Package ‘PRECISION’

June 9, 2016

Type Package

Title PaiREd miCrorna sImulation Study desIgn mOlecular classificatioN

Version 0.1.0

Date 05/26/2016

Author Huei-Chung Huang, Li-Xuan Qin

Maintainer Huei-Chung Huang <huangh4@mskcc.org>

Description
This package asllows users to reuse the unique paired Agilent microRNA datasets and to reproduce our simulation studies in the paper linked to via the URL below.

License GPL (>= 2)

Depends R (>= 3.0.2)

Imports glmnet, limma, pamr, preprocessCore, ruv, sva, vsn

URL http://clincancerres.aacrjournals.org/content/20/13/3371.long

LazyData TRUE

RoxygenNote 5.0.1

Suggests knitr, rmarkdown

VignetteBuilder knitr

R topics documented:

`amplify.ary.eff` *Array effect amplification*

Description

Amplifies array effect in specified slides in a training set by a multiplier.

Usage

```
amplify.ary.eff(ary.eff.tr, amplify.slide.id, amplify.level, type = "shift")
```

Arguments

`ary.eff.tr`	the array effect training set to be modified, rows as probes, columns as samples.
`amplify.slide.id`	the slide IDs specified to have its array effect amplified; a vector of slide IDs if type = "shift" or "scale1", a list of vectors of slide IDs if type = "scale2".
`amplify.level`	a multiplier specified to amplify array effect by; a multiplier if type = "shift" or "scale1"; a vector of multipliers if type = "scale2" which has to be equaal length to the amplify.slide.id list.
`type`	a choice of amplification type, either "shift", "scale1" or "scale2" for either location shift amplification or scale amplification; default is "shift". Location shift amplification shifts the entire specified arrays up by a constant. Scaling amplification scales the expressions towards maximum and minimum per array.

Value

a array-effect-amplified array effect training data

Examples

```
## Not run:
smp.eff <- estimate.smp.eff(r.data = r.data.pl)
ary.eff <- estimate.ary.eff(r.data = r.data.pl,
                             non.r.data = non.r.data.pl)
ctrl.genes <- unique(rownames(r.data.pl))[grep("NC", unique(rownames(r.data.pl)))]
smp.eff.nc <- smp.eff[!rownames(smp.eff) %in% ctrl.genes, ]
ary.eff.nc <- ary.eff[!rownames(ary.eff) %in% ctrl.genes, ]
ary.eff.nc.tr <- ary.eff.nc[, c(1:64, 129:192)]
ary.eff.nc.tr.shift <- amplify.ary.eff(ary.eff.tr = ary.eff.nc.tr,
                                       amplify.slide.id = colnames(ary.eff.nc.tr)[1:64],
                                       amplify.level = 2, type = "shift")
ary.eff.nc.tr.scale1 <- amplify.ary.eff(ary.eff.tr = ary.eff.nc.tr,
                                       amplify.slide.id = colnames(ary.eff.nc.tr)[1:64],
                                        amplify.level = 2, type = "scale1")
amplify.slide.id <- list(1:40, 41:64, (129:160) - 64, (161:192) - 64)
for(i in 1:length(amplify.slide.id))
  amplify.slide.id[[i]] <- colnames(ary.eff.nc.tr)[amplify.slide.id[[i]]]
amplify.level <- c(1.2, 1.3, 1/3, 2/3)
ary.eff.nc.tr.scale2 <- amplify.ary.eff(ary.eff.tr = ary.eff.nc.tr,
                                        amplify.slide.id = amplify.slide.id,
                                        amplify.level = amplify.level,
                                        type = "scale2")
par(mfrow = c(2, 2), mar = c(4, 3, 2, 2))
rng <- range(ary.eff.nc.tr, ary.eff.nc.tr.shift, ary.eff.nc.tr.scale1, ary.eff.nc.tr.scale2)
boxplot(ary.eff.nc.tr, main = "original",
        ylim = rng, pch = 20, cex = 0.2, xaxt = "n")
boxplot(ary.eff.nc.tr.shift, main = "shifted",
        ylim = rng, pch = 20, cex = 0.2, xaxt = "n")
boxplot(ary.eff.nc.tr.scale1, main = "scaled 1",
        ylim = rng, pch = 20, cex = 0.2, xaxt = "n")
boxplot(ary.eff.nc.tr.scale2, main = "scaled 2",
        ylim = rng, pch = 20, cex = 0.2, xaxt = "n")

## End(Not run)
```

`blocking.design` *Blocking Design*

Description

Assigns arrays to samples with blocking design.

Usage

```
blocking.design(seed, num.smp)
```

Arguments

`seed`	specifies seed for random assignment using set.seed().
`num.smp`	number of samples.

Value

array-to-sample assignment, first half for group 1 (endometrial), second half for group 2 (ovarian)

Examples

```
blocking.design(seed = 1, num.smp = 128)
```

---

calc.confounding.level

Level of confounding calculation

Description

Calculates level of confounding between handling effects of a dataset and sample-group labels.

Usage

```
calc.confounding.level(data, group.id, nbe.genes)
```

Arguments

`data`	expression dataset, rows as probes, columns as samples.
`group.id`	sample group ID for the dataset.
`nbe.genes`	a vector of non-biological genes indicated as TRUE or 1, equal length as number of probes in the data.

Value

the level of confounding and the most correlated principal component of the non-biological genes in the dataset with the sample-group labels

Examples

```
smp.eff <- estimate.smp.eff(r.data = r.data.pl)
ary.eff <- estimate.ary.eff(r.data = r.data.pl,
                            non.r.data = non.r.data.pl)
ctrl.genes <- unique(rownames(r.data.pl))[grep("NC", unique(rownames(r.data.pl)))]
smp.eff.nc <- smp.eff[!rownames(smp.eff) %in% ctrl.genes, ]
ary.eff.nc <- ary.eff[!rownames(ary.eff) %in% ctrl.genes, ]
group.id <- substr(colnames(smp.eff.nc), 7, 7)
smp.eff.train.ind <- colnames(smp.eff.nc)[c(sample(which(group.id == "E"), size = 64),
sample(which(group.id == "V"), size = 64))]
ary.eff.train.ind <- colnames(ary.eff.nc)[c(1:64, 129:192)]
# randomly created a vector of Boolean for nbe.genes
nbe.genes <- sample(c(TRUE, FALSE), size = nrow(smp.eff.nc), replace = TRUE)
calc.confounding.level(data = smp.eff.nc[, smp.eff.train.ind],
                       group.id = substr(smp.eff.train.ind, 7, 7),
                       nbe.genes = nbe.genes)
```

`classify.gene.type` *Gene type classification*

Description

Classifies genes into technical, biological or other based on the differential expression analysis results of the estimated sample and array effect data.

Usage

```
classify.gene.type(smp.eff, ary.eff, smp.eff.train.ind, ary.eff.train.ind,
  group.id, ary.to.smp.assign)
```

Arguments

`smp.eff`	estimated sample effect data, rows as probes, columns as samples; can only take in either probe-level data with 10 probe per unique probe or probe-set-level data.
`ary.eff`	estimated array effect data, rows as probes, columns as samples; must have same dimensions and same probe name as sample effect data; can only take in either probe-level data with 10 probe per unique probe or probe-set-level data; must be the same dimensions as the estimated sample effect data.
`smp.eff.train.ind`	training set index for samples from the estimated sample effect data.
`ary.eff.train.ind`	training set index for arrays from the estimaed array effect data.
`group.id`	sample group ID for the estimated sample effect data.
`ary.to.smp.assign`	array-to-sample assignment for the arrays of the estimated array effect data.

Value

gene category vector, -1 for technical, 0 for other, 1 for biological genes

Examples

```
## Not run:
smp.eff <- estimate.smp.eff(r.data = r.data.pl)
ary.eff <- estimate.ary.eff(r.data = r.data.pl,
                            non.r.data = non.r.data.pl)
ctrl.genes <- unique(rownames(r.data.pl))[grep("NC", unique(rownames(r.data.pl)))]
smp.eff.nc <- smp.eff[!rownames(smp.eff) %in% ctrl.genes, ]
ary.eff.nc <- ary.eff[!rownames(ary.eff) %in% ctrl.genes, ]
group.id <- substr(colnames(smp.eff.nc), 7, 7)
smp.eff.train.ind <- colnames(smp.eff.nc)[c(sample(which(group.id == "E"), size = 64),
sample(which(group.id == "V"), size = 64))]
smp.eff.test.ind <- colnames(smp.eff.nc)[!colnames(smp.eff.nc) %in% smp.eff.train.ind]
ary.eff.train.ind <- colnames(ary.eff.nc)[c(1:64, 129:192)]
group.id.list <- list("all" = group.id,
                      "tr" = substr(smp.eff.train.ind, 7, 7),
                      "te" = substr(smp.eff.test.ind, 7, 7))
ary.to.smp.assign <- list("all" = c(rep(c("E", "V"), each = 64),
                          rep(c("V", "E"), each = 32)),
                          "tr" = rep(c("E", "V"), each = 64),
                          "te" = rep(c("V", "E"), each = 32))
gene.cat <- classify.gene.type(smp.eff = smp.eff.nc,
                               ary.eff = ary.eff.nc,
                               smp.eff.train.ind = smp.eff.train.ind,
                               ary.eff.train.ind = ary.eff.train.ind,
                               group.id = group.id.list,
                               ary.to.smp.assign = ary.to.smp.assign)

## End(Not run)
```

`confounding.design` *Confounding Design*

Description

Assigns arrays to samples with confounding design.

Usage

```
confounding.design(seed, num.smp, degree = "complete", rev.order = FALSE)
```

Arguments

`seed`	specifies seed for random assignment using set.seed().
`num.smp`	number of samples.
`degree`	level of confounding; has to be either "complete" or "partial" for either "complete confounding" or "partial confounding" design; default is "complete".
`rev.order`	FALSE indicating no reverse order, first half arrays to group 1 (endometrial), second half arrays to group 2 (ovarian); default is FALSE.

Value

array-to-sample assignment, first half for group 1 (endometrial), second half for group 2 (ovarian)

Examples

```
cc.ind <- confounding.design(seed = 1, num.smp = 128,
                             degree = "complete", rev.order = FALSE)
cc.ind <- confounding.design(seed = 1, num.smp = 128,
                             degree = "complete", rev.order = FALSE)
```

`estimate.ary.eff` *Estimated array effect*

Description

Estimates array effect from taking the differences between the expressions of the non-randomized and the randomized data, matched by samples

Usage

```
estimate.ary.eff(r.data, non.r.data)
```

Arguments

`r.data`	randomized expression dataset, rows as probes, columns as samples.
`non.r.data`	non-randomized expression dataset, rows as probes, columns as samples; must have same dimensions and same probe name as randomized data.

Value

an estimation of the array effect

Examples

```
ary.eff <- estimate.ary.eff(r.data = r.data.pl, non.r.data = non.r.data.pl)
```

`estimate.smp.eff` *Estimated Sample Effect*

Description

Estimates sample effect from the expressions of the randomized data

Usage

```
estimate.smp.eff(r.data)
```

Arguments

`r.data`	randomized expression dataset, rows as probes, columns as samples.

Value

an estimation of the sample effect

Examples

```
smp.eff <- estimate.smp.eff(r.data = r.data.pl)
```

`lasso.intcv` *Least absolute shrinkage and selection operator through internal cross validation*

Description

Builds a LASSO classifier using internal cross validation, with a 5-fold cross validation as default.

Usage

```
lasso.intcv(kfold = 5, X, y, seed, alp = 1)
```

Arguments

`kfold`	number of folds; default is 5.
`X`	expression dataset to be trained, rows as probes, columns as samples.
`y`	sample group corresponding to the dataset to be trained; should have the equal length as the number of samples as X.
`seed`	specifies seed for random assignment using set.seed().
`alp`	alpha, the penalty type from 0 to 1; default alp = 1 for LASSO; alp = 0 for a ridge classifier.

Value

a LASSO classifier

Examples

```
set.seed(101)
smp.eff <- estimate.smp.eff(r.data = r.data.pl)
ctrl.genes <- unique(rownames(r.data.pl))[grep("NC", unique(rownames(r.data.pl)))]
smp.eff.nc <- smp.eff[!rownames(smp.eff) %in% ctrl.genes, ]
group.id <- substr(colnames(smp.eff.nc), 7, 7)

smp.eff.train.ind <- colnames(smp.eff.nc)[c(sample(which(group.id == "E"), size = 64),
                                            sample(which(group.id == "V"), size = 64))]
smp.eff.nc.tr <- smp.eff.nc[, smp.eff.train.ind]

lasso.int <- lasso.intcv(X = smp.eff.nc.tr,
                         y = substr(colnames(smp.eff.nc.tr), 7, 7),
                         kfold = 5, seed = 1, alp = 1)
```

`lasso.predict` *Prediction with least absolute shrinkage and selection operator classifier*

Description

Predicts from a least absolute shrinkage and selection operator fit.

Usage

```
lasso.predict(lasso.intcv.model, pred.obj, pred.obj.group.id)
```

Arguments

`lasso.intcv.model`	a LASSO classifier built with lasso.intcv().
`pred.obj`	expression dataset to have its sample group predicted, rows as probes, columns as samples; should have equal number of probes as the data trained.
`pred.obj.group.id`	sample group corresponding to the dataset to be predicted; should have equal length as the number of samples as pred.obj.

Value

predicted object, predicted error and predicted features

Examples

```
set.seed(101)
smp.eff <- estimate.smp.eff(r.data = r.data.pl)
ctrl.genes <- unique(rownames(r.data.pl))[grep("NC", unique(rownames(r.data.pl)))]
smp.eff.nc <- smp.eff[!rownames(smp.eff) %in% ctrl.genes, ]
group.id <- substr(colnames(smp.eff.nc), 7, 7)

smp.eff.train.ind <- colnames(smp.eff.nc)[c(sample(which(group.id == "E"), size = 64),
                                          sample(which(group.id == "V"), size = 64))]
smp.eff.test.ind <- colnames(smp.eff.nc)[!colnames(smp.eff.nc) %in% smp.eff.train.ind]
smp.eff.nc.tr <- smp.eff.nc[, smp.eff.train.ind]
smp.eff.nc.te <- smp.eff.nc[, smp.eff.test.ind]

lasso.int <- lasso.intcv(X = smp.eff.nc.tr,
                         y = substr(colnames(smp.eff.nc.tr), 7, 7),
                         kfold = 5, seed = 1, alp = 1)

lasso.pred <- lasso.predict(lasso.intcv.model = lasso.int,
                            pred.obj = smp.eff.nc.te,
                            pred.obj.group.id = substr(colnames(smp.eff.nc.te), 7, 7))
lasso.int$mc
lasso.pred$mc
```

`limma.pbset` *Differential expression analysis of probe-set data*

Description

Performs two-group differential expression analysis using "limma".

Usage

```
limma.pbset(data, group.id, group.id.level = c("E", "V"), pbset.id = NULL)
```

Arguments

`data`	expression dataset to be differentially expression analyzed, rows as unique probe-sets, columns as samples.
`group.id`	sample group label; must be a 2-level non-numeric factor vector.
`group.id.level`	sample group label level, the first one being the reference level; default = c("E", "V") in our studies when comparing endometrial to ovarian samples.
`pbset.id`	unique probe-set name; default is NULL, the rownames of the dataset.

Value

differential expression anlysis results, group means, group standard deviations

Examples

```
r.data.psl <- med.sum.pbset(data = r.data.pl,
                            num.per.unipbset = 10)
ctrl.genes <- unique(rownames(r.data.pl))[grep("NC", unique(rownames(r.data.pl)))]
r.data.psl.nc <- r.data.psl[!rownames(r.data.psl) %in% ctrl.genes, ]
group.id <- substr(colnames(r.data.psl.nc), 7, 7)
group.id.level <- levels(as.factor(group.id))
limma.fit.r.data<- limma.pbset(data = r.data.psl.nc,
                               group.id = group.id,
                               group.id.level = group.id.level)
                               table(limma.fit.r.data$P.Value < 0.01, dnn = "DE genes")
```

---

med.norm *Median nomalization*

Description

Normalizes training dataset so that each array shares a same median, stores the median from the training dataset as the reference to frozen median normalize test dataset.

Usage

```
med.norm(train, test = NULL)
```

Arguments

`train`	training data to be median normalized, rows as probes, columns as samples.
`test`	test data to be frozen median normalized, rows as probes with equal number of rows as the training set, columns as samples.

Value

a list of two datasets, the normalized training set and the frozen normalied test set

Examples

```
set.seed(101)
group.id <- substr(colnames(non.r.data.pl), 7, 7)
train.ind <- colnames(non.r.data.pl)[c(sample(which(group.id == "E"), size = 64),
                               sample(which(group.id == "V"), size = 64))]
train.dat <- non.r.data.pl[, train.ind]
test.dat <- non.r.data.pl[, !colnames(non.r.data.pl) %in%train.ind]
data.mn <- med.norm(train = train.dat)
str(data.mn)
data.mn <- med.norm(train = train.dat, test = test.dat)
str(data.mn)
```

med.sum.pbset *Probe-set median summarization*

Description

Summarizes probe-set using median of each unique probe, only taking in data matrix with a fixed number of probes per unique probe-set.

Usage

```
med.sum.pbset(data, pbset.id = NULL, num.per.unipbset = 10)
```

Arguments

`data`	expression data to be summarized, rows as probes, columns as samples; rownames as probe names; only accept data matrix with a fixed number of probes per unique probe-set. If the data is already on the probe-set level, no manipulation will be done.
`pbset.id`	unique probe-set name, if not specified then use the unique probe name of the data.
`num.per.unipbset`	number of probes for each unique probe-set; default is 10.

Value

probe-set median summarized data

Examples

```
r.data.psl <- med.sum.pbset(data = r.data.pl,
                            num.per.unipbset = 10)
```

non.r.data.pl *The non-randomized (test) probe-level dataset, 10 probes for each unique probe*

Description

The non-randomized probe-level dataset, non-control-probe-removed, 10 probes for each unique probe, no background adjusted and after logged 2.

Usage

```
non.r.data.pl
```

Format

A data matrix with 1810 rows (probes) and 192 columns (samples), column names ending with E/V are endometrial/ovarian samples.

`pam.intcv` *Nearest shrunken centroid through internal cross validation*

Description

Builds a PAM classifier using internal cross validation, with 5-fold cross validation as the default.

Usage

```
pam.intcv(X, y, vt.k = NULL, n.k = 30, kfold = 5, folds = NULL, seed)
```

Arguments

`X`	expression dataset to be trained, rows as probes, columns as samples.
`y`	sample group corresponding to the data to be trained; must have the equal length as the number of samples as X.
`vt.k`	custom-specified threshold list; default is NULL predetermined by the PAM package.
`n.k`	number of threshold values desired; default is 30.
`kfold`	number of folds for cross validation; default is 5
`folds`	prespecifies samples to folds; default is NULL for no prespecification.
`seed`	specifies seed for random assignment using set.seed().

Value

a PAM classifier

Examples

```
set.seed(101)
smp.eff <- estimate.smp.eff(r.data = r.data.pl)
ctrl.genes <- unique(rownames(r.data.pl))[grep("NC", unique(rownames(r.data.pl)))]
smp.eff.nc <- smp.eff[!rownames(smp.eff) %in% ctrl.genes, ]
group.id <- substr(colnames(smp.eff.nc), 7, 7)

smp.eff.train.ind <- colnames(smp.eff.nc)[c(sample(which(group.id == "E"), size = 64),
                                           sample(which(group.id == "V"), size = 64))]
smp.eff.nc.tr <- smp.eff.nc[, smp.eff.train.ind]

pam.int <- pam.intcv(X = smp.eff.nc.tr,
                     y = substr(colnames(smp.eff.nc.tr), 7, 7),
                     kfold = 5, seed = 1)
```

pam.predict *Prediction with nearest shrunken centroid classifier*

Description

Predicts from a nearest shrunken centroid fit.

Usage

```
pam.predict(pam.intcv.model, pred.obj, pred.obj.group.id)
```

Arguments

`pam.intcv.model`	a PAM classifier built with pam.intcv().
`pred.obj`	expression dataset to have its sample group predicted, rows as probes, columns as samples; should have equal number of probes as the data trained.
`pred.obj.group.id`	sample group corresponding to the data to be predicted; should have equal length as the number of samples as pred.obj.

Value

predicted object, predicted error and predicted features

Examples

```
set.seed(101)
smp.eff <- estimate.smp.eff(r.data = r.data.pl)
ctrl.genes <- unique(rownames(r.data.pl))[grep("NC", unique(rownames(r.data.pl)))]
smp.eff.nc <- smp.eff[!rownames(smp.eff) %in% ctrl.genes, ]
group.id <- substr(colnames(smp.eff.nc), 7, 7)

smp.eff.train.ind <- colnames(smp.eff.nc)[c(sample(which(group.id == "E"), size = 64),
                                            sample(which(group.id == "V"), size = 64))]
smp.eff.test.ind <- colnames(smp.eff.nc)[!colnames(smp.eff.nc) %in% smp.eff.train.ind]
smp.eff.nc.tr <- smp.eff.nc[, smp.eff.train.ind]
smp.eff.nc.te <- smp.eff.nc[, smp.eff.test.ind]

pam.int <- pam.intcv(X = smp.eff.nc.tr,
                     y = substr(colnames(smp.eff.nc.tr), 7, 7),
                     kfold = 5, seed = 1)

pam.pred <- pam.predict(pam.intcv.model = pam.int,
                        pred.obj = smp.eff.nc.te,
                        pred.obj.group.id = substr(colnames(smp.eff.nc.te), 7, 7))
pam.int$mc
pam.pred$mc
```

`per.unipbset.truncate` *Classification analysis of uniformly-handled data*

Description

Performs classification analysis on the uniformly-handled data by reassigning samples to training and test set in Qin et al. (see reference).

Usage

```
per.unipbset.truncate(data, pbset.id = NULL, num.per.unipbset = 10)
```

Arguments

`data`	expression data, rows as probes, columns as samples.
`pbset.id`	unique probe-set name; default is NULL, the rownames of the dataset.
`num.per.unipbset`	number of probes for each unique probe-set; default is 10.

Value

benchmark analysis results with list of models built and internal and external misclassification error stored, also a list of assignment stored

References

http://clincancerres.aacrjournals.org/content/20/13/3371.long

Examples

```
r.data.pl.p5 <- per.unipbset.truncate(data = r.data.pl,
num.per.unipbset = 5)
```

`precision.simulate` *Classification analysis of simulation study*

Description

Performs simulation study in Qin et al. (see reference).

Usage

```
precision.simulate(myseed, N, smp.eff.tr, smp.eff.te, ary.eff.tr, ary.eff.te,
  group.id.tr, group.id.te, design.list = c("CC+", "CC-", "PC+", "PC-"),
  norm.list = c("NN", "QN"), class.list = c("PAM", "LASSO"),
  batch.id = NULL, icombat = FALSE, isva = FALSE, iruv = FALSE,
  smp.eff.tr.ctrl = NULL, ary.eff.tr.ctrl = NULL, norm.funcs = NULL,
  class.funcs = NULL, pred.funcs = NULL)
```

Arguments

`myseed`	specifies seed for random assignment using set.seed().
`N`	number of simulation runs.
`smp.eff.tr`	sample effect training data, rows as probes, columns as samples.
`smp.eff.te`	sample effect test data, rows as probes, columns as samples; must have same number of probes and probe names as sample effect training data.
`ary.eff.tr`	array effect training data, rows as probes, columns as samples; must have same dimensions and same probe name as sample effect training data.
`ary.eff.te`	array effect test data, rows as probes, columns as samples; must have same dimensions and same probe name as array effect test data.
`group.id.tr`	sample group ID of the sample effect training data; has to be "E" and "V".
`group.id.te`	sample group ID of the sample effect test data; has to be "E" and "V".
`design.list`	a list of strings for study designs compared in the simulation study; built-in designs are "CC+", "CC-", "PC+", "PC-", "BLK", and "STR" for "Complete Confounding 1", "Complete Confounding 2", "Partial Confounding 1", "Partial Confounding 2", "Blocking", "Stratification" in Qin et al.
`norm.list`	a list of strings for normalization methods compared in the simulation study; build-in available normalization methods are "NN", "QN", "MN", "VSN" for "No Normalization", "Quantile Normalization", "Median Normalization", "Variance Stablizing Normalization"; user can provide a list of normalization methods given the functions are supplied (also see norm.funcs).
`class.list`	a list of strings for classification methods compared in the simulation study; built-in classification methods are "PAM" and "LASSO" for "prediction analysis for microarrays" and "least absolute shrinkage and selection operator"; user can provide a list of classification methods given the correponding model-building and predicting functions are supplied (also see class.funcs and pred.funcs).
`batch.id`	a list of batch id by the number of batches when stratification study design is specified; default is NULL.
`icombat`	indicator for combat adjustment; default is not to adjust (icombat = FALSE).
`isva`	indicator for sva adjustment; default is not to adjust (isva = FALSE).
`iruv`	indicator for RUV-4 adjustment; default is not to adjust (iruv = FALSE).
`smp.eff.tr.ctrl`	negative-control gene sample effect data if iruv = TRUE, rows as probes, columns as samples; must have same number of probes and probe names as non-control gene sample effect training data.
`ary.eff.tr.ctrl`	negative-control gene array effect data if iruv = TRUE, rows as probes, columns as samples; must have same number of probes and probe names as non-control gene array effect training data.
`norm.funcs`	a list of strings for names of user-defined normalization method functions, in the order of norm.list excluding any built-in normalization methods.
`class.funcs`	a list of strings for names of user-defined classification model-building functions, in the order of class.list excluding any built-in classification methods.
`pred.funcs`	a list of strings for names of user-defined classification predicting functions, in the order of class.list excluding any built-in classification methods.

Value

simulated results with list of models built and internal and external misclassification error stored, also a list of assignment stored

References

http://clincancerres.aacrjournals.org/content/20/13/3371.long

Examples

```
## Not run:
set.seed(101)
smp.eff <- estimate.smp.eff(r.data = r.data.pl)
ary.eff <- estimate.ary.eff(r.data = r.data.pl,
                             non.r.data = non.r.data.pl)

ctrl.genes <- unique(rownames(r.data.pl))[grep("NC", unique(rownames(r.data.pl)))]

smp.eff.nc <- smp.eff[!rownames(smp.eff) %in% ctrl.genes, ]
ary.eff.nc <- ary.eff[!rownames(ary.eff) %in% ctrl.genes, ]

group.id <- substr(colnames(smp.eff.nc), 7, 7)

# randomly split sample effect data into training and test set with
# equal number of endometrial and ovarian samples
smp.eff.train.ind <- colnames(smp.eff.nc)[c(sample(which(group.id == "E"), size = 64),
                                           sample(which(group.id == "V"), size = 64))]
smp.eff.test.ind <- colnames(smp.eff.nc)[!colnames(smp.eff.nc) %in% smp.eff.train.ind]
smp.eff.train.test.split =
  list("tr" = smp.eff.train.ind,
       "te" = smp.eff.test.ind)

# non-randomly split array effect data into training and test set
ary.eff.train.test.split =
  list("tr" = c(1:64, 129:192),
       "te" = 65:128)

smp.eff.nc.tr <- smp.eff.nc[, smp.eff.train.ind]
smp.eff.nc.te <- smp.eff.nc[, smp.eff.test.ind]
ary.eff.nc.tr <- ary.eff.nc[, c(1:64, 129:192)]
ary.eff.nc.te <- ary.eff.nc[, 65:128]

# Simulation without batch adjustment
precision.results <- precision.simulate(myseed = 1, N = 3,
                                        smp.eff.tr = smp.eff.nc.tr,
                                        smp.eff.te = smp.eff.nc.te,
                                        ary.eff.tr = ary.eff.nc.tr,
                                        ary.eff.te = ary.eff.nc.te,
                                    group.id.tr = substr(colnames(smp.eff.nc.tr), 7, 7),
                                    group.id.te = substr(colnames(smp.eff.nc.te), 7, 7),
                                        design.list = c("PC-", "STR"),
                                        norm.list = c("NN", "QN"),
                                              class.list = c("PAM", "LASSO"),
                                              batch.id = list(1:40,
                                                              41:64,
                                                              (129:160) - 64,
                                                              (161:192) - 64))

# Simulation with RUV-4 batch adjustment
smp.eff.ctrl <- smp.eff[rownames(smp.eff) %in% ctrl.genes, ]
ary.eff.ctrl <- ary.eff[rownames(ary.eff) %in% ctrl.genes, ]

smp.eff.tr.ctrl <- smp.eff.ctrl[, smp.eff.train.test.split$tr]
ary.eff.tr.ctrl <- ary.eff.ctrl[, ary.eff.train.test.split$tr]

precision.ruv4.results <- precision.simulate(myseed = 1, N = 3,
                                             smp.eff.tr = smp.eff.nc.tr,
                                             smp.eff.te = smp.eff.nc.te,
                                             ary.eff.tr = ary.eff.nc.tr,
                                             ary.eff.te = ary.eff.nc.te,
                                     group.id.tr = substr(colnames(smp.eff.nc.tr), 7, 7),
                                     group.id.te = substr(colnames(smp.eff.nc.te), 7, 7),
                                             design.list = c("PC-", "STR"),
                                             norm.list = c("NN", "QN"),
                                             class.list = c("PAM", "LASSO"),
                                             batch.id = list(1:40,
                                                             41:64,
                                                             (129:160) - 64,
                                                             (161:192) - 64),
                                             iruv = TRUE,
                                             smp.eff.tr.ctrl = smp.eff.tr.ctrl,
                                             ary.eff.tr.ctrl = ary.eff.tr.ctrl)

## End(Not run)
```

quant.norm *Quantile nomalization*

Description

Normalizes training dataset with quantile normalization, stores the quantiles from the training dataset as the references to frozen quantile normalize test dataset.

Usage

```
quant.norm(train, test = NULL)
```

Arguments

train	training data to be quantile normalized, rows as probes, columns as samples.
test	test data to be frozen quantile normalized, rows as probes with equal number of rows as the training set, columns as samples.

Value

a list of two datasets, the normalized training set and the frozen normalied test set

References

Bolstad, B. M., Irizarry R. A., Astrand, M, and Speed, T. P. (2003) A Comparison of Normalization Methods for High Density Oligonucleotide Array Data Based on Bias and Variance. Bioinformatics 19(2) ,pp 185-193. http://bmbolstad.com/misc/normalize/normalize.html

Examples

```
set.seed(101)
group.id <- substr(colnames(non.r.data.pl), 7, 7)
train.ind <- colnames(non.r.data.pl)[c(sample(which(group.id == "E"), size = 64),
                               sample(which(group.id == "V"), size = 64))]
train.dat <- non.r.data.pl[, train.ind]
test.dat <- non.r.data.pl[, !colnames(non.r.data.pl) %in%train.ind]
data.qn <- quant.norm(train = train.dat)
str(data.qn)
data.qn <- quant.norm(train = train.dat, test = test.dat)
str(data.qn)
```

`r.data.pl` *The randomized (benchmark) probe-level dataset, 10 probes for each unique probe*

Description

The randomized probe-level dataset, non-control-probe-removed, 10 probes for each unique probe, no background adjusted and after logged 2.

Usage

```
r.data.pl
```

Format

A data matrix with 1810 rows (probes) and 192 columns (samples), column names ending with E/V are endometrial/ovarian samples.

`reduce.signal` *Biological signal reduction*

Description

Reduces biological effect between sample group by a multiplier.

Usage

```
reduce.signal(smp.eff, group.id, group.id.level = c("E", "V"),
  reduce.multiplier = 1/2, pbset.id = NULL)
```

Arguments

`smp.eff`	estimated sample effect data, rows as probes, columns as samples; can only take in probe-level data with 10 probe per unique probe.
`group.id`	sample group ID for the estimated sample effect data.
`group.id.level`	sample group label level; default = c("E", "V") in our studies when comparing endometrial to ovarian samples.
`reduce.multiplier`	a multiplier specified to reduce between-sample-group signal by; default is 1/2.
`pbset.id`	unique probe-set name, if not specified then use the unique probe name of the data.

Value

estimated sample effect data with reduced biological signal

Examples

```
smp.eff <- estimate.smp.eff(r.data = r.data.pl)
ary.eff <- estimate.ary.eff(r.data = r.data.pl,
                            non.r.data = non.r.data.pl)
ctrl.genes <- unique(rownames(r.data.pl))[grep("NC", unique(rownames(r.data.pl)))]
smp.eff.nc <- smp.eff[!rownames(smp.eff) %in% ctrl.genes, ]
ary.eff.nc <- ary.eff[!rownames(ary.eff) %in% ctrl.genes, ]
group.id <- substr(colnames(smp.eff.nc), 7, 7)
redhalf.smp.eff.nc <- reduce.signal(smp.eff = smp.eff.nc,
                                    group.id = group.id,
                                    group.id.level = c("E", "V"),
                                    reduce.multiplier = 1/2)
```

rehybridize — *Rehybridization with an array-to-sample assignment*

Description

Creates simulated dataset through rehybridization with a specified array-to-sample assignment.

Usage

```
rehybridize(smp.eff, ary.eff, group.id, group.id.level = c("E", "V"),
  ary.to.smp.assign, icombat = FALSE, isva = FALSE, iruv = FALSE,
  smp.eff.ctrl = NULL, ary.eff.ctrl = NULL)
```

Arguments

`smp.eff`	sample effect data, rows as probes, columns as samples.
`ary.eff`	array effect data, rows as probes, columns as samples; must have same dimensions and same probe name as sample effect data.
`group.id`	sample group label; must be a 2-level non-numeric factor vector.
`group.id.level`	sample group label level, the first one being the reference level; default = c("E", "V") in our studies when comparing endometrial to ovarian samples.
`ary.to.smp.assign`	array-to-sample assignment, equal length as number of samples of sample effect data; first half of the vector assigning to endometrial, second half to ovarian.
`icombat`	indicator for combat adjustment; default is not to adjust, icombat = FALSE.
`isva`	indicator for sva adjustment; default is not to adjust, isva = FALSE.
`iruv`	indicator for RUV-4 adjustment; default is not to adjust, iruv = FALSE.
`smp.eff.ctrl`	negative-control gene sample effect data if iruv = TRUE.
`ary.eff.ctrl`	negative-control gene array effect data if iruv = TRUE

Value

simulated data, after batch adjustment if specified

Examples

```
## Not run:
smp.eff <- estimate.smp.eff(r.data = r.data.pl)
ary.eff <- estimate.ary.eff(r.data = r.data.pl,
                            non.r.data = non.r.data.pl)
ctrl.genes <- unique(rownames(r.data.pl))[grep("NC", unique(rownames(r.data.pl)))]
smp.eff.nc <- smp.eff[!rownames(smp.eff) %in% ctrl.genes, ]
ary.eff.nc <- ary.eff[!rownames(ary.eff) %in% ctrl.genes, ]
assign.ind <- confounding.design(seed = 1, num.smp = 192,
degree = "complete", rev.order = FALSE)
```

```
group.id <- substr(colnames(smp.eff.nc), 7, 7)
sim.data.raw <- rehybridize(smp.eff = smp.eff.nc,
                            ary.eff = ary.eff.nc,
                            group.id = group.id,
                            ary.to.smp.assign = assign.ind)
sim.data.sva <- rehybridize(smp.eff = smp.eff.nc,
                            ary.eff = ary.eff.nc,
                            group.id = group.id,
                            ary.to.smp.assign = assign.ind,
                            isva = TRUE)
smp.eff.ctrl <- smp.eff[rownames(smp.eff) %in% ctrl.genes, ]
ary.eff.ctrl <- ary.eff[rownames(ary.eff) %in% ctrl.genes, ]
sim.data.ruv <- rehybridize(smp.eff = smp.eff.nc,
                            ary.eff = ary.eff.nc,
                            group.id = group.id,
                            ary.to.smp.assign = assign.ind,
                            iruv = TRUE,
                            smp.eff.ctrl = smp.eff.ctrl,
                            ary.eff.ctrl = ary.eff.ctrl)

## End(Not run)
```

stratification.design *Stratification Design*

Description

Assigns arrays to samples with stratification design.

Usage

```
stratification.design(seed, num.smp, batch.id)
```

Arguments

seed	specifies seed for random assignment using set.seed().
num.smp	number of samples.
batch.id	sample group ID for the estimated sample effect data.

Value

array-to-sample assignment, first half for group 1 (endometrial), second half for group 2 (ovarian)

Examples

```
batch.id <- list(1:40, 41:64, (129:160) - 64, (161:192) - 64)
str.ind <- stratification.design(seed = 1, num.smp = 128,
                                 batch.id = batch.id)
```

`uni.handled.simulate` *Classification analysis of uniformly-handled data*

Description

Performs classification analysis on the uniformly-handled data by reassigning samples to training and test set in Qin et al. (see reference).

Usage

```
uni.handled.simulate(myseed, N, smp.eff, norm.list = c("NN", "QN"),
  class.list = c("PAM", "LASSO"), norm.funcs = NULL, class.funcs = NULL,
  pred.funcs = NULL)
```

Arguments

`myseed`	specifies seed for random assignment using set.seed().
`N`	number of simulation runs.
`smp.eff`	sample effect data, rows as probes, columns as samples.
`norm.list`	a list of strings for normalization methods compared in the simulation study; built-in normalization methods includes "NN", "QN", "MN", "VSN" for "No Normalization", "Quantile Normalization", "Median Normalization", "Variance Stablizing Normalization"; user can provide a list of normalization methods given the functions are supplied (also see norm.funcs).
`class.list`	a list of strings for classification methods compared in the simulation study; built-in classification methods are "PAM" and "LASSO" for "prediction analysis for microarrays" and "least absolute shrinkage and selection operator"; user can provide a list of classification methods given the correponding model-building and predicting functions are supplied (also see class.funcs and pred.funcs).
`norm.funcs`	a list of strings for names of user-defined normalization method functions, in the order of norm.list excluding any built-in normalization methods.
`class.funcs`	a list of strings for names of user-defined classification model-building functions, in the order of class.list excluding any built-in classification methods.
`pred.funcs`	a list of strings for names of user-defined classification predicting functions, in the order of class.list excluding any built-in classification methods.

Value

benchmark analysis results with list of models built and internal and external misclassification error stored, also a list of assignment stored

References

http://clincancerres.aacrjournals.org/content/20/13/3371.long

Examples

```
## Not run:
smp.eff <- estimate.smp.eff(r.data = r.data.pl)
ctrl.genes <- unique(rownames(r.data.pl))[grep("NC", unique(rownames(r.data.pl)))]
smp.eff.nc <- smp.eff[!rownames(smp.eff) %in% ctrl.genes, ]
uni.handled.results <- uni.handled.simulate(myseed = 1, N = 3,
                                            smp.eff = smp.eff.nc,
                                            norm.list = c("NN", "QN"),
                                            class.list = c("PAM", "LASSO"))

## End(Not run)
```

---

vs.norm *Variance stabalizing nomalization*

Description

Normalizes training dataset with vsn, stores the fitted vsn model from the training dataset as the reference to frozen variance stabalizng normalize test dataset.

Usage

```
vs.norm(train, test = NULL)
```

Arguments

`train`	training data to be variance stabalizing normalized, rows as probes, columns as samples.
`test`	test data to be frozen variance stabalizing normalized, rows as probes with equal number of rows as the training set, columns as samples.

Value

a list of two datasets, the normalized training set and the frozen normalied test set

References

Wolfgang Huber, Anja von Heydebreck, Holger Sueltmann, Annemarie Poustka and Martin Vingron. Variance Stabilization Applied to Microarray Data Calibration and to the Quantification of Differential Expression. Bioinformatics 18, S96-S104 (2002).

Examples

```
## Not run:
set.seed(101)
group.id <- substr(colnames(non.r.data.pl), 7, 7)
train.ind <- colnames(non.r.data.pl)[c(sample(which(group.id == "E"), size = 64),
                                      sample(which(group.id == "V"), size = 64))]
```

```
train.dat <- non.r.data.pl[, train.ind]
test.dat <- non.r.data.pl[, !colnames(non.r.data.pl) %in%train.ind]
data.vsn <- vs.norm(train = train.dat)
str(data.vsn)
data.vsn <- vs.norm(train = train.dat, test = test.dat)
str(data.vsn)

## End(Not run)
```

Index