Package: darwin (via r-universe)

May 25, 2026

Type Package

Title Multi-Objective Gene Selection Using Evolutionary Algorithms

Version 1.0.0

Date 2026-01-25

Author Zaoqu Liu [aut, cre] (<https://orcid.org/0000-0002-0452-742X>)

Maintainer Zaoqu Liu <liuzaoqu@163.com>

Description Automatic gene selection for bulk RNA-seq deconvolution using multi-objective optimization. Implements the NSGA-II algorithm to simultaneously minimize correlation and maximize distance between cell type expression profiles, yielding Pareto-optimal gene subsets. Supports Seurat objects (V4 and V5), SingleCellExperiment, and standard matrix inputs. Includes built-in deconvolution methods and parallel computing support.

License MIT + file LICENSE

URL https://zaoqu-liu.github.io/darwin/, https://github.com/Zaoqu-Liu/darwin

BugReports https://github.com/Zaoqu-Liu/darwin/issues

Encoding UTF-8

RoxygenNote 7.3.2

Roxygen list(markdown = TRUE)

Depends R (>= 4.0.0)

Imports Rcpp, R6, Matrix, stats, methods, parallel, future, future.apply, ggplot2, scales, cli, rlang, digest

Suggests Seurat, SeuratObject, SingleCellExperiment, SummarizedExperiment, e1071, nnls, testthat (>= 3.0.0), knitr, rmarkdown, BiocStyle

LinkingTo Rcpp, RcppArmadillo

VignetteBuilder knitr

Config/testthat/edition 3

Repository https://zaoqu-liu.r-universe.dev

Date/Publication 2026-01-25 16:43:13 UTC

RemoteUrl https://github.com/Zaoqu-Liu/darwin

RemoteRef main

RemoteSha a6d9027d5903c2e0104fbea2bf141be7cf07fdf9

Contents

`compute_condition` *Compute Condition Number*

Description

Computes the condition number of the data matrix. Lower values indicate better numerical stability for deconvolution.

Usage

```
compute_condition(data, use_cpp = getOption("darwin.use_cpp", TRUE))
```

Arguments

`data`	Numeric matrix (cell types x genes).
`use_cpp`	Use C++ implementation for speed. Default: TRUE.

Value

Condition number (ratio of largest to smallest singular value).

Examples

```
data <- matrix(rnorm(50), nrow = 5, ncol = 10)
cond <- compute_condition(data)
```

compute_correlation *Compute Pairwise Correlation*

Description

Computes the sum of absolute pairwise Pearson correlations between cell type expression profiles. Lower values indicate more distinct profiles.

Usage

```
compute_correlation(data, use_cpp = getOption("darwin.use_cpp", TRUE))
```

Arguments

data	Numeric matrix (cell types x genes).
use_cpp	Use C++ implementation for speed. Default: TRUE.

Value

Sum of absolute pairwise Pearson correlations between rows.

Examples

```
data <- matrix(rnorm(50), nrow = 5, ncol = 10)
corr <- compute_correlation(data)
```

compute_distance *Compute Pairwise Distance*

Description

Computes the sum of pairwise Euclidean distances between cell type expression profiles. Higher values indicate more distinct profiles.

Usage

```
compute_distance(data, use_cpp = getOption("darwin.use_cpp", TRUE))
```

Arguments

data	Numeric matrix (cell types x genes).
use_cpp	Use C++ implementation for speed. Default: TRUE.

Value

Sum of pairwise Euclidean distances between rows.

Examples

```
data <- matrix(rnorm(50), nrow = 5, ncol = 10)
dist_val <- compute_distance(data)
```

`crowding_distance` *Compute Crowding Distance*

Description

Computes the crowding distance for each individual, measuring solution density in objective space.

Usage

```
crowding_distance(fitness_values, ranks = NULL)
```

Arguments

`fitness_values`	Matrix of fitness values (individuals x objectives).
`ranks`	Vector of Pareto ranks (optional).

Details

Crowding distance measures how close an individual is to its neighbors in objective space. Higher values indicate more isolated solutions, which are preferred for maintaining diversity. Boundary solutions receive infinite crowding distance.

Value

Numeric vector of crowding distances.

Examples

```
fitness <- matrix(runif(20), nrow = 10, ncol = 2)
crowding <- crowding_distance(fitness)
```

darwin *Create a Darwin Object*

Description

Creates a Darwin object for multi-objective gene selection optimization. This is the recommended way to create Darwin objects.

Usage

```
darwin(
  data,
  celltype_key = "celltype",
  assay = NULL,
  layer = "data",
  genes_key = NULL,
  use_highly_variable = FALSE
)
```

Arguments

data	Input data. Can be a matrix (cell types x genes), data.frame, Seurat object, or SingleCellExperiment object.
celltype_key	For Seurat/SCE objects, the metadata column containing cell type labels. Default: "celltype".
assay	For Seurat objects, which assay to use. Default: default assay.
layer	For Seurat V5, which layer to use. Default: "data".
genes_key	Column in feature metadata for gene pre-selection.
use_highly_variable	Use highly variable genes only. Default: FALSE.

Value

A Darwin-class R6 object.

See Also

Darwin-class for the R6 class documentation.

Examples

```
# Create example data
set.seed(42)
data <- matrix(rnorm(500), nrow = 5, ncol = 100)
rownames(data) <- paste0("CellType", 1:5)
colnames(data) <- paste0("Gene", 1:100)
```

```
# Initialize darwin
dw <- darwin(data)

# Run optimization
dw$optimize(ngen = 5, verbose = FALSE, parallel = FALSE)

# Select genes
dw$select()
genes <- dw$get_genes()
```

`Darwin-class` *Darwin R6 Class for Multi-Objective Gene Selection*

Description

R6 class implementing multi-objective gene selection using NSGA-II algorithm. Use the `darwin` constructor function to create instances.

Public fields

None directly exposed. Use methods to access data.

Public methods

`initialize(data, ...)` Create a new Darwin object. See `darwin`.

`optimize(ngen, mode, ...)` Run the NSGA-II optimization algorithm.

`plot()` Plot the Pareto front.

`select(weights, index, close_to)` Select a solution from the Pareto front.

`get_genes()` Get names of selected genes.

`get_selection()` Get logical vector of gene selection.

`get_pareto()` Get all Pareto-optimal solutions.

`get_fitness()` Get fitness values for Pareto front.

`deconvolve(bulk, method)` Perform bulk RNA-seq deconvolution.

`save(path)` Save object to file.

`print()` Print object summary.

See Also

`darwin` for the constructor function.

Examples

```
# Create example data
set.seed(42)
data <- matrix(rnorm(500), nrow = 5, ncol = 100)
rownames(data) <- paste0("CellType", 1:5)
colnames(data) <- paste0("Gene", 1:100)

# Create and use Darwin object
dw <- darwin(data)  # Using constructor function
dw$optimize(ngen = 5, verbose = FALSE, parallel = FALSE)
dw$select()
genes <- dw$get_genes()
```

---

nsga2_select *NSGA-II Selection*

Description

Performs NSGA-II selection based on non-dominated sorting and crowding distance.

Usage

```
nsga2_select(fitness_values, n, weights)
```

Arguments

fitness_values	Matrix of fitness values (individuals x objectives).
n	Number of individuals to select.
weights	Vector of weights (-1 for minimization, 1 for maximization).

Details

Selection proceeds by:

1. Non-dominated sorting to assign Pareto ranks
2. Computing crowding distance within each rank
3. Selecting individuals by rank (ascending), breaking ties by crowding distance (descending)

Value

Integer vector of selected indices.

References

Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation, 6(2), 182-197.

Examples

```
fitness <- matrix(runif(20), nrow = 10, ncol = 2)
selected <- nsga2_select(fitness, n = 5, weights = c(-1, 1))
```

Index