

Package ‘SingleR’

October 17, 2020

Title Reference-Based Single-Cell RNA-Seq Annotation

Version 1.2.4

Date 2020-05-24

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, BiocGenerics, DelayedArray, DelayedMatrixStats, BiocNeighbors, BiocParallel, BiocSingular, stats, utils, Rcpp, ExperimentHub

LinkingTo Rcpp, beachmat

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

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

SystemRequirements C++11

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 7.1.0

URL <https://github.com/LTLA/SingleR>

BugReports <https://support.bioconductor.org/>

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

git_branch RELEASE_3_11

git_last_commit 4653a3c

git_last_commit_date 2020-05-24

Date/Publication 2020-10-16

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:

.mockRefData	2
aggregateReference	3
BlueprintEncodeData	5
classifySingleR	6
combine-predictions	10
combineCommonResults	11
combineRecomputedResults	12
DatabaseImmuneCellExpressionData	15
getDeltaFromMedian	17
HumanPrimaryCellAtlasData	18
ImmGenData	19
matchReferences	20
MonacoImmuneData	21
MouseRNAseqData	23
NovershternHematopoieticData	24
plotScoreDistribution	27
plotScoreHeatmap	30
pruneScores	34
SingleR	37
trainSingleR	40
Index	45

.mockRefData	<i>Mock data for examples</i>
--------------	-------------------------------

Description

Make up some test and reference data for the various examples in the **SingleR** package.

Usage

```
.mockRefData(ngroups = 5, nreps = 4, ngenes = 1000, prop = 0.5)

.mockTestData(mock.ref, ncells = 100)
```

Arguments

ngroups	Integer scalar specifying the number of groups.
nreps	Integer scalar specifying the number of replicates per group.
ngenes	Integer scalar specifying the number of genes in the dataset.
prop	Numeric scalar specifying the proportion of genes that are DE between groups.
mock.ref	A SummarizedExperiment object produced by .mockRefData.
ncells	Integer scalar specifying the number of cells to simulate.

Details

This functions are simply provided to simulate some data in the Examples of the documentation. The simulations are very simple and should not be used for performance comparisons.

Value

Both functions return a [SummarizedExperiment](#) object containing simulated counts in the counts assay, with the group assignment of each sample in the "label" field of the [colData](#).

Author(s)

Aaron Lun

Examples

```
ref <- .mockRefData()
test <- .mockTestData(ref)
```

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,
  ncenters = NULL,
  power = 0.5,
  assay.type = "logcounts",
  rank = 20,
  subset.row = NULL,
  check.missing = TRUE,
  BPPARAM = SerialParam(),
  BSPARAM = bsparam()
)
```

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.
ncenters	Integer scalar specifying the maximum number of aggregated profiles to produce for each label.
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.

rank	Integer scalar specifying the number of principal components to use during clustering.
subset.row	Integer, character or logical vector indicating the rows of ref to use for k-means clustering.
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.
BSPARAM	A BiocSingularParam object indicating which SVD algorithm should be used in <code>runPCA</code> .

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 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. 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.

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 via k-means clustering on all cells in a particular label. Cells in each cluster are subsequently averaged to create one pseudo-bulk sample that serves as a representative for that location in the expression space. This reduces the number of separate observations (for speed) while preserving some level of population heterogeneity (for fidelity).

The number of pseudo-bulk samples per label is controlled by `ncenters`. By default, we set the number of clusters to X^{power} where X is the number of cells for that label. This ensures that labels with more cells have more resolved representatives. If `ncenters` is greater than the number of samples for a label and/or `power=1`, no aggregation is performed. Setting `power=0` will aggregate all cells of a label into a single pseudo-bulk profile.

k-means clustering is actually performed on the first rank principal components as computed using `runPCA`. The use of PCs compacts the data for more efficient operation of `kmeans`; it also removes some of the high-dimensional noise to highlight major factors of within-label heterogeneity. Note that the PCs are only used for clustering and the full expression profiles are still used for the final averaging. Users can disable the PCA step by setting `rank=Inf`.

Setting `subset.row` to an appropriate indexing vector will subset the matrix by gene prior to PCA. However, again, the aggregation is performed on the full expression profiles. This option may be useful for clustering based on known genes of interest but retaining all genes in the aggregated results.

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,
  cell.ont = c("all", "nonna", "none")
)
```

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.
cell.ont	String specifying whether Cell Ontology terms should be included in the colData . If "nonna", all samples without a valid term are discarded; if "all", all samples are returned with (possibly NA) terms; if "none", terms are not added.

Details

This function provides normalized expression values for 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). 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).

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. All together, this reference contains 259 samples with 43 cell types ("label.fine"), manually aggregated into 24 broad classes ("label.main"). The fine labels have also been mapped to the Cell Ontology ("label.ont", if cell.ont is not "none"), which can be used for further programmatic queries.

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 specifying 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, the per-reference statistics are combined into a single `DataFrame` of results. This is done using [combineRecomputedResults](#) if `recompute=TRUE` in [trainSingleR](#), otherwise it is done using [combineCommonResults](#).

Value

A `DataFrame` where each row corresponds to a cell in `test`. In the case of a single reference, this contains:

- `scores`, a numeric matrix of correlations at the specified quantile for each label (column) in each cell (row). This will contain NAs if multiple references were supplied to [trainSingleR](#) with `recompute=TRUE`.
- `first.labels`, a character vector containing the predicted label *before* fine-tuning. Only added if `fine.tune=TRUE`.
- `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. Only added if `fine.tune=TRUE`.
- `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". els are replaced with NAs. Only added if `prune=TRUE`.

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.

In the case of multiple references, the output of [combineCommonResults](#) or [combineRecomputedResults](#) is returned. This is a `DataFrame` containing:

- `scores`, a numeric matrix of scores for each cell (row) across all labels in all references (columns). This will contain NAs if recomputation is performed.
- `labels`, `first.labels` (if `fine.tune=TRUE`) and `pruned.labels` (if `prune=TRUE`), containing the consolidated labels of varying flavors as described above.
- `orig.results`, a `DataFrame` of `DataFrame`s containing the results of running [classifySingleR](#) against each individual reference. Each nested `DataFrame` has the same format as described above.

The `metadata` of the top-level `DataFrame` contains `common.genes` if recomputation is not performed.

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.
[combineCommonResults](#), 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)
```

Description

It is often desirable to combine information from separate references, thus improving the quality and breadth of the cell type annotation. However, it is not trivial due to the presence of batch effects across references (from differences in technology, experimental protocol or the biological system) as well as differences in the annotation vocabulary between investigators. This page describes some of the considerations with choosing a strategy to combine information from multiple reference datasets.

Option 1 - using reference-specific labels

This option nests each label within each reference data, (e.g., “Ref1-Bcell” vs “Ref2-Bcell”). It is most applicable if there are relevant biological differences between the references, e.g., one reference is concerned with healthy tissue while the other reference considers diseased tissue.

In practical terms, this option is easily implemented by just combining the expression matrices together and pasting the reference name onto the corresponding character vector of labels. There is no need for time-consuming label harmonization between references.

However, 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 - using harmonized labels

This option also involves combining the reference datasets into a single matrix but with harmonization of 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 most obvious problem with this approach is that it assumes that harmonized labels are available. This is not always the case due to differences in naming schemes (e.g. “B cell” vs “B”) between references. Another problem is that of 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”).

To mitigate this, **SingleR** datasets (e.g., [ImmGenData](#)) have all their labels mapped to the Cell Ontology, allowing the use of standard terms to refer to the same cell type across references. Users can then traverse the ontology graph to achieve a consistent label resolution across references.

Option 3 - comparing scores across the union of markers

This option 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 is also logistically simpler as it allows each reference to be processed separately (more or less, depending on the exact algorithm) for embarrassing parallelization.

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)

Aaron Lun

See Also

[combineCommonResults](#) and [combineRecomputedResults](#), for the functions that implement variants of Option 3.

[matchReferences](#), to harmonize labels between reference datasets.

combineCommonResults *Combine SingleR results with common genes*

Description

Combine results from multiple runs of [classifySingleR](#) (usually against different references) into a single [DataFrame](#). For each cell, the label from the result with the highest score is used as that cell's combined label. This assumes that each run of [classifySingleR](#) was performed using a common set of marker genes, hence the Common in the function name.

Usage

```
combineCommonResults(results)
```

Arguments

`results` A list of [DataFrame](#) prediction results as returned by [classifySingleR](#) when run on each reference separately.

Details

For each cell, we identify the reference with the highest score across all of its labels. The “combined label” is then defined as the label assigned to that cell in the highest-scoring reference. (The same logic is also applied to the first and pruned labels, if available.) See comments in [?“combine-predictions”](#) for the overall rationale.

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 also for this reason that we use the highest score prior to fine-tuning, even if it does not correspond to the score of the fine-tuned label.

It is highly unlikely that this function will be called directly by the end-user. Users are advised to use the multi-reference mode of [SingleR](#) and related functions, which will take care of the use of a common set of genes before calling this function to combine results across references.

Value

A `DataFrame` is returned containing the annotation statistics for each cell or cluster (row). This mimics the output of `classifySingleR` and contains the following fields:

- `scores`, a numeric matrix of correlations formed by combining the equivalent matrices from results.
- `labels`, a character vector containing the per-cell combined label across references.
- `references`, an integer vector specifying the reference from which the combined label was derived.
- `orig.results`, a `DataFrame` containing results.

It may also contain `first.labels` and `pruned.labels` if these were also present in results.

The `metadata` contains `common.genes`, a character vector of the common genes that were used across all references in results.

Author(s)

Jared Andrews, Aaron Lun

See Also

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

`combineRecomputedResults`, for another approach to combining predictions.

Examples

```
# Making up data (using one reference to seed another).
ref <- .mockRefData(nreps=8)
ref1 <- ref[,1:2%%2==0]
ref2 <- ref[,1:2%%2==1]
ref2$label <- tolower(ref2$label)

test <- .mockTestData(ref1)

# Applying classification with SingleR's multi-reference mode.
ref1 <- scater::logNormCounts(ref1)
ref2 <- scater::logNormCounts(ref2)
test <- scater::logNormCounts(test)

pred <- SingleR(test, list(ref1, ref2), labels=list(ref1$label, ref2$label))
pred[,1:5] # Only viewing the first 5 columns for visibility.
```

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. Unlike `combineCommonResults`, this does not assume that each run of `classifySingleR` was performed using the same set of common genes, instead recomputing the scores for comparison across references.

Usage

```
combineRecomputedResults(
  results,
  test,
  trained,
  quantile = 0.8,
  assay.type.test = "logcounts",
  check.missing = TRUE,
  BNPARAM = KmknnParam(),
  BPPARAM = SerialParam()
)
```

Arguments

<code>results</code>	A list of <code>DataFrame</code> prediction results as returned by <code>classifySingleR</code> when run on each reference separately.
<code>test</code>	A numeric matrix of single-cell expression values where rows are genes and columns are cells. Alternatively, a <code>SummarizedExperiment</code> object containing such a matrix.
<code>trained</code>	A list of <code>Lists</code> containing the trained outputs of multiple references, equivalent to either (i) the output of <code>trainSingleR</code> on multiple references with <code>recompute=TRUE</code> , or (ii) running <code>trainSingleR</code> on each reference separately and manually making a list of the trained outputs.
<code>quantile</code>	Further arguments to pass to <code>classifySingleR</code> .
<code>assay.type.test</code>	An integer scalar or string specifying the assay of <code>test</code> containing the relevant expression matrix, if <code>test</code> is a <code>SummarizedExperiment</code> object.
<code>check.missing</code>	Logical scalar indicating whether rows should be checked for missing values (and if found, removed).
<code>BNPARAM</code>	A <code>BiocNeighborParam</code> object specifying the algorithm to use for building nearest neighbor indices.
<code>BPPARAM</code>	A <code>BiocParallelParam</code> object specifying how parallelization should be performed, if any.

Details

This function implements a variant of Option 3 described in `"combine-predictions"`. For a given cell in `test`, we extract its assigned label from `results` for each reference. We also retrieve the marker genes associated with that label and take the union of markers across all references. This defines a common feature space in which the score for each reference's assigned label is recomputed using `ref`; the label from the reference with the top recomputed score is then reported as the combined annotation for that cell.

Unlike `combineCommonResults`, the union of markers is not used for the within-reference calls. This avoids the inclusion of noise from irrelevant genes in the within-reference assignments. Obviously, `combineRecomputedResults` is slower as it does require recomputation of the scores, but the within-reference calls are faster as there are fewer genes in the union of markers for assigned labels (compared to the union of markers across all labels, as required by `combineCommonResults`), so it is likely that the net compute time should be lower.

It is strongly recommended that the universe of genes be the same across all references. The intersection of genes across all ref and test is used when recomputing scores, and differences in the availability of genes between references may have unpredictable effects.

Value

A `DataFrame` is returned containing the annotation statistics for each cell or cluster (row). This mimics the output of `classifySingleR` and contains the following fields:

- `scores`, a numeric matrix of correlations containing the *recomputed* scores. For any given cell, entries of this matrix are only non-NA for the assigned label in each reference; scores are not recomputed for the other labels.
- `labels`, a character vector containing the per-cell combined label across references.
- `references`, an integer vector specifying the reference from which the combined label was derived.
- `orig.results`, a `DataFrame` containing results.

It may also contain `first.labels` and `pruned.labels` if these were also present in `results`.

Author(s)

Aaron Lun

References

Lun A, Bunis D, Andrews J (2020). Thoughts on a more scalable algorithm for multiple references. <https://github.com/LTLA/SingleR/issues/94>

See Also

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

`combineCommonResults`, for another approach to combining predictions.

Examples

```
# Making up data.
ref <- .mockRefData(nreps=8)
ref1 <- ref[,1:2%%2==0]
ref2 <- ref[,1:2%%2==1]
ref2$label <- tolower(ref2$label)

test <- .mockTestData(ref)

# Performing classification within each reference.
test <- scater::logNormCounts(test)

ref1 <- scater::logNormCounts(ref1)
train1 <- trainSingleR(ref1, labels=ref1$label)
```

```

pred1 <- classifySingleR(test, train1)

ref2 <- scater::logNormCounts(ref2)
train2 <- trainSingleR(ref2, labels=ref2$label)
pred2 <- classifySingleR(test, train2)

# Combining results with recomputation of scores.
combined <- combineRecomputedResults(
  results=list(pred1, pred2),
  test=test,
  trained=list(train1, train2))

combined[,1:5]

```

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,
  cell.ont = c("all", "nonna", "none")
)

```

Arguments

<code>ensembl</code>	Logical scalar indicating whether to convert row names to Ensembl IDs. Genes without a mapping to a non-duplicated Ensembl ID are discarded.
<code>cell.ont</code>	String specifying whether Cell Ontology terms should be included in the <code>colData</code> . If "nonna", all samples without a valid term are discarded; if "all", all samples are returned with (possibly NA) terms; if "none", terms are not added.

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 ("`label.main`"):

- B cells
- Monocytes
- NK cells

- T cells, CD8+
- T cells, CD4+

Samples were additionally annotated to 15 fine cell types ("label.fine"):

- 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

The subtypes have also been mapped to the Cell Ontology ("label.ont", if cell.ont is not "none"), which can be used for further programmatic queries.

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	<i>Compute the difference from median</i>
--------------------	---

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 function 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,  
  cell.ont = c("all", "nonna", "none")  
)
```

Arguments

<code>ensembl</code>	Logical scalar indicating whether to convert row names to Ensembl IDs. Genes without a mapping to a non-duplicated Ensembl ID are discarded.
<code>cell.ont</code>	String specifying whether Cell Ontology terms should be included in the <code>colData</code> . If "nonna", all samples without a valid term are discarded; if "all", all samples are returned with (possibly NA) terms; if "none", terms are not added.

Details

This function provides normalized expression values for 713 microarray samples from 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).

Each sample has been assigned to one of 37 main cell types ("`label.main`") and 157 subtypes ("`label.fine`"). The subtypes have also been mapped to the Cell Ontology ("`label.ont`", if `cell.ont` is not "none"), which can be used for further programmatic queries.

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	<i>Obtain mouse bulk expression data from the Immunologic Genome Project</i>
------------	--

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, cell.ont = c("all", "nonna", "none"))
```

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.
cell.ont	String specifying whether Cell Ontology terms should be included in the <code>colData</code> . If "nonna", all samples without a valid term are discarded; if "all", all samples are returned with (possibly NA) terms; if "none", terms are not added.

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.

This dataset consists of 20 broad cell types ("label.main") and 253 finely resolved cell subtypes ("label.fine"). The subtypes have also been mapped to the Cell Ontology ("label.ont", if cell.ont is not "none"), which can be used for further programmatic queries.

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	<i>Match labels from two references</i>
-----------------	---

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, cell.ont = c("all", "nonna", "none"))
```

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.
cell.ont	String specifying whether Cell Ontology terms should be included in the colData . If "nonna", all samples without a valid term are discarded; if "all", all samples are returned with (possibly NA) terms; if "none", terms are not added.

Details

The dataset contains 114 human RNA-seq samples annotated to 10 main cell types ("label.main"):

- 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 ("label.fine"):

- 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

The subtypes have also been mapped to the Cell Ontology ("label.ont", if cell.ont is not "none"), which can be used for further programmatic queries.

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	<i>Obtain mouse bulk expression data of sorted cell populations (RNA-seq)</i>
-----------------	---

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, cell.ont = c("all", "nonna", "none"))
```

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.
cell.ont	String specifying whether Cell Ontology terms should be included in the colData . If "nonna", all samples without a valid term are discarded; if "all", all samples are returned with (possibly NA) terms; if "none", terms are not added.

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 ("label.main"):

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

- NK cells
- Oligodendrocytes
- T cells

These are split further into 28 subtypes ("label.fine"). The subtypes have also been mapped to the Cell Ontology ("label.ont", if cell.ont is not "none"), which can be used for further programmatic queries.

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,
  cell.ont = c("all", "nonna", "none")
)
```

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.
cell.ont	String specifying whether Cell Ontology terms should be included in the colData . If "nonna", all samples without a valid term are discarded; if "all", all samples are returned with (possibly NA) terms; if "none", terms are not added.

Details

The dataset contains 211 human microarray samples annotated to 16 main cell types ("label.main"):

- 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 ("label.fine"):

- 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

The subtypes have also been mapped to the Cell Ontology ("label.ont", if cell.ont is not "none"), which can be used for further programmatic queries.

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("scores", "delta.med", "delta.next"),
  labels.use = colnames(results$scores),
  scores.use = NULL,
  calls.use = 0,
  pruned.use = 0,
  size = 0.5,
  ncol = 5,
  dots.on.top = TRUE,
  this.color = "#F0E442",
  pruned.color = "#E69F00",
  other.color = "gray60",
  show.nmads = 3,
  show.min.diff = NULL,
  grid.vars = list()
)
```

Arguments

- | | |
|-----------------------|--|
| results | A DataFrame containing the output from SingleR , classifySingleR , combineCommonResults , or combineRecomputedResults . |
| show | String specifying whether to show the scores ("scores"), the difference from the median ("delta.med") or the difference from the next-best score ("delta.next"). |
| labels.use | String vector indicating one or more labels to show. If NULL, all labels available in results are presented. |
| scores.use | Integer scalar specifying which scores to use. This can refer to any column index of results\$orig.results to use scores from individual references of a combined prediction (see ?combine-predictions), or to the top-level results which is indicated by the value 0.

Alternatively, scores.use can be an integer vector containing multiple such column indices (or zero). In such cases, multiple plots will be created showing multiple sets of scores.

Default setting, scores.use=NULL, will create plots for all targets that make sense: when show="scores" plots are created for top-level results & all individual references; when show="delta.med" or "delta.next", plots are created for all individual references. |
| calls.use, pruned.use | Integer scalar specifying which chosen labels or pruning calls to use, defaulting to those from the top-level results which is indicated by the value 0. However, |

	this can also refer to any column index of <code>results\$orig.results</code> to use labels from individual references of a combined prediction (see ?combine-predictions). Alternatively, an integer vector of the same length as <code>scores.use</code> , specifying the labels to use in each plot generated by <code>scores.use</code> .
<code>size</code>	Numeric scalar to set the size of the dots.
<code>ncol</code>	Integer scalar to set the number of labels to display per row.
<code>dots.on.top</code>	Logical specifying whether cell dots should be plotted on top of the violin plots.
<code>this.color</code>	String specifying the color for cells that were assigned to the label.
<code>pruned.color</code>	String specifying the color for cells that were assigned to the label but pruned.
<code>other.color</code>	String specifying the color for other cells not assigned to the label.
<code>show.nmads</code>	Numeric scalar that shows the threshold that would be used for pruning with pruneScores . Only used when <code>show="delta.med"</code> .
<code>show.min.diff</code>	Numeric scalar that shows the threshold that would be used for pruning with pruneScores . Only used when <code>show="delta.med"</code> or <code>"delta.next"</code> .
<code>grid.vars</code>	named list of extra variables to pass to grid.arrange , used when <code>scores.use</code> is of length greater than 1. If NULL, the function will not arrange plots in a grid and will instead output them as a list. Doing so outputs them one after another on the graphics device.

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, or for comparing scores across predictions when multiple references were used (see [?combine-predictions](#)).

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

- `"scores"`, the raw assignment scores (prior to fine-tuning).
- `"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.

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, except 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

One or more [ggplot](#) objects showing assignment scores in violin plots is generated on the current graphics device. Or such objects are returned one-by-one as a list if `scores.use` is of length greater than 1, and `grid.vars` is set to NULL.

Working with combined results

When results are the output of a combined prediction (see [?combine-predictions](#)), `scores.use`, `calls.use`, and `pruned.use` are used to indicate which prediction's scores, chosen labels, and pruning calls should be utilized.

- `scores.use` sets which prediction's scores to create distribution plots for.
- `calls.use` sets which prediction's label calls to use for grouping scores and coloring the distributions.
- `pruned.use` sets which prediction's pruning calls to use for splitting cells assigned to each label into "assigned" versus "pruned".

Values of these inputs can be:

- 0: the top-level "combined" scores or calls.
- Any positive integer: indicates the index of an individual prediction within `results$orig.results`.

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 quality cutoffs.

[grid.arrange](#), for tweaks to the how plots are arranged when multiple are output together.

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.use = "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 cutoffs
plotScoreDistribution(results = pred,
  show = "delta.med")

# 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 cutoffs
plotScoreDistribution(results = pred, show = "delta.next")

### Visualizing and adjusting pruning cutoffs ###

# The default nmads cutoff of 3 is displayed when 'show = "delta.med"', but
# this can be adjusted or turned off with 'show.nmads'
plotScoreDistribution(results = pred,
```

```

    show = "delta.med", show.nmads = 2)
plotScoreDistribution(results = pred,
  show = "delta.med", show.nmads = NULL)

# A min.diff cutoff can be shown using 'show.min.diff' when
# 'show = "delta.med"' or 'show = "delta.next"'
plotScoreDistribution(results = pred,
  show = "delta.med", show.min.diff = 0.03)
plotScoreDistribution(results = pred,
  show = "delta.next", show.min.diff = 0.03)

### Multi-Reference Compatibility ###

# When SingleR is run with multiple references, default output will contain
# separate plots for each original reference, as well as for the the combined
# set when 'show' = "scores".
example(combineRecomputedResults, echo = FALSE)
plotScoreDistribution(results = combined)

# 'scores.use' sets which original results to plot distributions for, and can
# be multiple or NULL (default)
plotScoreDistribution(results = combined, show = "scores",
  scores.use = 0)
plotScoreDistribution(results = combined, show = "scores",
  scores.use = 1:2)

# To color and group cells by non-final label and pruned calls,
# use 'calls.use' and 'pruned.use'
plotScoreDistribution(results = combined, show = "scores",
  calls.use = 1, pruned.use = 1)

# To have plots output in a grid rather than as separate pages, provide,
# a list of inputs for gridExtra::grid.arrange() to 'grids.vars'.
plotScoreDistribution(combined,
  grid.vars = list(ncol = 1))

# An empty list will use grid.arrange defaults
plotScoreDistribution(combined,
  grid.vars = list())

```

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 = c("labels", "clusters"),
scores.use = NULL,
calls.use = 0,
na.color = "gray30",
cluster_cols = FALSE,
annotation_col = NULL,
show_colnames = FALSE,
color = (grDevices::colorRampPalette(c("#D1147E", "white", "#00A44B")))(100),
silent = FALSE,
...,
grid.vars = list()
)

```

Arguments

results	A DataFrame containing the output from SingleR , classifySingleR , combineCommonResults , or combineRecomputedResults .
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 the target 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 chosen labels of cells should be shown as an annotation bar.
show.pruned	Logical indicating whether the pruning status of the cell labels, as defined by pruneScores , should be shown as an annotation bar.
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 or String vector specifying how to order the cells/columns of the heatmap. Regardless of cells.use, this input should be the the same length as the total number of cells. Subordinate to cluster_cols.
order.by	String providing the annotation to be used for cells/columns ordering. Can be "labels" (default) or "clusters" (when provided). Subordinate to cells.order and cluster_cols.
scores.use	Integer scalar specifying which scores to use. This can refer to any column index of results\$orig.results to use scores from individual references of a combined prediction (see ?combine-predictions), or to the top-level results which is indicated by the value 0. Alternatively, scores.use can be an integer vector containing multiple such column indices (or zero). In such cases, multiple heatmaps will be created showing multiple sets of scores.

	Default setting, <code>scores.use=NULL</code> , will create heatmaps for scores from both the top-level results and the individual references.
<code>calls.use</code>	Integer scalar specifying which labels to use, defaulting to those from the top-level results which is indicated by the value 0. However, this can also refer to any column index of <code>results\$orig.results</code> to use labels from individual references of a combined prediction (see ?combine-predictions). Alternatively, an integer vector of the same length as <code>scores.use</code> , specifying the labels to use in each heatmap generated by <code>scores.use</code> .
<code>na.color</code>	String specifying the color for non-calculated scores of combined results.
<code>annotation_col</code> , <code>cluster_cols</code> , <code>show_colnames</code> , <code>color</code> , <code>silent</code> , ...	Additional parameters for heatmap control passed to pheatmap .
<code>grid.vars</code>	A named list of extra variables to pass to grid.arrange , used when <code>scores.use</code> is of length greater than 1. If <code>NULL</code> , the function will not arrange plots on a grid and will instead output them as a list.

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 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 top labels are shown in the plot. Labels that were called most frequently are prioritized. Then remaining labels are selected based on:

- General case: Labels with max z-scores after per-cell centering and scaling of the scores matrix.
- Recomputed scores of combined predictions: Labels which were suggested most frequently by individual references.

Value

One or more heatmaps of assignment scores, generated by [pheatmap](#), are returned on the current graphics device. Or a list of such heatmaps are output if `scores.use` is of length greater than 1, and `grid.vars` is set to `NULL`.

Working with combined results

When results are the output of a combined prediction (see [?combine-predictions](#)), `scores.use` and `calls.use` are used to indicate which prediction's scores or labels should be presented.

- `scores.use` sets which prediction's scores to create a heatmap for.
- `calls.use` sets which prediction's label (and pruning) calls to show as annotations above the heatmap.

Values of these inputs can be:

- 0: the top-level "combined" scores or calls.
- Any positive integer: indicates the index of an individual prediction within `results$orig.results`.

Tweaking the output

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; `treeheight_row`, which sets the width of the clustering tree; 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.

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.

[grid.arrange](#), for tweaks to the how heatmaps are arranged when multiple are output together.

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 = 20)
plotScoreHeatmap(pred, clusters=clusts,
  treeheight_row = 15)
plotScoreHeatmap(pred, clusters=clusts, cluster_col = TRUE,
  cutree_cols = 5)

### Multi-Reference Compatibility ###

example(combineRecomputedResults, echo = FALSE)
plotScoreHeatmap(combined)

# 'scores.use' sets which particular run's scores to show, and can be multiple
plotScoreHeatmap(combined,
  scores.use = 1)
plotScoreHeatmap(combined,
  scores.use = c(0,2))

# 'calls.use' adjusts which run's labels and pruning calls to display.
plotScoreHeatmap(combined,
  calls.use = 1)

# To have plots output in a grid rather than as separate pages, provide,
# a list of inputs for gridExtra::grid.arrange() to 'grids.vars'.
plotScoreHeatmap(combined,
  grid.vars = list(ncol = 1))

# An empty list will use grid.arrange defaults
plotScoreHeatmap(combined,
  grid.vars = list())

```

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

<code>results</code>	A <code>DataFrame</code> containing the output generated by <code>SingleR</code> or <code>classifySingleR</code> .
<code>nmads</code>	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).
<code>min.diff.med</code>	Numeric scalar specifying the minimum acceptable delta for each cell.
<code>min.diff.next</code>	Numeric scalar specifying the minimum difference between the best score and the next best score in fine-tuning.
<code>get.thresholds</code>	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 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` would 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",
  sd.thresh = 1,
  de.method = "classic",
  de.n = NULL,
  de.args = list(),
  aggr.ref = FALSE,
  aggr.args = list(),
  recompute = TRUE,
  quantile = 0.8,
  fine.tune = TRUE,
  tune.thresh = 0.05,
  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 (usually log-transformed) expression values from a reference dataset, or a SummarizedExperiment object containing such a matrix; see trainSingleR for details. Alternatively, a list or List of SummarizedExperiment objects or numeric matrices containing multiple references. Row names may be different across entries but only the intersection will be used, see Details .
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, de.method, de.n, de.args	Arguments controlling the choice of marker genes used for annotation, see trainSingleR .
aggr.ref, aggr.args	Arguments controlling the aggregation of the references prior to annotation, see trainSingleR .
recompute	Logical scalar indicating whether to set up indices for later recomputation of scores, when ref contains multiple references from which the individual results are to be combined. (See the difference between combineCommonResults and combineRecomputedResults .)
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 (or is a list that contains one or more such objects).
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	<i>Train the SingleR classifier</i>
--------------	-------------------------------------

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(),
  aggr.ref = FALSE,
  aggr.args = list(),
  recompute = TRUE,
  assay.type = "logcounts",
  check.missing = TRUE,
  BNPARAM = KmknnParam()
)
```

Arguments

ref	<p>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.</p> <p>Alternatively, a SummarizedExperiment object containing such a matrix.</p> <p>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.</p>
labels	<p>A character vector or factor of known labels for all samples in ref.</p> <p>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.</p>
genes	<p>A string specifying the feature selection method to be used, see Details.</p> <p>Alternatively, if ref is <i>not</i> a list, genes can be either:</p> <ul style="list-style-type: none"> • A list of lists of character vectors containing DE genes between pairs of labels. • A list of character vectors containing marker genes for each label. <p>If ref is a list, genes can be a list of lists of (lists of) character vectors, i.e., either of the two above choices. Each element of the outer-most list should contain markers for labels in the corresponding entry of ref.</p>
sd.thresh	<p>A numeric scalar specifying the minimum threshold on the standard deviation per gene. Only used when genes="sd".</p>

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 $500 * (2/3) ^ \log_2(N)$ 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".
aggr.ref	Logical scalar indicating whether references should be aggregated to pseudo-bulk samples for speed, see aggregateReference .
aggr.args	Further arguments to pass to aggregateReference when aggr.ref=TRUE.
recompute	Logical scalar indicating whether to set up indices for later recomputation of scores, when ref contains multiple references from which the individual results are to be combined. (See the difference between combineCommonResults and combineRecomputedResults .)
assay.type	An integer scalar or string specifying the assay of ref containing the relevant expression matrix, if ref is a SummarizedExperiment object (or is a list that contains one or more such objects).
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.

`search`: A List of additional information on the feature selection, for use by [classifySingleR](#). This includes `mode`, a string containing the selection method; `args`, method-specific arguments that can be re-used during classification; and `extras`, method-specific structures that can be re-used 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 `genesAB`, 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 [?combine-predictions](#) for an explanation). To this end, if `ref` is a list with multiple references, marker genes are identified separately within each reference when `genes="de"` or `"sd"`. Rank calculation and index construction is then performed within each reference separately.

Alternatively, genes can still be used to explicitly specify marker genes for each label in each of multiple references. This is achieved by passing a list of lists to `genes`, where each inner list corresponds to a reference in `ref` and can be of any format described in “Custom feature specification”. Thus, it is possible for `genes` to be - wait for it - a list (per reference) of lists (per label) of lists (per label) of character vectors.

If `recompute=TRUE`, the output is exactly equivalent to running `trainSingleR` on each reference separately. If `recompute=FALSE`, `trainSingleR` is also run each reference but the difference is that the final common set of genes consists of the union of common genes across all references. This is necessary to ensure that correlations are computed from the same set of genes across reference and are thus reasonably comparable in [combineCommonResults](#).

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).

In addition, it is generally unnecessary to have single-cell resolution on the reference profiles. We can instead set `aggr.ref=TRUE` to aggregate per-cell references into a set of pseudo-bulk profiles using [aggregateReference](#). This improves classification speed while using vector quantization to preserve within-label heterogeneity and mitigate the loss of information. Note that any aggregation is done *after* marker gene detection; this ensures that the relevant tests can appropriately penalize within-label variation. Users should also be sure to set the seed as the aggregation involves randomization.

Author(s)

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

See Also

[classifySingleR](#), where the output of this function gets used.

[combineCommonResults](#) and [combineRecomputedResults](#), to combine results from multiple references.

Examples

```
# Making up some data for a quick demonstration.
ref <- .mockRefData()

# 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 <- scan::pairwiseTTests(assay(ref, 2), ref$label, direction="up")
markers <- scan::getTopMarkers(by.t[[1]], by.t[[2]], n=10)
trained <- trainSingleR(ref, ref$label, genes=markers)
length(trained$common.genes)
```


Index

`.mockRefData`, 2
`.mockTestData (.mockRefData)`, 2

`aggregateReference`, 3, 41, 43

`BiocNeighborIndex`, 42
`BiocNeighborParam`, 13, 38, 41
`BiocParallelParam`, 4, 7, 13, 38
`BiocSingularParam`, 4
`BlueprintEncodeData`, 5

`classifySingleR`, 4, 6, 8, 11–14, 17, 27, 31, 34–36, 38, 41–43
`colData`, 3, 5, 6, 15, 16, 18, 19, 21–24, 26
`combine-predictions`, 10
`combineCommonResults`, 8, 9, 11, 11, 13, 14, 27, 31, 38, 41, 43
`combineRecomputedResults`, 8, 11, 12, 12, 27, 31, 38, 41, 43

`DatabaseImmuneCellExpressionData`, 15
`DataFrame`, 8, 11–14, 17, 27, 31, 35, 38

`getDeltaFromMedian`, 17, 36
`ggplot`, 28
`grid.arrange`, 28, 29, 32, 33

`HumanPrimaryCellAtlasData`, 18

`ImmGenData`, 10, 19

`kmeans`, 4

`List`, 7, 13, 37, 40, 41

`matchReferences`, 11, 20
`metadata`, 8, 12
`MonacoImmuneData`, 21
`MouseRNAseqData`, 23

`NovershternHematopoieticData`, 24

`pairwiseTTests`, 41
`pairwiseWilcox`, 41
`pheatmap`, 32, 33
`plotScoreDistribution`, 27
`plotScoreHeatmap`, 30
`pruneScores`, 8, 9, 17, 28, 29, 31, 33, 34

`runPCA`, 4

`SingleR`, 4, 11, 12, 14, 17, 20, 21, 27, 29–33, 35, 37
`SummarizedExperiment`, 2–4, 6, 7, 13, 16, 18–20, 22, 24, 26, 37, 38, 40, 41

`trainSingleR`, 4, 7–9, 13, 37, 38, 40