Package 'GCSscore'

April 12, 2022

Type Package

Title GCSscore: an R package for microarray analysis for Affymetrix/Thermo Fisher arrays

Version 1.8.0

Author Guy M. Harris & Shahroze Abbas & Michael F. Miles

Maintainer Guy M. Harris <harrisgm@vcu.edu>

Description
For differential expression analysis of 3'IVT and WT-style microarrays from Affymetrix/Thermo-Fisher. Based on S-score algorithm originally described by Zhang et al 2002.

License GPL (>=3)

Encoding UTF-8

Depends R (>= 3.6)

Imports BiocManager, Biobase, utils, methods, RSQLite, devtools, dplR, stringr, graphics, stats, affxparser, data.table

Suggests siggenes, GEOquery, R.utils

biocViews DifferentialExpression, Microarray, OneChannel, ProprietaryPlatforms, DataImport

LazyData FALSE

NeedsCompilation no

git_url https://git.bioconductor.org/packages/GCSscore

git_branch RELEASE_3_14

git_last_commit d7f279c

git_last_commit_date 2021-10-26

Date/Publication 2022-04-12

R topics documented:

calcSF 2
computeSscore 3
GCSscore 4

get3primeIVTprobefileData 6
getClariomSprobefileData 8
getXTAprobefileData 9
mismatch 11
normalize 12
rawScore 12
zoneRQ 13

Index **15**

`calcSF` *calculates Scaling Factor (SF) values*

Description

This internally called function calculates the scaling factor (SF) values for Affymetrix microarrays, for use in computing GCS-score values

Usage

```
calcSF(diff, probetab, trim, clean.chip)
```

Arguments

`diff`	The GC-content background corrected probe groupings for every probesetID or transcriptionclusterID on the given array type. This is generated internally by the `computeSscore` function
`probetab`	The internal datafile that contains the probe groupings and annotations for each array type and method type
`trim`	The internal setting for the trimmed mean of every probe grouping on the array, as used in the calculation of SF. For 3' IVT arrays, the `trim` is set to 0.02 by default. For all newer WT-type arrays, the `trim` is set to 0.04 by default
`clean.chip`	The clean chiptype name, based on the platform design package name.

Value

`calcSF` returns a numeric SF value for a given CEL file

Examples

```
if (length(list.files(path = ".", pattern = "*.CEL")) != 0){

#Example of input, as the function would be called internally:
calcSF(diff, probetab, trim, clean.chip)
}
```

computeSscore *Computes GCS-score values*

Description

This internally called function computes the GCS-score values between two Affymetrix-style microarrays. The computeSscore function contains the majority of the GCS-score algorithm.

Usage

```
computeSscore(cel1, cel2, probeFile, bgp, method, infoKey, SF1 = NULL, SF2 = NULL,
              verbose = FALSE, trim = NULL, clean.chip)
```

Arguments

cel1	The 1st Affymetrix CEL file, as read in by the affxparser package
cel2	The 2nd Affymetrix CEL file, as read in by the affxparser package
probeFile	The internal datafile that contains the probe groupings and annotations for each array type and method type
bgp	The index of the probe location, GC-content, and annotations of the background probes of a given chip type. For WT-type arrays, the bgp consists of 16,943 antigenomic background probes. For 3' IVT arrays, the MisMatch (MM) probes are used to calculate the bgp list in both methods
method	Determines the method used to group and tally the probes when calculating GCS-score values
infoKey	The key of how to group the probes together for the GCS-score calculations. Determines the method used to group and tally the probes when calculating GCS-score values. For example, exon-level analysis groups probes by probeset_ids while gene-level groups probes by transcript_cluster_ids
SF1	If the user has predetermined scaling factors, input user Scaling Factor (SF) for the 1st CEL file. Otherwise, the computeSscore function will caluclate SF1 directly from the 1st CEL file
SF2	If the user has predetermined scaling factors, input user Scaling Factor (SF) for the 2nd CEL file. Otherwise, the computeSscore function will caluclate SF2 directly from the 2nd CEL file
verbose	If set to TRUE, additional information will be printed to the console while the algorithm is running.
trim	Internal parameter determined by chip type .trim=0.04 for WT-type arrays and 0.02 for 3' IVT type arrays
clean.chip	The clean chiptype name, based on the platform design package name.

Details

This internally called function computes the raw difference scores between the probes on each microarray, then groups the probes into probesets or transcript cluset ids, and normalizes the results to produce GCS-score values. The function returns the values to the main `GCscore2`, where BioConductor-based annotations are added to either the exon-level or gene-level probe groupings

Value

A `data.table` object with GCS-Score values for the probe groupings (determined by the `method` argument)

Examples

```
if (length(list.files(path = ".", pattern = "*.CEL")) != 0){

#Example of input, as the function would be called internally:
computeSscore(cel1, cel2, probeFile, bgp, infoKey, method, SF1 = NULL, SF2 = NULL,
              verbose = FALSE, trim = NULL)
}
```

`GCSscore` *Main GCS-score Function*

Description

The main function used to call and run the GCS-score algorithm.

Usage

```
GCSscore(celFile1 = NULL, celFile2 = NULL, celTable = NULL,celTab.names = FALSE,
typeFilter = 0, method = 1, rm.outmask = FALSE, SF1 = NULL, SF2 = NULL, fileout = FALSE,
gzip = FALSE, verbose = FALSE)
```

Arguments

`celFile1`	If a one comparison run is desired, enter the filename and path to the 1st Affymetrix CEL file
`celFile2`	If a one comparison run is desired, enter the filename and path to the 2nd Affymetrix CEL file
`celTable`	If a batch run is desired, enter the filename and path to the CSV file containing the batch information
`celTab.names`	If set to `TRUE`, then the GCS-score batch output is assigned the user-designated name (specified in the first column of the `celTable` CSV file (see examples))
`typeFilter`	If set to `0`, all available probe types are included in the calculation and normalization of the GCS-score values. If set to `1`, only probes well-annotated probe_ids (from BioConductor `.db` packages) are included in the calculation and normalization of the GCS-score output

method	This determines the method used to group and tally the probes_ids when calculating GCS-scores. For Whole Transcriptome arrays, for gene-level (transcript_cluster_id-based) analysis, set method = 1, and for exon-level (probeset_id-based) analysis, set method = 2. For the older generation arrays (3' IVT-style), if a PM-MM based background correction is desired, set method = 1 (PM-MM gives identical results to the original sscore package). If a GC-content based background correction is desired on the 3' IVT arrays, set method = 2
rm.outmask	If set to TRUE, then probes that are flagged as MASKED or OUTLIER in either CEL file 1 or CEL file 2 will be removed from the analysis. If set to FALSE, these probes are not filtered out and will be used in the GCS-score calculation
SF1	Input a pre-determined Scaling Factor (SF) for the 1st CEL file
SF2	Input a pre-determined Scaling Factor (SF) for the 2nd CEL file
fileout	Determines if the resulting GCS-score output is written to disk in a CSV format following the completion of the function.
gzip	If set to TRUE, the GCSscore output that is written to disk is compressed. This could prove useful if a large number runs are input using the batch submission
verbose	If set to TRUE, more information will be printed to the console during while the algorithm is running

Details

The input accepts individual CEL files or reads in a CSV file for batch runs. The user also inputs parameters to determine the method used by the GCS-score algorithm to group and tally the individual probes on a given array.

Value

An ExpressionSet object with GCS-score values for the probe groupings (determined by the method argument) and the relevant annotation informtaion

Examples

```
if (length(list.files(path = ".", pattern = "*.CEL")) != 0){

######################### Single run example ############################

# get the path to example CEL files provided with package:
celpath1 <- system.file("extdata/","MN_2_3.CEL", package = "GCSscore")
celpath2 <- system.file("extdata/","MN_4_1.CEL", package = "GCSscore")

# run GCSscore() function directly on the two .CEL files above:
GCSs.single <- GCSscore::GCSscore(celFile1 = celpath1, celFile2 = celpath2)

# convert GCSscore single-run from ExpressionSet to data.table:
GCSs.single.dt <-
  data.table::as.data.table(cbind(GCSs.single@featureData@data,
                                  GCSs.single@assayData[["exprs"]]))

# show all column names included in the output:
```

```
colnames(GCSs.single.dt)

# show simplified output of select columns and rows:
GCSs.single.dt[10000:10005,
               c("transcriptclusterid","symbol",
                 "ref_id","Sscore")]

######################## batch run example ############################

# get the path to example batch (.csv) file provided with package:
celtab_path <- system.file("extdata",
                           "GCSs_batch_ex.csv",
                           package = "GCSscore")

# read in the .CSV file using fread():
celtab <- data.table::fread(celtab_path)

# view structure of 'celTable' input:
celtab

# add the path to the sample CEL files to the batch input:
#   NOTE: this step is not necessary if the .CEL files
#         are in the working directory:

path <- system.file("extdata", package = "GCSscore")
celtab$CelFile1 <- celtab[,paste(path,CelFile1,sep="/")]
celtab$CelFile2 <- celtab[,paste(path,CelFile2,sep="/")]

# run GCSscore function on the batch input:
GCSs.batch <- GCSscore::GCSscore(celTable = celtab, celTab.names = TRUE)

# convert GCS-score output from 'ExpressionSet' to 'data.table':
GCSs.batch.dt <-
  data.table::as.data.table(cbind(GCSs.batch@featureData@data,
                                  GCSs.batch@assayData[["exprs"]]))

# show all columns included in the output:
colnames(GCSs.batch.dt)

# show simplified output of GCSscore batch example:
GCSs.batch.dt[10000:10005,
              c("transcriptclusterid","symbol",
                "example01","example02","example03")]
}
```

`get3primeIVTprobefileData` *Read a data file describing the probe sequences on an Affymetrix genechip*

Description

Read a data file describing the probe sequences on an Affymetrix genechip

Usage

```
get3primeIVTprobefileData(arraytype, datafile, pkgname, chip.pd, comparewithcdf = FALSE)
```

Arguments

arraytype	Character. Array type (e.g. 'HG-U133A')
datafile	Character. The filename of the input data file, or a connection (see example). If omitted a default name is constructed from arraytype (for details you will need to consult this function's source code).
pkgname	Character. Package name. If NULL the name is derived from arraytype.
chip.pd	Character. Name of the platform design file for the arraytype.
comparewithcdf	Logical. If TRUE, run a consistency check against a CDF package of the same name (what used to be Laurent's "extraparanoia".)

Details

This function serves as an interface between the (1) representation of array probe information data in the packages that are generated by makeProbePackageGCSs and (2) the vendor- and possibly version-specific way the data are represented in datafile.

datafile is a tabulator-separated file with one row per probe, and column names 'probesetid', 'fsetid', 'fid', 'x', 'y', and 'GC.count'. See the vignette for an example.

Value

A list with three components

dataEnv	an environment which contains the data frame with the probe sequences and the other probe data.
symVal	a named list of symbol value substitutions which can be used to customize the man pages. See createPackage.
pkgname	a character with the package name; will be the same as the function parameter pkgname if it was specified; otherwise, the name is constructed from the parameter arraytype.

See Also

makeProbePackageGCSs

Examples

```
if (length(list.files(path = ".", pattern = "*.CEL")) != 0){
  ## Example using the "Mouse430 2.0" chip-type

  ## Input the clean name for the given chip:
  chip <- "mouse4302"

  ## Input the .probe_tab file, as generated using the internal function:
     ##  IVT3primepFBuilder()

  probedata <- "GCSs.mouse4302.probeFile.probe_tab"

  ## Run function:
  get3primeIVTprobefileData(arraytype = chip, datafile = probedata)
  }
```

`getClariomSprobefileData` *Read a data file describing the probe sequences on an Affymetrix genechip*

Description

Read a data file describing the probe sequences on an Affymetrix genechip

Usage

```
getClariomSprobefileData(arraytype, datafile, pkgname, chip.pd, comparewithcdf = FALSE)
```

Arguments

`arraytype`	Character. Array type (e.g. 'HG-U133A')
`datafile`	Character with the filename of the input data file, or a connection (see example). If omitted a default name is constructed from `arraytype` (for details you will need to consult this function's source code).
`pkgname`	Character. Package name. If NULL the name is derived from `arraytype`.
`chip.pd`	Character. Name of the platform design file for the `arraytype`.
`comparewithcdf`	Logical. If TRUE, run a consistency check against a CDF package of the same name (what used to be Laurent's "extraparanoia".)

Details

This function serves as an interface between the (1) representation of array probe information data in the packages that are generated by `makeProbePackageGCSs` and (2) the vendor- and possibly version-specific way the data are represented in `datafile`.

`datafile` is a tabulator-separated file with one row per probe, and column names `'Probe X'`, `'Probe Y'`, `'Probe Sequence'`, and `'Probe.Set.Name'`. See the vignette for an example.

Value

A list with three components

dataEnv	an environment which contains the data frame with the probe sequences and the other probe data.
symVal	a named list of symbol value substitutions which can be used to customize the man pages. See createPackage.
pkgname	a character with the package name; will be the same as the function parameter pkgname if it was specified; otherwise, the name is constructed from the parameter arraytype.

See Also

makeProbePackage

Examples

```
if (length(list.files(path = ".", pattern = "*.CEL")) != 0){
## Example using the "Clariom S mouse" chip-type

## Input the clean name for the given chip:
chip <- "clariomsmouse"

## Input the .probe_tab file, as generated using the internal function:
   ##  ClariomSpFBuilder()

probedata <- "GCSs.clariomsmouse.probeFile.probe_tab"

## Run function:
getClariomSprobefileData(arraytype = chip, datafile = probedata)
}
```

---

getXTAprobefileData *Read a data file describing the probe sequences on an Affymetrix genechip*

Description

Read a data file describing the probe sequences on an Affymetrix genechip

Usage

```
getXTAprobefileData(arraytype, datafile, pkgname, chip.pd, comparewithcdf = FALSE)
```

Arguments

arraytype	Character. Array type (e.g. 'HG-U133A')
datafile	Character with the filename of the input data file, or a connection (see example). If omitted a default name is constructed from arraytype (for details you will need to consult this function's source code).
pkgname	Character. Package name. If NULL the name is derived from arraytype.
chip.pd	Character. Name of the platform design file for the arraytype.
comparewithcdf	Logical. If TRUE, run a consistency check against a CDF package of the same name (what used to be Laurent's "extraparanoia".)

Details

This function serves as an interface between the (1) representation of array probe information data in the packages that are generated by makeProbePackageGCSs and (2) the vendor- and possibly version-specific way the data are represented in datafile.

datafile is a tabulator-separated file with one row per probe, and column names 'Probe X', 'Probe Y', 'Probe Sequence', and 'Probe.Set.Name'. See the vignette for an example.

Value

A list with three components

dataEnv	an environment which contains the data frame with the probe sequences and the other probe data.
symVal	a named list of symbol value substitutions which can be used to customize the man pages. See createPackage.
pkgname	a character with the package name; will be the same as the function parameter pkgname if it was specified; otherwise, the name is constructed from the parameter arraytype.

See Also

makeProbePackage

Examples

```
if (length(list.files(path = ".", pattern = "*.CEL")) != 0){
## Example using the "MTA_1-0" chip-type:

## Input the clean name for the given chip:
chip <- "mta10"

## Input the .probe_tab file, as generated using the internal function:
   ##  ClariomDXTApFBuilder()

probedata <- "GCSs.mta10.probeFile.probe_tab"

## Run function:
```

```
getXTAprobefileData(arraytype = chip, datafile = probedata)
}
```

`mismatch` *Calculates mismatch values for probes*

Description

This internally called function calculates the background correction for each probe based on the median intensity of all background probes with the same GC-content of as the target probe in question

Usage

```
mismatch(probes, bgp, intensity)
```

Arguments

`probes`	probe indicies for target probes. Each probe index contains the gc-content of the probe
`bgp`	contains probe location (indicies), GC-content, and annotations of the background probes of a given chip type. For WT-type arrays, the bgp consists of 16,943 antigenomic background probes. For 3' IVT arrays, the mismatch (MM) probes are used to calculate the bgp list in both methods
`intensity`	The intensities value of the bgp probes as read in from the .CEL file

Details

This internally called function calculates the probe background correction based on the median intensity of all background probes with the same gc-content of as the target probe in question

Value

`mismatch` returns a numeric vector containing the gc-content based background correction for every probe included in the analysis

Examples

```
if (length(list.files(path = ".", pattern = "*.CEL")) != 0){

#Example of input, as the function would be called internally:
mismatch(probes, bgp, intensity)
}
```

normalize *Normalization of GCS-score values*

Description

Normalizes the GCS-score values using all scores within 3*SD of the mean. This normalization step occurs after the probes have been tallied and grouped into probe_ids, according to the method (probeset_id for exon-level or transcription_cluster_id for gene-level

Usage

```
normalize(Score)
```

Arguments

Score	The unnormalized GCS-score values (grouped and tallied according to the method selection) that are generated in the computeScore function

Value

normalize Returns a numeric vector containing normalized GCS-score values for every probe_id grouping included in the analysis

Examples

```
if (length(list.files(path = ".", pattern = "*.CEL")) != 0){

#Example of input, as the function would be called internally:
normalize(Score)
}
```

rawScore *Calculates the rawScore values*

Description

Calculates rawScore values based on differences between the two background corrected arrays in a given GCS-score analysis (e.g. CEL_1 vs. CEL_2), using the internally generated Statistical Difference Threshold (SDT) values.

Usage

```
rawScore(diff1, diff2, SDT1, SDT2)
```

Arguments

diff1	The gc-background-corrected values for the probe intensities on the 1st array
diff2	The gc-background-corrected values for the probe intensities on the 2nd array
SDT1	The internally calculated Statistical Difference Threshold (SDT=4*rawQ*SF) for the 1st array
SDT2	The internally calculated Statistical Difference Threshold (SDT=4*rawQ*SF) for the 2nd array

Value

rawScore returns a numeric vector containing the raw, ungrouped scores for every probe grouping included in the analysis (as determined by method)

Examples

```
if (length(list.files(path = ".", pattern = "*.CEL")) != 0){

#Example of input, as the function would be called internally:
rawScore(diff1, diff2, SDT1, SDT2)
}
```

---

zoneRQ *Calculates the zone-based RawQ values*

Description

This internally called function calculates the zone-based RawQ values. RawQ is a measure of the noise within a given zone on a microarray chip. This noise value is used in the error model contained in the GCS-score algorithm

Usage

```
zoneRQ(DT, affyCel, trim)
```

Arguments

DT	Internally generated data.table containing the .CEL data, generated from the list that is created by the affxparser package
affyCel	The .CEL file data, in list structure, as read in using the readCel function included in the affxparser package
trim	The internal setting for the trimmed mean of every probe grouping on the array. For 3' IVT arrays, the trim is set to 0.02 by default. For all newer WT-type arrays, the trim is set to 0.04 by default

Value

zoneRQ returns a numeric vector containing zone-based rawQ values for a given array

Examples

```
if (length(list.files(path = ".", pattern = "*.CEL")) != 0){

#Example of input, as the function would be called internally:
zoneRQ(DT, affyCel, trim)
}
```

Index

∗ **IO**
    get3primeIVTprobefileData, 6
    getClariomSprobefileData, 8
    getXTAprobefileData, 9
∗ **utilities**
    get3primeIVTprobefileData, 6
    getClariomSprobefileData, 8
    getXTAprobefileData, 9

calcSF, 2
computeSscore, 3
createPackage, *7*, *9*, *10*

GCSscore, 4
get3primeIVTprobefileData, 6
getClariomSprobefileData, 8
getXTAprobefileData, 9

mismatch, 11

normalize, 12

rawScore, 12

zoneRQ, 13