper` *Runs the Spiked Eigenvalue Test for Pathway data (SETPath) on multiple pathways in a dataset*

Description

For pathway in a list, compares the pathway's gene expression data from two classes, testing the null hypothesis that the first eigenvalue and the sum of the eigenvalues are equal between classes. Returns a matrix of results.

Usage

```
setpath.wrapper(d1, d2, pathwaygenes, pathwaynames, M = 1, transform = NULL,
minalpha = NULL, normalize = TRUE, pvalue = "chisq", npermutations = 10000)
```

Arguments

`d1`	A n1*p matrix from the pathway's data in the first class
`d2`	A n2*p matrix from the pathway's data in the second class, with the same genes (column names) as d1
`pathwaygenes`	A list of vectors containing the names of the genes in each of the pathways
`pathwaynames`	A vector of the names of the pathways
`M`	The null hypothesis specifies that the first M eigenvalues are equal between the two classes. SETPath was conceived as a test of just the first eigenvalue and the sum of the eigenvalues (M=1), but cases where multiple leading eigenvalues are of biological interest may indicate M>1.
`transform`	Default NULL. Otherwise, a matrix or vector specifying a linear transformation of the null hypothesis. The use of the transform argument modifies the standard SETPath test of equality of the first eigenvalue and the sum of eigenvalues between classes. Specifically, if the null hypothesis is that (alpha0.1, ..., alpha0.M,T0) = (alpha1.1, ..., alpha1.M,T1), the transform argument changes the null hypothesis to t(transform)%*%(alpha0.1, ..., alpha0.M,T0) = t(transform)%*%(alpha1.1, ..., alpha1.M,T1).
`minalpha`	Can be used to tweak the estimation of the leading eigenvalues under the null hypothesis. If NULL, eigenvalues are truncated below at 1 + 2*sqrt(gamma). Otherwise, eigenvalues are truncated below at 1 + sqrt(gamma) + minalpha.
`normalize`	SETPath assumes that the unspiked eigenvalues of each class's dataset are equal to 1. If the normalize argument is set to TRUE, the data will be rescaled by the average of the median of the non-zero eigenvalues of the two datasets. A further adjustment is performed when n<p.
`pvalue`	If pvalue=="chisq", the theoretical distribution of the test statistic will be used to compute a p-value. Alternatively, use pvalue=="permutation".
`npermutations`	If pvalue == "permutation", the number of permutations.

Value

A matrix of results, with a row for each pathway and (2*(M+2)) columns. For each pathway, the following is reported: the number of genes in the pathway, the leading M eigenvalues in each class, the sum of the eigenvalues in each class, and the p-value for the test.

Author(s)

Patrick Danaher

References

Patrick Danaher, Debashis Paul, and Pei Wang. "Covariance-based analyses of biological pathways." Biometrika (2015)

Examples

```
# use the function setpath.wrapper to analyze several pathways simultaneously
data(setpath.data)
setpath.wrapper(d1,d2,pathwaygenes,pathwaynames,M=1,transform=NULL,minalpha=NULL,normalize=TRUE,
pvalue="chisq",npermutations=10000)

## The function is currently defined as
function (d1, d2, pathwaygenes, pathwaynames, M = 1, transform = NULL,
    minalpha = NULL, normalize = TRUE, pvalue = "chisq", npermutations = 10000)
{
    K = length(pathwaynames)
    results = matrix(NA, K, 2 * (M + 1) + 2)
    dimnames(results)[[1]] = pathwaynames
    dimnames(results)[[2]] = c("n.genes", paste("alpha.0", 1:M,
        sep = "."), "T.0", paste("alpha.0", 1:M, sep = "."),
        "T.0", "pval")
    if (!identical(dimnames(d1)[[2]], dimnames(d2)[[2]])) {
        stop("d1 and d2 have different feature (column) names.")
    }
    for (k in 1:K) {
        missinggenes = setdiff(pathwaygenes[[k]], dimnames(d1)[[2]])
        if (length(missinggenes) > 0) {
            warning(c("The following pathway genes are missing from the dataset:",
                missinggenes))
            pathwaygenes[[k]] = intersect(pathwaygenes[[k]],
                dimnames(d1)[[2]])
        }
        temp = setpath(d1[, pathwaygenes[[k]]], d2[, pathwaygenes[[k]]],
            M = M, transform = transform, verbose = TRUE, minalpha = minalpha,
            normalize = normalize, pvalue = pvalue, npermutations = npermutations)
        results[k, ] = c(length(pathwaygenes[[k]]), temp$stats[,
            1], temp$stats[, 2], temp$pval)
    }
    return(results)
  }
```

unbias.eigens *Unbiased estimation of leading eigenvalues*

Description

A function called by setpath(), used to attain unbiased estimates of leading eigenvalues and to estimate the leading eigenvalues under the null hypothesis.

Usage

```
unbias.eigens(L, g, w, minalpha = NULL)
```

Arguments

L	A vector of length two containing a leading eigenvalue from each of the two datasets.
g	A vector of length two containing the gamma (p/n) values from the two datasets.
w	The weights to assign to the two classes when estimating common leading eigenvalues under the null hypothesis.
minalpha	Can be used to tweak the estimation of the leading eigenvalues under the null hypothesis. If NULL, eigenvalues are truncated below at 1 + 2*sqrt(gamma). Otherwise, eigenvalues are truncated below at 1 + sqrt(gamma) + minalpha.

Details

Called by the setpath() function, not useful on its own.

Value

QLcorrection	The correction factor to remove the bias in the difference between the two eigenvalues.
a0	The common eigenvalue estimated under the null hypothesis
a	The unbiased eigenvalue estimates from each class

Author(s)

Patrick Danaher

References

Patrick Danaher, Debashis Paul, and Pei Wang. "Covariance-based analyses of biological pathways." Biometrika (2015)

Examples

```
## The function is currently defined as
function (L, g, w, minalpha = NULL)
{
    if (length(minalpha) == 0) {
        minalpha = sqrt(max(g))
    }
    a = c()
    for (k in 1:2) {
        if (L[k] < (1 + sqrt(g[k]))^2) {
            a[k] = 1 + sqrt(max(g)) + minalpha
        }
        if (L[k] >= (1 + sqrt(g[k]))^2) {
            a[k] = ((1 + L[k] - g[k]) + sqrt((1 + L[k] - g[k])^2 -
                4 * L[k]))/2
        }
    }
    a0 = sum(a * w)
    a0 = max(c(a0, 1 + sqrt(g)))
    QLcorrection = (g[1] - g[2]) * a0/(a0 - 1)
    return(list(QLcorrection = QLcorrection, a0 = a0, a = a))
  }
```

Index