Package 'NSA'

July 2, 2014

Version 0.0.32

Date 2012-12-20

Title Post-normalization of total copy numbers

Author Maria Ortiz-Estevez <mortizest@gmail.com>, Ander Aranburu <aaranburu@ceit.es>, Angel Rubio <arubio@ceit.es>

Maintainer Maria Ortiz-Estevez <mortizest@gmail.com>

Depends R (>= 2.11.1), R.methodsS3 (>= 1.2.0), MASS, matrixStats (>= 0.2.1), R.oo (>= 1.7.3), R.utils (>= 1.4.2), aroma.core (>= 1.6.0), aroma.affymetrix, DNAcopy

Description Post-normalization of total copy-number estimates.

License LGPL (>= 2.1)

URL http://r-forge.r-project.org/projects/snpsprocessing/

LazyLoad TRUE

biocViews Infrastructure, Microarray, DNACopyNumber, Preprocessing,aCGH, SNP, CopyNumberVariants

Repository CRAN

Repository/R-Forge/Project snpsprocessing

Repository/R-Forge/Revision 202

Repository/R-Forge/DateTimeStamp 2012-12-20 19:24:38

Date/Publication 2012-12-21 09:17:05

NeedsCompilation no

R topics documented:

`NSA-package` *Package NSA*

Description

Post-normalization of total copy-number estimates.

This package should be considered to be in an alpha phase. You should expect the API to be changing over time.

Requirements

This package requires...

Installation and updates

TBA.

To get started

1. To process SNP and non-polymorphic signals, see `NSAByTotalAndFracB`.
2. For processing data in the aroma framework, see `NSANormalization`.

License

The releases of this package is licensed under LGPL version 2.1 or newer.

Author(s)

Maria Ortiz-Estevez <mortiz@ceit.es>, Ander Aramburu <aaramburu@ceit.es>, Pierre Neuvial <pierre@stat.berkeley.edu>,

References

TBA.

fitNSA.matrix *Find normal regions*

Description

Find normal regions.

Usage

```
## S3 method for class 'matrix'
fitNSA(data, ...)
```

Arguments

data	An JxI numeric array, where J is the number of SNPs, and I is the number of samples.
...	Additional arguments passed to internal calmate().

Value

Returns an JxI numeric array.

NSAByTotalAndFracB.matrix *Finds normal regions within tumoral samples (total,fracB)*

Description

Finds normal regions within tumoral samples (total,fracB), where total is the total (non-polymorphic) signal and fracB is the allele B fraction. It is only loci with a non-missing (NA) fracB value that are considered to be SNPs and normalized by CalMaTe. The other loci are left untouched.

Usage

```
## S3 method for class 'matrix'
NSAByTotalAndFracB(data, ..., verbose=FALSE)
```

Arguments

data	An Jx2 matrix, where J is the number of loci and 2 is total and fracB.
...	Not used.
verbose	See Verbose.

Value

Returns an Jx2 numeric array.

NSANormalization *The NSANormalization class*

Description

Package: NSA
Class NSANormalization

Object
~~|
~~+--NSANormalization

Directly known subclasses:

public static class **NSANormalization**
extends Object

This class represents the NSA normalization method [1], which looks for normal regions within the tumoral samples.

Usage

```
NSANormalization(data=NULL, tags="*", ...)
```

Arguments

data	A named list with data set named "total" and "fracB" where the former should be of class AromaUnitTotalCnBinarySet and the latter of class AromaUnitFracBCnBinarySet. The two data sets must be for the same chip type, have the same number of samples and the same sample names.
tags	Tags added to the output data sets.
...	Not used.

Details

...

Fields and Methods

Methods:

findArraysTodo	-
getDataSets	-
getFullName	-

getName	-
getOutputDataSets	-
getPath	-
getRootPath	-
getTags	-
nbrOfFiles	-
process	Finds normal regions within tumoral samples.
setTags	-

Methods inherited from Object:
asThis, $, $<-, [[, [[<-, as.character, attach, attachLocally, clearCache, clone, detach, equals, extend, finalize, gc, getEnvironment, getFields, getInstantiationTime, getStaticInstance, hasField, hashCode, ll, load, objectSize, print, registerFinalizer, save

References

[1] ...

See Also

Low-level versions of the NSA normalization method is available via the NSAByTotalAndFracB.matrix() methods.

Examples

```
## Not run:

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# CRMAv2 - Preprocess raw Affymetrix data
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
library("aroma.affymetrix");  # Needed for CRMAv2
#library("calmate");
library(MASS)
source("F:/MOrtiz/curro/Aroma/calmate/R/CalMaTeNormalization.R")
source("F:/MOrtiz/curro/Aroma/calmate/R/calmateByTotalAndFracB.R")
source("F:/MOrtiz/curro/Aroma/calmate/R/calmateByThetaAB.R")
source("F:/MOrtiz/curro/Aroma/calmate/R/fitCalMaTe.R")
source("F:/MOrtiz/curro/Aroma/calmate/R/fitCalMaTeCNprobes.R")
source("F:/MOrtiz/curro/Aroma/calmate/R/thetaAB2TotalAndFracB.R")

source("F:/MOrtiz/curro/Aroma/NSA/R/NSANormalization.R")
source("F:/MOrtiz/curro/Aroma/NSA/R/NSAByTotalAndFracB.R")
source("F:/MOrtiz/curro/Aroma/NSA/R/fitNSA.R")

source("F:/MOrtiz/curro/Aroma/NSA/R/SNPsNormalization.R")
source("F:/MOrtiz/curro/Aroma/NSA/R/SNPsNByTotalAndFracB.R")
source("F:/MOrtiz/curro/Aroma/NSA/R/fitSNPsN.R")

source("F:/MOrtiz/curro/Aroma/NSA/R/SampleNormalization.R")
source("F:/MOrtiz/curro/Aroma/NSA/R/SampleNByTotalAndFracB.R")
```

```
source("F:/MOrtiz/curro/Aroma/NSA/R/fitSample.R")
library("DNAcopy");

setwd("I:/aroma")

dataSet <- "breast cancer";
dataSet <- "GSE12702-prostateCancerPaired"
dataSet <- "GSE12702-prostateCancer"
dataSet <- "GSE14996,testSet"

chipType <- "Mapping250K_Nsp";
#chipType <- "GenomeWideSNP_6"
dsList <- doCRMAv2(dataSet, chipType=chipType, combineAlleles=FALSE,
                                            plm="RmaCnPlm", verbose=-10);
print(dsList);


# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# CalMaTe - Post-normalization of ASCNs estimates
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
asn <- CalMaTeNormalization(dsList);
print(asn);

# For speed issues, we will here only process loci on Chromosome 17.
chr <- 22;
ugp <- getAromaUgpFile(dsList$total);
units <- getUnitsOnChromosome(ugp, chr);

dsNList <- process(asn, units=units, verbose=verbose);
#dsNList <- process(asn, references = seq(2,40,2), verbose=verbose);
#dsNList <- process(asn, verbose=verbose);
print(dsNList);

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# NSA - Finding normal regions
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

asnN <- NSANormalization(dsNList);
print(asnN);

dsNNList <- process(asnN, verbose=verbose);
print(dsNNList);

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# NSA - SNPs Normalization
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

asnNsnps <- SNPsNormalization(dsNList);
print(asnNsnps);

dsNNListSNPs <- process(asnNsnps, references = dsNNList, verbose=verbose);
print(dsNNListSNPs);
```

```
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# NSA - Sample Normalization
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

asnNsample <- SampleNormalization(dsNNListSNPs);
print(asnNsample);

dsNNListSample <- process(asnNsample, references = dsNNList, verbose=verbose);
print(dsNNListSample);

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# NSA - SNPs Normalization
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

asnNsnps <- SNPsNormalization(dsNNListSample);
print(asnNsnps);

dsListSNPs <- process(asnNsnps, references = dsNNList, verbose=verbose);

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# NSA - Sample Normalization
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

asnNsample <- SampleNormalization(dsListSNPs);
print(asnNsample);

dsListSample <- process(asnNsample, references = dsNNList, verbose=verbose);
print(dsNNListSample);

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# CalMaTe - sigma delta validation  (for CRMAv2)
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

# Alt 1: Calculate CN ratios where the reference is a pool of specific references samples

references <- c(1:20);

dsR <- extract(dsList$total, references);
dfR <- getAverageFile(dsR);
cnm <- CopyNumberChromosomalModel(dsList$total, dfR);

# Alt 2: Calculate CN ratios where the reference is the pool of all samples
cnm <- CopyNumberChromosomalModel(dsList$total);
print(cnm);

# Extract CN ratios - C=theta/thetaR
cn <- extractRawCopyNumbers(cnm, array=ii, chromosome=chr, logBase=NULL);
cn$y <- 2*cn$y;
print(cn);

# As a data frame
#data <- as.data.frame(cn);
```

```
# Estimate std dev (via first-order variance estimator)
sigma <- estimateStandardDeviation(cn);
print(sigma);

# Plot
x11();plot(cn, ylim=c(0,6));
sigmaStr <- sprintf("%.3f", sigma);
stext(side=3, pos=0.5, line=-1, substitute(hat(sigma)[Delta]==x, list(x=sigmaStr)));

# Smooth (bins of 1.0Mb)
cnS <- binnedSmoothing(cn, by=1.0e6);
print(cnS);
points(cnS, col="red");
lines(cnS, col="red");

# Segmentation
fit <- segmentByCBS(cn);
cnr <- extractCopyNumberRegions(fit);
print(cnr);
drawLevels(cnr, col="blue", lwd=3);

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# CalMaTe - sigma delta validation  (for CalMaTe)
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

# Alt 1: Calculate CN ratios where the reference is a pool of specific references samples

references <- c(1:24)

# Alt 2: Calculate CN ratios where the reference is the pool of all samples
#cnm <- CopyNumberChromosomalModel(dsNNListSample$total);

#cnm <- CopyNumberChromosomalModel(dsNList$total);
#print(cnm);

# Extract CN ratios - C=theta/thetaR
#cn <- extractRawCopyNumbers(cnm, array=ii, chromosome=chr, logBase=NULL);

dataNSA <- extractMatrix(dsNList$total,units = units);
cnNSA <- cn;
cnNSA$y <- dataNSA[,ii];
cn <- cnNSA;


# As a data frame
data <- as.data.frame(cn);

# Estimate std dev (via first-order variance estimator)
sigma <- estimateStandardDeviation(cn);
print(sigma);

# Plot
x11();plot(cn, ylim=c(0,6));
```

```
sigmaStr <- sprintf("%.3f", sigma);
stext(side=3, pos=0.5, line=-1, substitute(hat(sigma)[Delta]==x, list(x=sigmaStr)));

# Smooth (bins of 1.0Mb)
cnS <- binnedSmoothing(cn, by=1.0e6);
print(cnS);
points(cnS, col="red");
lines(cnS, col="red");

# Segmentation
fit <- segmentByCBS(cn);
cnr <- extractCopyNumberRegions(fit);
print(cnr);
drawLevels(cnr, col="blue", lwd=3);


#You can obtain the above 'cn' manually as:

dsR <- extract(dsList$total, references);
dfR <- getAverageFile(dsR);

df <- getFile(dsList$total, ii);
theta <- extractRawCopyNumbers(df, chromosome=chr);
print(theta);
thetaR <- extractRawCopyNumbers(dfR, chromosome=chr);
print(thetaR);
cn <- clone(theta);
cn <- divideBy(cn, thetaR);
cn$y <- 2*cn$y;

#sigma delta
mad(diff(cn$y))

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Calculating the differences between CN and SNP probes
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -


cnCNP <- RawCopyNumbers(...);
cnSNP <- RawCopyNumbers(...);

xRange <- range(xRange(cnCNP), xRange(cnSNP));

by <- 50e3; # 50kb bins; you may want to try with other amounts of smoothing xOut <- seq(from=xRange[1], to=xRange

cnCNPS <- binnedSmoothing(cnCNP, xOut=xOut); cnSNPS <- binnedSmoothing(cnSNP, xOut=xOut);

Clim <- c(0,5);
plot(cnCNPS, ylim=Clim);
points(cnSNPS, col="blue");

plot(getSignals(cnCNPS), getSignals(cnSNPS), xlim=Clim, ylim=Clim); abline(a=0, b=1, col="red", lwd=2);
```

```
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Plot allele B fractions (before and after calmate)
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Sample #1 and Chromosome 17
ii <- 1;
chr <- 17;
df <- getFile(dsList$fracB, ii);
dfN <- getFile(dsNList$fracB, ii);

beta <- extractRawAlleleBFractions(df, chromosome=chr);
betaN <- extractRawAlleleBFractions(dfN, chromosome=chr);
x11();
subplots(2, ncol=1);
plot(beta);
title(sprintf("%s", getName(beta)));
plot(betaN);
title(sprintf("%s (CalMaTe)", getName(betaN)));


for(ii in 1:24){
  df <- getFile(dsList$total, ii);
  CN <- extractRawCopyNumbers(df, chromosome=chr);
  if(ii == 1){
    aux <- CN$y;
  }else{
    aux <- aux + CN$y;
  }
}
ref <- aux/24;

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Plot copy numbers (before and after calmate)
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Sample #1 and Chromosome 17
ii <- 3;
chr <- 1;

df <- getFile(dsList$total, ii);
#dfN <- getFile(dsNList$total, ii);

#units <- getUnitsOnChromosome(gi, ii);
#pos <- getPositions(gi, units = units);
#units <- units[order(pos)];

CN <- extractRawCopyNumbers(df, chromosome=chr);
#CNN <- extractRawCopyNumbers(dfN, chromosome=chr);
#CNN <- extractMatrix(dsNList$total, units = units);
#CNN <- CNN[,ii];


x11();
subplots(2, ncol=1);
```

```
plot(CN$x, log2(CN$y));
title(sprintf("%s", getName(CN)));
plot(CNN);
title(sprintf("%s (CalMaTe)", getName(CNN)));

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Plot Normal Regions
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Sample #3 and Chromosome 6
ii <- 1;
chr <- 6;
dfNN <- getFile(dsNNList$normalReg, ii);

betaNN <- extractRawAlleleBFractions(dfNN, chromosome=chr);

plot(betaNN);
title(sprintf("%s (NSA)", getName(betaNN)));

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Plot Normalized by SNP data
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Sample #9 and Chromosome 8
ii <- 9;
chr <- 8;
dfN <- getFile(dsNNListSNPs$fracB, ii);
fracBN <- extractRawAlleleBFractions(dfN, chromosome=chr);

dfN <- getFile(dsNNListSNPs$total, ii);
totalN <- extractRawCopyNumbers(dfN, chromosome=chr);

x11();
subplots(2, ncol=1);
plot(fracBN);
title(sprintf("%s", getName(fracBN)));
plot(totalN);
title(sprintf("%s ", getName(totalN)));


# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Plot Normalized by Sample data
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Sample #3 and Chromosome 6
ii <- 9;
chr <- 8;

dfC <- getFile(dsNList$total, ii);
CNC <- extractRawCopyNumbers(dfC, chromosome=chr);

dfN <- getFile(dsNNListSample$fracB, ii);
fracBN <- extractRawAlleleBFractions(dfN, chromosome=chr);

dfN <- getFile(dsNNListSample$total, ii);
totalN <- extractRawCopyNumbers(dfN, chromosome=chr);
```

```
x11();
subplots(2, ncol=1);
plot(fracBN);
title(sprintf("%s", getName(fracBN)));
plot(totalN$x, 2^totalN$y);
title(sprintf("%s ", getName(totalN)));

x11();
plot(CNC);
points(totalN, col="green");


## End(Not run)
```

`sampleNByTotalAndFracB.numeric` *Normalize total copy numbers by samples (total,fracB)*

Description

Normalize total copy numbers by samples (total,fracB), where total is the total (non-polymorphic) signal and fracB is the allele B fraction. It is only loci with a non-missing (NA) fracB value that are considered to be SNPs and normalized by NSA. The other loci are left untouched.

Usage

```
## S3 method for class 'numeric'
sampleNByTotalAndFracB(data, references=NULL, ..., verbose=FALSE)
```

Arguments

`data`	An J `numeric vector`, where J is the number of loci for an specific sample.
`references`	An object specifying the normal regions of each sample.
`...`	Additional arguments passed to fitSNPsN.
`verbose`	See `Verbose`.

Value

Returns an Jx2 `numeric array`.

SampleNormalization *The SampleNormalization class*

Description

Package: NSA
Class SampleNormalization

Object
~~|
~~+--SampleNormalization

Directly known subclasses:

public static class **SampleNormalization**
extends Object

This class represents the Sample normalization method [1], which looks for normal regions within the tumoral samples.

Usage

```
SampleNormalization(data=NULL, tags="*", ...)
```

Arguments

data	A named list with data set named "total" and "fracB" where the former should be of class AromaUnitTotalCnBinarySet and the latter of class AromaUnitFracBCnBinarySet. The two data sets must be for the same chip type, have the same number of samples and the same sample names.
tags	Tags added to the output data sets.
...	Not used.

Details

...

Fields and Methods

Methods:

findArraysTodo	-
getDataSets	-
getFullName	-

getName	-
getOutputDataSets	-
getPath	-
getRootPath	-
getTags	-
nbrOfFiles	-
process	Finds normal regions within tumoral samples.
setTags	-

Methods inherited from Object:
asThis, $, $<-, [[, [[<-, as.character, attach, attachLocally, clearCache, clone, detach, equals, extend, finalize, gc, getEnvironment, getFields, getInstantiationTime, getStaticInstance, hasField, hashCode, ll, load, objectSize, print, registerFinalizer, save

References

[1] ...

See Also

Low-level versions of the Sample normalization method is available via the sampleNByTotalAndFracB.numeric() methods.

---

snpsNByTotalAndFracB.matrix *Normalize allele-specific copy numbers (total,fracB)*

Description

Normalize allele-specific copy numbers (total,fracB), where total is the total (non-polymorphic) signal and fracB is the allele B fraction. It is only loci with a non-missing (NA) fracB value that are considered to be SNPs and normalized by CalMaTe. The other loci are left untouched.

Usage

```
## S3 method for class 'matrix'
snpsNByTotalAndFracB(data, references=NULL, ..., verbose=FALSE)
```

Arguments

data	An Jx2xI numeric array, where J is the number of loci, 2 is total and fracB, and I is the number of samples.
references	A logical or numeric vector specifying which samples should be used as the reference set. By default, all samples are considered.
...	Additional arguments passed to fitSNPsN.
verbose	See Verbose.

Value

Returns an Jx2xI numeric array.

SNPsNormalization *The SNPsNormalization class*

Description

Package: NSA
Class SNPsNormalization

```
Object
~~|
~~+--SNPsNormalization
```

Directly known subclasses:

public static class **SNPsNormalization**
extends Object

This class represents the SNPs normalization method [1], which corrects for SNP effects in allele-specific copy-number estimates (ASCNs).

Usage

```
SNPsNormalization(data=NULL, tags="*", ...)
```

Arguments

data	A named list with data set named "total" and "fracB" where the former should be of class AromaUnitTotalCnBinarySet and the latter of class AromaUnitFracBCnBinarySet. The two data sets must be for the same chip type, have the same number of samples and the same sample names.
tags	Tags added to the output data sets.
...	Not used.

Details

...

Fields and Methods

Methods:

```
findUnitsTodo      -
getDataSets        -
getFullName        -
getName            -
getOutputDataSets  -
getPath            -
getRootPath        -
getTags            -
nbrOfFiles         -
process            -
setTags            -
```

Methods inherited from Object:
asThis, $, $<-, [[, [[<-, as.character, attach, attachLocally, clearCache, clone, detach, equals, extend, finalize, gc, getEnvironment, getFields, getInstantiationTime, getStaticInstance, hasField, hashCode, ll, load, objectSize, print, registerFinalizer, save

References

[1] ...

See Also

Low-level versions of the SNPs normalization method is available via `snpsNByTotalAndFracB.matrix`() methods.

Index