Package 'mlica2'

February 20, 2015

Type Package

Title Independent Component Analysis using Maximum Likelihood

Version 2.1

Date 2012-11-02

Author Andrew Teschendorff

Maintainer Thomas Nelson <tnelson1@k-state.edu>

Description An R code implementation of the maximum likelihood (fixed point) algorithm of Hyvaerinen, Karhuna, and Oja for independent component analysis.

License GPL-2

Depends R (>= 2.10)

Repository CRAN

Date/Publication 2012-11-03 07:00:09

NeedsCompilation no

R topics documented:

`mlica2-package` *Maximum likelihood implementation of independent component analysis*

Description

An R code implementation of the maximum likelihood (fixed point) algorithm of Hyvaerinen, Karhuna, and Oja for independent component analysis.

Details

Package:	mlica2
Type:	Package
Version:	1.0
Date:	2012-07-17
License:	GPL-2

Author(s)

Andrew Teschendorff a.teschendorff@ucl.ac.uk (ported to R > 2.10 by Thomas Nelson)

Maintainer: Thomas Nelson

References

Hyvaerinen A., Karhunen J., and Oja E.: Independent Component Analysis, John Wiley and Sons, New York, (2001).

Kreil D. and MacKay D. (2003): Reproducibility Assessment of Independent Component Analysis of Expression Ratios from DNA microarrays, Comparative and Functional Genomics *4* (3),300-317.

Liebermeister W. (2002): Linear Modes of gene expression determined by independent component analysis, Bioinformatics *18*, no.1, 51-60.

Chiappetta P., Roubaud MC. and Torresani B.: Blind source separation and the analysis of microarray data, J. Comput. Biol. 2004; 11(6):1090-109.

`CheckStability` *Test stability of inferred ICA modes*

Description

Performs a correlation test to see which of the inferred ICA modes are reproducible across multiple runs using different random initialisations. Returns a set of consensus ICA modes and stability scores for each following the algorithm of Chiappetta,...et.al (2004)

Usage

```
CheckStability(a.best.l, corr.th = 0.7)
```

Arguments

a.best.l	List of 'a.best' objects from 'mlica' runs.
corr.th	Correlation threshold to use to decide whether a mode is reproducible.

Value

A list with the following components

consS: Consensus source matrix with columns labeling the consensus ICA modes. Has same number of rows as 'a.best$S'.

consA: Consensus mixing matrix with rows labeling the consensus ICA modes.

stabM: Vector of same length as 'consM' giving the stability measures of each consensus ICA mode. Stability or reproducibility measures are given as fractions, that is, the number of times the ICA mode correlates with one of the other runs at threshold level 'corr.th' divided by the number of runs (length of 'a.best.l').

Author(s)

Andrew Teschendorff a.teschendorff@ucl.ac.uk

References

Hyvaerinen A., Karhunen J., and Oja E.: Independent Component Analysis, John Wiley and Sons, New York, (2001).

Kreil D. and MacKay D. (2003): Reproducibility Assessment of Independent Component Analysis of Expression Ratios from DNA microarrays, Comparative and Functional Genomics *4* (3),300-317.

Liebermeister W. (2002): Linear Modes of gene expression determined by independent component analysis, Bioinformatics *18*, no.1, 51-60.

Chiappetta P., Roubaud MC. and Torresani B.: Blind source separation and the analysis of microarray data, J. Comput. Biol. 2004; 11(6):1090-109.

Examples

```
## The function is currently defined as
function (a.best.l, corr.th = 0.7)
{
```

```
nIruns <- length(a.best.l)
if (nIruns < 2) {
    print("Argument must be list of at least two a.best objects")
    stop
}
for (i in 2:nIruns) {
    if (a.best.l[[1]]$ncp != a.best.l[[i]]$ncp) {
        print("Stopping: the objects in list must have same number of ICA modes")
        stop
    }
}
ncp <- a.best.l[[1]]$ncp
X <- a.best.l[[1]]$X
nConn.v <- vector(length = nIruns * ncp)
Adj.a <- array(rep(0, nIruns * ncp * nIruns * ncp), dim = c(nIruns,
    ncp, nIruns, ncp))
v <- 1
for (j1 in 1:nIruns) {
    selrun <- setdiff(1:nIruns, j1)
    for (j2 in selrun) {
        for (i1 in 1:ncp) {
            cor.v <- as.vector(abs(cor(a.best.l[[j1]]$S[,
              i1], a.best.l[[j2]]$S)))
            i2 <- which.max(cor.v)
            Adj.a[j1, i1, j2, i2] <- 1
        }
    }
    for (i in 1:ncp) {
        nConn <- 0
        for (j2 in selrun) {
            if (sum(Adj.a[j1, i, j2, ]) > 0) {
              nConn <- nConn + 1
            }
        }
        nConn.v[v] <- nConn
        v <- v + 1
    }
}
selmodes.idx <- 1:(nIruns * ncp)
consS.lv <- list()
StabScore.l <- list()
c <- 1
while (c <= ncp) {
    max.idx <- which.max(nConn.v)
    j1.max <- as.integer(max.idx/(ncp + 1)) + 1
    i1.max <- max.idx - (j1.max - 1) * ncp
    selrun <- setdiff(1:nIruns, j1.max)
    consS <- a.best.l[[j1.max]]$S[, i1.max]
    corr.idx <- vector()
    for (j2 in selrun) {
        i2 <- which(Adj.a[j1.max, i1.max, j2, ] == 1)
        if (length(i2) > 0) {
            corr.idx <- c(corr.idx, (j2 - 1) * ncp + i2)
```

```
                tmp <- cor(a.best.l[[j1.max]]$S[, i1.max], a.best.l[[j2]]$S[,
                  i2])
                consS <- consS + sign(tmp) * a.best.l[[j2]]$S[,
                  i2]
            }
        }
        selmodes.idx <- setdiff(selmodes.idx, c(max.idx, corr.idx))
        consS.lv[[c]] <- consS/(nConn.v[max.idx] + 1)
        StabScore.l[[c]] <- (nConn.v[max.idx] + 1)/nIruns
        nConn.v[c(max.idx, corr.idx)] <- -1
        c <- c + 1
    }
    consS.m <- matrix(nrow = nrow(a.best.l[[1]]$S), ncol = ncp)
    StabScore.v <- vector()
    for (c in 1:ncol(consS.m)) {
        consS.m[, c] <- consS.lv[[c]]/sqrt(var(consS.lv[[c]]))
        StabScore.v[c] <- StabScore.l[[c]]
    }
    STS <- t(consS.m) %*% consS.m
    STSinv <- solve(STS)
    STX <- t(consS.m) %*% X
    consA.m <- STSinv %*% STX
    return(list(stabM = StabScore.v, consS = consS.m, consA = consA.m))
}
```

`mlica` *Maximum likelihood implementation of independent component analysis*

Description

This function performs ICA using a maximum likelihood framework and takes as arguments parameters to control the number of algorithm runs and convergence criteria.

Usage

```
mlica(prNCP, nruns = 10, tol = 1e-04, maxit = 300, fail.th = 5, learn.mu = 1)
```

Arguments

`prNCP`	The output object from 'proposeNCP'.
`nruns`	The number of converged algorithm runs sought (function returns the best solution according to the log-likelihood value).
`tol`	Tolerance level for establishing convergence of run.
`maxit`	Maximum number of iterations to allow per run.
`fail.th`	A threshold on the number of consecutive runs that fail to converge.
`learn.mu`	Learning parameter for fixed point algorithm (note that this need not be changed since it has already been optimised).

Value

A list with following components:

A: Estimate of the mixing matrix.

B: Estimate of the inverse mixing matrix.

S: Estimate of the source matrix.

X: Normalised data matrix.

ncp: Number of independent components.

NC: Binary number specifying whether best run converged or not.(=1 indicates convergence,=0 indicates no convergence).

LL: Log likelihood value of best run.

Author(s)

Andrew Teschendorff a.teschendorff@ucl.ac.uk

References

Hyvaerinen A., Karhunen J., and Oja E.: Independent Component Analysis, John Wiley and Sons, New York, (2001).

Kreil D. and MacKay D. (2003): Reproducibility Assessment of Independent Component Analysis of Expression Ratios from DNA microarrays, Comparative and Functional Genomics *4* (3),300-317.

Liebermeister W. (2002): Linear Modes of gene expression determined by independent component analysis, Bioinformatics *18*, no.1, 51-60.

Chiappetta P., Roubaud MC. and Torresani B.: Blind source separation and the analysis of microarray data, J. Comput. Biol. 2004; 11(6):1090-109.

Examples

```
## Not run:
data(simMAdata);
dataX <- simMAdata[[1]];
prPCA <- PriorNormPCA(dataX);
prNCP <- proposeNCP(prPCA,0.1);
a.best.l <- list();
for( i in 1:5){
 a.best.l[[i]] <- mlica(prNCP,nruns=5);
}
checkICA <- CheckStability(a.best.l,0.7);
sourceS <- simMAdata[[3]];
print(cor(a.best.l[[1]]$S,sourceS));
sModes <- SortModes(a.best.l[[1]],c.val=0.5);

## End(Not run)

## The function is currently defined as
function (prNCP, nruns = 10, tol = 1e-04, maxit = 300, fail.th = 5,
```

```
    learn.mu = 1)
{
    print("Entering mlica")
    print("Performing preliminary run")
    a <- mlicaMAIN(prNCP, tol = 1e-04, maxit = 10, mu = learn.mu)
    ncp <- dim(a$S)[2]
    max.logL <- a$LL
    a.best <- a
    print("Finished preliminary run")
    print("Starting runs")
    run.n <- 0
    fail.count <- 0
    v.logL <- vector()
    v.NC <- vector()
    while (run.n < nruns) {
        a <- mlicaMAIN(prNCP, tol = 1e-04, maxit = maxit, mu = learn.mu)
        v.logL <- c(v.logL, a$LL)
        v.NC <- c(v.NC, a$NC)
        if (a$NC == 0) {
            fail.count <- 0
            run.n <- run.n + 1
            if (a$LL > max.logL) {
                a.best <- a
            }
        }
        else {
            fail.count <- fail.count + 1
        }
        if (fail.count >= fail.th) {
            print("Stopping: Five consecutive runs failed to converge!")
          print("Consider either increasing the threshold for pca eigenvalues to perform ICA on a smaller subspac
            stop
        }
    }
    print("End of runs")
    return(a.best)
  }
```

---

mlicaMAIN *Main engine function that implements the fixed point algorithm for maximum likelihood of ICA modes*

Description

See references for detailed description.

Usage

```
mlicaMAIN(prNCP, tol = 1e-04, maxit = 300, mu = 1)
```

Arguments

prNCP	The output object of 'proposeNCP'.
tol	Tolerance level for convergence.
maxit	Maximum number of iterations to allow for convergence.
mu	Learning paramter for fixed point algorithm. This has already been optimised.

Value

A list with following components:

A: Estimate of the mixing matrix.

B: Estimate of the inverse mixing matrix.

S: Estimate of the source matrix.

X: Normalised data matrix.

ncp: Number of independent components.

NC: Binary number specifying whether best run converged,0, or not,1.

LL: Log likelihood value of best run.

Author(s)

Andrew Teschendorff a.teschendorff@ucl.ac.uk

References

Hyvaerinen A., Karhunen J., and Oja E.: Independent Component Analysis, John Wiley and Sons, New York, (2001).

Kreil D. and MacKay D. (2003): Reproducibility Assessment of Independent Component Analysis of Expression Ratios from DNA microarrays, Comparative and Functional Genomics *4* (3),300-317.

Liebermeister W. (2002): Linear Modes of gene expression determined by independent component analysis, Bioinformatics *18*, no.1, 51-60.

Examples

```
## The function is currently defined as
function (prNCP, tol = 1e-04, maxit = 300, mu = 1)
{
    print("Entered MLica")
    X <- prNCP$X
    x <- prNCP$x
    pEx <- prNCP$pEx
    pCorr <- prNCP$pCorr
    ntp <- dim(X)[1]
    ndim <- dim(X)[2]
    ncp <- ncol(x)
    Sest <- matrix(nrow = ntp, ncol = ncp)
    B.old <- matrix(runif(ncp * ncp, 0, 1), nrow = ncp, ncol = ncp)
```

```
B.o <- B.old
icount <- 0
not.conv <- c(1, 2)
y <- matrix(nrow = ntp, ncol = ncp)
tmp <- matrix(nrow = ncp, ncol = ncp)
beta <- vector(length = ncp)
alpha <- vector(length = ncp)
while ((length(not.conv) > 0) && (icount < maxit)) {
    print(c("Entering iteration loop ", icount))
    Cy <- B.old %*% t(B.old)
    svds <- eigen(Cy, symmetric = TRUE)
    D <- diag(svds$values)
    E <- svds$vectors
    Dinv <- solve(D)
    V <- E %*% sqrt(Dinv) %*% t(E)
    B.old <- V %*% B.old
    for (g in 1:ntp) {
        y[g, ] <- B.old %*% x[g, ]
    }
    for (c in 1:ncp) {
        beta[c] <- 2 * sum(y[, c] * tanh(y[, c]))/ntp
        alpha[c] <- -1/(beta[c] - 2 + 2 * sum(tanh(y[, c]) *
            tanh(y[, c]))/ntp)
        for (c2 in 1:ncp) {
            tmp[c, c2] <- -2 * sum(tanh(y[, c]) * y[, c2])/ntp
        }
    }
    print("Checkpt1")
    tmp <- diag(beta) + tmp
    B <- B.old + mu * diag(alpha) %*% tmp %*% B.old
    Dev <- abs(B - B.o)
    AvDev <- sum(Dev)/(ncp * ncp)
    print(c("AvDev=", AvDev))
    not.conv <- vector()
    not.conv <- as.vector(Dev[Dev > tol])
    B.old <- B
    B.o <- B
    icount <- icount + 1
    for (g in 1:ntp) {
        Sest[g, ] <- B %*% x[g, ]
    }
    logL <- -2 * sum(log(cosh(Sest))) + ntp * log(abs(det(B)))
    print("iterated logL")
    print(logL)
}
Cy <- B %*% t(B)
svds <- eigen(Cy, symmetric = TRUE)
D <- diag(svds$values)
E <- svds$vectors
Dinv <- solve(D)
V <- E %*% sqrt(Dinv) %*% t(E)
B <- V %*% B
for (g in 1:ntp) {
```

```
        Sest[g, ] <- B %*% x[g, ]
    }
    Aest <- t(pEx %*% sqrt(pCorr) %*% t(B))
    if (length(not.conv) > 0) {
        NotConv <- 1
    }
    else {
        NotConv <- 0
    }
    logL <- -2 * sum(log(cosh(Sest))) + ntp * log(abs(det(B)))
    return(list(A = Aest, B = B, S = Sest, X = X, ncp = dim(Sest)[2],
        NC = NotConv, LL = logL))
}
```

PriorNormPCA *Prior PCA analysis for threshold setting and noise removal*

Description

This function performs a simple PCA analysis to aid in threshold setting and noise removal.

Usage

```
PriorNormPCA(X)
```

Arguments

X	Data Matrix (need not be normalised). Subsequent ICA seeks independent modes as independent distributions with values "down the rows".

Details

This function performs a simple PCA analysis and is used prior to application of the main ICA algorithm. The objective of the prior PCA is to help determine the dimensionality of a subspace on which the further ICA converges. The convention used here is that the rows of 'X' label the space over which independent components are sought. For a typical microarray application in which ICA is being used as a generative model for gene expression, rows should label genes and columns should label samples. If, however, ICA is to be used as an unsupervised projection pursuit algorithm, rows should label samples and columns genes. For the latter application, the number of genes should be less than the number of samples.

Value

A list with following components:

X: Normalised data matrix with the mean of each column set to zero.

Dx: Eigenvalues in a diagonal matrix.

Ex: Eigenvectors

Author(s)

Andrew Teschendorff a.teschendorff@ucl.ac.uk

References

Hyvaerinen A., Karhunen J., and Oja E.: Independent Component Analysis, John Wiley and Sons, New York, (2001).

Kreil D. and MacKay D. (2003): Reproducibility Assessment of Independent Component Analysis of Expression Ratios from DNA microarrays, Comparative and Functional Genomics *4* (3),300-317.

Liebermeister W. (2002): Linear Modes of gene expression determined by independent component analysis, Bioinformatics *18*, no.1, 51-60.

Examples

```
## The function is currently defined as
function (X)
{
    ndim <- ncol(X)
    ntp <- nrow(X)
    for (s in 1:ndim) {
        X[, s] <- X[, s] - mean(X[, s])
    }
    print("Performing SVD")
    svd.o <- svd(X, LINPACK = TRUE)
    Dx <- diag(svd.o$d * svd.o$d)/ntp
    Ex <- svd.o$v
    barplot(Dx, main = "Singular values")
    return(list(X = X, Dx = Dx, Ex = Ex))
  }
```

---

proposeNCP *Number of independent components proposal function*

Description

This function takes the output of 'PriorNormPCA' and returns for a given threshold the number of components to be inferred for subsequent ICA.

Usage

```
proposeNCP(prPCA, thresh = 0.1)
```

Arguments

prPCA	The output object from 'PriorNormPCA'.
thresh	Threshold on eigenvalues.

Value

A list with following components:

X: Normalised data matrix.

x: Normalised data matrix projected onto selected subspace.

pEx: Selected eigenvectors defining subspace for projection.

pCorr: Projected correlation matrix.

ncp: The dimension of the selected subspace(=number of independent components to be inferred with subsequent ICA).

Author(s)

Andrew Teschendorff a.teschendorff@ucl.ac.uk

References

Hyvaerinen A., Karhunen J., and Oja E.: Independent Component Analysis, John Wiley and Sons, New York, (2001).

Kreil D. and MacKay D. (2003): Reproducibility Assessment of Independent Component Analysis of Expression Ratios from DNA microarrays, Comparative and Functional Genomics *4* (3),300-317.

Liebermeister W. (2002): Linear Modes of gene expression determined by independent component analysis, Bioinformatics *18*, no.1, 51-60.

Examples

```
## The function is currently defined as
function (prPCA, thresh = 0.1)
{
    X <- prPCA$X
    eigenvals.v <- diag(prPCA$Dx)
    Ex <- prPCA$Ex
    ntp <- nrow(X)
    ndim <- ncol(X)
    print("About to find ncp")
    p.cpts <- eigenvals.v[eigenvals.v > thresh]
    ncp <- length(p.cpts)
    pCorr <- diag(eigenvals.v[1:ncp])
    pEx <- Ex[, 1:ncp]
    x <- matrix(nrow = ntp, ncol = ncp)
    for (g in 1:ntp) {
        for (c in 1:ncp) {
            x[g, c] <- sum(X[g, ] * Ex[, c])/sqrt(diag(pCorr)[c])
        }
    }
    return(list(X = X, x = x, pEx = pEx, pCorr = pCorr, ncp = ncp))
  }
```

simMAdata *Simulated microarray data for testing purposes*

Description

This data set contains a mock microarray data set of 1000 genes and 60 samples where the data has been generated from an underlying Independent Component Analysis model of 5 supergaussian modes.

Usage

```
data(simMAdata)
```

Value

A list with the components:

A list with three components. First, second and third components are the data matrix, mixing matrix and source matrix, respectively.

References

Hyvaerinen A., Karhunen J., and Oja E.: Independent Component Analysis, John Wiley and Sons, New York (2001).

SortModes *Sorting of ICA Modes*

Description

Sorts inferred ICA modes using two criteria: Relative data power or the Liebermeister criterion, which is based on a measure that is a weighted linear combination of non-gaussianity and data variance measures.

Usage

```
SortModes(a.best,c.val = 0.25)
```

Arguments

a.best	The output object of 'mlica'.
c.val	A parameter to control the relative weight of the two measures when using the Liebermeister criterion. Should be between 0 (pure data variance measure) and 1 (pure non-gaussianity).

Value

A list with the components:

a.best: The output of 'mlica'.

rdp: The relative data power values obtained for each independent component.

lbm: The Liebermeister contrast value for each component.

Author(s)

Andrew Teschendorff a.teschendorff@ucl.ac.uk

References

Hyvaerinen A., Karhunen J., and Oja E.: Independent Component Analysis, John Wiley and Sons, New York, (2001).

Kreil D. and MacKay D. (2003): Reproducibility Assessment of Independent Component Analysis of Expression Ratios from DNA microarrays, Comparative and Functional Genomics *4* (3),300-317.

Liebermeister W. (2002): Linear Modes of gene expression determined by independent component analysis, Bioinformatics *18*, no.1, 51-60.

Examples

```
#This function is currently defined as
function(a.best,c.val=0.25){

ncp <- ncol(a.best$S);
Ng <- nrow(a.best$S);

#SORTING CRITERION

# A) Computation of relative data power. Store values in vector of size H=ncp. Could use different criterion here.
# need squared entries
 Ssq <- a.best$S * a.best$S ;
 Asq <- a.best$A * a.best$A ;
 Xsq <- a.best$X * a.best$X ;
 rdp <- rep(0, times=ncp);
 for ( k in 1:ncp ) {
  rdp[k]<- sum(Ssq[,k])*sum(Asq[k,])/sum(Xsq) ;
 }
 rdp.s <- sort(rdp, na.last=NA,decreasing=TRUE, index.return=TRUE);

# B) sorting with mixture of contrast and data variance (Liebermeister)
 JG <- rep(0, times=ncp);
 JA <- rep(0, times=ncp);
 # generate values from std. normal distribution
 nu <- rnorm(10000,0,1);
 G0 <- mean(log(cosh(nu)));
```

```
 for ( k in 1:ncp ){
   # compute contrast for mode using logcosh
   G1 <- mean(log(cosh(a.best$S[,k])));
   JG[k] <- abs(G1-G0);
   JA[k] <- sum(Asq[k,]);
 }
   sumJG <- sum(JG) ; sumJA <- sum(JA) ;
   J <- JG*(c.val/sumJG)+ JA*(1-c.val)/sumJA ;
   J.s <- sort(J, na.last=NA,decreasing=TRUE, index.return=TRUE);

   return(list(a.best=a.best,rdp=rdp.s,lbm=J.s));

}
```

Index