Package 'SingleR'

April 15, 2020

Title Reference-Based Single-Cell RNA-Seq Annotation

Version 1.0.6

Date 2020-04-08

Description Performs unbiased cell type recognition from single-cell RNA sequencing data, by leveraging reference transcriptomic datasets of pure cell types to infer the cell of origin of each single cell independently.

License GPL-3 + file LICENSE

Depends SummarizedExperiment

Imports methods, Matrix, S4Vectors, DelayedArray, DelayedMatrixStats, BiocNeighbors, BiocParallel, stats, utils, Rcpp, ExperimentHub

LinkingTo Rcpp, beachmat

Suggests testthat, knitr, rmarkdown, BiocStyle, beachmat, SingleCellExperiment, scater, scRNAseq, scran, BiocGenerics, ggplot2, pheatmap, grDevices, viridis, AnnotationHub, AnnotationDbi

biocViews Software, SingleCell, GeneExpression, Transcriptomics, Classification, Clustering, Annotation

SystemRequirements C++11

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 7.0.2

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

git_branch RELEASE_3_10

git_last_commit 5803a1f

git_last_commit_date 2020-04-08

Date/Publication 2020-04-14

Author Dvir Aran [aut, cph],
Aaron Lun [ctb, cre],
Daniel Bunis [ctb],
Jared Andrews [ctb],
Friederike Dündar [ctb]

Maintainer Aaron Lun <infinite.monkeys.with.keyboards@gmail.com>

R topics documented:

`aggregateReference` *Aggregate reference samples*

Description

Aggregate reference samples for a given label by averaging their count profiles. This can be done with varying degrees of resolution to preserve the within-label heterogeneity.

Usage

```
aggregateReference(
  ref,
  labels,
  power = 0.5,
  assay.type = "logcounts",
  check.missing = TRUE,
  BPPARAM = SerialParam()
)
```

Arguments

`ref`	A numeric matrix of reference expression values, usually containing log-expression values. Alternatively, a SummarizedExperiment object containing such a matrix.
`labels`	A character vector or factor of known labels for all cells in `ref`.
`power`	Numeric scalar between 0 and 1 indicating how much aggregation should be performed, see Details.
`assay.type`	An integer scalar or string specifying the assay of `ref` containing the relevant expression matrix, if `ref` is a SummarizedExperiment object.

check.missing — Logical scalar indicating whether rows should be checked for missing values (and if found, removed).

BPPARAM — A BiocParallelParam object indicating how parallelization should be performed. Only used if ref is or contains a DelayedMatrix.

Details

With single-cell reference datasets, it is often useful to aggregate individual cells into pseudo-bulk samples to serve as a reference. This improves speed (and to some extent, reduces noise) in downstream assignment with classifySingleR or SingleR.

The most obvious aggregation is to simply average all counts for all cells in a label to obtain a single pseudo-bulk profile. This can be achieved by setting power=0. However, this discards information about the within-label heterogeneity (e.g., the "shape" and spread of the population in expression space) that may be informative for assignment, especially for closely related labels.

Instead, the default approach in this function is to create a series of pseudo-bulk samples to represent each label. This is achieved by performing vector quantization using k-means clustering on all cells in a particular label. Cells in each cluster are subsequently averaged to create one pseudo-bulk sample. We set the number of clusters to be ncol(ref)^power so that labels with more cells have more resolved representatives.

If power=1, no aggregation is performed.

We use the average rather than the sum in order to be compatible with trainSingleR's internal marker detection. Moreover, unlike counts, the sum of transformed and normalized expression values generally has little meaning. We do not use the median to avoid consistently obtaining zeros for lowly expressed genes.

Value

A SummarizedExperiment object with a "logcounts" assay containing a matrix of aggregated expression values, and a label column metadata field specifying the label corresponding to each column.

Author(s)

Aaron Lun

Examples

```
library(scater)
sce <- mockSCE()
sce <- logNormCounts(sce)

# Making up some labels for demonstration purposes:
labels <- sample(LETTERS, ncol(sce), replace=TRUE)

# Aggregation at different resolutions:
(aggr <- aggregateReference(sce, labels, power=0.5))

(aggr <- aggregateReference(sce, labels, power=0))

# No aggregation:
(aggr <- aggregateReference(sce, labels, power=1))
```

`BlueprintEncodeData` *Obtain human bulk RNA-seq data from Blueprint and ENCODE*

Description

Download and cache the normalized expression values of 259 RNA-seq samples of pure stroma and immune cells as generated and supplied by Blueprint and ENCODE.

Usage

```
BlueprintEncodeData(rm.NA = c("rows", "cols", "both", "none"), ensembl = FALSE)
```

Arguments

`rm.NA`	String specifying how missing values should be handled. `"rows"` will remove genes with at least one missing value, `"cols"` will remove samples with at least one missing value, `"both"` will remove any gene or sample with at least one missing value, and `"none"` will not perform any removal.
`ensembl`	Logical scalar indicating whether to convert row names to Ensembl IDs. Genes without a mapping to a non-duplicated Ensembl ID are discarded.

Details

This function provides normalized expression values of 259 bulk RNA-seq samples generated by Blueprint and ENCODE from pure populations of stroma and immune cells (Martens and Stunnenberg, 2013; The ENCODE Consortium, 2012).

Blueprint Epigenomics contains 144 RNA-seq pure immune samples annotated to 28 cell types. ENCODE contains 115 RNA-seq pure stroma and immune samples annotated to 17 cell types. Altogether, this reference contains 259 samples with 43 cell types.

The samples were processed and normalized as described in Aran, Looney and Liu et al. (2019); i.e. the raw RNA-seq counts were downloaded from Blueprint and ENCODE in 2016 and normalized via edgeR (TPMs).

The data will be downloaded from ExperimentHub, returning a SummarizedExperiment object for further use.

Value

A SummarizedExperiment object with a `"logcounts"` assay containing the log-normalized expression values, along with cell type labels in the `colData`.

Author(s)

Friederike Dündar

References

The ENCODE Project Consortium (2012). An integrated encyclopedia of DNA elements in the human genome. *Nature* 489, pages 57–74.

Martens JHA and Stunnenberg HG (2013). BLUEPRINT: mapping human blood cell epigenomes. *Haematologica* 98, 1487–1489.

Aran D, Looney AP, Liu L et al. (2019). Reference-based analysis of lung single-cell sequencing reveals a transitional profibrotic macrophage. *Nat. Immunol.* 20, 163–172.

Examples

```
ref.se <- BlueprintEncodeData(rm.NA = "rows")
```

`classifySingleR` *Classify cells with SingleR*

Description

Assign labels to each cell in a test dataset, using a pre-trained classifier combined with an iterative fine-tuning approach.

Usage

```
classifySingleR(
  test,
  trained,
  quantile = 0.8,
  fine.tune = TRUE,
  tune.thresh = 0.05,
  sd.thresh = NULL,
  prune = TRUE,
  assay.type = "logcounts",
  check.missing = TRUE,
  BPPARAM = SerialParam()
)
```

Arguments

`test`	A numeric matrix of single-cell expression values where rows are genes and columns are cells. Alternatively, a SummarizedExperiment object containing such a matrix.
`trained`	A List containing the output of the `trainSingleR` function. Alternatively, a List of Lists produced by `trainSingleR` for multiple references.
`quantile`	A numeric scalar specifying the quantile of the correlation distribution to use to compute the score for each label.
`fine.tune`	A logical scalar indicating whether fine-tuning should be performed.
`tune.thresh`	A numeric scalar specifying the maximum difference from the maximum correlation to use in fine-tuning.
`sd.thresh`	A numeric scalar specifying the threshold on the standard deviation, for use in gene selection during fine-tuning. This is only used if `genes="sd"` when constructing `trained` and defaults to the value used in `trainSingleR`.
`prune`	A logical scalar indicating whether label pruning should be performed.
`assay.type`	Integer scalar or string specifying the matrix of expression values to use if `test` is a SummarizedExperiment.

check.missing	Logical scalar indicating whether rows should be checked for missing values (and if found, removed).
BPPARAM	A BiocParallelParam object specifyign the parallelization scheme to use.

Details

Consider each cell in the test set test and each label in the training set. We compute Spearman's rank correlations between the test cell and all cells in the training set with the given label, based on the expression profiles of the genes selected by trained. The score is defined as the quantile of the distribution of correlations, as specified by quantile. (Technically, we avoid explicitly computing all correlations by using a nearest neighbor search, but the resulting score is the same.) After repeating this across all labels, the label with the highest score is used as the prediction for that cell.

If fine.tune=TRUE, an additional fine-tuning step is performed for each cell to improve resolution. We identify all labels with scores that are no more than tune.thresh below the maximum score. These labels are used to identify a fresh set of marker genes, and the calculation of the score is repeated using only these genes. The aim is to refine the choice of markers and reduce noise when distinguishing between closely related labels. The best and next-best scores are reported in the output for use in diagnostics, e.g., pruneScores.

The default assay.type is set to "logcounts" simply for consistency with trainSingleR. In practice, the raw counts (for UMI data) or the transcript counts (for read count data) can also be used without normalization and log-transformation. Any monotonic transformation will have no effect the calculation of the correlation values other than for some minor differences due to numerical precision.

If prune=TRUE, label pruning is performed as described in pruneScores with default arguments. This aims to remove low-quality labels that are ambiguous or correspond to misassigned cells. However, the default settings can be somewhat aggressive and discard otherwise useful labels in some cases - see ?pruneScores for details.

If trained was generated from multiple references, combineResults is used to consolidate the per-reference statistics into a single DataFrame of results.

Value

A DataFrame where each row corresponds to a cell in test. If fine.tune=FALSE, fields are:

- scores, a numeric matrix of correlations at the specified quantile for each label (column) in each cell (row).
- labels, a character vector containing the predicted label based on the maximum entry in scores.
- pruned.labels, a character vector containing the pruned labels where "low-quality" labels are replaced with NAs. Only added if prune=TRUE.

If fine.tune=TRUE, fields are:

- scores, a numeric matrix of correlations as above.
- first.labels, a character vector containing the predicted label *before* fine-tuning.
- tuned.scores, a DataFrame containing first and second. These are numeric vectors containing the best and next-best scores at the final round of fine-tuning for each cell.
- labels, a character vector containing the predicted label after fine-tuning.
- pruned.labels, a character vector of pruned labels as above. Only added if prune=TRUE.

In both cases, the metadata of the DataFrame contains:

- common.genes, a character vector of genes used to compute the correlations prior to fine-tuning.
- de.genes, a list of list of genes used to distinguish between each pair of labels. Only returned if genes="de" when constructing trained, see ?trainSingleR for more details.

Author(s)

Aaron Lun, based on the original SingleR code by Dvir Aran.

See Also

trainSingleR, to prepare the training set for classification.

pruneScores, to remove low-quality labels based on the scores.

combineResults, to combine results from multiple references.

Examples

```
##############################
## Mocking up training data ##
##############################

Ngroups <- 5
Ngenes <- 1000
means <- matrix(rnorm(Ngenes*Ngroups), nrow=Ngenes)
means[1:900,] <- 0
colnames(means) <- LETTERS[1:5]

g <- rep(LETTERS[1:5], each=4)
ref <- SummarizedExperiment(
    list(counts=matrix(rpois(1000*length(g),
        lambda=10*2^means[,g]), ncol=length(g))),
    colData=DataFrame(label=g)
)
rownames(ref) <- sprintf("GENE_%s", seq_len(nrow(ref)))

ref <- scater::logNormCounts(ref)
trained <- trainSingleR(ref, ref$label)

##############################
## Mocking up some test data ##
##############################

N <- 100
g <- sample(LETTERS[1:5], N, replace=TRUE)
test <- SummarizedExperiment(
    list(counts=matrix(rpois(1000*N, lambda=2^means[,g]), ncol=N)),
    colData=DataFrame(cluster=g)
)

rownames(test) <- sprintf("GENE_%s", seq_len(nrow(test)))
test <- scater::logNormCounts(test)

##############################
## Performing classification ##
##############################
```

```
pred <- classifySingleR(test, trained)
table(predicted=pred$labels, truth=g)
```

`combineResults` *Combine SingleR results*

Description

Combine results from multiple runs of `classifySingleR` (usually against different references) into a single DataFrame. The label from the results with the highest score for each cell is retained.

Usage

```
combineResults(results)
```

Arguments

`results`	A list of DataFrame prediction results as returned by `classifySingleR` when run on each reference separately.

Details

Labels are combined across `results` based on the highest score in each reference. Each result should be generated from training sets that use a common set of genes during classification, i.e., `common.genes` should be the same in the `trained` argument to each `classifySingleR` call. This is because the scores are not comparable across results if they were generated from different sets of genes.

It is unlikely that this method will be called directly by the end-user. Users are advised to use the multi-reference mode of `SingleR`, `trainSingleR` and/or `classifySingleR`, which will take care of the use of a common set of genes before calling this function to combine results across references.

If this function must be called manually, users should ensure that `common.genes` is the same for all calls used to generate `results`. This is most easily achieved by calling `trainSingleR` on each reference; replacing each `common.genes` with the union of all `common.genes`; and then calling `classifySingleR` on the test with the modified training objects. The resulting DataFrames can then be passed as `results` above.

Value

A DataFrame is returned containing the annotation statistics for each cell or cluster (row). This has the same fields as the output of `classifySingleR`, where the `scores` are combined across all `results`. The set of labels for each cell are those from the DataFrame with the largest maximum score. The original results are available in the `orig.results` field.

Method rationale

There are three obvious options for combining reference datasets or classification results stemming from disparate references:

Option 1 would be to combine the reference datasets into a single matrix and treat each label as though it is specific to the reference from which it originated (e.g. Ref1-Bcell vs Ref2-Bcell), which is easily accomplished by `paste`ing the reference name onto the corresponding set of labels. This option avoids the need for time-consuming label harmonization between references, and may be the best approach if the differences between the reference sets are important (e.g., different experimental conditions).

That said, the fact that we are comparing across references means that the marker set is likely to contain genes responsible for uninteresting batch effects. This will increase noise during the calculation of the score in each reference, possibly leading to a loss of precision and a greater risk of technical variation dominating the classification results.

Option 2 would also involve combining the reference datasets into a single matrix but would harmonize the labels so that the same cell type is given the same label across references. This would allow feature selection methods to identify robust sets of label-specific markers that are more likely to generalize to other datasets. It would also simplify interpretation, as there is no need to worry about the reference from which the labels came.

The main obstacle to this method is the diffculty of harmonization. Putting aside trivial differences in naming schemes (e.g. `"B cell"` vs `"B"`), we must resolve additional challenges like differences in label resolution across references (e.g., how to harmonize `"B cell"` to another reference that splits to `"naive B cell"` and `"mature B cell"`), different sorting strategies for obtaining pure cell types, or other subtle biological differences that require domain expertise.

Option 3 is the method that is implemented in this function. It involves performing classification separately within each reference, then collating the results to choose the label with the highest score across references. This is a relatively expedient approach that avoids the need for explicit harmonization while also reduces the potential for reference-specific markers.

It leaves a mixture of labels in the final results that is up to the user to resolve, though perhaps this may be considered a feature as it smoothly handles differences in resolution between references, e.g., a cell that cannot be resolved as a CD4+ or CD8+ T cell may simply fall back to `"T cell"`. It will also be somewhat suboptimal if there are many reference-specific labels, as markers are not identified with the aim of distinguishing a label in one reference from another label in another reference.

Author(s)

Jared Andrews

See Also

`matchReferences`, to harmonize labels between reference datasets.

`SingleR` and `classifySingleR`, for generating predictions to use in `results`.

Examples

```
##############################
## Mocking up training data ##
##############################

Ngroups <- 5
Ngenes <- 1000
```

```
means <- matrix(rnorm(Ngenes*Ngroups), nrow=Ngenes)
means[1:900,] <- 0
colnames(means) <- LETTERS[1:5]

g <- rep(LETTERS[1:5], each=4)
g2 <- rep(LETTERS[6:10], each=4)
ref1 <- SummarizedExperiment(
    list(counts=matrix(rpois(1000*length(g),
    lambda=10*2^means[,g]), ncol=length(g))),
    colData=DataFrame(label=g)
)
ref2 <- SummarizedExperiment(
    list(counts=matrix(rpois(1000*length(g2),
    lambda=10*2^means[,g]), ncol=length(g2))),
    colData=DataFrame(label=g2)
)
rownames(ref1) <- sprintf("GENE_%s", seq_len(nrow(ref1)))
rownames(ref2) <- sprintf("GENE_%s", seq_len(nrow(ref2)))

ref1 <- scater::logNormCounts(ref1)
ref2 <- scater::logNormCounts(ref2)

################################
## Mocking up some test data ##
################################

N <- 100
g <- sample(LETTERS[1:5], N, replace=TRUE)
means <- matrix(rnorm(Ngenes*Ngroups), nrow=Ngenes)
means[1:900] <- 0
colnames(means) <- LETTERS[1:5]
test <- SummarizedExperiment(
    list(counts=matrix(rpois(1000*N, lambda=2^means[,g]), ncol=N)),
    colData=DataFrame(label=g)
)

rownames(test) <- sprintf("GENE_%s", seq_len(nrow(test)))
test <- scater::logNormCounts(test)

################################
## Performing classification ##
################################

pred1 <- SingleR(test, ref1, labels=ref1$label)
pred2 <- SingleR(test, ref2, labels=ref2$label)

pred3 <- SingleR(test, ref1, labels=ref1$label,
    method="cluster", clusters=test$label)
pred4 <- SingleR(test, ref2, labels=ref2$label,
    method="cluster", clusters=test$label)

################################
##     Combining results     ##
################################

pred.single <- combineResults(list("pred1" = pred1, "pred2" = pred2))
pred.clust <- combineResults(list("pred3" = pred3, "pred4" = pred4))
```

`DatabaseImmuneCellExpressionData` *Obtain human bulk RNA-seq data from DICE*

Description

Download and cache the normalized expression values of 1561 bulk RNA-seq samples of sorted cell populations from the Database of Immune Cell Expression (DICE).

Usage

```
DatabaseImmuneCellExpressionData(ensembl = FALSE)
```

Arguments

`ensembl`	Logical scalar indicating whether to convert row names to Ensembl IDs. Genes without a mapping to a non-duplicated Ensembl ID are discarded.

Details

This function provides normalized expression values of 1561 bulk RNA-seq samples generated by DICE from pure populations of human immune cells.

TPM normalized values for each cell type were downloaded from https://dice-database.org/downloads. Genes with no reads across samples were removed, and values were log2 normalized after a pseudocount of 1 was added.

The dataset contains 1561 human RNA-seq samples annotated to 5 main cell types:

- B cells
- Monocytes
- NK cells
- T cells, CD8+
- T cells, CD4+

Samples were additionally annotated to 15 fine cell types:

- B cells, naive
- Monocytes, CD14+
- Monocytes, CD16+
- NK cells
- T cells, memory TREG
- T cells, CD4+, naive
- T cells, CD4+, naive, stimulated
- T cells, CD4+, naive Treg
- T cells, CD4+, Th1
- T cells, CD4+, Th1_17

- T cells, CD4+, Th2
- T cells, CD8+, naïve
- T cells, CD8+, naïve, stimulated
- T cells, CD4+, TFH
- T cells, CD4+, Th17

Value

A SummarizedExperiment object with a `"logcounts"` assay containing the log-normalized expression values, along with cell type labels in the `colData`.

Author(s)

Jared Andrews

References

Schmiedel B et al. (2018). Impact of Genetic Polymorphisms on Human Immune Cell Gene Expression. *Cell* 175, 1701-1715.

Examples

```
ref.se <- DatabaseImmuneCellExpressionData()
```

`getDeltaFromMedian` *Compute the difference from median*

Description

Compute the delta value for each cell, defined as the difference between the score for the assigned label and the and median score across all labels.

Usage

```
getDeltaFromMedian(results)
```

Arguments

`results`	A DataFrame containing the output generated by `SingleR` or `classifySingleR`.

Details

This funciton computes the same delta value that is used in `pruneScores`, for users who want to apply more custom filters or visualizations.

Value

A numeric vector containing delta values for each cell in `results`.

Author(s)

Aaron Lun

See Also

pruneScores, where the delta values are used.

Examples

```
# Running the SingleR() example.
example(SingleR, echo=FALSE)

summary(getDeltaFromMedian(pred))
```

HumanPrimaryCellAtlasData *Obtain the HPCA data*

Description

Download and cache the normalized expression values of the data stored in the Human Primary Cell Atlas. The data will be downloaded from ExperimentHub, returning a SummarizedExperiment object for further use.

Usage

```
HumanPrimaryCellAtlasData(ensembl = FALSE)
```

Arguments

ensembl	Logical scalar indicating whether to convert row names to Ensembl IDs. Genes without a mapping to a non-duplicated Ensembl ID are discarded.

Details

This function provides normalized expression values 713 microarray samples of the Human Primary Cell Atlas (HPCA) (Mabbott et al., 2013). These 713 samples were processed and normalized as described in Aran, Looney and Liu et al. (2019) and each sample has been assigned to one of 37 main cell types and 157 subtypes.

Value

A SummarizedExperiment object with a "logcounts" assay containing the log-normalized expression values, along with cell type labels in the colData.

Author(s)

Friederike Dündar

References

Mabbott NA et al. (2013). An expression atlas of human primary cells: inference of gene function from coexpression networks. *BMC Genomics* 14, Article 632.

Aran D, Looney AP, Liu L et al. (2019). Reference-based analysis of lung single-cell sequencing reveals a transitional profibrotic macrophage. *Nat. Immunol.* 20, 163–172.

Examples

```
ref.se <- HumanPrimaryCellAtlasData()
```

`ImmGenData` *Obtain mouse bulk expression data from the Immunologic Genome Project*

Description

Download and cache the normalized expression values of 830 microarray samples of pure mouse immune cells, generated by the Immunologic Genome Project (ImmGen).

Usage

```
ImmGenData(ensembl = FALSE)
```

Arguments

`ensembl`	Logical scalar indicating whether to convert row names to Ensembl IDs. Genes without a mapping to a non-duplicated Ensembl ID are discarded.

Details

This function provides normalized expression values of 830 microarray samples generated by ImmGen from pure populations of murine immune cells (<http://www.immgen.org/>).

The samples were processed and normalized as described in Aran, Looney and Liu et al. (2019); i.e. CEL files from the Gene Expression Omnibus (GEO; GSE15907 and GSE37448), were downloaded, processed, and normalized using the robust multi-array average (RMA) procedure on probe-level data (with Matlab functions).

The data will be downloaded from ExperimentHub, returning a SummarizedExperiment object for further use.

Value

A SummarizedExperiment object with a `"logcounts"` assay containing the log-normalized expression values, along with cell type labels in the `colData`.

Author(s)

Friederike Dündar

References

Heng TS, Painter MW, Immunological Genome Project Consortium (2008). The Immunological Genome Project: networks of gene expression in immune cells. *Nat. Immunol.* 9, 1091-1094.

Aran D, Looney AP, Liu L et al. (2019). Reference-based analysis of lung single-cell sequencing reveals a transitional profibrotic macrophage. *Nat. Immunol.* 20, 163–172.

Examples

```
ref.se <- ImmGenData()
```

`matchReferences` *Match labels from two references*

Description

Match labels from a pair of references, corresponding to the same underlying cell type or state but with differences in nomenclature.

Usage

```
matchReferences(ref1, ref2, labels1, labels2, ...)
```

Arguments

`ref1, ref2`	Numeric matrices of single-cell (usually log-transformed) expression values where rows are genes and columns are cells. Alternatively, SummarizedExperiment objects containing such matrices.
`labels1, labels2`	A character vector or factor of known labels for all cells in `ref1` and `ref2`, respectively.
`...`	Further arguments to pass to `SingleR`.

Details

It is often the case that two references contain the same cell types for the same biological system, but the two sets of labels differ in their nomenclature. This makes it difficult to compare results from different references. It also interferes with attempts to combine multiple datasets to create a larger, more comprehensive reference.

The `matchReferences` function attempts to facilitate matching of labels across two reference datasets. It does so by using one of the references (say, `ref1`) to assign its labels to the other (`ref2`). For each label X in `labels2`, we compute the probability of assigning a sample of X to each label Y in `labels1`. We also use `ref2` to assign labels to `ref1`, to obtain the probability of assigning a sample of Y to label X.

We then consider the probability of mutual assignment, i.e., assigning a sample of X to Y *and* a sample of Y to X. This is computed by simply taking the product of the two probabilities mentioned earlier. The output matrix contains mutual assignment probabilities for all pairs of X (rows) and Y (columns).

The mutual assignment probabilities are only high if there is a 1:1 mapping between labels. A perfect mapping manifests as probabilities of 1 in the relevant entries of the output matrix. Lower

values are expected for ambiguous mappings and near-zero values for labels that are specific to one reference.

Value

A numeric matrix containing a probability table of mutual assignment. Values close to 1 represent a 1:1 mapping between labels across the two references.

Author(s)

Aaron Lun

See Also

SingleR, to do the actual cross-assignment.

Examples

```
example(SingleR, echo=FALSE)
test$label <- paste0(test$label, "_X") # modifying the labels.
matchReferences(test, ref, labels1=test$label, labels2=ref$label)
```

MonacoImmuneData *Obtain bulk RNA-seq data of sorted human immune cells*

Description

Download and cache the normalized expression values of 114 bulk RNA-seq samples of sorted immune cell populations that can be found in GSE107011.

Usage

```
MonacoImmuneData(ensembl = FALSE)
```

Arguments

ensembl	Logical scalar indicating whether to convert row names to Ensembl IDs. Genes without a mapping to a non-duplicated Ensembl ID are discarded.

Details

The dataset contains 114 human RNA-seq samples annotated to 10 main cell types:

- CD8+ T cells
- T cells
- CD4+ T cells
- Progenitors
- B cells
- Monocytes
- NK cells
- Dendritic cells

- Neutrophils
- Basophils

Samples were additionally annotated to 29 fine cell types:

- Naive CD8 T cells
- Central memory CD8 T cells
- Effector memory CD8 T cells
- Terminal effector CD8 T cells
- MAIT cells
- Vd2 gd T cells
- Non-Vd2 gd T cells
- Follicular helper T cells
- T regulatory cells
- Th1 cells
- Th1/Th17 cells
- Th17 cells
- Th2 cells
- Naive CD4 T cells
- Terminal effector CD4 T cells
- Progenitor cells
- Naive B cells
- Non-switched memory B cells
- Exhausted B cells
- Switched memory B cells
- Plasmablasts
- Classical monocytes
- Intermediate monocytes
- Non classical monocytes
- Natural killer cells
- Plasmacytoid dendritic cells
- Myeloid dendritic cells
- Low-density neutrophils
- Low-density basophils

Value

A SummarizedExperiment object with a `"logcounts"` assay containing the log-normalized expression values, along with cell type labels in the `colData`.

Author(s)

Jared Andrews

References

Monaco G et al. (2019). RNA-Seq Signatures Normalized by mRNA Abundance Allow Absolute Deconvolution of Human Immune Cell Types *Cell Rep.* 26, 1627-1640.

Examples

```
ref.se <- MonacoImmuneData()
```

`MouseRNAseqData` *Obtain mouse bulk expression data of sorted cell populations (RNA-seq)*

Description

Download and cache the normalized expression values of 358 bulk RNA-seq samples of sorted cell populations that can be found at GEO.

Usage

```
MouseRNAseqData(ensembl = FALSE)
```

Arguments

`ensembl`	Logical scalar indicating whether to convert row names to Ensembl IDs. Genes without a mapping to a non-duplicated Ensembl ID are discarded.

Details

This dataset was contributed by the Benayoun Lab that identified, downloaded and processed data sets on GEO that corresponded to sorted cell types (Benayoun et al., 2019).

The dataset contains 358 mouse RNA-seq samples annotated to 18 main cell types:

- Adipocytes
- Astrocytes
- B cells
- Cardiomyocytes
- Dendritic cells
- Endothelial cells
- Epithelial cells
- Erythrocytes
- Fibroblasts
- Granulocytes
- Hepatocytes
- Macrophages
- Microglia
- Monocytes

- Neurons
- NK cells
- Oligodendrocytes
- T cells

Value

A SummarizedExperiment object with a "logcounts" assay containing the log-normalized expression values, along with cell type labels in the colData.

Author(s)

Friederike Dündar

References

Benayoun B et al. (2019). Remodeling of epigenome and transcriptome landscapes with aging in mice reveals widespread induction of inflammatory responses. *Genome Res.* 29, 697-709.

Code at https://github.com/BenayounLaboratory/Mouse_Aging_Epigenomics_2018/tree/master/FigureS7_CIBERSORT/RNAseq_datasets_for_Deconvolution/2017-01-18

Examples

```
ref.se <- MouseRNAseqData()
```

NovershternHematopoieticData

Obtain bulk microarray expression for sorted hematopoietic cells

---

Description

Download and cache the normalized expression values of 211 bulk human microarray samples of sorted hematopoietic cell populations that can be found in GSE24759.

Usage

```
NovershternHematopoieticData(ensembl = FALSE)
```

Arguments

ensembl	Logical scalar indicating whether to convert row names to Ensembl IDs. Genes without a mapping to a non-duplicated Ensembl ID are discarded.

Details

The dataset contains 211 human microarray samples annotated to 16 main cell types:

- Basophils
- B cells
- CMPs
- Dendritic cells
- Eosinophils
- Erythroid cells
- GMPS
- Granulocytes
- HSCs
- Megakaryocytes
- MEPs
- Monocytes
- NK cells
- NK T cells
- CD8+ T cells
- CD4+ T cells

Samples were additionally annotated to 38 fine cell types:

- Basophils
- Naive B cells
- Mature B cells class able to switch
- Mature B cells
- Mature B cells class switched
- Common myeloid progenitors
- Plasmacytoid Dendritic Cells
- Myeloid Dendritic Cells
- Eosinophils
- Erythroid_CD34+ CD71+ GlyA-
- Erythroid_CD34- CD71+ GlyA-
- Erythroid_CD34- CD71+ GlyA+
- Erythroid_CD34- CD71lo GlyA+
- Erythroid_CD34- CD71- GlyA+
- Granulocyte/monocyte progenitors
- Colony Forming Unit-Granulocytes
- Granulocyte (Neutrophilic Metamyelocytes)
- Granulocyte (Neutrophils)
- Hematopoietic stem cells_CD133+ CD34dim
- Hematopoietic stem cell_CD38- CD34+

- Colony Forming Unit-Megakaryocytic
- Megakaryocytes
- Megakaryocyte/erythroid progenitors
- Colony Forming Unit-Monocytes
- Monocytes
- Mature NK cells_CD56- CD16+ CD3-
- Mature NK cells_CD56+ CD16+ CD3-
- Mature NK cells_CD56- CD16- CD3-
- NK T cells
- Early B cells
- Pro B cells
- CD8+ Effector Memory RA
- Naive CD8+ T cells
- CD8+ Effector Memory
- CD8+ Central Memory
- Naive CD4+ T cells
- CD4+ Effector Memory
- CD4+ Central Memory

Value

A SummarizedExperiment object with a `"logcounts"` assay containing the log-normalized expression values, along with cell type labels in the `colData`.

Author(s)

Jared Andrews

References

Novershtern N et al. (2011). Densely interconnected transcriptional circuits control cell states in human hematopoiesis. *Cell* 144, 296-309.

Examples

```
ref.se <- NovershternHematopoieticData()
```

plotScoreDistribution *Plot score distributions of labels.*

Description

Plot score distributions of labels.

Usage

```
plotScoreDistribution(
  results,
  show = c("delta.med", "delta.next", "scores"),
  labels = colnames(results$scores),
  size = 0.5,
  ncol = 5,
  dots.on.top = TRUE,
  this.color = "#F0E442",
  pruned.color = "#E69F00",
  other.color = "gray60",
  show.nmads = NULL,
  show.min.diff = NULL
)
```

Arguments

results	A DataFrame containing the output from SingleR or classifySingleR.
show	String specifying whether to show the scores, the difference from the median or the difference from the next-best score.
labels	String vector indicating one or more labels to show. If NULL, all labels available in results are presented.
size	Numeric scalar to set the size of the dots.
ncol	Integer scalar to set the number of labels to display per row.
dots.on.top	Logical specifying whether cell dots should be plotted on top of the violin plots.
this.color	String specifying the color for cells that were assigned to the label.
pruned.color	String specifying the color for cells that were assigned to the label but pruned.
other.color	String specifying the color for other cells not assigned to the label.
show.nmads	Numeric scalar that shows the threshold that would be used for pruning with pruneScores. Only used when show="delta.med".
show.min.diff	Numeric scalar that shows the threshold that would be used for pruning with pruneScores. Only used when show="delta.med" or "delta.next".

Details

This function creates jitter and violin plots showing assignment scores or related values for all cells across one or more labels. It is intended for visualizing and adjusting the nmads, min.diff.med, and min.diff.next cutoffs of the pruneScores function.

The show argument determines what values to show on the y-axis. Options are:

- "delta.med", the difference between the score of the assigned label and the median of all scores for each cell.
- "delta.next", the difference between best and second-best tuning scores of each cell.
- "scores", the raw assignment scores prior to fine-tuning.

For a given label X, cells distributions in several categories are shown:

- Was assigned to label X, and the label was not pruned away.
- Was assigned to label X, and the label was pruned away.
- Was assigned as any label, including label X.

Each category is grouped and colored separately based on this.color and related parameters.

Values are stratified according to the assigned labels in results$labels. If any fine-tuning was performed, the highest scoring label for an individual cell may not be its final label. This may manifest as negative values when show="delta.med".

Also note that pruneScores trims based on the min.diff.med and min.diff.next cutoffs first, before calculating the first-labels' delta medians. Thus, the actual nmads cut-off used in pruneScores may vary from the one portrayed in the plot.

Value

A ggplot object showing assignment scores in violin plots.

Author(s)

Daniel Bunis and Aaron Lun

See Also

SingleR, to generate scores.

pruneScores, to remove low-quality labels based on the scores, and to see more about the quailty cutoffs.

Examples

```
example(SingleR, echo=FALSE)

# To show the distribution of scores grouped by label:
plotScoreDistribution(results = pred)
# We can display a particular label using the label
plotScoreDistribution(results = pred, labels = "B")

# To show the distribution of deltas between cells' maximum and median scores,
#   grouped by label, change `show` to "delta.med":
#   This is useful for checking/adjusting nmads and min.diff.med
plotScoreDistribution(results = pred, show = "delta.med")
# The nmads cutoff can be displayed using show.nmads.
plotScoreDistribution(results = pred, show = "delta.med",
    show.nmads = 3)
# A min.diff.med cutoff can be shown using show.min.diff
plotScoreDistribution(results = pred, show = "delta.med",
    show.min.diff = 0.03)

# To show the distribution of deltas between cells' top 2 fine-tuning scores,
```

```
#   grouped by label, change `show` to "delta.next":
#   This is useful for checking/adjusting min.diff.next
plotScoreDistribution(results = pred, show = "delta.next")
# A min.diff.med cutoff can be shown using show.min.diff
plotScoreDistribution(results = pred, show = "delta.next",
    show.min.diff = 0.03)
```

`plotScoreHeatmap` *Plot a score heatmap*

Description

Create a heatmap of the SingleR assignment scores across all cell-label combinations.

Usage

```
plotScoreHeatmap(
  results,
  cells.use = NULL,
  labels.use = NULL,
  clusters = NULL,
  show.labels = TRUE,
  show.pruned = FALSE,
  max.labels = 40,
  normalize = TRUE,
  cells.order = NULL,
  order.by.clusters = FALSE,
  annotation_col = NULL,
  show_colnames = FALSE,
  ...
)
```

Arguments

`results`	A DataFrame containing the output from SingleR or classifySingleR.
`cells.use`	Integer or string vector specifying the single cells to show. If NULL, all cells are presented.
`labels.use`	String vector indicating what labels to show. If NULL, all labels available in `results` are presented.
`clusters`	String vector or factor containing cell cluster assignments, to be shown as an annotation bar in the heatmap.
`show.labels`	Logical indicating whether the final labels of cells should be shown as an annotation bar.
`show.pruned`	Logical indicating whether the pruning status of the labels should be shown as an annotation bar, as defined by pruneScores.
`max.labels`	Integer scalar specifying the maximum number of labels to show.
`normalize`	Logical specifying whether correlations should be normalized to lie in [0, 1].

`cells.order` Integer vector specifying the ordering of cells/columns of the heatmap. Regardless of `cells.use`, this input should be the the same length as the total number of cells. If set, turns off clustering of columns based on scoring.

`order.by.clusters` Logical scalar specifying if cells should be ordered by `clusters` and not by scores. If set, this takes precedence over `cells.order` input.

`annotation_col, show_colnames, ...` Additional parameters for heatmap control passed to `pheatmap`.

Details

This function creates a heatmap containing the `SingleR` initial assignment scores for each cell (columns) to each reference label (rows). Users can then easily identify the high-scoring labels associated with each cell and/or cluster of cells.

If `show.labels=TRUE`, an annotation bar will be added to the heatmap indicating final labels assigned to the cells. Note that scores shown in the heatmap are initial scores prior to the fine-tuning step, so the reported labels may not match up to the visual maximum for each cell in the heatmap.

If `max.labels` is less than the total number of unique labels, only the labels with the largest maximum scores in `results` are shown in the plot. Specifically, the set of scores for each cell is centred and scaled, and the maximum transformed score for each label is used to choose the labels to retain.

Additional arguments can be passed to `pheatmap` for further tweaking of the heatmap. Particularly useful parameters are `show_colnames`, which can be used to display cell/cluster names; and `annotation_col`, which can be used to add extra annotation layers. Clustering, pruning and label annotations are automatically generated and appended to `annotation_col` when available.

Value

A heatmap of assignment scores is generated on the current graphics device using **pheatmap**.

Normalization of colors

If `normalize=TRUE`, scores will be linearly adjusted for each cell so that the smallest score is 0 and the largest score is 1. This is followed by cubing of the adjusted scores to improve dynamic range near 1. Visually, the color scheme is changed to a blue-green-yellow scale.

The adjustment is intended to inflate differences between scores within a given cell for easier visualization. This is because the scores are often systematically shifted between cells, making the raw values difficult to directly compare. However, it may be somewhat misleading; fine-tuning may appear to assign a cell to a label with much lower score whereas the actual scores are much closer. It is for this reason that the color bar values are not shown as the absolute values of the score have little meaning.

Also note that this transformation is done *after* the choice of the top `max.labels` labels.

Author(s)

Daniel Bunis, based on code by Dvir Aran.

See Also

`SingleR`, to generate `scores`.

`pruneScores`, to remove low-quality labels based on the scores.

`pheatmap`, for additional tweaks to the heatmap.

Examples

```
# Running the SingleR() example.
example(SingleR, echo=FALSE)
# Grab the original identities of the cells as mock clusters
clusts <- g

# Creating a heatmap with just the labels.
plotScoreHeatmap(pred)

# Creating a heatmap with clusters also displayed.
plotScoreHeatmap(pred, clusters=clusts)

# Creating a heatmap with whether cells were pruned displayed.
plotScoreHeatmap(pred, show.pruned = TRUE)

# We can also turn off the normalization with Normalize = FALSE
plotScoreHeatmap(pred, clusters=clusts, normalize = FALSE)

# To only show certain labels, you can use labels.use or max.labels
plotScoreHeatmap(pred, clusters=clusts, labels.use = c("A","B","D"))
plotScoreHeatmap(pred, clusters=clusts, max.labels = 4)

# We can pass extra tweaks the heatmap as well
plotScoreHeatmap(pred, clusters=clusts, fontsize.row = 9)
plotScoreHeatmap(pred, clusters=clusts, cutree_col = 3)
```

---

pruneScores *Prune out low-quality assignments*

Description

Remove low-quality assignments based on the cell-label score matrix returned by `classifySingleR`.

Usage

```
pruneScores(
  results,
  nmads = 3,
  min.diff.med = -Inf,
  min.diff.next = 0,
  get.thresholds = FALSE
)
```

Arguments

`results`	A DataFrame containing the output generated by `SingleR` or `classifySingleR`.
`nmads`	Numeric scalar specifying the number of MADs to use for defining low outliers in the per-label distribution of delta values (i.e., difference from median).
`min.diff.med`	Numeric scalar specifying the minimum acceptable delta for each cell.
`min.diff.next`	Numeric scalar specifying the minimum difference between the best score and the next best score in fine-tuning.

`get.thresholds` Logical scalar indicating whether the per-label thresholds on the deltas should be returned.

Details

By itself, the SingleR algorithm will always assign a label to every cell. This occurs even if the cell's true label is not represented in the reference set of labels, resulting in assignment of an incorrect label to that cell. The `pruneScores` function aims to mitigate this effect by removing poor-quality assignments with "low" scores.

We compute a "delta" value for each cell, defined as the difference between the score for the assigned label and the and median score across all labels. If the delta is small, this indicates that the cell matches all labels with the same confidence such that the assigned label is not particularly meaningful. The aim is to discard low delta values caused by (i) ambiguous assignments with closely related reference labels and (ii) incorrect assignments that match poorly to all reference labels.

We use an outlier-based approach to obtain a minimum threshold for filtering "low" delta values. For each (pre-fine-tuning) label, we obtain a distribution of deltas across all assigned cells. Cells that are more than `nmads` below the median score for each label are ignored. This assumes that most cells are correctly assigned to their true label and that cells of the same label have a unimodal distribution of delta values.

Filtering on outliers is useful as it adapts to the spread and scale of delta values. For example, references with many closely related cell types will naturally yield lower deltas. By comparison, references with more distinct cell types would yield large deltas, even for cells that have no representative type in the reference and are incorrectly assigned to the next-most-related label. The outlier definition procedure adjusts naturally to these situations.

The default `nmads` is motivated by the fact that, for a normal distribution, 99 Smaller values for `nmads` will increase the stringency of the pruning.

Value

A logical vector is returned by default, specifying which assignments in `results` should be ignored.

If `get.thresholds=TRUE`, a numeric vector is returned containing the per-label thresholds on the deltas, as defined using the outlier-based approach with `nmads`.

Applying a hard filter on the deltas

If `min.diff.med` is specified, cells with deltas below this threshold are discarded. This is provided as an alternative filtering approach if the assumptions of outlier detection are violated. For example, if one label is consistently missassigned, the incorrect assignments would not be pruned. In such cases, one could set a threshold with `min.diff.med` to forcibly remove low-scoring cells.

It is possible for the per-label delta distribution to be multimodal yet still correct, e.g., due to cells belonging to subtypes nested within a main type label. This violates the unimodal assumption mentioned above for the outlier detection. In such cases, it may be better to set `nmads=Inf` and rely on `min.diff.med` for filtering instead.

Note that the deltas do not consider the effects of fine-tuning as scores are not comparable across different fine-tuning steps. In situations involving a majority of labels with only subtle distinctions, it is possible for the scores to be relatively similar but for the labels to be correctly assigned after fine-tuning. While outlier detection will automatically adapt to smaller scores, this effect should be considered if a threshold needs to be manually chosen for use in `min.diff.med`.

Filtering on fine-tuning scores

If fine-tuning was performed to generate `results`, we ignore any cell for which the fine-tuning score is not more than `min.diff.next` greater than the next best score. This aims to only retain labels for which there is no ambiguity in assignment, especially when some labels have similar scores because they are closely related (and thus easily confused).

Typical values of `min.diff.next` woud lie between [0, 0.1]. That said, the `min.diff.next` cutoff can be harmful in some applications involving highly related labels. From a user perspective, any confusion between these labels may not be a problem as the assignment is broadly correct; however, the best and next best scores will be very close and cause the labels to be unnecessarily discarded.

Author(s)

Aaron Lun and Daniel Bunis

See Also

`classifySingleR`, to generate `results`.

`getDeltaFromMedian`, to compute the per-cell deltas.

Examples

```
# Running the SingleR() example.
example(SingleR, echo=FALSE)

summary(pruneScores(pred))
pruneScores(pred, get.thresholds=TRUE)

# Less stringent:
summary(pruneScores(pred, min.diff.med=0))
summary(pruneScores(pred, nmads=5))

# More stringent:
summary(pruneScores(pred, min.diff.med=0.1))
summary(pruneScores(pred, nmads=2))
summary(pruneScores(pred, min.diff.next=0.1))
```

`SingleR` *Annotate scRNA-seq data*

Description

Returns the best annotation for each cell in a test dataset, given a labelled reference dataset in the same feature space.

Usage

```
SingleR(
  test,
  ref,
  labels,
  method = c("single", "cluster"),
```

```
  clusters = NULL,
  genes = "de",
  de.method = "classic",
  de.n = NULL,
  de.args = list(),
  quantile = 0.8,
  fine.tune = TRUE,
  tune.thresh = 0.05,
  sd.thresh = 1,
  prune = TRUE,
  assay.type.test = "logcounts",
  assay.type.ref = "logcounts",
  check.missing = TRUE,
  BNPARAM = KmknnParam(),
  BPPARAM = SerialParam()
)
```

Arguments

`test`	A numeric matrix of single-cell expression values where rows are genes and columns are cells. Alternatively, a SummarizedExperiment object containing such a matrix.
`ref`	A numeric matrix of expression values where rows are genes and columns are reference samples (individual cells or bulk samples). Each row should be named with the gene name. In general, the expression values are expected to be log-transformed, see Details. Alternatively, a SummarizedExperiment object containing such a matrix. Alternatively, a list or List of SummarizedExperiment objects or numeric matrices containing multiple references, in which case the row names are expected to be the same across all objects.
`labels`	A character vector or factor of known labels for all samples in `ref`. Alternatively, if `ref` is a list, `labels` should be a list of the same length. Each element should contain a character vector or factor specifying the label for the corresponding entry of `ref`.
`method`	String specifying whether annotation should be performed on single cells in `test`, or whether they should be aggregated into cluster-level profiles prior to annotation.
`clusters`	A character vector or factor of cluster identities for each cell in `test`. Only used if `method="cluster"`.
`genes, sd.thresh`	Arguments controlling the genes that are used for annotation, see `trainSingleR`.
`de.method`	String specifying how DE genes should be detected between pairs of labels. Defaults to `"classic"`, which sorts genes by the log-fold changes and takes the top `de.n`. Setting to `"wilcox"` or `"t"` will use Wilcoxon ranked sum test or Welch t-test between labels, respectively, and take the top `de.n` upregulated genes per comparison.
`de.n`	An integer scalar specifying the number of DE genes to use when `genes="de"`. If `de.method="classic"`, defaults to `500 * (2/3) ^ log2(N)` where `N` is the number of unique labels. Otherwise, defaults to 10.
`de.args`	Named list of additional arguments to pass to `pairwiseTTests` or `pairwiseWilcox` when `de.method="wilcox"` or `"t"`.

`quantile, fine.tune, tune.thresh, prune`
: Further arguments to pass to `classifySingleR`.

`assay.type.test`
: An integer scalar or string specifying the assay of `test` containing the relevant expression matrix, if `test` is a SummarizedExperiment object.

`assay.type.ref`
: An integer scalar or string specifying the assay of `ref` containing the relevant expression matrix, if `ref` is a SummarizedExperiment object.

`check.missing`
: Logical scalar indicating whether rows should be checked for missing values (and if found, removed).

`BNPARAM`
: A BiocNeighborParam object specifying the algorithm to use for building nearest neighbor indices.

`BPPARAM`
: A BiocParallelParam object specifying how parallelization should be performed, if any.

Details

If `method="single"`, this function is effectively just a convenient wrapper around `trainSingleR` and `classifySingleR`.

If `method="cluster"`, per-cell profiles are summed to obtain per-cluster profiles and annotation is performed on these clusters.

The function will automatically restrict the analysis to the intersection of the genes available in both `ref` and `test`. If this intersection is empty (e.g., because the two datasets use different annotation in their row names), an error will be raised.

`ref` can contain both single-cell or bulk data, but in the case of the former, read the Note in `?trainSingleR`.

Value

A DataFrame is returned containing the annotation statistics for each cell or cluster (row). This is identical to the output of `classifySingleR`.

Author(s)

Aaron Lun, based on code by Dvir Aran.

References

Aran D, Looney AP, Liu L et al. (2019). Reference-based analysis of lung single-cell sequencing reveals a transitional profibrotic macrophage. *Nat. Immunology* 20, 163–172.

Examples

```
##############################
## Mocking up training data ##
##############################

Ngroups <- 5
Ngenes <- 1000
means <- matrix(rnorm(Ngenes*Ngroups), nrow=Ngenes)
means[1:900,] <- 0
colnames(means) <- LETTERS[1:5]
```

```
g <- rep(LETTERS[1:5], each=4)
ref <- SummarizedExperiment(
    list(counts=matrix(rpois(1000*length(g),
        lambda=10*2^means[,g]), ncol=length(g))),
    colData=DataFrame(label=g)
)
rownames(ref) <- sprintf("GENE_%s", seq_len(nrow(ref)))

ref <- scater::logNormCounts(ref)
trained <- trainSingleR(ref, ref$label)

################################
## Mocking up some test data ##
################################

N <- 100
g <- sample(LETTERS[1:5], N, replace=TRUE)
test <- SummarizedExperiment(
    list(counts=matrix(rpois(1000*N, lambda=2^means[,g]), ncol=N)),
    colData=DataFrame(cluster=g)
)

rownames(test) <- sprintf("GENE_%s", seq_len(nrow(test)))
test <- scater::logNormCounts(test)

################################
## Performing classification ##
################################

pred <- SingleR(test, ref, labels=ref$label)
table(predicted=pred$labels, truth=g)

pred2 <- SingleR(test, ref, labels=ref$label,
    method="cluster", clusters=test$cluster)
table(predicted=pred2$labels, truth=rownames(pred2))
```

`trainSingleR` *Train the SingleR classifier*

Description

Train the SingleR classifier on one or more reference datasets with known labels.

Usage

```
trainSingleR(
  ref,
  labels,
  genes = "de",
  sd.thresh = 1,
  de.method = c("classic", "wilcox", "t"),
  de.n = NULL,
  de.args = list(),
```

```
  assay.type = "logcounts",
  check.missing = TRUE,
  BNPARAM = KmknnParam()
)
```

Arguments

ref	A numeric matrix of expression values where rows are genes and columns are reference samples (individual cells or bulk samples). Each row should be named with the gene name. In general, the expression values are expected to be log-transformed, see Details. Alternatively, a SummarizedExperiment object containing such a matrix. Alternatively, a list or List of SummarizedExperiment objects or numeric matrices containing multiple references, in which case the row names are expected to be the same across all objects.
labels	A character vector or factor of known labels for all samples in ref. Alternatively, if ref is a list, labels should be a list of the same length. Each element should contain a character vector or factor specifying the label for the corresponding entry of ref.
genes	A string specifying the feature selection method to be used, see Details. Alternatively, a list of lists of character vectors containing DE genes between pairs of labels. Alternatively, a list of character vectors containing marker genes for each label.
sd.thresh	A numeric scalar specifying the minimum threshold on the standard deviation per gene. Only used when genes="sd".
de.method	String specifying how DE genes should be detected between pairs of labels. Defaults to "classic", which sorts genes by the log-fold changes and takes the top de.n. Setting to "wilcox" or "t" will use Wilcoxon ranked sum test or Welch t-test between labels, respectively, and take the top de.n upregulated genes per comparison.
de.n	An integer scalar specifying the number of DE genes to use when genes="de". If de.method="classic", defaults to 500 * (2/3) ^ log2(N) where N is the number of unique labels. Otherwise, defaults to 10.
de.args	Named list of additional arguments to pass to pairwiseTTests or pairwiseWilcox when de.method="wilcox" or "t".
assay.type	An integer scalar or string specifying the assay of ref containing the relevant expression matrix, if ref is a SummarizedExperiment object.
check.missing	Logical scalar indicating whether rows should be checked for missing values (and if found, removed).
BNPARAM	A BiocNeighborParam object specifying the algorithm to use for building nearest neighbor indices.

Details

This function uses a training data set to select interesting features and construct nearest neighbor indices in rank space. The resulting objects can be re-used across multiple classification steps with different test data sets via classifySingleR. This improves efficiency by avoiding unnecessary repetition of steps during the downstream analysis.

Several options are available for feature selection:

- genes="de" identifies genes that are differentially expressed between labels. This is done by identifying the median expression within each label, and computing differences between medians for each pair of labels. For each label, the top de.n genes with the largest differences compared to another label are chosen as markers to distinguish the two labels. The set of all features is defined as the union of markers from all pairwise comparisons.
- genes="sd" identifies genes that are highly variable across labels. This is done by identifying the median expression within each label, and computing the standard deviation in the medians across all labels. The set of all features is defined as those genes with standard deviations above sd.thresh.
- genes="all" will not perform any feature selection.

If genes="de" or "sd", the expression values are expected to be log-transformed and normalized.

Value

For a single reference, a List is returned containing:

common.genes: A character vector of all genes that were chosen by the designated feature selection method.

nn.indices: A List of BiocNeighborIndex objects containing pre-constructed neighbor search indices.

original.exprs: A List of numeric matrices where each matrix contains all cells for a particular label.

extra: A List of additional information on the feature selection, for use by classifySingleR. This includes the selection method in genes and method-specific structures that can be reused during classification.

For multiple references, a List of Lists is returned where each internal List corresponds to a reference in ref and has the same structure as described above.

Custom feature specification

Rather than relying on the in-built feature selection, users can pass in their own features of interest to genes. The function expects a named list of named lists of character vectors, with each vector containing the DE genes between a pair of labels. For example:

```
genes <- list(
   A = list(A = character(0), B = "GENE_1", C = c("GENE_2", "GENE_3")),
   B = list(A = "GENE_100", B = character(0), C = "GENE_200"),
   C = list(A = c("GENE_4", "GENE_5"), B = "GENE_5", C = character(0))
)
```

If we consider the entry genes$A$B, this contains marker genes for label "A" against label "B". That is, these genes are upregulated in "A" compared to "B". The outer list should have one list per label, and each inner list should have one character vector per label. (Obviously, a label cannot have markers against itself, so this is just set to character(0).)

Alternatively, genes can be a named list of character vectors containing per-label markers. For example:

```
genes <- list(
     A = c("GENE_1", "GENE_2", "GENE_3"),
     B = c("GENE_100", "GENE_200"),
     C = c("GENE_4", "GENE_5")
)
```

The entry genes$A represent the genes that are upregulated in A compared to some or all other labels. This allows the function to handle pre-defined marker lists for specific cell populations. However, it obviously captures less information than marker sets for the pairwise comparisons.

If genes explicitly contains gene identities (as character vectors), ref can be the raw counts or any monotonic transformation thereof.

Dealing with multiple references

The default **SingleR** policy for dealing with multiple references is to perform the classification for each reference separately and combine the results (see ?combineResults for an explanation). To this end, if ref is a list with multiple references, marker genes are identified separately within each reference when de="genes" or "sd". This is almost equivalent to running trainSingleR on each reference separately, except that the final common set of genes consists of the union of common genes across all references. We take the union to ensure that correlations are computed from the same set of genes across reference and are thus reasonably comparable in combineResults.

Note on single-cell references

The default marker selection is based on log-fold changes between the per-label medians and is very much designed with bulk references in mind. It may not be effective for single-cell reference data where it is not uncommon to have more than 50% zero counts for a given gene such that the median is also zero for each group. Users are recommended to either set de.method to another DE ranking method, or detect markers externally and pass a list of markers to genes (see Examples).

Author(s)

Aaron Lun, based on the original SingleR code by Dvir Aran.

See Also

classifySingleR, where the output of this function gets used.

combineResults, to combine results from multiple references.

Examples

```
##############################
## Mocking up training data ##
##############################

Ngroups <- 5
Ngenes <- 1000
means <- matrix(rnorm(Ngenes*Ngroups), nrow=Ngenes)
means[1:900,] <- 0
colnames(means) <- LETTERS[1:5]

g <- rep(LETTERS[1:5], each=4)
ref <- SummarizedExperiment(
    list(counts=matrix(rpois(1000*length(g),
        lambda=10*2^means[,g]), ncol=length(g))),
    colData=DataFrame(label=g)
)
rownames(ref) <- sprintf("GENE_%s", seq_len(nrow(ref)))

########################
## Doing the training ##
```

```
########################

# Normalizing and log-transforming for automated marker detection.
ref <- scater::logNormCounts(ref)

trained <- trainSingleR(ref, ref$label)
trained
trained$nn.indices
length(trained$common.genes)

# Alternatively, computing and supplying a set of label-specific markers.
by.t <- scran::pairwiseTTests(assay(ref, 2), ref$label, direction="up")
markers <- scran::getTopMarkers(by.t[[1]], by.t[[2]], n=10)
trained <- trainSingleR(ref, ref$label, genes=markers)
length(trained$common.genes)
```

Index