# Package ‘cTRAP’

October 18, 2022

**Title** Identification of candidate causal perturbations from differential gene expression data

**Version** 1.14.1

**Description** Compare differential gene expression results with those from known cellular perturbations (such as gene knock-down, overexpression or small molecules) derived from the Connectivity Map. Such analyses allow not only to infer the molecular causes of the observed difference in gene expression but also to identify small molecules that could drive or revert specific transcriptomic alterations.

**Depends** R (>= 4.0)

**License** MIT + file LICENSE

**Encoding** UTF-8

**LazyData** true

**biocViews** DifferentialExpression, GeneExpression, RNASeq, Transcriptomics, Pathways, ImmunoOncology, GeneSetEnrichment

**URL** `https://nuno-agostinho.github.io/cTRAP`, `https://github.com/nuno-agostinho/cTRAP`

**BugReports** `https://github.com/nuno-agostinho/cTRAP/issues`

**Suggests** testthat, knitr, covr, rmarkdown, spelling, biomaRt, remotes

**RoxygenNote** 7.2.0

**Imports** AnnotationDbi, AnnotationHub, binr, cowplot, data.table, dplyr, DT, fastmatch, fgsea, ggplot2, ggrepel, graphics, highcharter, htmltools, httr, limma, methods, parallel, pbapply, purrr, qs, R.utils, readxl, reshape2, rhdf5, rlang, scales, shiny (>= 1.7.0), shinycssloaders, stats, tibble, tools, utils

**VignetteBuilder** knitr

**Language** en-GB

**Collate** 'utils.R' 'CMap.R' 'ENCODE.R' 'cTRAP-package.r' 'cmapR_subset.R' 'compare.R' 'drugSensitivity.R' 'drugSetEnrichment.R' 'floweRy.R' 'plots.R' 'shinyInterface.R' 'shinyInterface_session.R'

**git_url** https://git.bioconductor.org/packages/cTRAP

**git_branch** RELEASE_3_15

**git_last_commit** 0cfc240

**git_last_commit_date** 2022-07-14

**Date/Publication** 2022-10-18

**Author** Bernardo P. de Almeida [aut],
Nuno Saraiva-Agostinho [aut, cre],
Nuno L. Barbosa-Morais [aut, led]

**Maintainer** Nuno Saraiva-Agostinho <`nunodanielagostinho@gmail.com`>

# R topics documented:

---

`analyseDrugSetEnrichment` *Analyse drug set enrichment*

---

## Description

Analyse drug set enrichment

## Usage

```
analyseDrugSetEnrichment(
  sets,
  stats,
  col = NULL,
  nperm = 10000,
  maxSize = 500,
  ...,
  keyColSets = NULL,
  keyColStats = NULL
)
```

## Arguments

| | |
|---|---|
| `sets` | Named list of characters: named sets containing compound identifiers (obtain drug sets by running `prepareDrugSets()`) |
| `stats` | Named numeric vector or either a `similarPerturbations` or a `targetingDrugs` object (obtained after running `rankSimilarPerturbations` or `predictTargetingDrugs`, respectively) |
| `col` | Character: name of the column to use for statistics (only required if class of `stats` is either `similarPerturbations` or `targetingDrugs`) |
| `nperm` | Number of permutations to do. Minimial possible nominal p-value is about 1/nperm |
| `maxSize` | Maximal size of a gene set to test. All pathways above the threshold are excluded. |
| `...` | Arguments passed on to `fgsea::fgseaSimple`<br>`minSize` Minimal size of a gene set to test. All pathways below the threshold are excluded.<br>`scoreType` This parameter defines the GSEA score type. Possible options are ("std", "pos", "neg")<br>`nproc` If not equal to zero sets BPPARAM to use nproc workers (default = 0).<br>`gseaParam` GSEA parameter value, all gene-level statis are raised to the power of ‘gseaParam‘ before calculation of GSEA enrichment scores. |

| | |
|---|---|
| | `BPPARAM` Parallelization parameter used in bplapply. Can be used to specify cluster to run. If not initialized explicitly or by setting ‘nproc‘ default value ‘bpparam()‘ is used. |
| `keyColSets` | Character: column from `sets` to compare with column `keyColStats` from `stats`; automatically selected if `NULL` |
| `keyColStats` | Character: column from `stats` to compare with column `keyColSets` from `sets`; automatically selected if `NULL` |

## Value

Enrichment analysis based on GSEA

## See Also

Other functions for drug set enrichment analysis: `loadDrugDescriptors()`, `plotDrugSetEnrichment()`, `prepareDrugSets()`

## Examples

```
descriptors <- loadDrugDescriptors()
drugSets <- prepareDrugSets(descriptors)

# Analyse drug set enrichment in ranked targeting drugs for a differential
# expression profile
data("diffExprStat")
gdsc      <- loadExpressionDrugSensitivityAssociation("GDSC")
predicted <- predictTargetingDrugs(diffExprStat, gdsc)

analyseDrugSetEnrichment(drugSets, predicted)
```

---

# `as.table.referenceComparison`
## *Cross Tabulation and Table Creation*

## Description

Cross Tabulation and Table Creation

## Usage

```
## S3 method for class 'referenceComparison'
as.table(x, ..., clean = TRUE)
```

## Arguments

| | |
|---|---|
| `x` | `referenceComparison` object |
| `...` | Extra parameters not currently used |
| `clean` | Boolean: only show certain columns (to avoid redundancy)? |

### Value

Complete table with metadata based on a `targetingDrugs` object

### See Also

Other functions related with the ranking of CMap perturbations: `filterCMapMetadata()`, `getCMapConditions()`, `getCMapPerturbationTypes()`, `loadCMapData()`, `loadCMapZscores()`, `parseCMapID()`, `plot.perturbationChanges(`
`plot.referenceComparison()`, `plotTargetingDrugsVSsimilarPerturbations()`, `prepareCMapPerturbations()`, `print.similarPerturbations()`, `rankSimilarPerturbations()`

Other functions related with the prediction of targeting drugs: `listExpressionDrugSensitivityAssociation()`, `loadExpressionDrugSensitivityAssociation()`, `plot.referenceComparison()`, `plotTargetingDrugsVSsimilarPe`
`predictTargetingDrugs()`

---

`convertENSEMBLtoGeneSymbols`

## *Convert ENSEMBL gene identifiers to gene symbols*

### Description

Convert ENSEMBL gene identifiers to gene symbols

### Usage

```
convertENSEMBLtoGeneSymbols(
  genes,
  dataset = "hsapiens_gene_ensembl",
  mart = "ensembl"
)
```

### Arguments

| | |
|---|---|
| `genes` | Character: ENSEMBL gene identifiers |
| `dataset` | Character: `biomaRt` dataset name |
| `mart` | Character: `biomaRt` database name |

### Value

Named character vector where names are the input ENSEMBL gene identifiers and the values are the matching gene symbols

---

`convertGeneIdentifiers` *Convert gene identifiers*

---

## Description

Convert gene identifiers

## Usage

```
convertGeneIdentifiers(
  genes,
  annotation = "Homo sapiens",
  key = "ENSEMBL",
  target = "SYMBOL",
  ignoreDuplicatedTargets = TRUE
)
```

## Arguments

| | |
|---|---|
| `genes` | Character: genes to be converted |
| `annotation` | `OrgDb` with genome wide annotation for an organism or `character` with species name to query `OrgDb`, e.g. `"Homo sapiens"` |
| `key` | Character: type of identifier used, e.g. `ENSEMBL`; read `?AnnotationDbi::columns` |
| `target` | Character: type of identifier to convert to; read `?AnnotationDbi::columns` |
| `ignoreDuplicatedTargets` | Boolean: if `TRUE`, identifiers that share targets with other identifiers will not be converted |

## Value

Character vector of the respective targets of gene identifiers. The previous identifiers remain other identifiers have the same target (in case `ignoreDuplicatedTargets = TRUE`) or if no target was found.

## Examples

```
genes <- c("ENSG00000012048", "ENSG00000083093", "ENSG00000141510",
           "ENSG00000051180")
convertGeneIdentifiers(genes)
convertGeneIdentifiers(genes, key="ENSEMBL", target="UNIPROT")

# Explicit species name to automatically look for its OrgDb database
sp <- "Homo sapiens"
genes <- c("ENSG00000012048", "ENSG00000083093", "ENSG00000141510",
           "ENSG00000051180")
convertGeneIdentifiers(genes, sp)
```

```
# Alternatively, set the annotation database directly
ah <- AnnotationHub::AnnotationHub()
sp <- AnnotationHub::query(ah, c("OrgDb", "Homo sapiens"))[[1]]
columns(sp) # these attributes can be used to change the attributes

convertGeneIdentifiers(genes, sp)
```

---

## `cTRAP` *cTRAP package*

### Description

Compare differential gene expression results with those from big datasets (e.g. CMap), allowing to infer which types of perturbations may explain the observed difference in gene expression.

Optimised to run in ShinyProxy with Celery/Flower backend with argument `shinyproxy = TRUE`.

### Usage

```
cTRAP(
  ...,
  commonPath = "data",
  expire = 14,
  fileSizeLimitMiB = 50,
  flowerURL = NULL,
  port = getOption("shiny.port"),
  host = getOption("shiny.host", "127.0.0.1")
)
```

### Arguments

| | |
|---|---|
| `...` | Objects |
| `commonPath` | Character: path where to store data common to all sessions |
| `expire` | Character: days until a session expires (message purposes only) |
| `fileSizeLimitMiB` | Numeric: file size limit in MiB |
| `flowerURL` | Character: Flower REST API's URL (`NULL` to avoid using Celery/Flower backend) |
| `port` | The TCP port that the application should listen on. If the `port` is not specified, and the `shiny.port` option is set (with `options(shiny.port = XX)`), then that port will be used. Otherwise, use a random port between 3000:8000, excluding ports that are blocked by Google Chrome for being considered unsafe: 3659, 4045, 5060, 5061, 6000, 6566, 6665:6669 and 6697. Up to twenty random ports will be tried. |
| `host` | The IPv4 address that the application should listen on. Defaults to the `shiny.host` option, if set, or `"127.0.0.1"` if not. See Details. |

## Details

**Input:** To use this package, a named vector of differentially expressed gene metric is needed, where its values represent the significance and magnitude of the differentially expressed genes (e.g. t-statistic) and its names are gene symbols.

**Workflow:** The differentially expressed genes will be compared against selected perturbation conditions by:

- Spearman or Pearson correlation with z-scores of differentially expressed genes after perturbations from CMap. Use function `rankSimilarPerturbations` with `method = "spearman"` or `method = "pearson"`
- Gene set enrichment analysis (GSEA) using the (around) 12 000 genes from CMap. Use function `rankSimilarPerturbations` with `method = gsea`.

Available perturbation conditions for CMap include:

- Cell line(s).
- Perturbation type (gene knockdown, gene upregulation or drug intake).
- Drug concentration.
- Time points.

Values for each perturbation type can be listed with `getCMapPerturbationTypes()`

**Output:** The output includes a data frame of ranked perturbations based on the associated statistical values and respective p-values.

## Value

Launches result viewer and plotter (returns `NULL`)

## See Also

Other visual interface functions: `launchCMapDataLoader()`, `launchDiffExprLoader()`, `launchDrugSetEnrichmentAnaly`
`launchMetadataViewer()`, `launchResultPlotter()`

---

## `dimnames.expressionDrugSensitivityAssociation`

### *Operations on* `expressionDrugSensitivityAssociation` *objects*

## Description

Operations on `expressionDrugSensitivityAssociation` objects

### Usage

```
## S3 method for class 'expressionDrugSensitivityAssociation'
dimnames(x)

## S3 method for class 'expressionDrugSensitivityAssociation'
dim(x)

## S3 method for class 'expressionDrugSensitivityAssociation'
x[i, j, drop = FALSE, ...]
```

### Arguments

| | |
|---|---|
| `x` | An `expressionDrugSensitivityAssociation` object |
| `i, j` | Character or numeric indexes specifying elements to extract |
| `drop` | Boolean: coerce result to the lowest possible dimension? |
| `...` | Extra arguments given to other methods |

### Value

Subset, dimension or dimension names

---

## `downloadENCODEknockdownMetadata` *Download metadata for ENCODE knockdown experiments*

### Description

Download metadata for ENCODE knockdown experiments

### Usage

```
downloadENCODEknockdownMetadata(
  cellLine = NULL,
  gene = NULL,
  file = "ENCODEmetadata.rds"
)
```

### Arguments

| | |
|---|---|
| `cellLine` | Character: cell line |
| `gene` | Character: target gene |
| `file` | Character: RDS filepath with metadata (if file doesn't exist, it will be created) |

### Value

Data frame containing ENCODE knockdown experiment metadata

### See Also

Other functions related with using ENCODE expression data: `loadENCODEsamples()`, `performDifferentialExpression(`
`prepareENCODEgeneExpression()`

### Examples

```
downloadENCODEknockdownMetadata("HepG2", "EIF4G1")
```

---

## `filterCMapMetadata` *Filter CMap metadata*

### Description

Filter CMap metadata

### Usage

```
filterCMapMetadata(
  metadata,
  cellLine = NULL,
  timepoint = NULL,
  dosage = NULL,
  perturbationType = NULL
)
```

### Arguments

| | |
|---|---|
| `metadata` | Data frame (CMap metadata) or character (respective filepath) |
| `cellLine` | Character: cell line (if `NULL`, all values are loaded) |
| `timepoint` | Character: timepoint (if `NULL`, all values are loaded) |
| `dosage` | Character: dosage (if `NULL`, all values are loaded) |
| `perturbationType` | Character: type of perturbation (if `NULL`, all perturbation types are loaded) |

### Value

Filtered CMap metadata

### See Also

Other functions related with the ranking of CMap perturbations: `as.table.referenceComparison()`,
`getCMapConditions()`, `getCMapPerturbationTypes()`, `loadCMapData()`, `loadCMapZscores()`,
`parseCMapID()`, `plot.perturbationChanges()`, `plot.referenceComparison()`, `plotTargetingDrugsVSsimilarPertu`
`prepareCMapPerturbations()`, `print.similarPerturbations()`, `rankSimilarPerturbations()`

### Examples

```
cmapMetadata <- loadCMapData("cmapMetadata.txt", "metadata")
filterCMapMetadata(cmapMetadata, cellLine="HEPG2", timepoint="2 h",
                   dosage="25 ng/mL")
```

---

## `getCMapConditions` *List available conditions in CMap datasets*

### Description

Downloads metadata if not available

### Usage

```
getCMapConditions(
  metadata,
  cellLine = NULL,
  timepoint = NULL,
  dosage = NULL,
  perturbationType = NULL,
  control = FALSE
)
```

### Arguments

| | |
|---|---|
| `metadata` | Data frame (CMap metadata) or character (respective filepath) |
| `cellLine` | Character: cell line (if `NULL`, all values are loaded) |
| `timepoint` | Character: timepoint (if `NULL`, all values are loaded) |
| `dosage` | Character: dosage (if `NULL`, all values are loaded) |
| `perturbationType` | Character: type of perturbation (if `NULL`, all perturbation types are loaded) |
| `control` | Boolean: show controls for perturbation types? |

### Value

List of conditions in CMap datasets

### See Also

Other functions related with the ranking of CMap perturbations: `as.table.referenceComparison()`, `filterCMapMetadata()`, `getCMapPerturbationTypes()`, `loadCMapData()`, `loadCMapZscores()`, `parseCMapID()`, `plot.perturbationChanges()`, `plot.referenceComparison()`, `plotTargetingDrugsVSsimilarPertu` `prepareCMapPerturbations()`, `print.similarPerturbations()`, `rankSimilarPerturbations()`

## Examples

```
## Not run:
cmapMetadata <- loadCMapData("cmapMetadata.txt", "metadata")

## End(Not run)
getCMapConditions(cmapMetadata)
```

---

# getCMapPerturbationTypes

*Get CMap perturbation types*

---

## Description

Get CMap perturbation types

## Usage

```
getCMapPerturbationTypes(control = FALSE)
```

## Arguments

| | |
|---|---|
| `control` | Boolean: return perturbation types used as control? |

## Value

Perturbation types and respective codes as used by CMap datasets

## See Also

Other functions related with the ranking of CMap perturbations: `as.table.referenceComparison()`, `filterCMapMetadata()`, `getCMapConditions()`, `loadCMapData()`, `loadCMapZscores()`, `parseCMapID()`, `plot.perturbationChanges()`, `plot.referenceComparison()`, `plotTargetingDrugsVSsimilarPerturbations()`, `prepareCMapPerturbations()`, `print.similarPerturbations()`, `rankSimilarPerturbations()`

## Examples

```
getCMapPerturbationTypes()
```

## `launchCMapDataLoader` *Load CMap data via a visual interface*

### Description

Load CMap data via a visual interface

### Usage

```
launchCMapDataLoader(
  metadata = "cmapMetadata.txt",
  zscores = "cmapZscores.gctx",
  geneInfo = "cmapGeneInfo.txt",
  compoundInfo = "cmapCompoundInfo.txt",
  cellLine = NULL,
  timepoint = NULL,
  dosage = NULL,
  perturbationType = NULL
)
```

### Arguments

| | |
|---|---|
| `metadata` | Data frame (CMap metadata) or character (respective filepath) |
| `zscores` | Data frame (GCTX z-scores) or character (respective filepath to load data from file) |
| `geneInfo` | Data frame (CMap gene info) or character (respective filepath to load data from file) |
| `compoundInfo` | Data frame (CMap compound info) or character (respective filepath to load data from file) |
| `cellLine` | Character: cell line (if `NULL`, all values are loaded) |
| `timepoint` | Character: timepoint (if `NULL`, all values are loaded) |
| `dosage` | Character: dosage (if `NULL`, all values are loaded) |
| `perturbationType` | Character: type of perturbation (if `NULL`, all perturbation types are loaded) |

### Value

CMap data

### See Also

Other visual interface functions: `cTRAP()`, `launchDiffExprLoader()`, `launchDrugSetEnrichmentAnalyser()`, `launchMetadataViewer()`, `launchResultPlotter()`

---

## `launchDiffExprLoader` *Load differential expression data via a visual interface*

### Description

Currently only supports loading data from ENCODE knockdown experiments

### Usage

```
launchDiffExprLoader(
  cellLine = NULL,
  gene = NULL,
  file = "ENCODEmetadata.rds",
  path = "."
)
```

### Arguments

| | |
|---|---|
| `cellLine` | Character: cell line |
| `gene` | Character: target gene |
| `file` | Character: RDS filepath with metadata (if file doesn't exist, it will be created) |
| `path` | Character: path where to download files |

### Value

Differential expression data

### See Also

Other visual interface functions: `cTRAP()`, `launchCMapDataLoader()`, `launchDrugSetEnrichmentAnalyser()`, `launchMetadataViewer()`, `launchResultPlotter()`

---

## `launchDrugSetEnrichmentAnalyser` *View and plot results via a visual interface*

### Description

View and plot results via a visual interface

### Usage

```
launchDrugSetEnrichmentAnalyser(sets, ...)
```

## Arguments

| | |
|---|---|
| `sets` | Named list of characters: named sets containing compound identifiers (obtain drug sets by running `prepareDrugSets()`) |
| `...` | Objects |

## Value

Launches result viewer and plotter (returns `NULL`)

## See Also

Other visual interface functions: `cTRAP()`, `launchCMapDataLoader()`, `launchDiffExprLoader()`, `launchMetadataViewer()`, `launchResultPlotter()`

---

# `launchMetadataViewer` *View metadata via a visual interface*

## Description

View metadata via a visual interface

## Usage

```
launchMetadataViewer(...)
```

## Arguments

| | |
|---|---|
| `...` | Objects |

## Value

Metadata viewer (retunrs `NULL`)

## See Also

Other visual interface functions: `cTRAP()`, `launchCMapDataLoader()`, `launchDiffExprLoader()`, `launchDrugSetEnrichmentAnalyser()`, `launchResultPlotter()`

## `launchResultPlotter` *View and plot results via a visual interface*

### Description

View and plot results via a visual interface

### Usage

```
launchResultPlotter(...)
```

### Arguments

| | |
|---|---|
| `...` | Objects |

### Value

Launches result viewer and plotter (returns `NULL`)

### See Also

Other visual interface functions: `cTRAP()`, `launchCMapDataLoader()`, `launchDiffExprLoader()`, `launchDrugSetEnrichmentAnalyser()`, `launchMetadataViewer()`

## `listExpressionDrugSensitivityAssociation` *List available gene expression and drug sensitivity correlation matrices*

### Description

List available gene expression and drug sensitivity correlation matrices

### Usage

```
listExpressionDrugSensitivityAssociation(url = FALSE)
```

### Arguments

| | |
|---|---|
| `url` | Boolean: return download link? |

### Value

Character vector of available gene expression and drug sensitivity correlation matrices

### See Also

Other functions related with the prediction of targeting drugs: `as.table.referenceComparison()`, `loadExpressionDrugSensitivityAssociation()`, `plot.referenceComparison()`, `plotTargetingDrugsVSsimilarPer`
`predictTargetingDrugs()`

### Examples

```
listExpressionDrugSensitivityAssociation()
```

---

## `loadCMapData` *Load CMap data*

### Description

Load CMap data (if not found, `file` will be automatically downloaded)

### Usage

```
loadCMapData(
  file,
  type = c("metadata", "geneInfo", "zscores", "compoundInfo"),
  zscoresID = NULL
)
```

### Arguments

| | |
|---|---|
| `file` | Character: path to file |
| `type` | Character: type of data to load (`metadata`, `geneInfo`, `zscores` or `compoundInfo`) |
| `zscoresID` | Character: identifiers to partially load z-scores file (for performance reasons; if `NULL`, all identifiers will be loaded) |

### Value

Metadata as a data table

### Note

If `type = "compoundInfo"`, two files from **The Drug Repurposing Hub** will be downloaded containing information about drugs and perturbations. The files will be named `file` with `_drugs` and `_samples` before their extension, respectively.

### See Also

Other functions related with the ranking of CMap perturbations: `as.table.referenceComparison()`, `filterCMapMetadata()`, `getCMapConditions()`, `getCMapPerturbationTypes()`, `loadCMapZscores()`, `parseCMapID()`, `plot.perturbationChanges()`, `plot.referenceComparison()`, `plotTargetingDrugsVSsimilarPertu`
`prepareCMapPerturbations()`, `print.similarPerturbations()`, `rankSimilarPerturbations()`

### Examples

```
# Load CMap metadata (data is automatically downloaded if not available)
cmapMetadata <- loadCMapData("cmapMetadata.txt", "metadata")

# Load CMap gene info
loadCMapData("cmapGeneInfo.txt", "geneInfo")
## Not run:
# Load CMap zscores based on filtered metadata
cmapMetadataKnockdown <- filterCMapMetadata(
  cmapMetadata, cellLine="HepG2",
  perturbationType="Consensus signature from shRNAs targeting the same gene")
loadCMapData("cmapZscores.gctx.gz", "zscores", cmapMetadataKnockdown$sig_id)

## End(Not run)
```

---

## `loadCMapZscores` *Load matrix of CMap perturbation's differential expression z-scores (optional)*

### Description

Load matrix of CMap perturbation's differential expression z-scores (optional)

### Usage

```
loadCMapZscores(data, inheritAttrs = FALSE, verbose = TRUE)
```

### Arguments

| | |
|---|---|
| `data` | `perturbationChanges` object |
| `inheritAttrs` | Boolean: convert to `perturbationChanges` object and inherit attributes from `data`? |
| `verbose` | Boolean: print additional details? |

### Value

Matrix containing CMap perturbation z-scores (genes as rows, perturbations as columns)

### See Also

Other functions related with the ranking of CMap perturbations: `as.table.referenceComparison()`, `filterCMapMetadata()`, `getCMapConditions()`, `getCMapPerturbationTypes()`, `loadCMapData()`, `parseCMapID()`, `plot.perturbationChanges()`, `plot.referenceComparison()`, `plotTargetingDrugsVSsimilarPertu` `prepareCMapPerturbations()`, `print.similarPerturbations()`, `rankSimilarPerturbations()`

### Examples

```
metadata <- loadCMapData("cmapMetadata.txt", "metadata")
metadata <- filterCMapMetadata(metadata, cellLine="HepG2")
## Not run:
perts <- prepareCMapPerturbations(metadata, "cmapZscores.gctx",
                                  "cmapGeneInfo.txt")
zscores <- loadCMapZscores(perts[ , 1:10])

## End(Not run)
```

---

## `loadDrugDescriptors` *Load table with drug descriptors*

---

### Description

Load table with drug descriptors

### Usage

```
loadDrugDescriptors(
  source = c("NCI60", "CMap"),
  type = c("2D", "3D"),
  file = NULL,
  path = NULL
)
```

### Arguments

| | |
|---|---|
| `source` | Character: source of compounds used to calculate molecular descriptors (`NCI60` or `CMap`) |
| `type` | Character: load `2D` or `3D` molecular descriptors |
| `file` | Character: filepath to drug descriptors (automatically downloaded if file does not exist) |
| `path` | Character: folder where to find files (optional; `file` may contain the full filepath if preferred) |

### Value

Data table with drug descriptors

### See Also

Other functions for drug set enrichment analysis: `analyseDrugSetEnrichment()`, `plotDrugSetEnrichment()`, `prepareDrugSets()`

### Examples

```
loadDrugDescriptors()
```

---

## `loadENCODEsamples` *Load ENCODE samples*

---

### Description

Samples are automatically downloaded if they are not found in the current working directory.

### Usage

```
loadENCODEsamples(metadata, path = ".")
```

### Arguments

| | |
|---|---|
| `metadata` | Character: ENCODE metadata |
| `path` | Character: path where to download files |

### Value

List of loaded ENCODE samples

### See Also

Other functions related with using ENCODE expression data: `downloadENCODEknockdownMetadata()`, `performDifferentialExpression()`, `prepareENCODEgeneExpression()`

### Examples

```
if (interactive()) {
  # Load ENCODE metadata for a specific cell line and gene
  cellLine <- "HepG2"
  gene <- c("EIF4G1", "U2AF2")
  ENCODEmetadata <- downloadENCODEknockdownMetadata(cellLine, gene)

  # Load samples based on filtered ENCODE metadata
  loadENCODEsamples(ENCODEmetadata)
}
```

---

## `loadExpressionDrugSensitivityAssociation` *Load gene expression and drug sensitivity correlation matrix*

---

### Description

Load gene expression and drug sensitivity correlation matrix

### Usage

```
loadExpressionDrugSensitivityAssociation(
  source,
  file = NULL,
  path = NULL,
  rows = NULL,
  cols = NULL,
  loadValues = FALSE
)
```

### Arguments

| | |
|---|---|
| `source` | Character: source of matrix to load; see `listExpressionDrugSensitivityAssociation` |
| `file` | Character: filepath to gene expression and drug sensitivity association dataset (automatically downloaded if file does not exist) |
| `path` | Character: folder where to find files (optional; `file` may contain the full filepath if preferred) |
| `rows` | Character or integer: rows |
| `cols` | Character or integer: columns |
| `loadValues` | Boolean: load data values (if available)? If `FALSE`, downstream functions will load and process directly from the file chunk by chunk, resulting in a lower memory footprint |

### Value

Correlation matrix between gene expression (rows) and drug sensitivity (columns)

### See Also

Other functions related with the prediction of targeting drugs: `as.table.referenceComparison()`, `listExpressionDrugSensitivityAssociation()`, `plot.referenceComparison()`, `plotTargetingDrugsVSsimilarPe` `predictTargetingDrugs()`

### Examples

```
gdsc <- listExpressionDrugSensitivityAssociation()[[1]]
loadExpressionDrugSensitivityAssociation(gdsc)
```

---

## `parseCMapID` *Parse CMap identifier*

### Description

Parse CMap identifier

### Usage

```
parseCMapID(id, cellLine = FALSE)
```

### Arguments

| | |
|---|---|
| `id` | Character: CMap identifier |
| `cellLine` | Boolean: if `TRUE`, return cell line information from CMap identifier; else, return the CMap identifier without the cell line |

### Value

Character vector with information from CMap identifiers

### See Also

Other functions related with the ranking of CMap perturbations: `as.table.referenceComparison()`, `filterCMapMetadata()`, `getCMapConditions()`, `getCMapPerturbationTypes()`, `loadCMapData()`, `loadCMapZscores()`, `plot.perturbationChanges()`, `plot.referenceComparison()`, `plotTargetingDrugsVSsimilarP` `prepareCMapPerturbations()`, `print.similarPerturbations()`, `rankSimilarPerturbations()`

### Examples

```
id <- c("CVD001_HEPG2_24H:BRD-K94818765-001-01-0:4.8",
        "CVD001_HEPG2_24H:BRD-K96188950-001-04-5:4.3967",
        "CVD001_HUH7_24H:BRD-A14014306-001-01-1:4.1")
parseCMapID(id, cellLine=TRUE)
parseCMapID(id, cellLine=FALSE)
```

---

## `performDifferentialExpression`

*Perform differential gene expression based on ENCODE data*

### Description

Perform differential gene expression based on ENCODE data

### Usage

```
performDifferentialExpression(counts)
```

### Arguments

| | |
|---|---|
| `counts` | Data frame: gene expression |

### Value

Data frame with differential gene expression results between knockdown and control

## See Also

Other functions related with using ENCODE expression data: `downloadENCODEknockdownMetadata()`, `loadENCODEsamples()`, `prepareENCODEgeneExpression()`

## Examples

```
if (interactive()) {
  # Download ENCODE metadata for a specific cell line and gene
  cellLine <- "HepG2"
  gene <- "EIF4G1"
  ENCODEmetadata <- downloadENCODEknockdownMetadata(cellLine, gene)

  # Download samples based on filtered ENCODE metadata
  ENCODEsamples <- loadENCODEsamples(ENCODEmetadata)[[1]]

  counts <- prepareENCODEgeneExpression(ENCODEsamples)

  # Remove low coverage (at least 10 counts shared across two samples)
  minReads   <- 10
  minSamples <- 2
  filter <- rowSums(counts[ , -c(1, 2)] >= minReads) >= minSamples
  counts <- counts[filter, ]

  # Convert ENSEMBL identifier to gene symbol
  counts$gene_id <- convertGeneIdentifiers(counts$gene_id)

  # Perform differential gene expression analysis
  diffExpr <- performDifferentialExpression(counts)
}
```

---

# `plot.perturbationChanges`

## *Operations on a* `perturbationChanges` *object*

## Description

Operations on a `perturbationChanges` object

## Usage

```
## S3 method for class 'perturbationChanges'
plot(
  x,
  perturbation,
  input,
  method = c("spearman", "pearson", "gsea"),
  geneSize = 150,
  genes = c("both", "top", "bottom"),
```

```
  ...,
  title = NULL
)

## S3 method for class 'perturbationChanges'
x[i, j, drop = FALSE, ...]

## S3 method for class 'perturbationChanges'
dim(x)

## S3 method for class 'perturbationChanges'
dimnames(x)
```

## Arguments

| | |
|---|---|
| `x` | `perturbationChanges` object |
| `perturbation` | Character (perturbation identifier) or a `similarPerturbations` table (from which the respective perturbation identifiers are retrieved) |
| `input` | `Named numeric vector` of differentially expressed genes whose names are gene identifiers and respective values are a statistic that represents significance and magnitude of differentially expressed genes (e.g. t-statistics); or `character` of gene symbols composing a gene set that is tested for enrichment in reference data (only used if `method` includes `gsea`) |
| `method` | Character: comparison method (`spearman`, `pearson` or `gsea`; multiple methods may be selected at once) |
| `geneSize` | Numeric: number of top up-/down-regulated genes to use as gene sets to test for enrichment in reference data; if a 2-length numeric vector, the first index is the number of top up-regulated genes and the second index is the number of down-regulated genes used to create gene sets; only used if `method` includes `gsea` and if `input` is not a gene set |
| `genes` | Character: when plotting gene set enrichment analysis (GSEA), plot most up-regulated genes (`genes = "top"`), most down-regulated genes (`genes = "bottom"`) or both (`genes = "both"`); only used if `method = "gsea"` and `geneset = NULL` |
| `...` | Extra arguments |
| `title` | Character: plot title (if `NULL`, the default title depends on the context; ignored when plotting multiple perturbations) |
| `i, j` | Character or numeric indexes specifying elements to extract |
| `drop` | Boolean: coerce result to the lowest possible dimension? |

## Value

Subset, plot or return dimensions or names of a `perturbationChanges` object

## See Also

Other functions related with the ranking of CMap perturbations: `as.table.referenceComparison()`, `filterCMapMetadata()`, `getCMapConditions()`, `getCMapPerturbationTypes()`, `loadCMapData()`,

loadCMapZscores(), parseCMapID(), plot.referenceComparison(), plotTargetingDrugsVSsimilarPerturbations(
prepareCMapPerturbations(), print.similarPerturbations(), rankSimilarPerturbations()

### Examples

```
data("diffExprStat")
data("cmapPerturbationsKD")

compareKD <- rankSimilarPerturbations(diffExprStat, cmapPerturbationsKD)
EIF4G1knockdown <- grep("EIF4G1", compareKD[[1]], value=TRUE)
plot(cmapPerturbationsKD, EIF4G1knockdown, diffExprStat, method="spearman")
plot(cmapPerturbationsKD, EIF4G1knockdown, diffExprStat, method="pearson")
plot(cmapPerturbationsKD, EIF4G1knockdown, diffExprStat, method="gsea")

data("cmapPerturbationsCompounds")
pert <- "CVD001_HEPG2_24H:BRD-A14014306-001-01-1:4.1"
plot(cmapPerturbationsCompounds, pert, diffExprStat, method="spearman")
plot(cmapPerturbationsCompounds, pert, diffExprStat, method="pearson")
plot(cmapPerturbationsCompounds, pert, diffExprStat, method="gsea")

# Multiple cell line perturbations
pert <- "CVD001_24H:BRD-A14014306-001-01-1:4.1"
plot(cmapPerturbationsCompounds, pert, diffExprStat, method="spearman")
plot(cmapPerturbationsCompounds, pert, diffExprStat, method="pearson")
plot(cmapPerturbationsCompounds, pert, diffExprStat, method="gsea")
```

---

## plot.referenceComparison

## *Plot data comparison*

---

### Description

If `element = NULL`, comparison is plotted based on all elements. Otherwise, show scatter or GSEA plots for a single element compared with previously given differential expression results.

### Usage

```
## S3 method for class 'referenceComparison'
plot(
  x,
  element = NULL,
  method = c("spearman", "pearson", "gsea", "rankProduct"),
  n = c(3, 3),
  showMetadata = TRUE,
  plotNonRankedPerturbations = FALSE,
  alpha = 0.3,
  genes = c("both", "top", "bottom"),
  ...,
  zscores = NULL,
```

```
  title = NULL
)
```

### Arguments

| | |
|---|---|
| `x` | `referenceComparison` object: obtained after running `rankSimilarPerturbations()` or `predictTargetingDrugs()` |
| `element` | Character: identifier in the first column of `x` |
| `method` | Character: method to plot results; choose between `spearman`, `pearson`, `gsea` or `rankProduct` (the last one is only available if `element = NULL`) |
| `n` | Numeric: number of top and bottom genes to label (if a vector of two numbers is given, the first and second numbers will be used as the number of top and bottom genes to label, respectively); only used if `element = NULL` |
| `showMetadata` | Boolean: show available metadata information instead of identifiers (if available)? Only used if `element = NULL` |
| `plotNonRankedPerturbations` | Boolean: plot non-ranked data in grey? Only used if `element = NULL` |
| `alpha` | Numeric: transparency; only used if `element = NULL` |
| `genes` | Character: when plotting gene set enrichment analysis (GSEA), plot most up-regulated genes (`genes = "top"`), most down-regulated genes (`genes = "bottom"`) or both (`genes = "both"`); only used if `method = "gsea"` and `geneset = NULL` |
| `...` | Extra arguments currently not used |
| `zscores` | Data frame (GCTX z-scores) or character (respective filepath to load data from file) |
| `title` | Character: plot title (if `NULL`, the default title depends on the context; ignored when plotting multiple perturbations) |

### Value

Plot illustrating the reference comparison

### See Also

Other functions related with the ranking of CMap perturbations: `as.table.referenceComparison()`, `filterCMapMetadata()`, `getCMapConditions()`, `getCMapPerturbationTypes()`, `loadCMapData()`, `loadCMapZscores()`, `parseCMapID()`, `plot.perturbationChanges()`, `plotTargetingDrugsVSsimilarPerturbations(` `prepareCMapPerturbations()`, `print.similarPerturbations()`, `rankSimilarPerturbations()`

Other functions related with the prediction of targeting drugs: `as.table.referenceComparison()`, `listExpressionDrugSensitivityAssociation()`, `loadExpressionDrugSensitivityAssociation()`, `plotTargetingDrugsVSsimilarPerturbations()`, `predictTargetingDrugs()`

### Examples

```
# Example of a differential expression profile
data("diffExprStat")
```

```
## Not run:
# Download and load CMap perturbations to compare with
cellLine <- "HepG2"
cmapMetadataKD <- filterCMapMetadata(
  "cmapMetadata.txt", cellLine=cellLine,
  perturbationType="Consensus signature from shRNAs targeting the same gene")

cmapPerturbationsKD <- prepareCMapPerturbations(
  cmapMetadataKD, "cmapZscores.gctx", "cmapGeneInfo.txt", loadZscores=TRUE)

## End(Not run)

# Rank similar CMap perturbations
compareKD <- rankSimilarPerturbations(diffExprStat, cmapPerturbationsKD)

# Plot ranked list of CMap perturbations
plot(compareKD, method="spearman")
plot(compareKD, method="spearman", n=c(7, 3))
plot(compareKD, method="pearson")
plot(compareKD, method="gsea")

# Plot results for a single perturbation
pert <- compareKD[[1, 1]]
plot(compareKD, pert, method="spearman", zscores=cmapPerturbationsKD)
plot(compareKD, pert, method="pearson", zscores=cmapPerturbationsKD)
plot(compareKD, pert, method="gsea", zscores=cmapPerturbationsKD)

# Predict targeting drugs based on a given differential expression profile
gdsc <- loadExpressionDrugSensitivityAssociation("GDSC 7")
predicted <- predictTargetingDrugs(diffExprStat, gdsc)

# Plot ranked list of targeting drugs
plot(predicted, method="spearman")
plot(predicted, method="spearman", n=c(7, 3))
plot(predicted, method="pearson")
plot(predicted, method="gsea")

# Plot results for a single targeting drug
drug <- predicted$compound[[4]]
plot(predicted, drug, method="spearman")
plot(predicted, drug, method="pearson")
plot(predicted, drug, method="gsea")
```

## `plotDrugSetEnrichment` *Plot drug set enrichment*

### Description

Plot drug set enrichment

## Usage

```
plotDrugSetEnrichment(
  sets,
  stats,
  col = "rankProduct_rank",
  selectedSets = NULL,
  keyColSets = NULL,
  keyColStats = NULL
)
```

## Arguments

| | |
|---|---|
| `sets` | Named list of characters: named sets containing compound identifiers (obtain drug sets by running `prepareDrugSets()`) |
| `stats` | Named numeric vector or either a `similarPerturbations` or a `targetingDrugs` object (obtained after running `rankSimilarPerturbations` or `predictTargetingDrugs`, respectively) |
| `col` | Character: name of the column to use for statistics (only required if class of `stats` is either `similarPerturbations` or `targetingDrugs`) |
| `selectedSets` | Character: drug sets to plot (if `NULL`, plot all) |
| `keyColSets` | Character: column from `sets` to compare with column `keyColStats` from `stats`; automatically selected if `NULL` |
| `keyColStats` | Character: column from `stats` to compare with column `keyColSets` from `sets`; automatically selected if `NULL` |

## Value

List of GSEA plots per drug set

## See Also

Other functions for drug set enrichment analysis: `analyseDrugSetEnrichment()`, `loadDrugDescriptors()`, `prepareDrugSets()`

## Examples

```
descriptors <- loadDrugDescriptors()
drugSets <- prepareDrugSets(descriptors)

# Analyse drug set enrichment in ranked targeting drugs for a differential
# expression profile
data("diffExprStat")
gdsc      <- loadExpressionDrugSensitivityAssociation("GDSC")
predicted <- predictTargetingDrugs(diffExprStat, gdsc)

plotDrugSetEnrichment(drugSets, predicted)
```

---

`plotTargetingDrugsVSsimilarPerturbations`

*Plot similar perturbations against predicted targeting drugs*

---

## Description

Plot similar perturbations against predicted targeting drugs

## Usage

```
plotTargetingDrugsVSsimilarPerturbations(
  targetingDrugs,
  similarPerturbations,
  column,
  labelBy = "pert_iname",
  quantileThreshold = 0.25,
  showAllScores = FALSE,
  keyColTargetingDrugs = NULL,
  keyColSimilarPerturbations = NULL
)
```

## Arguments

| | |
|---|---|
| `targetingDrugs` | `targetingDrugs` object |
| `similarPerturbations` | `similarPerturbations` object |
| `column` | Character: column to plot (must be available in both databases) |
| `labelBy` | Character: column in `as.table(similarPerturbations)` or `as.table(targetingDrugs)` to be used for labelling |
| `quantileThreshold` | Numeric: quantile (between 0 and 1) to highlight values of interest |
| `showAllScores` | Boolean: show all scores? If `FALSE`, only the best score per compound will be plotted |
| `keyColTargetingDrugs` | Character: column from `targetingDrugs` to compare with column `keyColSimilarPerturbations` from `similarPerturbations`; automatically selected if `NULL` |
| `keyColSimilarPerturbations` | Character: column from `similarPerturbations` to compare with column `keyColTargetingDrugs` from `targetingDrugs`; automatically selected if `NULL` |

## Value

`ggplot2` plot

### See Also

Other functions related with the ranking of CMap perturbations: `as.table.referenceComparison()`, `filterCMapMetadata()`, `getCMapConditions()`, `getCMapPerturbationTypes()`, `loadCMapData()`, `loadCMapZscores()`, `parseCMapID()`, `plot.perturbationChanges()`, `plot.referenceComparison()`, `prepareCMapPerturbations()`, `print.similarPerturbations()`, `rankSimilarPerturbations()`

Other functions related with the prediction of targeting drugs: `as.table.referenceComparison()`, `listExpressionDrugSensitivityAssociation()`, `loadExpressionDrugSensitivityAssociation()`, `plot.referenceComparison()`, `predictTargetingDrugs()`

### Examples

```
# Rank similarity against CMap compound perturbations
similarPerts <- rankSimilarPerturbations(diffExprStat,
                                         cmapPerturbationsCompounds)

# Predict targeting drugs
gdsc <- loadExpressionDrugSensitivityAssociation("GDSC 7")
predicted <- predictTargetingDrugs(diffExprStat, gdsc)

plotTargetingDrugsVSsimilarPerturbations(predicted, similarPerts,
                                         "spearman_rank")
```

---

## `predictTargetingDrugs` *Predict targeting drugs*

### Description

Identify compounds that may target the phenotype associated with a user-provided differential expression profile by comparing such against a correlation matrix of gene expression and drug sensitivity.

### Usage

```
predictTargetingDrugs(
  input,
  expressionDrugSensitivityCor,
  method = c("spearman", "pearson", "gsea"),
  geneSize = 150,
  isDrugActivityDirectlyProportionalToSensitivity = NULL,
  threads = 1,
  chunkGiB = 1,
  verbose = FALSE
)
```

## Arguments

| | |
|---|---|
| `input` | Named `numeric` `vector` of differentially expressed genes whose names are gene identifiers and respective values are a statistic that represents significance and magnitude of differentially expressed genes (e.g. t-statistics); or `character` of gene symbols composing a gene set that is tested for enrichment in reference data (only used if `method` includes `gsea`) |
| `expressionDrugSensitivityCor` | Matrix or character: correlation matrix of gene expression (rows) and drug sensitivity (columns) across cell lines or path to file containing such data; see `loadExpressionDrugSensitivityAssociation()`. |
| `method` | Character: comparison method (`spearman`, `pearson` or `gsea`; multiple methods may be selected at once) |
| `geneSize` | Numeric: number of top up-/down-regulated genes to use as gene sets to test for enrichment in reference data; if a 2-length numeric vector, the first index is the number of top up-regulated genes and the second index is the number of down-regulated genes used to create gene sets; only used if `method` includes `gsea` and if `input` is not a gene set |
| `isDrugActivityDirectlyProportionalToSensitivity` | Boolean: are the values used for drug activity directly proportional to drug sensitivity? If `NULL`, the argument `expressionDrugSensitivityCor` must have a non-`NULL` value for attribute `isDrugActivityDirectlyProportionalToSensitivity`. |
| `threads` | Integer: number of parallel threads |
| `chunkGiB` | Numeric: if second argument is a path to an HDF5 file (`.h5` extension), that file is loaded and processed in chunks of a given size in gibibytes (GiB); lower values decrease peak RAM usage (see details below) |
| `verbose` | Boolean: print additional details? |

## Value

Data table with correlation and/or GSEA score results

## Process data by chunks

If a file path to a valid HDF5 (`.h5`) file is provided instead of a data matrix, that file can be loaded and processed in chunks of size `chunkGiB`, resulting in decreased peak memory usage.

The default value of 1 GiB (1 GiB = 1024^3 bytes) allows loading chunks of ~10000 columns and 14000 rows (`10000 * 14000 * 8 bytes / 1024^3 = 1.04 GiB`).

## GSEA score

When `method = "gsea"`, weighted connectivity scores (WTCS) are calculated (`https://clue.io/connectopedia/cmap_algorithms`).

## See Also

Other functions related with the prediction of targeting drugs: `as.table.referenceComparison()`, `listExpressionDrugSensitivityAssociation()`, `loadExpressionDrugSensitivityAssociation()`, `plot.referenceComparison()`, `plotTargetingDrugsVSsimilarPerturbations()`

## Examples

```
# Example of a differential expression profile
data("diffExprStat")

# Load expression and drug sensitivity association derived from GDSC data
gdsc <- loadExpressionDrugSensitivityAssociation("GDSC 7")

# Predict targeting drugs on a differential expression profile
predictTargetingDrugs(diffExprStat, gdsc)
```

---

`prepareCMapPerturbations`

# *Prepare CMap perturbation data*

## Description

Prepare CMap perturbation data

## Usage

```
prepareCMapPerturbations(
  metadata,
  zscores,
  geneInfo,
  compoundInfo = NULL,
  ...,
  loadZscores = FALSE
)
```

## Arguments

| | |
|---|---|
| `metadata` | Data frame (CMap metadata) or character (respective filepath to load data from file) |
| `zscores` | Data frame (GCTX z-scores) or character (respective filepath to load data from file) |
| `geneInfo` | Data frame (CMap gene info) or character (respective filepath to load data from file) |
| `compoundInfo` | Data frame (CMap compound info) or character (respective filepath to load data from file) |
| `...` | Arguments passed on to `filterCMapMetadata`<br>`cellLine` Character: cell line (if `NULL`, all values are loaded)<br>`timepoint` Character: timepoint (if `NULL`, all values are loaded)<br>`dosage` Character: dosage (if `NULL`, all values are loaded)<br>`perturbationType` Character: type of perturbation (if `NULL`, all perturbation types are loaded) |

| | |
|---|---|
| `loadZscores` | Boolean: load matrix of perturbation z-scores? Not recommended in systems with less than 30GB of RAM; if `FALSE`, downstream functions will load and process the file directly chunk by chunk, resulting in a lower memory footprint |

## Value

CMap perturbation data attributes and filename

## See Also

Other functions related with the ranking of CMap perturbations: `as.table.referenceComparison()`, `filterCMapMetadata()`, `getCMapConditions()`, `getCMapPerturbationTypes()`, `loadCMapData()`, `loadCMapZscores()`, `parseCMapID()`, `plot.perturbationChanges()`, `plot.referenceComparison()`, `plotTargetingDrugsVSsimilarPerturbations()`, `print.similarPerturbations()`, `rankSimilarPerturbations()`

## Examples

```
metadata <- loadCMapData("cmapMetadata.txt", "metadata")
metadata <- filterCMapMetadata(metadata, cellLine="HepG2")
## Not run:
prepareCMapPerturbations(metadata, "cmapZscores.gctx", "cmapGeneInfo.txt")

## End(Not run)
```

---

# `prepareDrugSets` *Prepare drug sets from a table with compound descriptors*

## Description

Create a list of drug sets for each character and numeric column. For each character column, drugs are split across that column's unique values (see argument `maxUniqueElems`). For each numeric column, drugs are split across evenly-distributed bins.

## Usage

```
prepareDrugSets(
  table,
  id = 1,
  maxUniqueElems = 15,
  maxBins = 15,
  k = 5,
  minPoints = NULL
)
```

### Arguments

| | |
|---|---|
| `table` | Data frame: drug descriptors |
| `id` | Integer or character: index or name of the identifier column |
| `maxUniqueElems` | Numeric: ignore character columns with more unique elements than `maxUniqueElems` |
| `maxBins` | Numeric: maximum number of bins for numeric columns |
| `k` | Numeric: constant; the higher the constant, the smaller the bin size (check `minpts`) |
| `minPoints` | Numeric: minimum number of points in a bin (if `NULL`, the minimum number of points is the number of non-missing values divided by `maxBins` divided by `k`) |

### Value

Named list of characters: named drug sets with respective compound identifiers as list elements

### See Also

Other functions for drug set enrichment analysis: `analyseDrugSetEnrichment()`, `loadDrugDescriptors()`, `plotDrugSetEnrichment()`

### Examples

```
descriptors <- loadDrugDescriptors("NCI60")
prepareDrugSets(descriptors)
```

---

## `prepareENCODEgeneExpression`

*Load ENCODE gene expression data*

### Description

Load ENCODE gene expression data

### Usage

```
prepareENCODEgeneExpression(samples)
```

### Arguments

| | |
|---|---|
| `samples` | List of loaded ENCODE samples |

### Value

Data frame containing gene read counts

### See Also

`convertGeneIdentifiers()`

Other functions related with using ENCODE expression data: `downloadENCODEknockdownMetadata()`, `loadENCODEsamples()`, `performDifferentialExpression()`

### Examples

```
if (interactive()) {
  # Load ENCODE metadata for a specific cell line and gene
  cellLine <- "HepG2"
  gene <- "EIF4G1"
  ENCODEmetadata <- downloadENCODEknockdownMetadata(cellLine, gene)

  # Load samples based on filtered ENCODE metadata
  ENCODEsamples <- loadENCODEsamples(ENCODEmetadata)[[1]]

  prepareENCODEgeneExpression(ENCODEsamples)
}
```

---

## `print.similarPerturbations`

## *Print a* `similarPerturbations` *object*

### Description

Print a `similarPerturbations` object

### Usage

```
## S3 method for class 'similarPerturbations'
print(x, perturbation = NULL, ...)
```

### Arguments

| | |
|---|---|
| `x` | `similarPerturbations` object |
| `perturbation` | Character (perturbation identifier) or numeric (perturbation index) |
| `...` | Extra parameters passed to `print` |

### Value

Information on `perturbationChanges` object or on specific perturbations

### See Also

Other functions related with the ranking of CMap perturbations: `as.table.referenceComparison()`, `filterCMapMetadata()`, `getCMapConditions()`, `getCMapPerturbationTypes()`, `loadCMapData()`, `loadCMapZscores()`, `parseCMapID()`, `plot.perturbationChanges()`, `plot.referenceComparison()`, `plotTargetingDrugsVSsimilarPerturbations()`, `prepareCMapPerturbations()`, `rankSimilarPerturbations()`

---

## `rankSimilarPerturbations` *Rank differential expression profile against CMap perturbations by similarity*

---

### Description

Compare differential expression results against CMap perturbations.

### Usage

```
rankSimilarPerturbations(
  input,
  perturbations,
  method = c("spearman", "pearson", "gsea"),
  geneSize = 150,
  cellLineMean = "auto",
  rankPerCellLine = FALSE,
  threads = 1,
  chunkGiB = 1,
  verbose = FALSE
)
```

### Arguments

| | |
|---|---|
| `input` | Named `numeric vector` of differentially expressed genes whose names are gene identifiers and respective values are a statistic that represents significance and magnitude of differentially expressed genes (e.g. t-statistics); or `character` of gene symbols composing a gene set that is tested for enrichment in reference data (only used if `method` includes `gsea`) |
| `perturbations` | `perturbationChanges` object: CMap perturbations (check `prepareCMapPerturbations()`) |
| `method` | Character: comparison method (`spearman`, `pearson` or `gsea`; multiple methods may be selected at once) |
| `geneSize` | Numeric: number of top up-/down-regulated genes to use as gene sets to test for enrichment in reference data; if a 2-length numeric vector, the first index is the number of top up-regulated genes and the second index is the number of down-regulated genes used to create gene sets; only used if `method` includes `gsea` and if `input` is not a gene set |
| `cellLineMean` | Boolean: add rows with the mean of `method` across cell lines? If `cellLineMean = "auto"` (default), rows will be added when data for more than one cell line is available. |
| `rankPerCellLine` | Boolean: rank results based on both individual cell lines and mean scores across cell lines (`TRUE`) or based on mean scores alone (`FALSE`)? If `cellLineMean = FALSE`, individual cell line conditions are always ranked. |

| | |
|---|---|
| `threads` | Integer: number of parallel threads |
| `chunkGiB` | Numeric: if second argument is a path to an HDF5 file (`.h5` extension), that file is loaded and processed in chunks of a given size in gibibytes (GiB); lower values decrease peak RAM usage (see details below) |
| `verbose` | Boolean: print additional details? |

## Value

Data table with correlation and/or GSEA score results

## Process data by chunks

If a file path to a valid HDF5 (`.h5`) file is provided instead of a data matrix, that file can be loaded and processed in chunks of size `chunkGiB`, resulting in decreased peak memory usage.

The default value of 1 GiB (1 GiB = 1024^3 bytes) allows loading chunks of ~10000 columns and 14000 rows (`10000 * 14000 * 8 bytes / 1024^3 = 1.04 GiB`).

## GSEA score

When `method = "gsea"`, weighted connectivity scores (WTCS) are calculated (`https://clue.io/connectopedia/cmap_algorithms`).

## See Also

Other functions related with the ranking of CMap perturbations: `as.table.referenceComparison()`, `filterCMapMetadata()`, `getCMapConditions()`, `getCMapPerturbationTypes()`, `loadCMapData()`, `loadCMapZscores()`, `parseCMapID()`, `plot.perturbationChanges()`, `plot.referenceComparison()`, `plotTargetingDrugsVSsimilarPerturbations()`, `prepareCMapPerturbations()`, `print.similarPerturbations()`

## Examples

```
# Example of a differential expression profile
data("diffExprStat")

## Not run:
# Download and load CMap perturbations to compare with
cellLine <- c("HepG2", "HUH7")
cmapMetadataCompounds <- filterCMapMetadata(
    "cmapMetadata.txt", cellLine=cellLine, timepoint="24 h",
    dosage="5 \u00B5M", perturbationType="Compound")

cmapPerturbationsCompounds <- prepareCMapPerturbations(
    cmapMetadataCompounds, "cmapZscores.gctx", "cmapGeneInfo.txt",
    "cmapCompoundInfo_drugs.txt", loadZscores=TRUE)

## End(Not run)
perturbations <- cmapPerturbationsCompounds

# Rank similar CMap perturbations (by default, Spearman's and Pearson's
# correlation are used, as well as GSEA with the top and bottom 150 genes of
```

```
# the differential expression profile used as reference)
rankSimilarPerturbations(diffExprStat, perturbations)

# Rank similar CMap perturbations using only Spearman's correlation
rankSimilarPerturbations(diffExprStat, perturbations, method="spearman")
```

# Index