# Package ‘CRMetrics’

December 14, 2022

**Title** Cell Ranger Output Filtering and Metrics Visualization

**Version** 0.2.3

**Description** Sample and cell filtering as well as visualisation of output metrics from 'Cell Ranger' by Grace X.Y. Zheng et al. (2017) <doi:10.1038/ncomms14049>. 'CRMetrics' allows for easy plotting of output metrics across multiple samples as well as comparative plots including statistical assessments of these. 'CRMetrics' allows for easy removal of ambient RNA using 'SoupX' by Matthew D Young and Sam Behjati (2020) <doi:10.1093/gigascience/giaa151> or 'CellBender' by Stephen J Fleming et al. (2022) <doi:10.1101/791699>. Furthermore, it is possible to preprocess data using 'Pagoda2' by Nikolas Barkas et al. (2021) <`https://github.com/kharchenkolab/pagoda2`> or 'Seurat' by Yuhan Hao et al. (2021) <doi:10.1016/j.cell.2021.04.048> followed by embedding of cells using 'Conos' by Nikolas Barkas et al. (2019) <doi:10.1038/s41592-019-0466-z>. Finally, doublets can be detected using 'scrublet' by Samuel L. Wolock et al. (2019) <doi:10.1016/j.cels.2018.11.005> or 'DoubletDetection' by Gayoso et al. (2020) <doi:10.5281/zenodo.2678041>. In the end, cells are filtered based on user input for use in downstream applications.

**License** GPL-3

**Encoding** UTF-8

**Depends** R (>= 4.0.0)

**biocViews**

**Imports** cowplot, dplyr, ggbeeswarm, ggplot2, ggpmisc, ggpubr, ggrepel, magrittr, Matrix, methods, R6, scales, sccore, sparseMatrixStats, stats, tibble, tidyr, utils

**Suggests** conos, data.table, markdown, pagoda2, reticulate, rhdf5, Seurat, SoupX, testthat (>= 3.0.0)

**RoxygenNote** 7.2.2

**URL** `https://github.com/khodosevichlab/CRMetrics`

**BugReports** `https://github.com/khodosevichlab/CRMetrics/issues`

**Maintainer** Rasmus Rydbirk <`rasmus.rydbirk@bric.ku.dk`>

**Config/testthat/edition** 3

**NeedsCompilation** no

**Author** Rasmus Rydbirk [aut, cre],
Fabienne Kick [aut],
Henrietta Holze [aut],
Xian Xin [ctb]

**Repository** CRAN

**Date/Publication** 2022-12-14 11:30:06 UTC

# R topics documented:

---

## `CRMetrics` *CRMetrics class object*

### Description

Functions to analyze Cell Ranger count data

### Public fields

`metadata` (default = NULL)

`data.path` (default = NULL)

`summary.metrics` (default = NULL)

`detailed.metrics` (default = NULL)

`comp.group` (default = NULL)

`verbose` (default = TRUE)

`theme` (default = NULL)

`n.cores` (default = 1) Initialize a CRMetrics object

### Methods

#### Public methods:

- `CRMetrics$new()`
- `CRMetrics$addDetailedMetrics()`
- `CRMetrics$addComparison()`
- `CRMetrics$plotSamples()`
- `CRMetrics$plotSummaryMetrics()`
- `CRMetrics$plotDetailedMetrics()`
- `CRMetrics$plotEmbedding()`

- `CRMetrics$plotDepth()`
- `CRMetrics$plotMitoFraction()`
- `CRMetrics$detectDoublets()`
- `CRMetrics$doPreprocessing()`
- `CRMetrics$createEmbedding()`
- `CRMetrics$filterCms()`
- `CRMetrics$selectMetrics()`
- `CRMetrics$plotFilteredCells()`
- `CRMetrics$getConosDepth()`
- `CRMetrics$getMitoFraction()`
- `CRMetrics$prepareCellbender()`
- `CRMetrics$saveCellbenderScript()`
- `CRMetrics$getExpectedCells()`
- `CRMetrics$getTotalDroplets()`
- `CRMetrics$addCms()`
- `CRMetrics$plotCbTraining()`
- `CRMetrics$plotCbCellProbs()`
- `CRMetrics$plotCbAmbExp()`
- `CRMetrics$plotCbAmbGenes()`
- `CRMetrics$addSummaryFromCms()`
- `CRMetrics$runSoupX()`
- `CRMetrics$plotSoupX()`
- `CRMetrics$plotCbCells()`
- `CRMetrics$clone()`

**Method** `new()`**:** To initialize new object, 'data.path' or 'cms' is needed. 'metadata' is also recommended, but not required.

*Usage:*

```
CRMetrics$new(
  data.path = NULL,
  metadata = NULL,
  cms = NULL,
  sample.names = NULL,
  unique.names = TRUE,
  sep.cells = "!!",
  comp.group = NULL,
  verbose = TRUE,
  theme = theme_bw(),
  n.cores = 1,
  sep.meta = ",",
  raw.meta = FALSE
)
```

*Arguments:*

`data.path` character Path to directory with Cell Ranger count data, one directory per sample (default = NULL).

`metadata` data.frame or character Path to metadata file (comma-separated) or name of metadata dataframe object. Metadata must contain a column named 'sample' containing sample names that must match folder names in 'data.path' (default = NULL).

`cms` list List with count matrices (default = NULL)

`sample.names` character Sample names. Only relevant is cms is provided (default = NULL)

`unique.names` logical Create unique cell names. Only relevant if cms is provided (default = TRUE)

`sep.cells` character Sample-cell separator. Only relevant if cms is provided and `unique.names=TRUE` (default = "!!")

`comp.group` character A group present in the metadata to compare the metrics by, can be added with addComparison (default = NULL).

`verbose` logical Print messages or not (default = TRUE).

`theme` ggplot2 theme (default: theme_bw()).

`n.cores` integer Number of cores for the calculations (default = self$n.cores).

`sep.meta` character Separator for metadata file (default = ",")

`raw.meta` logical Keep metadata in its raw format. If FALSE, classes will be converted using "type.convert" (default = FALSE)

*Returns:* CRMetrics object

*Examples:*

```
\dontrun{
crm <- CRMetrics$new(data.path = "/path/to/count/data/")
}
```

**Method** `addDetailedMetrics()`: Function to read in detailed metrics. This is not done upon initialization for speed.

*Usage:*

```
CRMetrics$addDetailedMetrics(
  min.transcripts.per.cell = 100,
  raw = FALSE,
  symbol = TRUE,
  sep = "!!",
  cellbender = FALSE,
  unique.names = TRUE,
  data.path = self$data.path,
  sample.names = self$metadata$sample,
  n.cores = self$n.cores,
  verbose = self$verbose
)
```

*Arguments:*

`min.transcripts.per.cell` numeric Minimal number of transcripts per cell (default = 100)

`raw` logical Add raw count matrices from Cell Ranger output. Cannot be combined with `cellbender=TRUE` (default = FALSE)

`symbol` character The type of gene IDs to use, SYMBOL (TRUE) or ENSEMBLE (default = TRUE)

`sep` character Separator for cell names (default = "!!").

`cellbender` logical Add CellBender filtered count matrices in HDF5 format. Requires that "cellbender" is in the names of the files (default = FALSE)

`unique.names` logical Make cell names unique based on `sep` parameter (default = TRUE)

`data.path` character Path to cellranger count data (default = self$data.path).

`sample.names` character Vector containing sample names (default = self$metadata$sample).

`n.cores` integer Number of cores for the calculations (default = self$n.cores).

`verbose` logical Print messages or not (default = self$verbose).

*Returns:* Count matrices

*Examples:*

```
# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
sapply(seq_len(1e3), \(x) paste0("cell",x)))
return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, sample.names = c("sample1", "sample2"), n.cores = 1)

# Run function
crm$addDetailedMetrics()
```

**Method** `addComparison()`**:** Add comparison group for statistical testing.

*Usage:*

```
CRMetrics$addComparison(comp.group, metadata = self$metadata)
```

*Arguments:*

`comp.group` character Comparison metric (default = self$comp.group).

`metadata` data.frame Metadata for samples (default = self$metadata).

*Returns:* Vector

*Examples:*

```
# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
sapply(seq_len(1e3), \(x) paste0("cell",x)))
return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, sample.names = c("sample1", "sample2"), n.cores = 1)

# Add metadata
```

```
crm$metadata$sex <- c("male","female")

# Add comparison group
crm$addComparison(comp.group = "sex")
```

**Method** `plotSamples()`**:** Plot the number of samples.

*Usage:*

```
CRMetrics$plotSamples(
  comp.group = self$comp.group,
  h.adj = 0.05,
  exact = FALSE,
  metadata = self$metadata,
  second.comp.group = NULL
)
```

*Arguments:*

`comp.group` character Comparison metric, must match a column name of metadata (default = self$comp.group).

`h.adj` numeric Position of statistics test p value as % of max(y) (default = 0.05).

`exact` logical Whether to calculate exact p values (default = FALSE).

`metadata` data.frame Metadata for samples (default = self$metadata).

`second.comp.group` character Second comparison metric, must match a column name of metadata (default = NULL).

*Returns:* ggplot2 object

*Examples:*

```
sample.names <- c("sample1", "sample2")

# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
sapply(seq_len(1e3), \(x) paste0("cell",x)))
return(out)
})
names(testdata.cms) <- sample.names

# Create metadata
metadata <- data.frame(sample = sample.names,
sex = c("male","female"),
condition = c("a","b"))

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, metadata = metadata, n.cores = 1)

# Plot
crm$plotSamples(comp.group = "sex", second.comp.group = "condition")
```

**Method** `plotSummaryMetrics()`**:** Plot all summary stats or a selected list.

*Usage:*

```
CRMetrics$plotSummaryMetrics(
  comp.group = self$comp.group,
  second.comp.group = NULL,
  metrics = NULL,
  h.adj = 0.05,
  plot.stat = TRUE,
  stat.test = c("non-parametric", "parametric"),
  exact = FALSE,
  metadata = self$metadata,
  summary.metrics = self$summary.metrics,
  plot.geom = "bar",
  se = FALSE,
  group.reg.lines = FALSE,
  secondary.testing = TRUE
)
```

*Arguments:*

`comp.group` character Comparison metric (default = self$comp.group).

`second.comp.group` character Second comparison metric, used for the metric "samples per group" or when "comp.group" is a numeric or an integer (default = NULL).

`metrics` character Metrics to plot (default = NULL).

`h.adj` numeric Position of statistics test p value as % of max(y) (default = 0.05)

`plot.stat` logical Show statistics in plot. Will be FALSE if "comp.group" = "sample" or if "comp.group" is a numeric or an integer (default = TRUE)

`stat.test` character Statistical test to perform to compare means. Can either be "non-parametric" or "parametric" (default = "non-parametric").

`exact` logical Whether to calculate exact p values (default = FALSE).

`metadata` data.frame Metadata for samples (default = self$metadata).

`summary.metrics` data.frame Summary metrics (default = self$summary.metrics).

`plot.geom` character Which geometric is used to plot the data (default = "point").

`se` logical For regression lines, show SE (default = FALSE)

`group.reg.lines` logical For regression lines, if FALSE show one line, if TRUE show line per group defined by second.comp.group (default = FALSE)

`secondary.testing` logical Whether to show post hoc testing (default = TRUE)

*Returns:* ggplot2 object

*Examples:*

```
\donttest{
# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
sapply(seq_len(1e3), \(x) paste0("cell",x)))
return(out)
```

```
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, sample.names = c("sample1", "sample2"), n.cores = 1)

# Add summary metrics
crm$addSummaryFromCms()

crm$plotSummaryMetrics(plot.geom = "point")
}
```

**Method** `plotDetailedMetrics()`**:** Plot detailed metrics from the detailed.metrics object

*Usage:*

```
CRMetrics$plotDetailedMetrics(
  comp.group = self$comp.group,
  detailed.metrics = self$detailed.metrics,
  metadata = self$metadata,
  metrics = NULL,
  plot.geom = "violin",
  data.path = self$data.path,
  hline = TRUE
)
```

*Arguments:*

| | |
|---|---|
| `comp.group` | character Comparison metric (default = self$comp.group). |
| `detailed.metrics` | data.frame Object containing the count matrices (default = self$detailed.metrics). |
| `metadata` | data.frame Metadata for samples (default = self$metadata). |
| `metrics` | character Metrics to plot. NULL plots both plots (default = NULL). |
| `plot.geom` | character How to plot the data (default = "violin"). |
| `data.path` | character Path to cellranger count data (default = self$data.path). |
| `hline` | logical Whether to show median as horizontal line (default = TRUE) |

*Returns:* ggplot2 object

*Examples:*

```
\donttest{
# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
sapply(seq_len(1e3), \(x) paste0("cell",x)))
return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, sample.names = c("sample1", "sample2"), n.cores = 1)

# Add detailed metrics
```

```
crm$addDetailedMetrics()

# Plot
crm$plotDetailedMetrics()
}
```

**Method** `plotEmbedding()`**:** Plot cells in embedding using Conos and color by depth and doublets.

*Usage:*

```
CRMetrics$plotEmbedding(
  depth = FALSE,
  doublet.method = NULL,
  doublet.scores = FALSE,
  depth.cutoff = 1000,
  mito.frac = FALSE,
  mito.cutoff = 0.05,
  species = c("human", "mouse"),
  size = 0.3,
  sep = "!!",
  ...
)
```

*Arguments:*

`depth` logical Plot depth or not (default = FALSE).

`doublet.method` character Doublet detection method (default = NULL).

`doublet.scores` logical Plot doublet scores or not (default = FALSE).

`depth.cutoff` numeric Depth cutoff (default = 1e3).

`mito.frac` logical Plot mitochondrial fraction or not (default = FALSE).

`mito.cutoff` numeric Mitochondrial fraction cutoff (default = 0.05).

`species` character Species to calculate the mitochondrial fraction for (default = c("human","mouse")).

`size` numeric Dot size (default = 0.3)

`sep` character Separator for creating unique cell names (default = "!!")

`...` Plotting parameters passed to `sccore::embeddingPlot`.

*Returns:* ggplot2 object

*Examples:*

```
\donttest{
if (requireNamespace("pagoda2", quietly = TRUE)) {
if (requireNamespace("conos", quietly = TRUE)) {
# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
sapply(seq_len(1e3), \(x) paste0("cell",x)))
return(out)
})
```

```
# Initialize
crm <- CRMetrics$new(cms = testdata.cms, sample.names = c("sample1", "sample2"), n.cores = 1)

# Create embedding
crm$doPreprocessing()
crm$createEmbedding()

crm$plotEmbedding()
} else {
message("Package 'conos' not available.")
}
} else {
message("Package 'pagoda2' not available.")
}
}
```

**Method** `plotDepth()`**:** Plot the sequencing depth in histogram.

*Usage:*

```
CRMetrics$plotDepth(cutoff = 1000, samples = self$metadata$sample, sep = "!!")
```

*Arguments:*

`cutoff` numeric The depth cutoff to color the cells in the embedding (default = 1e3).

`samples` character Sample names to include for plotting (default = $metadata$sample).

`sep` character Separator for creating unique cell names (default = "!!")

*Returns:* ggplot2 object

*Examples:*

```
\donttest{
if (requireNamespace("pagoda2", quietly = TRUE)) {
if (requireNamespace("conos", quietly = TRUE)) {
# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
sapply(seq_len(1e3), \(x) paste0("cell",x)))
return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, sample.names = c("sample1", "sample2"), n.cores = 1)

# Create embedding
crm$doPreprocessing()
crm$createEmbedding()

# Plot
```

```
crm$plotDepth()
} else {
message("Package 'conos' not available.")
}
} else {
message("Package 'pagoda2' not available.")
}
}
```

**Method** `plotMitoFraction()`**:** Plot the mitochondrial fraction in histogram.

*Usage:*

```
CRMetrics$plotMitoFraction(
  cutoff = 0.05,
  species = c("human", "mouse"),
  samples = self$metadata$sample,
  sep = "!!"
)
```

*Arguments:*

`cutoff` numeric The mito. fraction cutoff to color the embedding (default = 0.05)

`species` character Species to calculate the mitochondrial fraction for (default = "human")

`samples` character Sample names to include for plotting (default = $metadata$sample)

`sep` character Separator for creating unique cell names (default = "!!")

*Returns:* ggplot2 object

*Examples:*

```
\donttest{
if (requireNamespace("pagoda2", quietly = TRUE)) {
if (requireNamespace("conos", quietly = TRUE)) {
# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
sapply(seq_len(1e3), \(x) paste0("cell",x)))
return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, sample.names = c("sample1", "sample2"), n.cores = 1)

# Create embedding
crm$doPreprocessing()
crm$createEmbedding()

# Plot
crm$plotMitoFraction()
} else {
```

```
message("Package 'conos' not available.")
}
} else {
message("Package 'pagoda2' not available.")
}
}
```

**Method** `detectDoublets()`**:** Detect doublet cells.

*Usage:*

```
CRMetrics$detectDoublets(
  method = c("scrublet", "doubletdetection"),
  cms = self$cms,
  env = "r-reticulate",
  conda.path = system("whereis conda"),
  n.cores = self$n.cores,
  verbose = self$verbose,
  args = list()
)
```

*Arguments:*

`method` character Which method to use, either `scrublet` or `doubletdetection` (default="scrublet").

`cms` list List containing the count matrices (default=self$cms).

`env` character Environment to run python in (default="r-reticulate").

`conda.path` character Path to conda environment (default=system("whereis conda")).

`n.cores` integer Number of cores to use (default = self$n.cores)

`verbose` logical Print messages or not (default = self$verbose)

`args` list A list with additional arguments for either `DoubletDetection` or `scrublet`. Please check the respective manuals.

*Returns:* data.frame

*Examples:*

```
\dontrun{
# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
sapply(seq_len(1e3), \(x) paste0("cell",x)))
return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, sample.names = c("sample1", "sample2"), n.cores = 1)


# Detect doublets
crm$detectDoublets(method = "scrublet",
conda.path = "/opt/software/miniconda/4.12.0/condabin/conda")
}
```

**Method** `doPreprocessing()`**:** Perform conos preprocessing.

*Usage:*

```
CRMetrics$doPreprocessing(
  cms = self$cms,
  preprocess = c("pagoda2", "seurat"),
  min.transcripts.per.cell = 100,
  verbose = self$verbose,
  n.cores = self$n.cores,
  get.largevis = FALSE,
  tsne = FALSE,
  make.geneknn = FALSE,
  cluster = FALSE,
  ...
)
```

*Arguments:*

`cms` list List containing the count matrices (default = self$cms).

`preprocess` character Method to use for preprocessing (default = c("pagoda2","seurat")).

`min.transcripts.per.cell` numeric Minimal transcripts per cell (default = 100)

`verbose` logical Print messages or not (default = self$verbose).

`n.cores` integer Number of cores for the calculations (default = self$n.cores).

`get.largevis` logical For Pagoda2, create largeVis embedding (default = FALSE)

`tsne` logical Create tSNE embedding (default = FALSE)

`make.geneknn` logical For Pagoda2, estimate gene kNN (default = FALSE)

`cluster` logical For Seurat, estimate clusters (default = FALSE)

`...` Additional arguments for `Pagaoda2::basicP2Proc` or `conos:::basicSeuratProc`

*Returns:* Conos object

*Examples:*

```
\donttest{
if (requireNamespace("pagoda2", quietly = TRUE)) {
# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
sapply(seq_len(1e3), \(x) paste0("cell",x)))
return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, sample.names = c("sample1", "sample2"), n.cores = 1)

# Perform preprocessing
crm$doPreprocessing(preprocess = "pagoda2")
} else {
message("Package 'pagoda2' not available.")
```

```
}
}
```

**Method** `createEmbedding()`**:** Create Conos embedding.

*Usage:*

```
CRMetrics$createEmbedding(
  cms = self$cms.preprocessed,
  verbose = self$verbose,
  n.cores = self$n.cores,
  arg.buildGraph = list(),
  arg.findCommunities = list(n.iterations = 1),
  arg.embedGraph = list(method = "UMAP")
)
```

*Arguments:*

`cms` list List containing the preprocessed count matrices (default = self$cms.preprocessed).

`verbose` logical Print messages or not (default = self$verbose).

`n.cores` integer Number of cores for the calculations (default = self$n.cores).

`arg.buildGraph` list A list with additional arguments for the `buildGraph` function in Conos (default = list())

`arg.findCommunities` list A list with additional arguments for the `findCommunities` function in Conos (default = list(n.iterations = 1)) # Should be updated when Conos issue #123 is resolved

`arg.embedGraph` list A list with additional arguments for the `embedGraph` function in Conos (default = list(method = "UMAP"))

*Returns:* Conos object

*Examples:*

```
\donttest{
if (requireNamespace("pagoda2", quietly = TRUE)) {
if (requireNamespace("conos", quietly = TRUE)) {
# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
sapply(seq_len(1e3), \(x) paste0("cell",x)))
return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, sample.names = c("sample1", "sample2"), n.cores = 1)

# Create embedding
crm$doPreprocessing()
crm$createEmbedding()
} else {
message("Package 'conos' not available.")
```

```
}
} else {
message("Package 'pagoda2' not available.")
}
}
```

**Method** `filterCms()`**:** Filter cells based on depth, mitochondrial fraction and doublets from the count matrix.

*Usage:*

```
CRMetrics$filterCms(
  min.transcripts.per.cell = 100,
  depth.cutoff = NULL,
  mito.cutoff = NULL,
  doublets = NULL,
  species = c("human", "mouse"),
  samples.to.exclude = NULL,
  verbose = self$verbose,
  sep = "!!"
)
```

*Arguments:*

`min.transcripts.per.cell` numeric Minimal transcripts per cell (default = 100)

`depth.cutoff` numeric Depth cutoff (default = NULL).

`mito.cutoff` numeric Mitochondrial fraction cutoff (default = NULL).

`doublets` character Doublet detection method to use (default = NULL).

`species` character Species to calculate the mitochondrial fraction for (default = "human").

`samples.to.exclude` character Sample names to exclude (default = NULL)

`verbose` logical Show progress (default = self$verbose)

`sep` character Separator for creating unique cell names (default = "!!")

*Returns:* list of filtered count matrices

*Examples:*

```
\donttest{
if (requireNamespace("pagoda2", quietly = TRUE)) {
if (requireNamespace("conos", quietly = TRUE)) {
# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
sapply(seq_len(1e3), \(x) paste0("cell",x)))
return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, sample.names = c("sample1", "sample2"), n.cores = 1)

# Create embedding
```

```
crm$doPreprocessing()
crm$createEmbedding()


# Filter CMs
crm$filterCms(depth.cutoff = 1e3, mito.cutoff = 0.05)
} else {
message("Package 'conos' not available.")
}
} else {
message("Package 'pagoda2' not available.")
}
}
```

**Method** `selectMetrics()`**:** Select metrics from summary.metrics

*Usage:*

```
CRMetrics$selectMetrics(ids = NULL)
```

*Arguments:*

`ids` character Metric id to select (default = NULL).

*Returns:* vector

*Examples:*

```
# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
sapply(seq_len(1e3), \(x) paste0("cell",x)))
return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, sample.names = c("sample1", "sample2"), n.cores = 1)

# Select metrics
crm$selectMetrics()
selection.metrics <- crm$selectMetrics(c(1:4))
```

**Method** `plotFilteredCells()`**:** Plot filtered cells in an embedding, in a bar plot, on a tile or export the data frame

*Usage:*

```
CRMetrics$plotFilteredCells(
  type = c("embedding", "bar", "tile", "export"),
  depth = TRUE,
  depth.cutoff = 1000,
  doublet.method = NULL,
  mito.frac = TRUE,
```

```
  mito.cutoff = 0.05,
  species = c("human", "mouse"),
  size = 0.3,
  sep = "!!",
  ...
)
```

*Arguments:*

`type` character The type of plot to use: embedding, bar, tile or export (default = c("embedding","bar","tile","export")).

`depth` logical Plot the depth or not (default = TRUE).

`depth.cutoff` numeric Depth cutoff, either a single number or a vector with cutoff per sample and with sampleIDs as names (default = 1e3).

`doublet.method` character Method to detect doublets (default = NULL).

`mito.frac` logical Plot the mitochondrial fraction or not (default = TRUE).

`mito.cutoff` numeric Mitochondrial fraction cutoff, either a single number or a vector with cutoff per sample and with sampleIDs as names (default = 0.05).

`species` character Species to calculate the mitochondrial fraction for (default = c("human","mouse")).

`size` numeric Dot size (default = 0.3)

`sep` character Separator for creating unique cell names (default = "!!")

`...` Plotting parameters passed to `sccore::embeddingPlot`.

*Returns:* ggplot2 object or data frame

*Examples:*

```
\donttest{
if (requireNamespace("pagoda2", quietly = TRUE)) {
if (requireNamespace("conos", quietly = TRUE)) {
# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
sapply(seq_len(1e3), \(x) paste0("cell",x)))
return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, sample.names = c("sample1", "sample2"), n.cores = 1)

# Create embedding
crm$doPreprocessing()
crm$createEmbedding()

# Plot and extract result
crm$plotFilteredCells(type = "embedding")
filtered.cells <- crm$plotFilteredCells(type = "export")
} else {
message("Package 'conos' not available.")
}
```

```
} else {
message("Package 'pagoda2' not available.")
}
}
```

**Method** `getConosDepth()`**:** Extract sequencing depth from Conos object.

*Usage:*

```
CRMetrics$getConosDepth()
```

*Returns:* data frame

*Examples:*

```
\donttest{
if (requireNamespace("pagoda2", quietly = TRUE)) {
if (requireNamespace("conos", quietly = TRUE)) {
# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
sapply(seq_len(1e3), \(x) paste0("cell",x)))
return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, sample.names = c("sample1", "sample2"), n.cores = 1)

# Create embedding
crm$doPreprocessing()
crm$createEmbedding()

# Get Conos depth
crm$getConosDepth()
} else {
message("Package 'conos' not available.")
}
} else {
message("Package 'pagoda2' not available.")
}
}
```

**Method** `getMitoFraction()`**:** Calculate the fraction of mitochondrial genes.

*Usage:*

```
CRMetrics$getMitoFraction(species = c("human", "mouse"))
```

*Arguments:*

`species` character Species to calculate the mitochondrial fraction for (default = "human").

`force` logical Force update of stored vector (default = FALSE)

*Returns:* data frame

*Examples:*

```
\donttest{
if (requireNamespace("pagoda2", quietly = TRUE)) {
if (requireNamespace("conos", quietly = TRUE)) {
# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
sapply(seq_len(1e3), \(x) paste0("cell",x)))
return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, sample.names = c("sample1", "sample2"), n.cores = 1)

# Create embedding
crm$doPreprocessing()
crm$createEmbedding()

# Get mito. fraction
crm$getMitoFraction(species = c("human", "mouse"))
} else {
message("Package 'conos' not available.")
}
} else {
message("Package 'pagoda2' not available.")
}
}
```

**Method** `prepareCellbender()`**:** Create plots and script call for CellBender

*Usage:*

```
CRMetrics$prepareCellbender(
  shrinkage = 100,
  show.expected.cells = TRUE,
  show.total.droplets = TRUE,
  expected.cells = NULL,
  total.droplets = NULL,
  cms.raw = self$cms.raw,
  umi.counts = self$cellbender$umi.counts,
  data.path = self$data.path,
  samples = self$metadata$sample,
  verbose = self$verbose,
  n.cores = self$n.cores,
  unique.names = FALSE,
  sep = "!!"
)
```

*Arguments:*

`shrinkage` integer Select every nth UMI count per cell for plotting. Improves plotting speed drastically. To plot all cells, set to 1 (default = 100)

`show.expected.cells` logical Plot line depicting expected number of cells (default = TRUE)

`show.total.droplets` logical Plot line depicting total droplets included for CellBender run (default = TRUE)

`expected.cells` named numeric If NULL, expected cells will be deduced from the number of cells per sample identified by Cell Ranger. Otherwise, a named vector of expected cells with sample IDs as names. Sample IDs must match those in summary.metrics (default: stored named vector)

`total.droplets` named numeric If NULL, total droplets included will be deduced from expected cells multiplied by 3. Otherwise, a named vector of total droplets included with sample IDs as names. Sample IDs must match those in summary.metrics (default: stored named vector)

`cms.raw` list Raw count matrices from HDF5 Cell Ranger outputs (default = self$cms.raw)

`umi.counts` list UMI counts calculated as column sums of raw count matrices from HDF5 Cell Ranger outputs (default: stored list)

`data.path` character Path to Cell Ranger outputs (default = self$data.path)

`samples` character Sample names to include (default = self$metadata$sample)

`verbose` logical Show progress (default: stored vector)

`n.cores` integer Number of cores (default: stored vector)

`unique.names` logical Create unique cell names (default = FALSE)

`sep` character Separator for creating unique cell names (default = "!!")

*Returns:* ggplot2 object and bash script

*Examples:*

```
\dontrun{
crm <- CRMetrics$new(data.path = "/path/to/count/data")
crm$prepareCellbender()
}
```

**Method** `saveCellbenderScript()`:

*Usage:*

```
CRMetrics$saveCellbenderScript(
  file = "cellbender_script.sh",
  fpr = 0.01,
  epochs = 150,
  use.gpu = TRUE,
  expected.cells = NULL,
  total.droplets = NULL,
  data.path = self$data.path,
  samples = self$metadata$sample,
  args = NULL
)
```

*Arguments:*

`file` character File name for CellBender script. Will be stored in `data.path` (default: "cellbender_script.sh") `fpr` numeric False positive rate for CellBender (default = 0.01)

`epochs` integer Number of epochs for CellBender (default = 150)

`use.gpu` logical Use CUDA capable GPU (default = TRUE)

`expected.cells` named numeric If NULL, expected cells will be deduced from the number of cells per sample identified by Cell Ranger. Otherwise, a named vector of expected cells with sample IDs as names. Sample IDs must match those in summary.metrics (default: stored named vector)

`total.droplets` named numeric If NULL, total droplets included will be deduced from expected cells multiplied by 3. Otherwise, a named vector of total droplets included with sample IDs as names. Sample IDs must match those in summary.metrics (default: stored named vector)

`data.path` character Path to Cell Ranger outputs (default = self$data.path)

`samples` character Sample names to include (default = self$metadata$sample)

`args` character (optional) Additional parameters for CellBender

*Returns:* bash script

*Examples:*

```
\dontrun{
crm <- CRMetrics$new(data.path = "/path/to/count/data/")
crm$prepareCellbender()
crm$saveCellbenderScript()
}
```

**Method** `getExpectedCells()`**:** Extract the expected number of cells per sample based on the Cell Ranger summary metrics

*Usage:*

```
CRMetrics$getExpectedCells(samples = self$metadata$sample)
```

*Arguments:*

`samples` character Sample names to include (default = self$metadata$sample)

*Returns:* A numeric vector

*Examples:*

```
# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
sapply(seq_len(1e3), \(x) paste0("cell",x)))
return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, sample.names = c("sample1", "sample2"), n.cores = 1)

# Get summary
crm$addSummaryFromCms()
```

```
# Get no. cells
crm$getExpectedCells()
```

**Method** `getTotalDroplets()`**:** Get the total number of droplets included in the CellBender estimations. Based on the Cell Ranger summary metrics and multiplied by a preset multiplier.

*Usage:*

```
CRMetrics$getTotalDroplets(samples = self$metadata$sample, multiplier = 3)
```

*Arguments:*

`samples` character Samples names to include (default = self$metadata$sample)

`multiplier` numeric Number to multiply expected number of cells with (default = 3)

*Returns:* A numeric vector

*Examples:*

```
# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
sapply(seq_len(1e3), \(x) paste0("cell",x)))
return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, sample.names = c("sample1", "sample2"), n.cores = 1)

# Add summary
crm$addSummaryFromCms()

# Get no. droplets
crm$getTotalDroplets()
```

**Method** `addCms()`**:** Add a list of count matrices to the CRMetrics object.

*Usage:*

```
CRMetrics$addCms(
  cms,
  sample.names = NULL,
  unique.names = TRUE,
  sep = "!!",
  n.cores = self$n.cores
)
```

*Arguments:*

`cms` list List of count matrices

`sample.names` character Vector of sample names. If NULL, sample.names are extracted from cms (default = NULL)

`unique.names` logical Create unique cell names (default = TRUE)

`sep` character Separator used to create unique cell names (default = "!!")

`n.cores` integer Number of cores to use (default = self$n.cores)

*Returns:* A ggplot2 object

*Examples:*

```
\dontrun{
crm <- CRMetrics$new(data.path = "/path/to/count/data/")

# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
sapply(seq_len(1e3), \(x) paste0("cell",x)))
return(out)
})

crm$addCms(cms = testdata.cms)
}
```

**Method** `plotCbTraining()`**:** Plot the results from the CellBender estimations

*Usage:*

```
CRMetrics$plotCbTraining(
  data.path = self$data.path,
  samples = self$metadata$sample
)
```

*Arguments:*

`data.path` character Path to Cell Ranger outputs (default = self$data.path)

`samples` character Sample names to include (default = self$metadata$sample)

*Returns:* A ggplot2 object

*Examples:*

```
\dontrun{
crm <- CRMetrics$new(data.path = "/path/to/count/data/")
crm$prepareCellbender()
crm$saveCellbenderScript()
## Run CellBender script
crm$plotCbTraining()
}
```

**Method** `plotCbCellProbs()`**:** Plot the CellBender assigned cell probabilities

*Usage:*

```
CRMetrics$plotCbCellProbs(
  data.path = self$data.path,
  samples = self$metadata$sample
)
```

*Arguments:*

`data.path` character Path to Cell Ranger outputs (default = self$data.path) `samples` character Sample names to include (default = self$metadata$sample)

*Returns:* A ggplot2 object

*Examples:*

```
\dontrun{
crm <- CRMetrics$new(data.path = "/path/to/count/data/")
crm$prepareCellbender()
crm$saveCellbenderScript()
## Run the CellBender script
crm$plotCbCellProbs()
}
```

**Method** `plotCbAmbExp()`**:** Plot the estimated ambient gene expression per sample from CellBender calculations

*Usage:*

```
CRMetrics$plotCbAmbExp(
  cutoff = 0.005,
  data.path = self$data.path,
  samples = self$metadata$sample
)
```

*Arguments:*

`cutoff` numeric Horizontal line included in the plot to indicate highly expressed ambient genes (default = 0.005)

`data.path` character Path to Cell Ranger outputs (default = self$data.path)

`samples` character Sample names to include (default = self$metadata$sample)

*Returns:* A ggplot2 object

*Examples:*

```
\dontrun{
crm <- CRMetrics$new(data.path = "/path/to/count/data/")
crm$prepareCellbender()
crm$saveCellbenderScript()
## Run CellBender script
crm$plotCbAmbExp()
}
```

**Method** `plotCbAmbGenes()`**:** Plot the most abundant estimated ambient genes from the CellBender calculations

*Usage:*

```
CRMetrics$plotCbAmbGenes(
  cutoff = 0.005,
  data.path = self$data.path,
  samples = self$metadata$sample
)
```

*Arguments:*

`cutoff` numeric Cutoff of ambient gene expression to use to extract ambient genes per sample `data.path` character Path to Cell Ranger outputs (default = self$data.path)

`samples` character Sample names to include (default = self$metadata$sample)

*Returns:* A ggplot2 object

*Examples:*

```
\dontrun{
crm <- CRMetrics$new(data.path = "/path/to/count/data/")
crm$prepareCellbender()
crm$saveCellbenderScript()
## Run CellBender script
crm$plotCbAmbGenes()
}
```

**Method** `addSummaryFromCms()`**:** Add summary metrics from a list of count matrices

*Usage:*

```
CRMetrics$addSummaryFromCms(
  cms = self$cms,
  n.cores = self$n.cores,
  verbose = self$verbose
)
```

*Arguments:*

`cms` list A list of filtered count matrices (default = self$cms)

`n.cores` integer Number of cores to use (default = self$n.cores)

`verbose` logical Show progress (default = self$verbose)

*Returns:* data.frame

*Examples:*

```
# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
sapply(seq_len(1e3), \(x) paste0("cell",x)))
return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, sample.names = c("sample1", "sample2"), n.cores = 1)

# Add summary
crm$addSummaryFromCms()
```

**Method** `runSoupX()`**:** Run SoupX ambient RNA estimation and correction

*Usage:*

```
CRMetrics$runSoupX(
  data.path = self$data.path,
  samples = self$metadata$sample,
```

```
  n.cores = self$n.cores,
  verbose = self$verbose,
  arg.load10X = list(),
  arg.autoEstCont = list(),
  arg.adjustCounts = list()
)
```

*Arguments:*

`data.path` character Path to Cell Ranger outputs (default = self$data.path)

`samples` character Sample names to include (default = self$metadata$sample)

`n.cores` numeric Number of cores (default = self$n.cores)

`verbose` logical Show progress (default = self$verbose)

`arg.load10X` list A list with additional parameters for `SoupX::load10X` (default = list())

`arg.autoEstCont` list A list with additional parameters for `SoupX::autoEstCont` (default = list())

`arg.adjustCounts` list A list with additional parameters for `SoupX::adjustCounts` (default = list())

*Returns:* List containing a list with corrected counts, and a data.frame containing plotting estimations

*Examples:*

```
\dontrun{
crm <- CRMetrics$new(data.path = "/path/to/count/data/")
crm$runSoupX()
}
```

**Method** `plotSoupX()`**:** Plot the results from the SoupX estimations

*Usage:*

```
CRMetrics$plotSoupX(plot.df = self$soupx$plot.df)
```

*Arguments:*

`plot.df` data.frame SoupX estimations (default = self$soupx$plot.df)

*Returns:* A ggplot2 object

*Examples:*

```
\dontrun{
crm <- CRMetrics$new(data.path = "/path/to/count/data/")
crm$runSoupX()
crm$plotSoupX()
}
```

**Method** `plotCbCells()`**:** Plot CellBender cell estimations against the estimated cell numbers from Cell Ranger

*Usage:*

```
CRMetrics$plotCbCells(
  data.path = self$data.path,
  samples = self$metadata$sample
)
```

*Arguments:*

`data.path` character Path to Cell Ranger outputs (default = self$data.path)

`samples` character Sample names to include (default = self$metadata$sample)

*Returns:* A ggplot2 object

*Examples:*

```
\dontrun{
crm <- CRMetrics$new(data.path = "/path/to/count/data/")
crm$prepareCellbender()
crm$saveCellbenderScript()
## Run CellBender script
crm$plotCbCells()
}
```

**Method** `clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
CRMetrics$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

### Examples

```
## ------------------------------------------------
## Method `CRMetrics$new`
## ------------------------------------------------

## Not run:
crm <- CRMetrics$new(data.path = "/path/to/count/data/")

## End(Not run)

## ------------------------------------------------
## Method `CRMetrics$addDetailedMetrics`
## ------------------------------------------------

# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
sapply(seq_len(1e3), \(x) paste0("cell",x)))
return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, sample.names = c("sample1", "sample2"), n.cores = 1)

# Run function
crm$addDetailedMetrics()
```

```
## ------------------------------------------------
## Method `CRMetrics$addComparison`
## ------------------------------------------------

# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
sapply(seq_len(1e3), \(x) paste0("cell",x)))
return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, sample.names = c("sample1", "sample2"), n.cores = 1)

# Add metadata
crm$metadata$sex <- c("male","female")

# Add comparison group
crm$addComparison(comp.group = "sex")

## ------------------------------------------------
## Method `CRMetrics$plotSamples`
## ------------------------------------------------

sample.names <- c("sample1", "sample2")

# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
sapply(seq_len(1e3), \(x) paste0("cell",x)))
return(out)
})
names(testdata.cms) <- sample.names

# Create metadata
metadata <- data.frame(sample = sample.names,
sex = c("male","female"),
condition = c("a","b"))

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, metadata = metadata, n.cores = 1)

# Plot
crm$plotSamples(comp.group = "sex", second.comp.group = "condition")

## ------------------------------------------------
## Method `CRMetrics$plotSummaryMetrics`
## ------------------------------------------------
```

```
# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
sapply(seq_len(1e3), \(x) paste0("cell",x)))
return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, sample.names = c("sample1", "sample2"), n.cores = 1)

# Add summary metrics
crm$addSummaryFromCms()

crm$plotSummaryMetrics(plot.geom = "point")


## ------------------------------------------------
## Method `CRMetrics$plotDetailedMetrics`
## ------------------------------------------------


# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
sapply(seq_len(1e3), \(x) paste0("cell",x)))
return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, sample.names = c("sample1", "sample2"), n.cores = 1)

# Add detailed metrics
crm$addDetailedMetrics()

# Plot
crm$plotDetailedMetrics()


## ------------------------------------------------
## Method `CRMetrics$plotEmbedding`
## ------------------------------------------------


if (requireNamespace("pagoda2", quietly = TRUE)) {
if (requireNamespace("conos", quietly = TRUE)) {
# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
```

```
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
sapply(seq_len(1e3), \(x) paste0("cell",x)))
return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, sample.names = c("sample1", "sample2"), n.cores = 1)

# Create embedding
crm$doPreprocessing()
crm$createEmbedding()

crm$plotEmbedding()
} else {
message("Package 'conos' not available.")
}
} else {
message("Package 'pagoda2' not available.")
}


## ------------------------------------------------
## Method `CRMetrics$plotDepth`
## ------------------------------------------------


if (requireNamespace("pagoda2", quietly = TRUE)) {
if (requireNamespace("conos", quietly = TRUE)) {
# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
sapply(seq_len(1e3), \(x) paste0("cell",x)))
return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, sample.names = c("sample1", "sample2"), n.cores = 1)

# Create embedding
crm$doPreprocessing()
crm$createEmbedding()

# Plot
crm$plotDepth()
} else {
message("Package 'conos' not available.")
}
} else {
message("Package 'pagoda2' not available.")
```

```
}


## ------------------------------------------------
## Method `CRMetrics$plotMitoFraction`
## ------------------------------------------------


if (requireNamespace("pagoda2", quietly = TRUE)) {
if (requireNamespace("conos", quietly = TRUE)) {
# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
sapply(seq_len(1e3), \(x) paste0("cell",x)))
return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, sample.names = c("sample1", "sample2"), n.cores = 1)

# Create embedding
crm$doPreprocessing()
crm$createEmbedding()

# Plot
crm$plotMitoFraction()
} else {
message("Package 'conos' not available.")
}
} else {
message("Package 'pagoda2' not available.")
}


## ------------------------------------------------
## Method `CRMetrics$detectDoublets`
## ------------------------------------------------

## Not run:
# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
sapply(seq_len(1e3), \(x) paste0("cell",x)))
return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, sample.names = c("sample1", "sample2"), n.cores = 1)
```

```
# Detect doublets
crm$detectDoublets(method = "scrublet",
conda.path = "/opt/software/miniconda/4.12.0/condabin/conda")

## End(Not run)

## ------------------------------------------------
## Method `CRMetrics$doPreprocessing`
## ------------------------------------------------


if (requireNamespace("pagoda2", quietly = TRUE)) {
# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
sapply(seq_len(1e3), \(x) paste0("cell",x)))
return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, sample.names = c("sample1", "sample2"), n.cores = 1)

# Perform preprocessing
crm$doPreprocessing(preprocess = "pagoda2")
} else {
message("Package 'pagoda2' not available.")
}


## ------------------------------------------------
## Method `CRMetrics$createEmbedding`
## ------------------------------------------------


if (requireNamespace("pagoda2", quietly = TRUE)) {
if (requireNamespace("conos", quietly = TRUE)) {
# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
sapply(seq_len(1e3), \(x) paste0("cell",x)))
return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, sample.names = c("sample1", "sample2"), n.cores = 1)

# Create embedding
crm$doPreprocessing()
```

```
crm$createEmbedding()
} else {
message("Package 'conos' not available.")
}
} else {
message("Package 'pagoda2' not available.")
}


## ------------------------------------------------
## Method `CRMetrics$filterCms`
## ------------------------------------------------


if (requireNamespace("pagoda2", quietly = TRUE)) {
if (requireNamespace("conos", quietly = TRUE)) {
# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
sapply(seq_len(1e3), \(x) paste0("cell",x)))
return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, sample.names = c("sample1", "sample2"), n.cores = 1)

# Create embedding
crm$doPreprocessing()
crm$createEmbedding()


# Filter CMs
crm$filterCms(depth.cutoff = 1e3, mito.cutoff = 0.05)
} else {
message("Package 'conos' not available.")
}
} else {
message("Package 'pagoda2' not available.")
}


## ------------------------------------------------
## Method `CRMetrics$selectMetrics`
## ------------------------------------------------

# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
sapply(seq_len(1e3), \(x) paste0("cell",x)))
```

```
return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, sample.names = c("sample1", "sample2"), n.cores = 1)

# Select metrics
crm$selectMetrics()
selection.metrics <- crm$selectMetrics(c(1:4))

## ------------------------------------------------
## Method `CRMetrics$plotFilteredCells`
## ------------------------------------------------


if (requireNamespace("pagoda2", quietly = TRUE)) {
if (requireNamespace("conos", quietly = TRUE)) {
# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
sapply(seq_len(1e3), \(x) paste0("cell",x)))
return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, sample.names = c("sample1", "sample2"), n.cores = 1)

# Create embedding
crm$doPreprocessing()
crm$createEmbedding()

# Plot and extract result
crm$plotFilteredCells(type = "embedding")
filtered.cells <- crm$plotFilteredCells(type = "export")
} else {
message("Package 'conos' not available.")
}
} else {
message("Package 'pagoda2' not available.")
}


## ------------------------------------------------
## Method `CRMetrics$getConosDepth`
## ------------------------------------------------


if (requireNamespace("pagoda2", quietly = TRUE)) {
if (requireNamespace("conos", quietly = TRUE)) {
# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
```

```
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
sapply(seq_len(1e3), \(x) paste0("cell",x)))
return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, sample.names = c("sample1", "sample2"), n.cores = 1)

# Create embedding
crm$doPreprocessing()
crm$createEmbedding()

# Get Conos depth
crm$getConosDepth()
} else {
message("Package 'conos' not available.")
}
} else {
message("Package 'pagoda2' not available.")
}


## ------------------------------------------------
## Method `CRMetrics$getMitoFraction`
## ------------------------------------------------


if (requireNamespace("pagoda2", quietly = TRUE)) {
if (requireNamespace("conos", quietly = TRUE)) {
# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
sapply(seq_len(1e3), \(x) paste0("cell",x)))
return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, sample.names = c("sample1", "sample2"), n.cores = 1)

# Create embedding
crm$doPreprocessing()
crm$createEmbedding()

# Get mito. fraction
crm$getMitoFraction(species = c("human", "mouse"))
} else {
message("Package 'conos' not available.")
}
} else {
```

```
message("Package 'pagoda2' not available.")
}


## ------------------------------------------------
## Method `CRMetrics$prepareCellbender`
## ------------------------------------------------

## Not run:
crm <- CRMetrics$new(data.path = "/path/to/count/data")
crm$prepareCellbender()

## End(Not run)

## ------------------------------------------------
## Method `CRMetrics$saveCellbenderScript`
## ------------------------------------------------

## Not run:
crm <- CRMetrics$new(data.path = "/path/to/count/data/")
crm$prepareCellbender()
crm$saveCellbenderScript()

## End(Not run)

## ------------------------------------------------
## Method `CRMetrics$getExpectedCells`
## ------------------------------------------------

# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
sapply(seq_len(1e3), \(x) paste0("cell",x)))
return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, sample.names = c("sample1", "sample2"), n.cores = 1)

# Get summary
crm$addSummaryFromCms()

# Get no. cells
crm$getExpectedCells()

## ------------------------------------------------
## Method `CRMetrics$getTotalDroplets`
## ------------------------------------------------

# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
```

```
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
sapply(seq_len(1e3), \(x) paste0("cell",x)))
return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, sample.names = c("sample1", "sample2"), n.cores = 1)

# Add summary
crm$addSummaryFromCms()

# Get no. droplets
crm$getTotalDroplets()

## ------------------------------------------------
## Method `CRMetrics$addCms`
## ------------------------------------------------

## Not run:
crm <- CRMetrics$new(data.path = "/path/to/count/data/")

# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
sapply(seq_len(1e3), \(x) paste0("cell",x)))
return(out)
})

crm$addCms(cms = testdata.cms)

## End(Not run)

## ------------------------------------------------
## Method `CRMetrics$plotCbTraining`
## ------------------------------------------------

## Not run:
crm <- CRMetrics$new(data.path = "/path/to/count/data/")
crm$prepareCellbender()
crm$saveCellbenderScript()
## Run CellBender script
crm$plotCbTraining()

## End(Not run)

## ------------------------------------------------
## Method `CRMetrics$plotCbCellProbs`
## ------------------------------------------------
```

```
## Not run:
crm <- CRMetrics$new(data.path = "/path/to/count/data/")
crm$prepareCellbender()
crm$saveCellbenderScript()
## Run the CellBender script
crm$plotCbCellProbs()

## End(Not run)

## ------------------------------------------------
## Method `CRMetrics$plotCbAmbExp`
## ------------------------------------------------

## Not run:
crm <- CRMetrics$new(data.path = "/path/to/count/data/")
crm$prepareCellbender()
crm$saveCellbenderScript()
## Run CellBender script
crm$plotCbAmbExp()

## End(Not run)

## ------------------------------------------------
## Method `CRMetrics$plotCbAmbGenes`
## ------------------------------------------------

## Not run:
crm <- CRMetrics$new(data.path = "/path/to/count/data/")
crm$prepareCellbender()
crm$saveCellbenderScript()
## Run CellBender script
crm$plotCbAmbGenes()

## End(Not run)

## ------------------------------------------------
## Method `CRMetrics$addSummaryFromCms`
## ------------------------------------------------

# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
sapply(seq_len(1e3), \(x) paste0("cell",x)))
return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, sample.names = c("sample1", "sample2"), n.cores = 1)

# Add summary
crm$addSummaryFromCms()
```

```
## ------------------------------------------------
## Method `CRMetrics$runSoupX`
## ------------------------------------------------

## Not run:
crm <- CRMetrics$new(data.path = "/path/to/count/data/")
crm$runSoupX()

## End(Not run)

## ------------------------------------------------
## Method `CRMetrics$plotSoupX`
## ------------------------------------------------

## Not run:
crm <- CRMetrics$new(data.path = "/path/to/count/data/")
crm$runSoupX()
crm$plotSoupX()

## End(Not run)

## ------------------------------------------------
## Method `CRMetrics$plotCbCells`
## ------------------------------------------------

## Not run:
crm <- CRMetrics$new(data.path = "/path/to/count/data/")
crm$prepareCellbender()
crm$saveCellbenderScript()
## Run CellBender script
crm$plotCbCells()

## End(Not run)
```

---

## `read10xH5` *Read 10x HDF5 files*

### Description

Read 10x HDF5 files

### Usage

```
read10xH5(
  data.path,
  sample.names = NULL,
  type = c("raw", "filtered", "cellbender", "cellbender_filtered"),
  symbol = TRUE,
  sep = "!!",
```

```
  n.cores = 1,
  verbose = TRUE,
  unique.names = FALSE
)
```

## Arguments

| | |
|---|---|
| `data.path` | character |
| `sample.names` | character vector, select specific samples for processing (default = NULL) |
| `type` | name of H5 file to search for, "raw" and "filtered" are Cell Ranger count outputs, "cellbender" is output from CellBender after running script from saveCellbenderScript |
| `symbol` | logical Use gene SYMBOLs (TRUE) or ENSEMBL IDs (FALSE) (default = TRUE) |
| `sep` | character Separator for creating unique cell names from sample IDs and cell IDs (default = "!!") |
| `n.cores` | integer Number of cores (default = 1) |
| `verbose` | logical Print progress (default = TRUE) |
| `unique.names` | logical Create unique cell IDs (default = FALSE) |

## Value

list with sparse count matrices

## Examples

```
## Not run:
cms.h5 <- read10xH5(data.path = "/path/to/count/data")

## End(Not run)
```

# Index