Package 'SPA3G'

January 27, 2015

Type Package

Title SPA3G: R package for the method of Li and Cui (2012)

Version 1.0

Date 2012-02-28

Author Shaoyu Li and Yuehua Cui

Maintainer ORPHANED

Description The package implements the model-based kernel machine method for detecting gene-centric gene-gene interactions of Li and Cui (2012).

License GPL (>= 3)

Repository CRAN

Date/Publication 2012-03-23 06:56:29

NeedsCompilation no

X-CRAN-Original-Maintainer Shaoyu Li<shaoyu.li@stjude.org>

X-CRAN-Comment Orphaned on 2014-12-07 as maintainer address <shaoyu.li@stjude.org> bounced.

R topics documented:

SPA3G-package *SPA3G*

Description

The package implements the model-based kernel machine method for detecting gene-centric gene-gene interactions of Li and Cui (2012).

Details

Package:	SPA3G
Type:	Package
Version:	1.0
Date:	2012-02-28
License:	GPL-3.0

SPA3G conducts statistical test for overall genetic effects of a gene pair and interaction effect between them. The overall test is conducted first and users can decide when to perform an interaction test by setting a cutoff value for the overall test p-value. REML estimates of variance components can also be reported as required by users.

To run SPA, appropriately prepared phenotype and genotype datasets are required. For the format of the input data sets, please run "data(SPA_example)" after install the package.

Author(s)

Shaoyu Li and Yuehua Cui

Shaoyu Li, shaoyu.li@stjude.org\ Yuehua Cui, cui@stt.msu.edu\ Erik Segur, segur@stt.msu.edu

References

Li, S and Cui, Y. (2012) Gene-centric gene-gene interaction: a model-based kernel machine method. Annals of Applied Statistics

Examples

```
data(SPA.example)
spam.res <- SPA(SPA.example$pheno, SPA.example$geno, g.size=c(1, 3), cutoff=1)
spam.res
```

KERNEL *Calculate Kernel Matrix*

Description

A function for calculating the kernel matrix using genotype data

Usage

```
KERNEL(G, weight)
```

Arguments

G	matrix: genotypes of SNP markers in one gene.
weight	numerical vector: prior weight for each marker

Value

KERNEL function returns a kernel (similarity) matrix.

Examples

```
## The function is currently defined as
function (G, weight)
{
    if (length(dim(G)) == 0) {
        size <- length(G)
        k <- matrix(1, size, size)
        for (i in 1:(size - 1)) {
            j <- seq(1, i, 1)
            remain <- G[-j]
            Ones <- matrix(1, length(remain), 1)
            leading <- Ones * G[i]
            D <- abs(remain - leading)
            AM <- D
            AM[AM == 0] <- 4
            AM[AM == 2] <- 0
            AM[AM == 1] <- 2
            AM[remain == 1 & leading == 1] <- 2
            k[i, (i + 1):size] <- k[(i + 1):size, i] <- AM *
                weight/sum(4 * weight)
        }
    }
    if (length(dim(G)) > 0) {
        size <- nrow(G)
        k <- matrix(1, size, size)
        for (i in 1:(size - 1)) {
```

```
            j <- seq(1, i, 1)
            if (i < (size - 1)) {
                remain = as.matrix(G[-j, ])
            }
            if (i == (size - 1)) {
                remain <- t(as.matrix(G[-j, ]))
            }
            Ones <- matrix(1, nrow(remain), 1)
            leading <- Ones %*% G[i, ]
            D <- abs(remain - leading)
            AM <- as.matrix(D)
            AM[AM == 0] <- 4
            AM[AM == 2] <- 0
            AM[AM == 1] <- 2
            AM[remain == 1 & leading == 1] <- 2
            k[i, (i + 1):size] <- k[(i + 1):size, i] <- AM %*%
                weight/sum(4 * weight)
        }
    }
    return(k)
}
```

PROJECT *Column-wise Mean Centered*

Description

PROJECT returns a columnwise mean-centered matrix of the input matrix.

Usage

```
PROJECT(MM)
```

Arguments

MM	matrix

Value

An object of the same type of M, but with every element been column mean centered.

Examples

```
## The function is currently defined as
function (MM)
{
    PMM <- MM - (matrix(1, n, 1) %*% apply(MM, 2, sum))/n
    return(PMM)
  }
```

`Score.Test.Interact` *Implement the gene-centric gene-gene interaction effect test for H0:*

Description

Score.Test.Interact returns results of interaction test, including score statistic, p-value, and estimates of variance components.

Usage

```
Score.Test.Interact(Y, K1, K2, K3, par, method = "BFGS", test = TRUE)
```

Arguments

`Y`	numerical vector: quantitative phenotypes.
`K1`	matrix: kernel matrix of the first gene.
`K2`	matrix: kernel matrix of the second gene.
`K3`	matrix: elementwise multiplication of K1 and K2.
`par`	numerical vector: initial values of varicance components.
`method`	the method to be used in maximazing REML. the defalt method is "BFGS". Other options are Average Information "AI" and Fisher Scoreing "FS".
`test`	logical: if TRUE conduct the test.

Details

The length of the initial values (par) should be the same as the number of variance components you intend to estimate. And the score test can only be implemented under the null model (H0:) which has 3 variance components.

Value

`VCs`	REML estimats of variance components
`Fisher.info`	fisher information matrix
`Beta`	ML estimate of the overall mean
`restricted.logLik`	restricted log-likelihood
`Score`	score statistic
`df`	estimated degree of freedom for the scaled chi-square
`scale`	estimated scale parameter for the scaled chi-square
`p.value`	p-value of the test

Examples

```
## The function is currently defined as
function (Y, K1, K2, K3, par, method = "BFGS", test = TRUE)
{
    p <- length(par)
    if (p != 3 & test == TRUE)
        cat("Error: Not matched initial values!")
    theta.new <- par
    theta.old <- rep(0, p)
    X <- matrix(1, n, 1)
    Vs <- array(0, c(n, n, 4))
    Vs[, , 1] <- diag(1, n)
    Vs[, , 2] <- K1
    Vs[, , 3] <- K2
    Vs[, , 4] <- K3
    Sigma <- 0
    for (i in 1:p) {
        Sigma <- Sigma + theta.new[i] * Vs[, , i]
    }
    W <- solve(Sigma)
    R <- W - W %*% X %*% solve(t(X) %*% W %*% X) %*% t(X) %*%
        W
    kk <- g.old <- 0
    tt <- c()
    while (sum(abs(theta.new - theta.old)) > 1e-05 & kk < 100) {
        if (method == "BFGS") {
            s <- theta.new - theta.old
            theta.old <- theta.new
            g <- c()
            for (i in 1:p) {
                g[i] <- -t(Y) %*% R %*% Vs[, , i] %*% R %*% Y +
                  TT(R, Vs[, , i])
            }
            delta <- g - g.old
            g.old <- g
            if (kk == 0 | t(s) %*% delta <= 0) {
                AI <- matrix(0, p, p)
                for (i in 1:p) {
                  for (j in i:p) {
                    AI[i, j] <- AI[j, i] <- t(Y) %*% R %*% Vs[,
                      , i] %*% R %*% Vs[, , j] %*% R %*% Y
                  }
                }
                H_inv <- solve(AI)
            }
            else {
                rho <- c(1/(t(delta) %*% s))
                H_inv <- (diag(1, p) - (s %*% t(delta)) * rho) %*%
                  H_inv %*% (diag(1, p) - rho * delta %*% t(s)) +
                  rho * s %*% t(s)
```

```
            }
        }
        if (method == "AI") {
            theta.old <- theta.new
            g <- c()
            for (i in 1:p) {
                g[i] <- t(Y) %*% R %*% Vs[, , i] %*% R %*% Y -
                  TT(R, Vs[, , i])
            }
            H <- matrix(0, p, p)
            for (i in 1:p) {
                for (j in i:p) {
                  H[i, j] <- H[j, i] <- -t(Y) %*% R %*% Vs[,
                    , i] %*% R %*% Vs[, , j] %*% R %*% Y
                }
            }
            H_inv <- solve(H)
        }
        if (method == "FS") {
            theta.old <- theta.new
            g <- c()
            for (i in 1:p) {
                g[i] <- t(Y) %*% R %*% Vs[, , i] %*% R %*% Y -
                  TT(R, Vs[, , i])
            }
            H <- matrix(0, p, p)
            for (i in 1:p) {
                AA <- R %*% Vs[, , i]
                for (j in i:p) {
                  BB <- R %*% Vs[, , j]
                  H[i, j] <- H[j, i] <- -TRACE(AA %*% BB)
                }
            }
            H_inv <- solve(H)
        }
        theta.new <- theta.old - H_inv %*% (g)
        alpha <- 0.5
        while (length(which(theta.new < 0)) > 0 & alpha > 1e-08) {
            theta.new <- theta.old - alpha * H_inv %*% (g)
            alpha <- alpha/2
        }
        theta.new[which(theta.new < 0)] <- 0
        Sigma.new <- 0
        for (i in 1:p) {
            Sigma.new <- Sigma.new + theta.new[i] * Vs[, , i]
        }
        W.new <- solve(Sigma.new)
        R <- W.new - W.new %*% X %*% solve(t(X) %*% W.new %*%
            X) %*% t(X) %*% W.new
        kk <- kk + 1
    }
    a1 <- R %*% Vs[, , 1]
    a2 <- R %*% Vs[, , 2]
```

```
    a3 <- R %*% Vs[, , 3]
    a4 <- R %*% Vs[, , 4]
    b11 <- TT(a1, a1)
    b12 <- TT(a1, a2)
    b13 <- TT(a1, a3)
    b14 <- TT(a1, a4)
    b22 <- TT(a2, a2)
    b23 <- TT(a2, a3)
    b24 <- TT(a2, a4)
    b33 <- TT(a3, a3)
    b34 <- TT(a3, a4)
    b44 <- TT(a4, a4)
    if (test == FALSE) {
        eigen.sigma <- eigen(Sigma.new)
        lR <- -(sum(log(eigen.sigma$values)) + log(det(t(X) %*%
            W.new %*% X)) + t(Y) %*% R %*% Y)/2
        H <- matrix(c(b11, b12, b13, b14, b12, b22, b23, b24,
            b13, b23, b33, b34, b14, b24, b34, b44), 4, 4)/2
        beta <- solve(t(X) %*% W.new %*% X) %*% t(X) %*% W.new %*%
            Y
        object <- list(VCs = theta.new, fisher.info = H, Beta = beta,
            restricted.logLik = lR)
        return(object)
    }
    if (test == TRUE) {
        eigen.sigma <- eigen(Sigma.new)
        lR <- -(sum(log(eigen.sigma$values)) + log(det(t(X) %*%
            W.new %*% X)) + t(Y) %*% R %*% Y)/2
        W0 <- W.new
        beta <- solve(t(X) %*% W0 %*% X) %*% t(X) %*% W0 %*%
            Y
        Q <- t(Y - X %*% beta) %*% W0 %*% K3 %*% W0 %*% (Y -
            X %*% beta)/2
        e <- TT(R, K3)/2
        Its <- c(b14, b24, b34)
        Iss <- matrix(c(b11, b12, b13, b12, b22, b23, b13, b23,
            b33), 3, 3)
        Itt <- (b44 - Its %*% solve(Iss) %*% Its)/2
        k <- Itt/e/2
        v = 2 * e^2/Itt
        pvalue <- pchisq(Q/k, df = v, lower.tail = F)
        object <- list(VCs = theta.new, fisher.info = Iss/2,
            Beta = beta, restricted.logLik = lR, Score = Q, df = v,
            scale = k, p.value = pvalue)
        class(object) <- "Score Test: tau3=0"
        return(object)
    }
}
```

`Score.Test.Overall` *Implement the overall genetic effect test for H0:*

Description

Score.Test.Overall returns results of overall genetic effect test, including score test statistic, estimated degree of freedom and scale parameter, and test p-value.

Usage

```
Score.Test.Overall(Y, K1, K2, K3)
```

Arguments

`Y`	numerical vector: quantitative phenotypes
`K1`	matrix: kernel matrix of the first gene.
`K2`	matrix: kernel matrix of the second gene.
`K3`	matrix: elementwise multiplication of K1 and K2

Value

`Score`	score statistic
`df`	estimated degree of freedom for the scaled chi-square
`scale`	estimated scale parameter for the scaled chi-square
`p.value`	test p-value

Examples

```
## The function is currently defined as
  function (Y, K1, K2, K3)
  {
    b <- mean(Y)
    sig2 <- var(Y)
    U <- t(Y - b) %*% (K1 + K2 + K3) %*% (Y - b)/(2 * sig2)
    M <- (K1 + K2 + K3)
    e <- TRACE(PROJECT(M))/2
    c11 <- TT(PROJECT(K1), PROJECT(K1))
    c12 <- TT(PROJECT(K1), PROJECT(K2))
    c13 <- TT(PROJECT(K1), PROJECT(K3))
    c22 <- TT(PROJECT(K2), PROJECT(K2))
    c23 <- TT(PROJECT(K2), PROJECT(K3))
    c33 <- TT(PROJECT(K3), PROJECT(K3))
    COV <- matrix(c(c11, c12, c13, c12, c22, c23, c13, c23, c33),
        3, 3)
    Its <- c(TRACE(PROJECT(K1)), TRACE(PROJECT(K2)), TRACE(PROJECT(K3)))
    correct_COV <- (COV - Its %*% t(Its)/(n - 1))/2
    Itt <- sum(correct_COV)
    k <- Itt/(2 * e)
    v <- 2 * e^2/Itt
    pvalue <- pchisq(U/k, df = v, lower.tail = FALSE)
    object <- list(Score = U, p.value = pvalue, df = v, scale = k)
```

```
  class(object) <- "Score Test: tau1=tau2=tau3=0"
  return(object)
}
```

SPA *run SPA*

Description

SPA function for testing overall genetic effect and ineraction effect of a pair of genes.

Usage

```
SPA(Y, G, g.size, cutoff = 0.05, par = NULL, est.alt = FALSE)
```

Arguments

Y	numerical vector: phenotype values.
G	matrix: genotypes of the gene pair, where columns are SNP markers and rows are samples.
g.size	numerical vector: with two elements indicating number of SNP markers in each gene of the gene pair.
cutoff	numerical value: cutoff for the overall test pvalue indicating when to perform interaction test.
par	numerical vector: initial values of variance components under null model of interaction test
est.alt	logical: if TRUE estimate variance comonents under the full model.

Details

SPA implements the model based kernel machine method for testing gene-centric gene-gene interaction of Li, S and Cui, Y. (2012). SPA takes a numerical vector as phenotypes and a numerical data matrix of SNP markers as columns and rows as samples. Markers in two genes are ordered as (gene 1, gene 2) and combined together into one matrix.

This function performs overall genetic effect test and interaction effect test as judged by users. Variance components can also be estimated by setting alt.est=TRUE.

For a detailed description of usage, input and output, see the example.

Value

test.overall	results of the overall test
test.interaction	results of the interaction test
parameter.est.alter	estimates of variance components under the full model

Author(s)

Yuehua Cui<cui@stt.msu.edu> Shaoyu Li<shaoyu.li@stjude.org>

References

Li, S and Cui, Y. (2012) Gene-centric gene-gene interaction: a model-based kernel machine method. Annals of Applied Statistics

Examples

```
## The function is currently defined as
function (Y, G, g.size, cutoff = 0.05, par = NULL, est.alt = FALSE)
{
    L1 <- g.size[1]
    L2 <- g.size[2]
    Gene1 <- G[, 1:L1]
    Gene2 <- G[, (L1 + 1):ncol(G)]
    w1 <- rep(1, L1)
    w2 <- rep(1, L2)
    K1 <- KERNEL(Gene1, w1)
    K2 <- KERNEL(Gene2, w2)
    K3 <- K1 * K2
    test_o <- Score.Test.Overall(Y, K1, K2, K3)
    if (test_o$p.value < cutoff) {
        if (is.null(par)) {
            grid <- c(0, 1e-05, 1e-04, 0.001, 0.01, 0.1, 1)
            test_i <- est <- vector("list", length(grid))
            for (i in 1:length(grid)) {
                initials <- c(var(Y), rep(grid[i], 2))
                test_i[[i]] <- Score.Test.Interact(Y, K1, K2,
                  K3, initials, method = "BFGS", test = TRUE)
            }
        }
        if (!is.null(par)) {
            initials <- par
            test_i <- list(Score.Test.Interact(Y, K1, K2, K3,
                initials, method = "BFGS", test = TRUE))
        }
        test.lr <- c()
        for (i in 1:length(test_i)) {
            test.lr[i] <- test_i[[i]]$restricted.logLik
        }
        test_int <- test_i[[which.max(test.lr)]]
        if (est.alt) {
            initials <- c(test_int$VCs, 0)
            est_res <- Score.Test.Interact(Y, K1, K2, K3, initials,
                method = "BFGS", test = FALSE)
            res <- list(test.overall = test_o, test.interaction = test_int,
                parameter.est.alter = est_res)
        }
```

```
        else {
            res <- list(test.overall = test_o, test.interaction = test_int)
        }
    }
    else {
        res <- list(test.overall = test_o)
    }
    return(res)
}
```

SPA.example *An example data set*

Description

The example data set containts fomatted phenotype and genotype data.

Usage

```
data(SPA.example)
```

Format

The format is: List of 2 $ pheno: Named num [1:500] 1.74 1.25 1.53 1.94 1.73 ... ..- attr(*, "names")= chr [1:500] "1" "2" "3" "4" ... $ geno : int [1:500, 1:4] 1 1 0 0 0 0 0 0 0 0 ... ..- attr(*, "dimnames")=List of 2 .. ..$: chr [1:500] "1" "2" "3" "4" ... .$: chr [1:4] "X634820282" "X634820324" "X632197333" "X632197358"

Examples

```
data(SPA.example)
```

TRACE *Returns trace of a square matrix*

Description

TRACE calculates the trace of a square matrix and returns a scale value.

Usage

```
TRACE(M)
```

Arguments

M	Square matrix

Examples

```
## The function is currently defined as
function (M)
{
    return(sum(diag(M)))
  }
```

TT *Returns the trace of the product of two matrices*

Description

TT function calculates diagal elements of the product of two matrices and sum them up to return as the trace.

Usage

```
TT(M1, M2)
```

Arguments

M1	matrix
M2	matrix

Value

scale value: trace of the product of the two input matrices

Examples

```
## The function is currently defined as
function (M1, M2)
{
    nn <- nrow(M1)
    S <- c()
    for (itt in 1:nn) {
        S[itt] <- sum(M1[itt, ] * M2[, itt])
    }
    trace <- sum(S)
    return(trace)
  }
```

WEIGHT_maf *Returns minor allele frequency based weights*

Description

WEIGHT_maf calculates a weighting scheme based on the minor allele frequency: 1/sqrt(maf)

Usage

```
WEIGHT_maf(G)
```

Arguments

G	matrix: genotypes data with columns as samples and rows as SNP markers

Value

a numeric vector of weights defined as 1/sqrt(maf)

Examples

```
## The function is currently defined as
function (G)
{
    qs <- apply(G, 1, sum)/nrow(G)
    return(1/sqrt(qs))
}
```

Index