Package 'BEARscc'

April 15, 2019

Type Package

Title BEARscc (Bayesian ERCC Assesstment of Robustness of Single Cell Clusters)

Version 1.2.1

Author David T. Severson <david_severson@hms.harvard.edu>

Maintainer Benjamin Schuster-Boeckler <benjamin.schuster-boeckler@ludwig.ox.ac.uk>

Description BEARscc is a noise estimation and injection tool that is designed to assess putative single-cell RNA-seq clusters in the context of experimental noise estimated by ERCC spike-in controls.

License GPL-3

Imports ggplot2, SingleCellExperiment, data.table, stats, utils, graphics, compiler

Suggests testthat, cowplot, knitr, rmarkdown, BiocStyle, NMF

VignetteBuilder knitr

RoxygenNote 6.0.1

biocViews ImmunoOncology, SingleCell, Clustering, Transcriptomics

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

git_branch RELEASE_3_8

git_last_commit b16e1c3

git_last_commit_date 2019-01-04

Date/Publication 2019-04-15

R topics documented:

BEARscc-package *BEARscc (Bayesian ERCC Assesstment of Robustness of Single Cell Clusters)*

Description

BEARscc is a noise estimation and injection tool that is designed to assess putative single-cell RNA-seq clusters in the context of experimental noise estimated by ERCC spike-in controls.

Details

Single-cell transcriptome sequencing data are subject to substantial technical variation and batch effects that can confound the classification of cellular sub-types. Unfortunately, current clustering algorithms don't account for this uncertainty. To address this shortcoming, we have developed a noise perturbation algorithm called BEARscc that is designed to determine the extent to which classifications by existing clustering algorithms are robust to observed technical variation.

BEARscc makes use of ERCC spike-in measurements to model technical variance as a function of gene expression and technical dropout effects on lowly expressed genes. In our benchmarks, we found that BEARscc accurately models read count fluctuations and drop-out effects across transcripts with diverse expression levels. Applying our approach to publicly available single-cell transcriptome data of mouse brain and intestine, we have demonstrated that BEARscc identified cells that cluster consistently, irrespective of technical variation. For more details, see the manuscript that is now available on bioRxiv.

Author(s)

David T. Severson <david_severson@hms.harvard.edu>

Maintainer: Benjamin Schuster-Boeckler <benjamin.schuster-boeckler@ludwig.ox.ac.uk>

References

Source code and README: <https://bitbucket.org/bsblabludwig/bearscc/overview> Associated preprint: <https://www.biorxiv.org/content/early/2017/06/05/118919>

analysis_examples *BEARscc downstream example objects.*

Description

The `analysis_examples` Rdata object contains downstream data objects for use in various help pages for dynamic execution resulting from running tutorial in README and vignette on `BEARscc_examples`. The objects are a result of applying BEARscc functions as described in the README found at https://bitbucket.org/bsblabludwig/bearscc.git or the vignette that accompanies this package.

Usage

```
data("analysis_examples")
```

Format

An R data file with:

data.frame	"BEARscc_clusts.df"	Toy dataframe of previously computed cluster labels for each cell across various cl
	[,1]	2 cluster case
	[,2]	3 cluster case
	[,3]	4 cluster case
	[,4]	5 cluster case
	[,5]	Original cluster case
data.frame	"clusters.df"	Toy dataframe of previously computed hierarchical clustering of 10 BEARscc simu
	[,1]	Cluster labels from simulated replicate 1.
	[,2]	Cluster labels from simulated replicate 2.
	[,3]	Cluster labels from simulated replicate 3.
	[,4]	Cluster labels from simulated replicate 4.
	[,5]	Cluster labels from simulated replicate 5.
	[,6]	Cluster labels from simulated replicate 6.
	[,7]	Cluster labels from simulated replicate 7.
	[,8]	Cluster labels from simulated replicate 8.
	[,9]	Cluster labels from simulated replicate 9.
	[,10]	Cluster labels from simulated replicate 10.
	[,11]	Original cluster case
function	"recluster"	A function used to quickly illustrate replicate-wise clustering and the resulting con
matrix	"noise_consensus"	50 by 50 matrix of previously computed output from `compute_consensus()`, whic
SCEList	"BEAR_analyzed.sce"	A `SingleCellExperiment` object discussed extensively in the vignette and ReadM

Value

An R data file with a `function`, `matrix`, two `data.frame` objects and a `SingleCellExperiment` object.

Source

These data are the result of running the README, https://bitbucket.org/bsblabludwig/bearscc.git, on a subset of observations obtained by Drs. Michael White and Richard Owen in the Xin Lu Lab. Samples were sequenced by the Wellcome Trust Center for Genomics, Oxford, UK. The original data used to generate these objects are available in full with GEO accession number, GSE95155.

References

Source code and README: <https://bitbucket.org/bsblabludwig/bearscc/overview> Associated preprint: <https://www.biorxiv.org/content/early/2017/06/05/118919>

Examples

```
data(analysis_examples)
```

BEARscc_examples *Example data for BEARscc.*

Description

A toy dataset for applying BEARscc functions as described in the README on https://bitbucket.org/bsblabludwig/bearsc and vignette accompanying this package on Bioconductor.

Usage

```
data("BEARscc_examples")
```

Format

And R data file with:

df	"data.counts.df"	Toy dataframe of endogenous counts with 117 genes and 50 samples.
df	"ERCC.counts.df"	Toy dataframe of ERCC counts with 57 spike-ins and 50 samples.
df	"ERCC.meta.df"	Toy dataframe of spike-in concentration values and spike-in labels as row names.
	[,1]	Spike-in actual concentration.
SCEList	"BEAR_examples.sce"	A `SingleCellExpression` object described in more detail in the accompanying vig

Value

An R data file containing three `data.frame` objects and a single `SingleCellExpression` object for the purpose of tutorials, testing, and help file examples.

Source

These data are a subset of observations Drs. Michael White and Richard Owen in the Xin Lu Lab. Samples were sequenced by the Wellcome Trust Center for Genomics, Oxford, UK. These data are available in full with GEO accession number, GSE95155.

References

Source code and README: <https://bitbucket.org/bsblabludwig/bearscc/overview> Associated preprint: <https://www.biorxiv.org/content/early/2017/06/05/118919>

Examples

```
data(BEARscc_examples)
```

`cluster_consensus` *Cluster the consensus matrix.*

Description

This function will perform hierarchical clustering on the noise consensus matrix allowing the user to investigate the appropriate number of clusters, *k*, considering the noise within the experiment.

Usage

```
cluster_consensus(consensus_matrix, cluster_num, method = "complete")
```

Arguments

`consensus_matrix`	A noise consensus output by `compute_consensus()`.
`cluster_num`	The number of clusters expected from the hierarchical clustering of the noise consensus matrix.
`method`	The hierarchical clustering method to be used on the consensus.

Details

We have found it useful to identify the optimal number of clusters in terms of resiliance to noise by examining these metrics by cutting hierarchical clustering dendograms of the noise consensus and comparing the results to the original clustering labels. To do this create a vector containing each number of clusters one wishes to examine (the function automatically determines the results for the dataset as a single cluster) and then cluster the consensus with this function.

Frequently one will want to assess multiple possible cluster number situations at once. In this case it is recommended that one use a `lapply` in conjunction with a vector of all biologically reasonable cluster numbers to fulfill the task of attempting to identify the optimal cluster number.

Value

The output is a vector of cluster labels based on hierarchical clustering of the noise consensus. In the event that a vector is supplied for number of clusters in conjunction with `lapply`, then the output is a data.frame of the cluster labels for each of the various number of clusters deemed biologically reasonable by the user.

Author(s)

David T. Severson <david_severson@hms.harvard.edu>

Maintainer: Benjamin Schuster-Boeckler <benjamin.schuster-boeckler@ludwig.ox.ac.uk>

See Also

`compute_consensus` `report_cluster_metrics` `report_cell_metrics`

Examples

```
data(analysis_examples)

vector <- seq(from=2, to=5, by=1)
BEARscc_clusts.df <- cluster_consensus(noise_consensus, vector)
BEARscc_clusts.df
```

`compute_consensus` *Compute consensus matrix.*

Description

Computes the consensus matrix using a data.frame of cluster labels across different BEARscc simulated technical replicates.

Usage

```
compute_consensus(cluster_labels)
```

Arguments

`cluster_labels`	A data.frame of labels assigned to each sample (rownames) across various simulated technical replicates designed by BEARscc (colnames).

Details

We provide a visual and quantitative representation of the clustering variation on a cell-by-cell level by using cluster labels to compute the number of times any given pair of cells associates in the same cluster; this forms the 'noise consensus matrix'. Each element of this matrix represents the fraction of simulated technical replicates in which two cells cluster together (the 'association frequency'), after using a clustering method of the user's choice to generate a data.frame of clustering labels. This consensus matrix may be used to compute BEARscc metrics at both the cluster and cell level.

Value

When the number of samples are n, then the noise consensus resulting from this function is an n x n matrix describing the fraction of simulated technical replicates in which each cell of the experiment associates with another cell.

A brief description of subfunctions

`compute_consensus` relies on the following subfunction to compute the noise consensus. This function obtains all of the necessary information form the options of `compute_consensus`.

- `names=rownames(cluster_labels)`
- `create_cm(cluster_labels, names)`

Author(s)

David T. Severson <david_severson@hms.harvard.edu>

Maintainer: Benjamin Schuster-Boeckler <benjamin.schuster-boeckler@ludwig.ox.ac.uk>

See Also

- cluster_consensus()
- report_cluster_metrics()
- report_cell_metrics()

Examples

```
data("analysis_examples")

noise_consensus <- compute_consensus(clusters.df)
noise_consensus
```

---

estimate_noiseparameters

Estimates noise in single cell data.

Description

Estimates the drop-out model and technical variance from spike-ins present in the sample.

Usage

```
estimate_noiseparameters(SCEList, plot=FALSE, sd_inflate=0, max_cumprob=0.9999,
    bins=10, write.noise.model=TRUE, file="noise_estimation",
    dropout_inflate=1, model_view=c("Observed", "Optimized"),
    alpha_resolution=0.005, tie_function="maximum")
```

Arguments

SCEList	A SingleCellExperiment object that must contain the observed counts matrix as "observed_expression" in assays, and must have the relevant spike-in samples identified using isSpike() as well as contain the expected actual concentrations of these spike-ins as spikeConcentrations in metadata. Please see the vignette for more detail about constructing the appropriate SCEList.
plot	When plot=TRUE produces plots to investigate quality of data fits with root file name set by file option.
sd_inflate	An optional parameter to modulate the estimated noise. The estimated standard deviation of spike-ins can be scaled by this factor. We recommend leaving the value at the default of 0.
bins	The parameter determines the number of bins for comparison of the quality of fit between the mixed-model and observed data for each spike-in alpha in order to calculate the relationship between alpha and mean in the noise model. This should be set lower for small datasets and higher for datasets with more observations
max_cumprob	Because a cumulative distribution will range from n=0 to a countable infinity, the event space needs to be set to cover a reasonable fraction of the probability density. This parameter determines the the fraction of probability density covered by the event space, which in turn defines the highes count number in the event space. We recommend users use the default value of 0.9999.

`write.noise.model`
: When `write.noise.model=TRUE` outputs two tab-delimited files containing the dropout effects and noise model parameters; this allows users to apply the noise generation on a seperate high compute node. The root file name is set by `file` option.

`file`
: Describes the root name for files written out by `write.noise.model` and `plot` options.

`dropout_inflate`
: A scaling parameter for increasing explicitly the number of drop-outs present beyond those estimated by spike-ins. The value must be greater than 0 or an error will occur. Values below one will diminish drop-outs in simulated replicates, and values above one will increase drop-outs in simulated replicates. We recommend users use the default value of 1.

`model_view`
: `model_view=c("Observed", "Optimized", "Poisson", "Neg. Binomial"` determines the statistical distributions that should be plotted for the ERCC plots output by `plot=TRUE`.

`alpha_resolution`
: Because the alpha parameter is enumerated discretely and empirically evaluated for each value for each spike-in, it is necessary to specify the resolution (how small the step is between each explicit alpha test); this parameter defines the resolution of alpha values tested for maximum empirical fit to spike-ins. It is recommended that users utilize the default resolution.

`tie_function`
: The parameter `tie_function=c("minimum", "maximum")` tells BEARscc how to handle a tie alpha value for fitting the mixture model to an individual spike-in. If `maximum`, then BEARscc will chose the maximum alpha value with the best fit; conversely, if `minimum` is set, then BEARscc will choose the minimum alpha value with the best fit.

Details

BEARscc consists of three steps: modelling technical variance based on spike-ins (Step 1); simulating technical replicates (Step 2); and clustering simulated replicates (Step 3). In Step 1, an experiment-specific model of technical variability ("noise") is estimated using observed spike-in read counts. This model consists of two parts. In the first part, expression-dependent variance is approximated by fitting read counts of each spike-in across cells to a mixture model (see Methods). The second part, addresses drop-out effects. Based on the observed drop-out rate for spike-ins of a given concentration, the 'drop-out injection distribution' models the likelihood that a given transcript concentration will result in a drop-out. The 'drop-out recovery distribution' is estimated from the drop-out injection distribution using Bayes' theorem and models the likelihood that a transcript that had no observed counts in a cell was a false negative. This function performs the first step of BEARscc. For further algorithmic detail please refer to our manuscript methods.

Value

The resulting output of `estimate_noiseparameters()` is another `SingleCellExperiment` class object; however four new annotations that describe the drop-out and variance models computed by `BEARscc` have been added to the `metadata` of the `SingleCellExperiment` object. Specifically.

`dropout_parameters`
: A `data.frame` listing gene-wise parameters necessary for computing drop-oout recovery and injection probabilities in order to define the two drop-out models for zero observation and positive values within the drop-out range by `simulate_replicates()`.

spikein_parameters
: A data.frame of the estimated noise model parameters utilized by simulate_replicates() to simulate replicates in non-zero observations.

genewiseDropouts
: A data.frame of the estimated probabilities used in the Bayes' calculation of the probabilities described in dropout_parameters. While these are not use in further analysis, they are supplied here for the user's reference.

Note

Frequently, the user will want to compute simulated technical replicates in a high performance computational environment. While the function outputs the necessary information for create_noiseinjected_counts(), with the option write.noise.model=TRUE users are able to save two tab delimited files necessary to run HPC_generate_noise_matrices.R on a high performance computational cluster. The option file is used to indicate the desired root label of the files, "*_bayesianestimates.xls" and "*_parameters4randomize.xls".

In the examples section, the parameter, alpha_resolution is set to 0.25, which is a terrible resolution for estimating noise, but allows the example to run in reasonable to time for checking the help files. We recommend the default parameter: alpha_resolution=0.005.

Author(s)

David T. Severson <david_severson@hms.harvard.edu>

Maintainer: Benjamin Schuster-Boeckler <benjamin.schuster-boeckler@ludwig.ox.ac.uk>

Examples

```
library("SingleCellExperiment")
data("BEARscc_examples")

#For execution on local machine
BEAR_examples.sce <- estimate_noiseparameters(BEAR_examples.sce,
    alpha_resolution=0.25, write.noise.model=FALSE)
BEAR_examples.sce

#To save results as files for abnalysis on a
#high performance computational cluster
estimate_noiseparameters(BEAR_examples.sce, write.noise.model=TRUE,
    alpha_resolution=0.25, file="noise_estimation",
    model_view=c("Observed","Optimized"))
```

---

report_cell_metrics *Reports BEARscc metrics for cells.*

Description

To quantitatively evaluate the results, three metrics are calculated from the noise consensus matrix: 'stability' is the average frequency with which cells within a cluster associate with each other across simulated replicates; 'promiscuity' measures the association frequency between cells within a cluster and those outside of it; and 'score' is the difference between 'stability' and 'promiscuity'. Importantly, 'score' reflects the overall "robustness" of a cluster and its constitutive samples to technical variance. These metrics may be calculated on cell or cluster-wise basis; here, they are calculated cell-wise.

Usage

```
report_cell_metrics(cluster_labels, consensus_matrix)
```

Arguments

`cluster_labels`	Cluster labels for each cell across various cluster numbers and the original clustering.
`consensus_matrix`	A noise consensus output by `compute_consensus()`

Value

A melted `data.frame` of BEARscc metrics for each cell:

[,1]	"Cluster.identity"	The number of the cluster within the respective clustering
[,2]	"Cell"	The identifier of the sample in question.
[,3]	"Cluster.size"	Number of samples in the cluster.
[,4]	"Metric"	Whether the metric is the BEARscc Score, Promiscuity, or Stability.
[,5]	"Value"	Value of the relevant BEARscc metric for the cell in a given clustering.
[,6]	"Clustering"	The clustering pertinant to the cell-wise metrics described.

Author(s)

David T. Severson <david_severson@hms.harvard.edu>

Maintainer: Benjamin Schuster-Boeckler <benjamin.schuster-boeckler@ludwig.ox.ac.uk>

Examples

```
data(analysis_examples)

cell_scores.df <- report_cell_metrics(BEARscc_clusts.df, noise_consensus)
cell_scores.df
```

`report_cluster_metrics` *Reports BEARscc metrics for clusters.*

Description

To quantitatively evaluate the results, three metrics are calculated from the noise consensus matrix: 'stability' is the average frequency with which cells within a cluster associate with each other across simulated replicates; 'promiscuity' measures the association frequency between cells within a cluster and those outside of it; and 'score' is the difference between 'stability' and 'promiscuity'. Importantly, 'score' reflects the overall "robustness" of a cluster to technical variance. These metrics may be calculated on cell or cluster-wise basis; here, they are calculated cluster-wise.

Usage

```
report_cluster_metrics(cluster_labels, consensus_matrix,
    weighted_mean = FALSE, plot = FALSE, file = "Rplot")
```

Arguments

cluster_labels	Cluster labels for each cell across various cluster numbers and the original clustering.
consensus_matrix	A noise consensus output by compute_consensus()
weighted_mean	A flag indicating whether to weigh observed clusters evenly or scale them by the number of samples in the cluster.
plot	A flag to determine whether to plot the boxplot of cluster metrics evaluated from the noise consensus with root file.
file	A string indicating the root desired for the resulting plots of the function.

Value

A melted data.frame of BEARscc metrics for each cluster:

[,1]	"Cluster.identity"	The number of the cluster within the respective clustering.
[,2]	"Cluster.size"	Number of samples in the cluster.
[,3]	"Metric"	Whether the metric is the BEARscc score, promiscuity, or stability.
[,4]	"Value"	Value of the relevant BEARscc metric for the cluster in a clustering.
[,5]	"Clustering"	The clustering pertinant to the cell-wise metrics described.
[,6]	"Singlet"	A binary output concerning whether the cluster consists of a single sample.
[,7]	"Clustering.Mean"	The average of the respective metric across cells of the clsuter.

Author(s)

David T. Severson <david_severson@hms.harvard.edu>

Maintainer: Benjamin Schuster-Boeckler <benjamin.schuster-boeckler@ludwig.ox.ac.uk>

Examples

```
data(analysis_examples)

cluster_scores.df <- report_cluster_metrics(BEARscc_clusts.df, noise_consensus,
    plot=TRUE, file="example")
cluster_scores.df
```

---

simulate_replicates *Computes* BEARscc *simulated technical replicates.*

Description

Computes BEARscc simulated technical replicates from the previously estimated noise parameters computed with the function estimate_noise_parameters().

Usage

```
simulate_replicates(SCEList, max_cumprob=0.9999, n = 3)
```

Arguments

SCEList	A SingleCellExpression object that has been appropriately processed by estimate_noiseparamet to add the necessary parameters describing the noise model for drop-outs and variance in the single cell experiment.
max_cumprob	Because a cumulative distribution will range from n=0 to a countable infinity, the event space needs to be set to cover a reasonable fraction of the probability density. This parameter determines the the fraction of probability density covered by the event space, which in turn defines the highes count number in the event space. We recommend users use the default value of 0.9999. However, if the default value was altered in estimate_noiseparameters(), then the value used in that function is most definitely what should be input here!
n	The number of simulated technical replicates to generate.

Details

In the second step of BEARscc, the algorithm applies the model from first step to produce simulated technical replicates. For every observed gene count below which drop-outs occurred amongst the spike-ins, BEARscc assesses whether to convert the count to zero (using the drop-out injection distribution). For observations where the count is zero, the drop-out recovery distribution is used to estimate a new value, based on the overall drop-out frequency for that gene. After this drop-out processing, all non-zero counts are substituted with a value generated by the model of expression variance created in the first step. parameterized to the observed counts for each gene. This second step is repeated any number of times (as prescribed by parameter n) to generate a collection of simulated technical replicates for downstream analysis.

Value

The resulting object is a list of counts data that is added to the metadata of the SingleCellExpression object as a long list titled "simulated_replicates". Each element of the list is a data.frame of the counts representing a BEARscc simulated technical replicate, e.g for n=10 we would have the list:

[,1]	Counts data.frame of simulated replicate 1.
[,2]	Counts data.frame of simulated replicate 2.
[,3]	Counts data.frame of simulated replicate 3.
[,4]	Counts data.frame of simulated replicate 4.
[,5]	Counts data.frame of simulated replicate 5.
[,6]	Counts data.frame of simulated replicate 6.
[,7]	Counts data.frame of simulated replicate 7.
[,8]	Counts data.frame of simulated replicate 8.
[,9]	Counts data.frame of simulated replicate 9.
[,10]	Counts data.frame of simulated replicate 10.
[,11]	Counts data.frame of observed data.

A brief description of subfunctions

simulate_replicates relies on the following subfunctions to generate simulated technical replicates. These functions share many common options with the user interactive function. For those options that are internal to the programming; these are annotated to give an idea of flow. For further detail please examine source code in the R directory of this package:

- spikes_prepared <- execute_noiseinjected_counts(n=1, noise_parameters=estimated_noise,
- probs4detection.genes<-t(data.frame(noise_parameters$bayes_parameters, row.names = "k
- probs4detection.k<-data.frame(noise_parameters$bayes_parameters[,2:4, with=FALSE],row
- noisy_counts<-data.table(noise_parameters$original.counts, keep.rownames = TRUE)[,apply
- probabilityA<-probs4detection.genes[gsub("-",".",x[1]),]
- apply(data.frame(as.numeric(x[-1])),1, `permute_count`, probs4detection.k, probability
- Under various conditions some form of nx<-randomizer(x, parameters, total_sampling) is invoked.

Note

Frequently, the user will want to compute simulated technical replicates in a high performance computational environment. When running estimate_noiseparameters() using the option write.noise.model=TRUE, the user recives the files with root file="noise_estimation", "noise_estimation_counts4clusterperturbation.xls", "noise_estimation_bayesianestimates.xls" and "noise_estimation_parameters4randomize.xls". These files may be input into the example code, HPC_generate_noise_matrices.R, on a high performance computational environment for faster processing.

Author(s)

David T. Severson <david_severson@hms.harvard.edu>

Maintainer: Benjamin Schuster-Boeckler <benjamin.schuster-boeckler@ludwig.ox.ac.uk>

See Also

The example code for running the simulation of technical replicates on a high performance computing cluster can be found in inst/example/.

The code for generating simulated technical replicates on a high powered compute node requires the function, HPC_simulate_replicates().

Examples

```
library("SingleCellExperiment")
data(analysis_examples)

BEAR_simreplicates.sce<-simulate_replicates(BEAR_analyzed.sce, n=3)
BEAR_simreplicates.sce
```

Index