Package 'SingleR'

May 20, 2024

Title Reference-Based Single-Cell RNA-Seq Annotation

Version 2.6.0

Date 2024-02-13

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, BiocParallel, BiocSingular, stats, utils, Rcpp, beachmat, parallel

LinkingTo Rcpp, beachmat, BiocNeighbors

Suggests testthat, knitr, rmarkdown, BiocStyle, BiocGenerics, SingleCellExperiment, scuttle, scater, scran, scRNAseq, ggplot2, pheatmap, grDevices, gridExtra, viridis, celldex

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

SystemRequirements C++17

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 7.3.1

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

BugReports https://support.bioconductor.org/

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

git_branch RELEASE_3_19

git_last_commit 1999dd0

git_last_commit_date 2024-04-30

Repository Bioconductor 3.19

Date/Publication 2024-05-19

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>

Contents

`.mockRefData` *Mock data for examples*

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,
  ntop = 1000,
  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.
ntop	Integer scalar specifying the number of highly variable genes to use for the PCA step.
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 specfiying 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 runPCA.

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.

In practice, 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 heterogenity. 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`.

By default, we speed things up by only using the top `ntop` genes with the largest variances in the PCA. More subsetting of the matrix prior to the PCA can be achieved by setting `subset.row` to an appropriate indexing vector. This option may be useful for clustering based on known genes of interest but retaining all genes in the aggregated results. (If both options are set, subsetting by `subset.row` is done first, and then the top `ntop` genes are selected.) In both cases, though, the aggregation is performed on the full expression profiles.

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(scuttle)
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))
```

---

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,
  num.threads = bpnworkers(BPPARAM),
  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	Deprecated and ignored.
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).
num.threads	Integer scalar specifying the number of threads to use for classification.
BPPARAM	A BiocParallelParam object specifying the parallelization scheme to use for NA scanning, when check.missing=TRUE.

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.

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.
- labels, a character vector containing the predicted label. If fine.tune=FALSE, this is based only on the maximum entry in scores.
- delta.next, a numeric vector containing the difference between tbe best and next-best score. If fine.tune=TRUE, this is reported for scores after fine-tuning.
- pruned.labels, a character vector containing the pruned labels where "low-quality" labels 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 character vectors, containing the genes used to distinguish between each pair of labels.

If trained was generated from multiple references, the per-reference statistics are automatically combined into a single DataFrame of results using combineRecomputedResults. The output of combineRecomputedResults is then directly returned.

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 data with log-normalized expression values:
ref <- .mockRefData()
test <- .mockTestData(ref)

ref <- scuttle::logNormCounts(ref)
test <- scuttle::logNormCounts(test)

# Setting up the training:
trained <- trainSingleR(ref, label=ref$label)

# Performing the classification:
pred <- classifySingleR(test, trained)
table(predicted=pred$labels, truth=test$label)
```

`combineCommonResults` *Combine SingleR results with common genes*

Description

Combine results from multiple runs of `classifySingleR` (usually against different references) into a single DataFrame. This assumes that each run of `classifySingleR` was performed using a common set of marker genes.

Usage

```
combineCommonResults(results)
```

Arguments

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

Details

Here, the strategy is to performed classification separately within each reference, then collating the results to choose the label with the highest score across references. 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 those are available.)

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 `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`; and `label.origin`, a DataFrame specifying the reference of origin for each label in `scores`.

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 <- scuttle::logNormCounts(ref1)
ref2 <- scuttle::logNormCounts(ref2)
test <- scuttle::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.
```

combineRecomputedResults *Combine SingleR results with recomputation*

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,
  allow.lost = FALSE,
  warn.lost = TRUE,
  num.threads = bpnworkers(BPPARAM),
  BPPARAM = SerialParam()
)
```

Arguments

results	A list of DataFrame prediction results as returned by classifySingleR when run on each reference separately.
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 of Lists containing the trained outputs of multiple references, equivalent to either (i) the output of trainSingleR on multiple references with recompute=TRUE, or (ii) running trainSingleR on each reference separately and manually making a list of the trained outputs.
quantile	Numeric scalar specifying the quantile of the correlation distribution to use for computing the score, see 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.
check.missing	Logical scalar indicating whether rows should be checked for missing values (and if found, removed).

allow.lost	Deprecated.
warn.lost	Logical scalar indicating whether to emit a warning if markers from one reference in trained are "lost" in other references.
num.threads	Integer scalar specifying the number of threads to use for index building and classification.
BPPARAM	A BiocParallelParam object specifying how parallelization should be performed in other steps, see ?trainSingleR and ?classifySingleR for more details.

Details

Here, the strategy is to perform classification separately within each reference, then collate the results to choose the label with the highest score across references. 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.

A key aspect of this approach is that each entry of results is generated with reference-specific markers. This avoids the inclusion of noise from irrelevant genes in the within-reference assignments. Similarly, the common feature space for each cell is defined from the most relevant markers across all references, analogous to one iteration of fine-tuning using only the best labels from each reference. Compare this to the alternative approach of creating a common feature space, where we force all per-reference classifications to use the same set of markers; this would slow down each individual classification as many more genes are involved.

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 pruned.labels if these were also present in results.

The metadata contains label.origin, a DataFrame specifying the reference of origin for each label in scores.

Dealing with mismatching gene availabilities

It is recommended that the universe of genes be the same across all references in trained. (Or, at the very least, markers used in one reference are available in the others.) This ensures that a common feature space can be generated when comparing correlations across references. Differences in the

availability of markers between references will have unpredictable effects on the comparability of correlation scores, so a warning will be emitted by default when `warn.lost=TRUE`. Callers can protect against this by subsetting each reference to the intersection of features present across all references - this is done by default in `SingleR`.

That said, this requirement may be too strict when dealing with many references with diverse feature annotations. In such cases, the recomputation for each reference will automatically use all available markers in that reference. The idea here is to avoid penalizing all references by removing an informative marker when it is only absent in a single reference. We hope that the recomputed scores are still roughly comparable if the number of lost markers is relatively low, coupled with the use of ranks in the calculation of the Spearman-based scores to reduce the influence of individual markers. This is perhaps as reliable as one might imagine.

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 <- scuttle::logNormCounts(test)

ref1 <- scuttle::logNormCounts(ref1)
train1 <- trainSingleR(ref1, labels=ref1$label)
pred1 <- classifySingleR(test, train1)

ref2 <- scuttle::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]
```

`datasets` *Reference dataset extractors*

Description

These dataset getter functions are deprecated as they have been migrated to the **celldex** package for more general use throughout the Bioconductor package ecosystem.

Usage

```
HumanPrimaryCellAtlasData(...)

BlueprintEncodeData(...)

ImmGenData(...)

MouseRNAseqData(...)

DatabaseImmuneCellExpressionData(...)

NovershternHematopoieticData(...)

MonacoImmuneData(...)
```

Arguments

`...`	Further arguments to pass to the **celldex** function of the same name.

Value

A SummarizedExperiment object containing the reference dataset.

Author(s)

Aaron Lun

`getClassicMarkers` *Get classic markers*

Description

Find markers between pairs of labels using the "classic" approach, i.e., based on the log-fold change between the medians of labels.

Usage

```
getClassicMarkers(
  ref,
  labels,
  assay.type = "logcounts",
  check.missing = TRUE,
  de.n = NULL,
  num.threads = bpnworkers(BPPARAM),
  BPPARAM = SerialParam()
)
```

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`.
`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).
`de.n`	An integer scalar specifying the number of DE genes to use. Defaults to `500 * (2/3) ^ log2(N)` where `N` is the number of unique labels.
`num.threads`	Integer scalar specifying the number of threads to use.
`BPPARAM`	A BiocParallelParam object specifying how parallelization should be performed.

Details

This function implements the classic mode of marker detection in **SingleR**, based only on the magnitude of the log-fold change between labels. In many respects, this approach may be suboptimal as it does not consider the variance within each label and has limited precision when the expression values are highly discrete. Nonetheless, it is often the only possible approach when dealing with reference datasets that lack replication and thus cannot be used with more advanced marker detection methods.

If multiple references are supplied, ranking is performed based on the average of the log-fold changes within each reference. This avoids comparison of expression values across references that can be distorted by batch effects. If a pair of labels does not co-occur in at least one reference, no attempt is made to perform the comparison and the corresponding character vector is left empty in the output.

The character vector corresponding to the comparison of a label to itself is always empty.

Value

A list of lists of character vectors, where both the outer and inner lists have names equal to the unique levels of `labels`. The character vector contains the names of the top `de.n` genes with the largest positive log-fold changes in one label (entry of the outer list) against another label (entry of the inner list).

Author(s)

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

See Also

`trainSingleR` and `SingleR`, where this function is used when `genes="de"` and `de.method="classic"`.

Examples

```
ref <- .mockRefData()
ref <- scuttle::logNormCounts(ref)
out <- getClassicMarkers(ref, labels=ref$label)
str(out)

# Works with multiple references:
ref2 <- .mockRefData()
ref2 <- scuttle::logNormCounts(ref2)
out2 <- getClassicMarkers(list(ref, ref2), labels=list(ref$label, ref2$label))
str(out2)
```

`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))
```

`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)
```

---

plotDeltaDistribution *Plot delta distributions*

Description

Plot the distribution of deltas (i.e., the gap between the assignment score for the assigned label and those of the remaining labels) across cells assigned to each reference label.

Usage

```
plotDeltaDistribution(
  results,
  show = c("delta.med", "delta.next"),
  labels.use = colnames(results$scores),
  references = NULL,
  chosen.only = TRUE,
  size = 2,
  ncol = 5,
  dots.on.top = TRUE,
  this.color = "#000000",
  pruned.color = "#E69F00",
  grid.vars = list()
)
```

Arguments

results	A DataFrame containing the output from SingleR, classifySingleR, combineCommonResults, or combineRecomputedResults.
show	String specifying whether to show the difference from the median ("delta.med") or the difference from the next-best score ("delta.next").
labels.use	Character vector specifying the labels to show in the plot facets. Defaults to all labels in results.
references	Integer scalar or vector specifying the references to visualize. This is only relevant for combined results, see Details.

chosen.only	Logical scalar indicating whether to only show deltas for individual labels that were chosen as the final label in a combined result.
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 scalar 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.
grid.vars	Named list of extra variables to pass to grid.arrange, used to arrange the multiple plots generated when references is of length greater than 1.

Details

This function creates jitter and violin plots showing the deltas for all cells across one or more labels. The idea is to provide a visual diagnostic for the confidence of assignment of each cell to its label. The show argument determines what values to show on the y-axis:

- "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 scores of each cell at the last round of fine-tuning.

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". show="delta.next" is guaranteed to be positive but may be overly stringent for references involving very similar labels.

Pruned calls are identified as NAs in the pruned.labels field in results. Points corresponding to cells with pruned calls are colored by pruned.color; this can be disabled by setting pruned.color=NA.

For combined results (see ?combineRecomputedResults), this function will show the deltas faceted by the assigned label within each individual reference. The references to show in this manner can be specified using the references argument, entries of which refer to columns of results$orig.results.

By default, a separate plot is created for each individual reference in a combined results. Deltas are only shown in each plot if the label in the corresponding reference was chosen as the overall best label in the combined results. However, this can be changed to show all deltas for an individual reference by setting chosen.only=FALSE.

Value

If references specifies a single set of deltas, a ggplot object is returned showing the deltas in violin plots.

If references specifies multiple deltas for a combined result, multiple ggplot objects are generated in a grid on the current graphics device.

If delta.use specifies multiple deltas and grid.vars is set to NULL, a list is returned containing the ggplot objects for manual display.

Author(s)

Daniel Bunis and Aaron Lun

See Also

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

plotScoreDistribution and plotScoreHeatmap, for alternative diagnostic plots.

Examples

```
example(SingleR, echo=FALSE)

# Showing the delta to the median:
plotDeltaDistribution(pred)

# Showing the delta to the next-highest score:
plotDeltaDistribution(pred, show = "delta.next")

# Multi-reference compatibility:
example(combineRecomputedResults, echo = FALSE)

plotDeltaDistribution(results = combined)

plotDeltaDistribution(results = combined, chosen.only=FALSE)

# Tweaking the grid controls:
plotDeltaDistribution(combined, grid.vars = list(ncol = 2))
```

---

plotScoreDistribution *Plot score distributions*

Description

Plot the distribution of assignment scores across all cells assigned to each reference label.

Usage

```
plotScoreDistribution(
  results,
  show = NULL,
  labels.use = colnames(results$scores),
  references = NULL,
  scores.use = NULL,
  calls.use = 0,
  pruned.use = NULL,
  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	Deprecated, use plotDeltaDistribution instead for show!="scores".
labels.use	Character vector specifying the labels to show in the plot facets. Defaults to all labels in results.
references	Integer scalar or vector specifying the references to visualize. This is only relevant for combined results, see Details.
scores.use	Deprecated, see references.
calls.use	Deprecated and ignored.
pruned.use	Deprecated and ignored.
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 scalar 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, show.min.diff	Deprecated, use plotDeltaDistribution instead.
grid.vars	Named list of extra variables to pass to grid.arrange, used to arrange the multiple plots generated when references is of length greater than 1.

Details

This function creates jitter and violin plots showing assignment scores for all cells across one or more labels. Each facet represents a label in labels.use and contains three violin plots:

- "Assigned", containing scores for all cells assigned to that label. Colored according to this.color.
  "Pruned", containing scores for all cells assigned to that label but pruned out, e.g., by pruneScores. Colored according to pruned.color, and can be omitted by setting pruned.color=NA.
- "Other", containing the scores for all cells assigned to other labels. Colored according to other.color.

The expectation is that the former is higher than the latter, though the deltas generated by plotDeltaDistribution are often more informative in this regard.

For combined results (see ?combineRecomputedResults), this function can show both the combined and individual scores. This is done using the references argument, entries of which refer to columns of results$orig.results if positive or to the combined results if zero. For example:

- If we set `references=2`, we will plot the scores from the second individual reference.
- If we set `references=1:2`, we will plot the scores from first and second references (in separate plots) faceted by their corresponding labels.
- By default, the function will create a separate plot for the combined scores and each individual reference, equivalent to `references=0:N` for `N` individual references.

Value

If `references` specifies a single set of scores, a ggplot object is returned showing the scores in violin plots.

If `references` specifies multiple scores for a combined result, multiple ggplot objects are generated in a grid on the current graphics device.

If `references` specifies multiple scores and `grid.vars=NULL`, a list is returned containing the ggplot objects for manual display.

Author(s)

Daniel Bunis and Aaron Lun

See Also

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

`plotDeltaDistribution` and `plotScoreHeatmap`, for alternative diagnostic plots.

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")

# For multiple references, default output will contain separate plots for
# each original reference as well as for the the combined scores.
example(combineRecomputedResults, echo = FALSE)
plotScoreDistribution(results = combined)

# 'references' specifies which original results to plot distributions for.
plotScoreDistribution(results = combined, references = 0)
plotScoreDistribution(results = combined, references = 1:2)

# Tweaking the grid arrangement:
plotScoreDistribution(combined, grid.vars = list(ncol = 2))
```

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"),
  rows.order = NULL,
  scores.use = NULL,
  calls.use = 0,
  na.color = "gray30",
  color = NA,
  breaks = NA,
  legend_breaks = NA,
  legend_labels = NA,
  cluster_cols = FALSE,
  annotation_col = NULL,
  show_colnames = FALSE,
  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 (i.e., rows of results) to show. If NULL, all cells are shown.
labels.use	Character vector specifying the labels to show in the heatmap rows. Defaults to all labels in results.
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 assigned labels 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. Ignored if cluster_cols is set.

order.by — String providing the annotation to be used for cells/columns ordering. Can be "labels" (default) or "clusters" (when provided). Ignored if cells.order or cluster_cols are specified.

rows.order — String vector specifying how to order rows of the heatmap. Contents should be the reference-labels in the order you would like them to appear, from top-to-bottom. For combined results, include labels for all plots in a single vector and labels relevant to each plot will be extracted.

scores.use — Integer scalar or vector specifying the individual annotation result from which to take scores. This is only relevant for combined results, see Details.

calls.use — Integer scalar or vector specifying the individual annotation result from which to take labels, for use in the annotation bar when show.labels=TRUE. This is only relevant for combined results, see Details.

na.color — String specifying the color for non-calculated scores of combined results. This will always be displayed in the legend if any NA values are present in the scores.

color — Character vector of colors passed to pheatmap. If NA and normalize=TRUE, the viridis color scheme is used by default; while if normalize=FALSE, a default red-blue color scheme is chosen that should be symmetric around zero (see breaks).

breaks — Numeric vector to map scores to colors, see the argument of the same name in pheatmap. If NA, this defaults to a sequence from 0 to 1 when normalize=TRUE, or a sequence from -T to T where T is the largest absolute score when normalize=FALSE.

legend_breaks, legend_labels — Arguments passed to pheatmap to label the legend. If NA, only the legend extremes are labelled by default; and when normalize=TRUE, the legend extremes are only labelled as "Lower" and "Higher", as actual normalized values have little meaning.

annotation_col, cluster_cols, show_colnames, silent, ... — Additional parameters for heatmap control passed to pheatmap.

grid.vars — A named list of extra variables to pass to grid.arrange, used to arrange the multiple plots generated when scores.use is of length greater than 1.

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 showing the label assigned to each cell. This is also used to order the columns for a more organized visualization when order.by="label". 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. The remaining labels are then selected based on:

- Labels with max z-scores after per-cell centering and scaling of the scores matrix, if results does not contain combined scores.
- Labels which were suggested most frequently by individual references, if results contains combined scores.

Value

If scores.use specifies a single set of scores, the output of pheatmap is returned showing the heatmap on the current graphics device.

If scores.use specifies multiple scores for a combined result, multiple heatmaps are generated in a grid on the current graphics device.

If scores.use specifies multiple scores and grid.vars is set to NULL, a list is returned containing the pheatmap globs for manual display.

Working with combined results

For combined results (see ?combineRecomputedResults), this function can show both the combined and individual scores or labels. This is done using the scores.use and calls.use arguments, entries of which refer to columns of results$orig.results if positive or to the combined results if zero. For example:

- If we set scores.use=2 and calls.use=1, we will plot the scores from the second individual reference with the annotation bar containing label assignments from the first reference.
- If we set scores.use=1:2 and calls.use=1:2, we will plot the scores from first and second references (in separate plots) with the annotation bar in each plot containing the corresponding label assignments.
- By default, the function will create a separate plot the combined scores and each individual reference. In each plot, the annotation bar contains the combined labels; this is equivalent to scores.use=0:N and calls.use=0 for N individual references.

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.

Note that this transformation is not dependent on the choice of the top `max.labels` labels. Altering `max.labels` will not change the normalized values, only the labels that are shown. However, the transformation will respond to `labels.use`.

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 <- test$label

# 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 defaluts
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

`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 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% of observations lie within 3 standard deviations from the mean. 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))
```

`rebuildIndex` *Rebuild the index*

Description

Rebuild the index (or indices), typically after restarting the R session. This is because the indices are held in external memory and are not serialized correctly by R.

Usage

```
rebuildIndex(trained, num.threads = 1)
```

Arguments

`trained`	List containing the output of `trainSingleR`, possibly after some operations that invalidate the indices.
`num.threads`	Integer specifying the number of threads to use for training.

Value

`trained` is returned with valid indices. If it already had valid indices, this function is a no-op.

Author(s)

Aaron Lun

Examples

```
# Making up the training set.
ref <- .mockRefData()
ref <- scuttle::logNormCounts(ref)
trained <- trainSingleR(ref, ref$label)
trained$built # a valid address

# Saving and reloading the index.
tmp <- tempfile(fileext=".rds")
saveRDS(trained, file=tmp)
reloaded <- readRDS(tmp)
reloaded$built # not valid anymore

rebuilt <- rebuildIndex(reloaded)
rebuilt$built # back to validity
```

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 = NULL,
  clusters = NULL,
  genes = "de",
  sd.thresh = 1,
  de.method = "classic",
  de.n = NULL,
  de.args = list(),
  aggr.ref = FALSE,
  aggr.args = list(),
  recompute = TRUE,
  restrict = NULL,
  quantile = 0.8,
  fine.tune = TRUE,
  tune.thresh = 0.05,
  prune = TRUE,
  assay.type.test = "logcounts",
  assay.type.ref = "logcounts",
  check.missing = TRUE,
  num.threads = bpnworkers(BPPARAM),
  BNPARAM = NULL,
  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` Deprecated.

`clusters` A character vector or factor of cluster identities for each cell in `test`. If set, annotation is performed on the aggregated cluster profiles, otherwise it defaults to per-cell annotation.

`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` Deprecated and ignored.

`restrict` A character vector of gene names to use for marker selection. By default, all genes in `ref` are used.

`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).

`num.threads` Integer scalar specifying the number of threads to use for index building and classification.

`BNPARAM` Deprecated and ignored.

`BPPARAM` A BiocParallelParam object specifying how parallelization should be performed in other steps, see ?`trainSingleR` and ?`classifySingleR` for more details.

Details

This function is just a convenient wrapper around `trainSingleR` and `classifySingleR`. The function will automatically restrict the analysis to the intersection of the genes in both `ref` and `test`. If this intersection is empty (e.g., because the two datasets use different gene annotations), an error will be raised.

If `clusters` is specified, per-cell profiles are summed to obtain per-cluster profiles. Annotation is then performed by running `classifySingleR` on these profiles. This yields a DataFrame with one row per level of `clusters`.

The default settings of this function are based on the assumption that `ref` contains or bulk data. If it contains single-cell data, this usually requires a different `de.method` choice. Read the Note in `?trainSingleR` for more details.

Value

A DataFrame is returned containing the annotation statistics for each cell (one cell per 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 data with log-normalized expression values:
ref <- .mockRefData()
test <- .mockTestData(ref)

ref <- scuttle::logNormCounts(ref)
test <- scuttle::logNormCounts(test)

# Running the classification with different options:
pred <- SingleR(test, ref, labels=ref$label)
table(predicted=pred$labels, truth=test$label)

k.out<- kmeans(t(assay(test, "logcounts")), center=5) # mock up a clustering
pred2 <- SingleR(test, ref, labels=ref$label, clusters=k.out$cluster)
table(predicted=pred2$labels, cluster=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 = NULL,
  de.method = c("classic", "wilcox", "t"),
  de.n = NULL,
  de.args = list(),
  aggr.ref = FALSE,
  aggr.args = list(),
  recompute = TRUE,
  restrict = NULL,
  assay.type = "logcounts",
  check.missing = TRUE,
  approximate = FALSE,
  num.threads = bpnworkers(BPPARAM),
  BNPARAM = NULL,
  BPPARAM = SerialParam()
)
```

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 containing `"de"`, indicating that markers should be calculated from `ref`. For back compatibility, other string values are allowed but will be ignored with a deprecation warning. Alternatively, if `ref` is *not* a list, `genes` can be either: • 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. If `ref` *is* a list, `genes` can be a list of length equal to `ref`. Each element of the list should be one of the two above choices described for non-list `ref`, containing markers for labels in the corresponding entry of `ref`.
`sd.thresh`	Deprecated and ignored.
`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. Ignored if `genes` is a list of markers/DE genes.

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. Ignored if genes is a list of markers/DE genes.
de.args	Named list of additional arguments to pass to pairwiseTTests or pairwiseWilcox when de.method="wilcox" or "t". Ignored if genes is a list of markers/DE genes.
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	Deprecated and ignored.
restrict	A character vector of gene names to use for marker selection. By default, all genes in ref are used.
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).
approximate	Logical scalar indicating whether a faster approximate method should be used to compute the quantile.
num.threads	Integer scalar specifying the number of threads to use for index building.
BNPARAM	Deprecated and ignored.
BPPARAM	A BiocParallelParam object specifying how parallelization should be performed. Relevant for marker detection if genes = NULL, aggregation if aggr.ref = TRUE, and NA checking if check.missing = TRUE.

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.

The automatic marker detection (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 expression values are expected to be log-transformed and normalized.

If restrict is specified, ref is subsetted to only include the rows with names that are in restrict. Marker selection and all subsequent classification will be performed using this restrictive subset of genes. This can be convenient for ensuring that only appropriate genes are used (e.g., not pseudogenes or predicted genes).

Value

For a single reference, a List is returned containing:

`built`: An external pointer to various indices in C++ space. Note that this cannot be serialized and should be removed prior to any `saveRDS` step.

`ref`: The reference expression matrix. This may have fewer columns than the input `ref` if `aggr.ref = TRUE`.

`markers`: A list containing `unique`, a character vector of all marker genes used in training; and `full`, a list of list of character vectors containing the markers for each pairwise comparison between labels.

`labels`: A list containing `unique`, a character vector of all unique reference labels; and `full`, a character vector containing the assigned label for each column in `ref`.

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` is manually passed, `ref` can be the raw counts or any monotonic transformation thereof. There is no need to supply (log-)normalized expression values for the benefit of the automatic marker detection. Similarly, for manual `genes`, the values of `de.method`, `de.n` and `sd.thresh` have no effect.

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 ?combineRecomputedResults for an explanation). To this end, if ref is a list with multiple references, marker genes are identified separately within each reference if genes = NULL. Rank calculation and index construction is then performed within each reference separately. The result is identical to lapplying over a list of references and runing trainSingleR on each reference.

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.

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.

combineRecomputedResults, to combine results from multiple references.

rebuildIndex, to rebuild the index after external memory is invalidated.

Examples

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

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

trained <- trainSingleR(ref, ref$label)
trained
```

```
length(trained$markers$unique)

# 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$markers$unique)
```

Index