Package 'crumblr'

November 5, 2025

Type Package

Title Count ratio uncertainty modeling base linear regression

Version 1.3.0

Date 2025-03-11

Description Crumblr enables analysis of count ratio data using precision weighted linear (mixed) models. It uses an asymptotic normal approximation of the variance following the centered log ration transform (CLR) that is widely used in compositional data analysis. Crumblr provides a fast, flexible alternative to GLMs and GLMM's while retaining high power and controlling the false positive rate.

VignetteBuilder knitr

License Artistic-2.0

Encoding UTF-8

URL https://DiseaseNeurogenomics.github.io/crumblr

BugReports https://github.com/DiseaseNeurogenomics/crumblr/issues

Suggests BiocStyle, RUnit, knitr, rmarkdown, dreamlet, muscat, ExperimentHub, scater, HMP, reshape2, glue, tidyverse, BiocGenerics, compositions

biocViews RNASeq, GeneExpression, DifferentialExpression, BatchEffect, QualityControl, SingleCell, Regression, Epigenetics, FunctionalGenomics, Transcriptomics, Normalization, Clustering, DimensionReduction, Preprocessing, Software

Depends R (>= 4.4.0), ggplot2, methods

Imports Rdpack, viridis, tidytree, variancePartition (>= 1.36.3), SingleCellExperiment, ggtree, dplyr, stats, MASS, Rfast

RoxygenNote 7.3.2

RdMacros Rdpack

LazyData false

NeedsCompilation no

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

git_branch devel

git_last_commit 00871a5

git_last_commit_date 2025-10-29

Repository Bioconductor 3.23

Date/Publication 2025-11-05

Author Gabriel Hoffman [aut, cre] (ORCID:
<https://orcid.org/0000-0002-0957-0224>)

Maintainer Gabriel Hoffman <gabriel.hoffman@mssm.edu>

Contents

`crumblr-package` *crumblr*

Description

Crumblr enables analysis of count ratio data using precision weighted linear (mixed) models. It uses an asymptotic normal approximation of the variance following the centered log ration transform (CLR) that is widely used in compositional data analysis. Crumblr provides a fast, flexible alternative to GLMs and GLMM's while retaining high power and controlling the false positive rate.

Value

none

`buildClusterTree` *Perform hierarchical clustering on reducedDim*

Description

Perform hierarchical clustering dimension reduction from single cell expression data

Usage

```
buildClusterTree(
  sce,
  reduction,
  labelCol,
 method.dist = c("cosine", "euclidean", "maximum", "manhattan", "canberra", "binary",
    "minkowski"),
  method.hclust = c("complete", "ward.D", "ward.D2")
)
```

Arguments

`sce`	`SingleCellExperiment` object
`reduction`	field of reducedDims(sce) to use
`labelCol`	column in `SingleCellExperiment` storing cell type annotations
`method.dist`	method for `dist(..,method=method.dist)`
`method.hclust`	method for `hclust(..,method=method.hclust)`

Value

hierarchical clustering computed by `hclust()`

Examples

```
library(muscat)

data(example_sce)

hcl_test = buildClusterTree(example_sce, "TSNE", "cluster_id")
```

clr *Centered log ratio transform*

Description

Compute the centered log ratio (CLR) transform of a count matrix.

Usage

```
clr(counts, pseudocount = 0.5)
```

Arguments

counts	count data with samples as rows and variables are columns
pseudocount	added to counts to avoid issues with zeros

Details

The CLR of a vector x of counts in D categories is defined as clr(x) = log(x) - mean(log(x)). For details see van den Boogaart and Tolosana-Delgado (2013).

Value

matrix of CLR transformed counts

References

Van den Boogaart, K. Gerald, and Raimon Tolosana-Delgado. Analyzing compositional data with R. Vol. 122. Berlin: Springer, 2013.

See Also

compositions::clr()

Examples

```
# set probability of each category
prob <- c(0.1, 0.2, 0.3, 0.5)

# number of total counts
countsTotal <- 300

# number of samples
n_samples <- 5

# simulate info for each sample
info <- data.frame(Age = rgamma(n_samples, 50, 1))
rownames(info) <- paste0("sample_", 1:n_samples)
```

```
# simulate counts from multinomial
counts <- t(rmultinom(n_samples, size = countsTotal, prob = prob))
colnames(counts) <- paste0("cat_", 1:length(prob))
rownames(counts) <- paste0("sample_", 1:n_samples)

# centered log ratio
clr(counts)
```

---

clrInv *Inverse of Centered log ratio transform*

Description

Compute the inverse centered log ratio (CLR) transform of a count matrix.

Usage

```
clrInv(x)
```

Arguments

x	CLR transform values

Details

Given the CLR transformed values, compute the original fractions

Value

matrix of fractions

References

Van den Boogaart, K. Gerald, and Raimon Tolosana-Delgado. Analyzing compositional data with R. Vol. 122. Berlin: Springer, 2013.

See Also

`compositions::clrInv()`

Examples

```
# set probability of each category
prob <- c(0.1, 0.2, 0.3, 0.5)

# number of total counts
countsTotal <- 300
```

```
# number of samples
n_samples <- 5

# simulate info for each sample
info <- data.frame(Age = rgamma(n_samples, 50, 1))
rownames(info) <- paste0("sample_", 1:n_samples)

# simulate counts from multinomial
counts <- t(rmultinom(n_samples, size = countsTotal, prob = prob))
colnames(counts) <- paste0("cat_", 1:length(prob))
rownames(counts) <- paste0("sample_", 1:n_samples)

# Fractions
counts / rowSums(counts)

# centered log ratio, with zero pseudocount
clr(counts, 0)

# recover fractions from CLR transformed values
clrInv(clr(counts, 0))
```

`crumblr` *Count ratio uncertainty modeling based linear regression*

Description

Count ratio uncertainty modeling based linear regression (crumblr) returns CLR-transformed counts and observation-level inverse-variance weights for use in weighted linear models.

Usage

```
crumblr(
  counts,
  pseudocount = 0.5,
  method = c("clr", "clr_2class"),
  tau = 1,
  max.ratio = 5,
  quant = 0.05
)

## S4 method for signature 'matrix'
crumblr(
  counts,
  pseudocount = 0.5,
  method = c("clr", "clr_2class"),
  tau = 1,
  max.ratio = 5,
  quant = 0.05
```

```
)

## S4 method for signature 'data.frame'
crumblr(
  counts,
  pseudocount = 0.5,
  method = c("clr", "clr_2class"),
  tau = 1,
  max.ratio = 5,
  quant = 0.05
)
```

Arguments

counts	count data with samples as rows and variables are columns
pseudocount	added to counts to avoid issues with zeros
method	"clr" computes standard centered log ratio and precision weights based on the delta approximation. "clr_2class" computes the clr() transform for category i using 2 classes: 1) counts in category i, and 2) counts _not_ in category i.
tau	overdispersion parameter for Dirichlet multinomial. If NULL, estimate from observed counts.
max.ratio	regularize estimates of the weights to have a maximum ratio of max.ratio between the maximum and quant quantile value
quant	quantile value used for max.ratio

Details

Evaluate the centered log ratio (CLR) transform of the count matrix, and the asymptotic theoretical variances of each transformed observation. The asymptotic normal approximation is increasingly accurate for small overdispersion τ, large total counts C, and large proportions p, but shows good agreement with the empirical results in most situations. In practice, it is often reasonable to assume a sufficient number of counts before a variable is included in an analysis anyway. But the feasibility of this assumption is up to the user to determine.

Given the array p storing proportions for one sample across all categories, the delta approximation uses the term 1/p. This can be unstable for small values of p, and the estimated variances can be sensitive to small changes in the proportions. To address this, the "clr_2class" method computes the clr() transform for category i using 2 classes: 1) counts in category i, and 2) counts _not_ in category i. Since class (2) now sums counts across all other categories, the small proportions are avoided and the variance estimates are more stable.

For real data, the asymptotic variance formula can give weights that vary substantially across samples and give very high weights for a subset of samples. In order to address this, we regularize the weights to reduce the variation in the weights to have a maximum ratio of max.ratio between the maximum and quant quantile value.

Value

An EList object with the following components:

E: numeric matrix of CLR transformed counts

weights: numeric matrix of observation-level inverse-variance weights

See Also

`limma::voom()`, `variancePartition::dream()`

Examples

```
# set probability of each category
prob <- c(0.1, 0.2, 0.3, 0.5)

# number of total counts
countsTotal <- 300

# number of samples
n_samples <- 100

# simulate info for each sample
info <- data.frame(Age = rgamma(n_samples, 50, 1))
rownames(info) <- paste0("sample_", 1:n_samples)

# simulate counts from multinomial
counts <- t(rmultinom(n_samples, size = countsTotal, prob = prob))
colnames(counts) <- paste0("cat_", 1:length(prob))
rownames(counts) <- paste0("sample_", 1:n_samples)

# run crumblr on counts
cobj <- crumblr(counts)

# run standard variancePartition analysis on crumblr results
library(variancePartition)

fit <- dream(cobj, ~Age, info)
fit <- eBayes(fit)

topTable(fit, coef = "Age", sort.by = "none")
```

---

diffTree *Compare difference in estimates between two trees*

Description

Compare difference in coefficient estimates between two trees. For node `i`, the test evaluates `tree1[i] - tree2[i] = 0`.

Usage

```
diffTree(tree1, tree2)
```

Arguments

tree1	object of type treedata from treeTest()
tree2	object of type treedata from treeTest()

Details

When a fixed effect test is performed at each node using treeTest() with method = "FE.empirical" or method = "FE", a coefficient estimate and standard error are estimated for each node based on the children. This function performs a two-sample z-test to test if a given coefficient from tree1 is significantly different from the corresponding coefficient in tree2.

Value

a comparison of the coefficient estimates at each node

Examples

```
library(variancePartition)

# Load cell counts, clustering and metadata
# from Kang, et al. (2018) https://doi.org/10.1038/nbt.4042
data(IFNCellCounts)

# Simulate a factor with 2 levels called DiseaseRand
set.seed(123)
info$DiseaseRand <- sample(LETTERS[seq(2)], nrow(info), replace = TRUE)
info$DiseaseRand <- factor(info$DiseaseRand, LETTERS[seq(2)])

# Apply crumblr transformation
cobj <- crumblr(df_cellCounts)

# Use dream workflow to analyze each cell separately
fit <- dream(cobj, ~ StimStatus + ind, info)
fit <- eBayes(fit)

# Perform multivariate test across the hierarchy
res1 <- treeTest(fit, cobj, hcl, coef = "StimStatusstim")

# Perform same test, but on DiseaseRand
fit2 <- dream(cobj, ~DiseaseRand, info)
fit2 <- eBayes(fit2)
res2 <- treeTest(fit2, cobj, hcl, coef = "DiseaseRandB")

# Compare the coefficient estimates at each node
# Test if res1 - res2 is significantly different from zero
resDiff <- diffTree(res1, res2)

resDiff

plotTreeTest(resDiff)
```

```
plotTreeTestBeta(resDiff)
```

`dmn_mle` *MLE for Dirichlet Multinomial*

Description

MLE for Dirichlet Multinomial

Usage

```
dmn_mle(counts, ...)
```

Arguments

`counts`	matrix with rows as samples and columns as categories
`...`	additional arguments passed to `optim()`

Details

Maximize Dirichlet Multinomial (DMN) log-likelihood with `optim()` using log likelihood function and its gradient. This method uses a second round of optimization to estimate the scale of α parameters, which is necessary for accurate estimation of overdispersion metric.

The covariance between counts in each category from DMN distributed data is $n(diag(p)-pp^T)(1+\rho^2(n-1))$ for n total counts, and vector of proportions p, where $\rho^2 = 1/(a_0+1)$ and $a_0 = \sum_i \alpha_i$. The count data is overdispersed by a factor of $1+\rho^2(n-1)$ compared to a multinomial (MN) distribution. As a_0 increases, the DMN converges to the MN.

See https://en.wikipedia.org/wiki/Dirichlet-multinomial_distribution#Matrix_notation

Value

list storing `alpha` parameter estimates, logLik, and details about convergence

`alpha` estimated $alpha$ parameters

`overdispersion` Overdispersion value $1+\rho^2(n-1)$ compared to multinomial

`logLik` value of function

`scale` scaling of α parameters computed in a second optimization step

`evals` number of function evaluations in step 1

`convergence` convergence details from step 1

See Also

Other functions also estimate DMN parameters. `MGLM::MGLMfit()` and `dirmult::dirmult()` give good parameter estimates but are slower. `Rfast::dirimultinom.mle()` often fails to converge

Examples

```
library(HMP)

set.seed(1)

n_samples <- 1000
n_counts <- 5000
alpha <- c(500, 1000, 2000)

# Dirichlet.multinomial
counts <- Dirichlet.multinomial(rep(n_counts, n_samples), alpha)

fit <- dmn_mle(counts)

fit

# overdispersion: true value
a0 <- sum(alpha)
rhoSq <- 1 / (a0 + 1)
1 + rhoSq * (n_counts - 1)

# multinomial, so overdispersion is 1
counts <- t(rmultinom(n_samples, n_counts, prob = alpha / sum(alpha)))

dmn_mle(counts)
#
#
```

`IFNCellCounts` *Cell counts following interferon treatment*

Description

Counts are from single cell RNA-seq data from treated and untreated samples from Kang, et al (2018).

Usage

```
data(IFNCellCounts)

info

df_cellCounts

hcl
```

Format

- `info` is metadata for each sample
- `df_cellCounts` data.frame of counts for each sample
- `hcl` cluster of cell types based on pseudobulk expression

An object of class `data.frame` with 16 rows and 4 columns.

An object of class `matrix` (inherits from `array`) with 16 rows and 8 columns.

An object of class `hclust` of length 7.

References

Kang, Hyun Min, et al. "Multiplexed droplet single-cell RNA-sequencing using natural genetic variation." Nature Biotechnology 36.1 (2018): 89-94.

`logFrac` *Log fractions and precision weights*

Description

Compute log fractions and precision weights from matrix of c ounts, where columns are variables and rows are samples

Usage

```
logFrac(counts, pseudocount = 0.5, max.ratio = 5, quant = 0.05)
```

Arguments

`counts`	count data with samples as rows and variables are columns
`pseudocount`	added to counts to avoid issues with zeros
`max.ratio`	regularize estimates of the weights to have a maximum ratio of `max.ratio` between the maximum and `quant` quantile value
`quant`	quantile value used for `max.ratio`

Details

For real data, the asymptotic variance formula can give weights that vary substantially across samples and give very high weights for a subset of samples. In order to address this, we regularize the weights to reduce the variation in the weights to have a maximum ratio of `max.ratio` between the maximum and `quant` quantile value.

Value

An `EList` object with the following components:

E: numeric matrix of log transformed counts

weights: numeric matrix of observation-level inverse-variance weights

See Also

limma::voom(), variancePartition::dream()

Examples

```
# set probability of each category
prob <- c(0.1, 0.2, 0.3, 0.5)

# number of total counts
countsTotal <- 300

# number of samples
n_samples <- 100

# simulate info for each sample
info <- data.frame(Age = rgamma(n_samples, 50, 1))
rownames(info) <- paste0("sample_", 1:n_samples)

# simulate counts from multinomial
counts <- t(rmultinom(n_samples, size = countsTotal, prob = prob))
colnames(counts) <- paste0("cat_", 1:length(prob))
rownames(counts) <- paste0("sample_", 1:n_samples)

# run logFrac on counts
cobj <- logFrac(counts)

# run standard variancePartition analysis on crumblr results
library(variancePartition)

fit <- dream(cobj, ~ Age, info)
fit <- eBayes(fit)

topTable(fit, coef = "Age", sort.by = "none")
```

---

meanSdPlot *Plot row standard deviations versus rank of row means*

Description

Diagnositic plot for homoscedasticity across variables

Usage

```
meanSdPlot(x)
```

Arguments

x	data matrix

Details

Plot the sd versus rank mean of each row like `vsn::meanSdPlot`. Also show the coefficient of variation of the variances. A lower value indicates stronger variance stabilization

Value

ggplot2 object

See Also

`vsn::meanSdPlot`

Examples

```
# set probability of each category
prob <- runif(300)

# number of samples
n_samples <- 1000

# number of counts
nCounts <- 3000

# simulate counts from multinomial
counts <- t(rmultinom(n_samples, size = nCounts, prob = prob))
colnames(counts) <- paste0("cat_", 1:length(prob))
rownames(counts) <- paste0("sample_", 1:n_samples)

# keep categories with at least 5 counts in at least 10 samples
keep <- colSums(counts > 5) > 10

# run crumblr on counts
cobj <- crumblr(counts[, keep], max.ratio = 10)

# Plot for CLR
# For each sample, plot rank of mean vs sd
fig1 <- meanSdPlot(cobj$E) + ggtitle("CLR")

# run crumblr::standardize()
df_std <- standardize(cobj)

# Standardized crumblr
fig2 <- meanSdPlot(df_std) + ggtitle("Standardized crumblr")

# Standardizing the crumblr results better stabilizes
# the variances across variables
fig1 | fig2
```

plotForest *Forest plot*

Description

Forest plot

Forest plot of effect size estimates at the leaves of the tree

Usage

```
plotForest(x, ...)

## S4 method for signature 'treedata'
plotForest(x, ..., hide = FALSE)
```

Arguments

x	result from treeTest()
...	other arguments
hide	hide rownames and legend

Value

ggplot2 object

Examples

```
library(variancePartition)

# Load cell counts, clustering and metadata
# from Kang, et al. (2018) https://doi.org/10.1038/nbt.4042
data(IFNCellCounts)

# Apply crumblr transformation
cobj <- crumblr(df_cellCounts)

# Use dream workflow to analyze each cell separately
fit <- dream(cobj, ~ StimStatus + ind, info)
fit <- eBayes(fit)

# Perform multivariate test across the hierarchy
res <- treeTest(fit, cobj, hcl, coef = "StimStatusstim")

# Plot log fold changes from coef
plotForest(res)
```

plotScatterDensity *Scatter plot with 2D density using viridis colors*

Description

Scatter plot with 2D density using viridis colors

Usage

```
plotScatterDensity(x, y, size = 1)
```

Arguments

x	the x-coordinates of points in the plot
y	the y-coordinates of points in the plot
size	size of point

Value

ggplot2 object

Examples

```
# simulate data
M <- Rfast::rmvnorm(1000, mu = c(0, 0), sigma = diag(1, 2))

# create 2D density plot
plotScatterDensity(M[, 1], M[, 2])
```

plotTreeTest *Plot tree with results from multivariate testing*

Description

Plot tree with results from multivariate testing

Usage

```
plotTreeTest(
  tree,
  low = "grey90",
  mid = "red",
  high = "darkred",
  xmax.scale = 1.5
)
```

Arguments

tree	phylo object storing tree
low	low color on gradient
mid	mid color on gradient
high	high color on gradient
xmax.scale	expand the x-axis by this factor so leaf labels fit in the plot

Value

ggplot2 object

Examples

```
library(variancePartition)

# Load cell counts, clustering and metadata
# from Kang, et al. (2018) https://doi.org/10.1038/nbt.4042
data(IFNCellCounts)

# Apply crumblr transformation
cobj <- crumblr(df_cellCounts)

# Use dream workflow to analyze each cell separately
fit <- dream(cobj, ~ StimStatus + ind, info)
fit <- eBayes(fit)

# Perform multivariate test across the hierarchy
res <- treeTest(fit, cobj, hcl, coef = "StimStatusstim")

# Plot hierarchy and testing results
plotTreeTest(res)

# Extract results for first 3 nodes
res[1:3, ]
```

---

plotTreeTestBeta *Plot tree coefficients from multivariate testing*

Description

Plot tree coefficients from multivariate testing at each node. Only applicable top fixed effect tests

Usage

```
plotTreeTestBeta(
  tree,
  low = "blue",
  mid = "white",
  high = "red",
  xmax.scale = 1.5
)
```

Arguments

tree	phylo object storing tree
low	low color on gradient
mid	mid color on gradient
high	high color on gradient
xmax.scale	expand the x-axis by this factor so leaf labels fit in the plot

Value

ggplot2 object

Examples

```
library(variancePartition)

# Load cell counts, clustering and metadata
# from Kang, et al. (2018) https://doi.org/10.1038/nbt.4042
data(IFNCellCounts)

# Apply crumblr transformation
cobj <- crumblr(df_cellCounts)

# Use dream workflow to analyze each cell separately
fit <- dream(cobj, ~ StimStatus + ind, info)
fit <- eBayes(fit)

# Perform multivariate test across the hierarchy
res <- treeTest(fit, cobj, hcl, coef = "StimStatusstim")

# Plot hierarchy, no tests are significant
plotTreeTestBeta(res)
```

`standardize` *Standardize observations using precision weights*

Description

Compute standardized observations by dividing the observed values by their standard deviations based on the precision weights

Usage

```
standardize(x, ...)

## S4 method for signature 'EList'
standardize(x, ...)
```

Arguments

`x`	object storing data to be transformed
`...`	other arguments

Details

Weighted response by their standard deviation so that resulting values have approximately equal sample variance. This is a key property that improves downstream PCA and clustering analysis.

Value

matrix of standardized values

Examples

```
# set probability of each category
prob <- c(0.1, 0.2, 0.3, 0.5)

# number of total counts
countsTotal <- 300

# number of samples
n_samples <- 100

# simulate counts from multinomial
counts <- t(rmultinom(n_samples, size = countsTotal, prob = prob))
colnames(counts) <- paste0("cat_", 1:length(prob))
rownames(counts) <- paste0("sample_", 1:n_samples)

# run crumblr on counts
cobj <- crumblr(counts)

# Standardize crumblr responses
```

```
df_std <- standardize(cobj)

# Perform PCA on student transformed data
pca <- prcomp(t(df_std))
df_pca <- as.data.frame(pca$x)

ggplot(df_pca, aes(PC1, PC2)) +
  geom_point() +
  theme_classic() +
  theme(aspect.ratio = 1)
```

`treeTest` *Perform multivariate testing along a hierarchy*

Description

Perform multivariate testing using `mvTest()` along the nodes of tree

Usage

```
treeTest(
  fit,
  obj,
  hc,
  coef,
  method = c("FE.empirical", "FE", "RE2C", "tstat", "sidak", "fisher"),
  shrink.cov = TRUE
)
```

Arguments

`fit`	`MArrayLM` object return by `lmFit()` or `dream()`
`obj`	`EList` object returned by `voom()`
`hc`	hierarchical clustering as an `hclust` object
`coef`	name of coefficient to be extracted
`method`	statistical method used to perform multivariate test. See details. `'FE'` is a fixed effect test that models the covariance between coefficients. `'FE.empirical'` use compute empirical p-values by sampling from the null distribution and fitting with a gamma. `'RE2C'` is a random effect test of heterogeneity of the estimated coefficients that models the covariance between coefficients, and also incorporates a fixed effects test too. `'tstat'` combines the t-statistics and models the covariance between coefficients. `'sidak'` returns the smallest p-value and accounting for the number of tests. `'fisher'` combines the p-value using Fisher's method assuming independent tests.
`shrink.cov`	shrink the covariance matrix between coefficients using the Schafer-Strimmer method

Details

See package remaCor for details about the remaCor::RE2C() test, and see remaCor::LS() for details about the fixed effect test. When only 1 feature is selected, the original t-statistic and p-value are returned.

Value

object of type treedata storing results

See Also

variancePartition::mvTest()

Examples

```
library(variancePartition)

# Load cell counts, clustering and metadata
# from Kang, et al. (2018) https://doi.org/10.1038/nbt.4042
data(IFNCellCounts)

# Apply crumblr transformation
cobj <- crumblr(df_cellCounts)

# Use dream workflow to analyze each cell separately
fit <- dream(cobj, ~ StimStatus + ind, info)
fit <- eBayes(fit)

# Perform multivariate test across the hierarchy
res <- treeTest(fit, cobj, hcl, coef = "StimStatusstim")

# Plot hierarchy and testing results
plotTreeTest(res)

# Extract results for first 3 nodes
res[1:3, ]
```

Index