Package: CellMentor (via r-universe)

May 25, 2026

Type Package

Title Supervised Non-negative Matrix Factorization for Dimensional Reduction in Single-Cell Analysis

Version 1.0.1

Description Implements supervised cell type-aware non-negative matrix factorization (NMF) for dimensional reduction in single-cell RNA sequencing analysis. The package provides methods for incorporating cell type information into the dimensionality reduction process, enabling improved visualization and downstream analysis of single-cell data while preserving biological structure. CellMentor employs a unique loss function that simultaneously minimizes variation within known cell populations while maximizing distinctions between different cell types, enabling effective transfer of learned patterns from labeled reference datasets to new unlabeled data.

License Apache License (>= 2)

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Depends R (>= 4.5.0)

Imports methods, Matrix, BiocParallel, SingleR, Seurat (>= 4.0.0), utils, stats, parallel, progress, ggplot2, data.table, magrittr, graphics, RMTstat, sparsesvd, cluster, skmeans, MLmetrics, tibble, lsa, nnls, SingleCellExperiment, entropy, irlba, aricode

Suggests testthat (>= 3.0.0), covr, withr, rmarkdown, knitr, BiocStyle, scater, scRNAseq

VignetteBuilder knitr

biocViews Software, SingleCell, Transcriptomics, DimensionReduction

URL https://github.com/petrenkokate/CellMentor

BugReports https://github.com/petrenkokate/CellMentor/issues

Config/testthat/edition 3

Config/pak/sysreqs cmake libglpk-dev make libicu-dev libpng-dev libuv1-dev libxml2-dev libssl-dev python3 zlib1g-dev

Repository https://bioc-release.r-universe.dev

Date/Publication 2026-05-25 16:23:09 UTC

RemoteUrl https://github.com/bioc/CellMentor

RemoteRef RELEASE_3_23

RemoteSha a2a13e2bd5b55274c16ce0806ebfaae56d9c94db

Contents

`CellMentor` *CellMentor: Cell-Type Aware Dimensionality Reduction for Single-Cell RNA-Seq*

Description

CellMentor is a supervised dimensionality reduction method based on non-negative matrix factorization (NMF) that integrates cell type labels directly into its optimization objective. By minimizing variation within known populations while maximizing distinctions between types, CellMentor produces low-dimensional embeddings optimized for cell type identification in single-cell RNA sequencing analysis.

Tests different combinations of hyperparameters using `RunCSFNMF` to find the optimal configuration. The function performs a grid search over specified parameter ranges, evaluating the model's performance for each combination. Parameters `alpha` and `beta` are kept equal during optimization. The rank (k) can be provided, taken from an existing object, or determined automatically using `SelectRank`.

Usage

```
CellMentor(
  object,
  k = NULL,
  init_methods = c("regulated"),
  alpha_range = c(1, 5),
  beta_range = c(1, 5),
  gamma_range = c(0.1),
  delta_range = c(1),
  n_iter = 1,
  verbose = TRUE,
  num_cores = 1,
  seed = 1
)
```

Arguments

`object`	CSFNMF object containing reference and query data matrices, with required matrices for `data` and `ref` under `object@matrices`.
`k`	Optional rank value (number of factors). If `NULL`: - Uses existing rank from `object` if available - Otherwise determines automatically using `SelectRank`
`init_methods`	Vector of initialization methods to test. Options: - "uniform": Random uniform initialization - "regulated": Cell-type guided initialization - "NNDSVD": Non-negative Double SVD - "skmeanGenes": Gene clustering-based - "skmeanCells": Cell clustering-based Default: c("regulated")
`alpha_range`	Vector of `alpha` values to test. Controls within-class scatter (cell similarity within the same type). Default: c(0.1, 0.5, 1)
`beta_range`	Vector of `beta` values to test. Controls between-class scatter (cell separation between different types). Default: c(1, 2, 5)
`gamma_range`	Vector of sparsity parameter values to test. Controls sparsity of the factorization. Default: c(0, 0.1)
`delta_range`	Vector of orthogonality parameter values to test. Controls orthogonality between factors. Default: c(0, 0.5)
`n_iter`	Number of repetitions per configuration for averaging results (default: 3).
`verbose`	Logical; whether to show progress messages during optimization. Default: `TRUE`
`num_cores`	Number of cores to use for parallel processing. If > 1, parameter combinations are tested in parallel. Default: 1
`seed`	Random seed

Value

List containing: - `best_params`: List with the overall best parameter configuration: * `k`: Selected rank * `init_method`: Best initialization method * `alpha`: Best alpha parameter * `beta`: Best beta parameter * `gamma`: Best gamma value * `delta`: Best delta value * `accuracy`: Best achieved accuracy * `loss`: Corresponding loss value - `results`: Data frame of all combinations tested, including:

* `init_method`: Initialization method used * `alpha`: Alpha parameter value * `beta`: Beta parameter value * `gamma`: Gamma parameter value * `delta`: Delta parameter value * `accuracy`: Achieved accuracy * `loss`: Final loss value * `convergence_iter`: Number of iterations for convergence - `best_model`:CSFNMF model object trained with the best parameters.

Key Features

- **Supervised NMF Framework:** Incorporates labels via discriminative constraints
- **Superior Cell Type Separation:** Maximally separable embeddings
- **Robust Batch Handling:** Preserves biology while mitigating technical effects
- **Rare Population Detection:** Sensitive to low-frequency types
- **Automated Parameter Optimization:** Built-in hyperparameter tuning

Two-Phase Workflow

1. **Decomposition (Training):** Learn W (genes × K) and H (K × cells)
2. **Projection (Inference):** Project queries with non-negative least squares

Main Functions

- `CellMentor` — supervised NMF / hyperparameter search
- `project_data` — project queries using learned W
- `CreateCSFNMFobject` — initialize a CellMentor object

Getting Help

- Package docs: `help(package = "CellMentor")`
- GitHub: `https://github.com/petrenkokate/CellMentor`
- Issues: `https://github.com/petrenkokate/CellMentor/issues`

Author(s)

Or Hevdeli (equal contribution)
Ekaterina Petrenko (equal contribution) <`petrenko.kate@icloud.com`>
Dvir Aran (corresponding author) <`dviraran@technion.ac.il`>

References

Hevdeli, O., Petrenko, E., & Aran, D. (2025). CellMentor: Cell-Type Aware Dimensionality Reduction for Single-cell RNA-Sequencing Data. *bioRxiv*. doi:10.1101/2025.06.17.660094

See Also

Useful links:

- `https://github.com/petrenkokate/CellMentor`
- Report bugs at `https://github.com/petrenkokate/CellMentor/issues`

Examples

```
data(obj_toy, package = "CellMentor")
# Run lightweight CellMentor
result <- CellMentor(
  object        = obj_toy,
  k             = 2,
  init_methods  = "regulated",
  alpha_range   = 1,
  beta_range    = 1,
  gamma_range   = 0.1,
  delta_range   = 1,
  n_iter        = 1,
  verbose       = FALSE,
  num_cores     = 1
)

# Inspect results (should run in <10 seconds)
names(result)
if ("best_params" %in% names(result)) {
  print(result$best_params)
}
```

`cm_annotation` *Access cell annotations from a CellMentor object*

Description

Retrieve cell type or metadata annotations.

Usage

```
cm_annotation(x)
```

Arguments

`x`	A CellMentor object.

Value

A data frame containing cell annotations.

Examples

```
data(obj_toy, package = "CellMentor")
cm_annotation(obj_toy)
```

`cm_rank` *Access the rank of a CellMentor object*

Description

Retrieve the factorization rank used during CSFNMF training.

Usage

```
cm_rank(x)
```

Arguments

`x`	A CellMentor object.

Value

A numeric value representing the selected rank.

Examples

```
## Not run:
# Access the rank of the model
cm_rank(cs_obj)

## End(Not run)
```

`CreateCSFNMFobject` *Create CSFNMF Object for Cell Type Analysis*

Description

Creates and initializes a Constrained Supervised Factorization NMF (CSFNMF) object for analyzing single-cell RNA sequencing data. This is the main function for starting analysis with CellMentor.

Usage

```
CreateCSFNMFobject(
  ref_matrix,
  ref_celltype,
  data_matrix,
  norm = TRUE,
  most.variable = TRUE,
  scale = TRUE,
  scale_by = "cells",
```

```
    gene_list = NULL,
    verbose = TRUE,
    num_cores = 1
  )
```

Arguments

`ref_matrix`	Reference matrix (genes × cells) with known cell types
`ref_celltype`	Vector of cell type labels for reference cells
`data_matrix`	Query matrix (genes × cells) to be analyzed
`norm`	Logical: perform normalization (default: TRUE)
`most.variable`	Logical: select variable genes (default: TRUE)
`scale`	Logical: perform scaling (default: TRUE)
`scale_by`	Character: scaling method, either "cells" or "genes" (default: "cells")
`gene_list`	Optional vector of genes to include (default: NULL)
`verbose`	Logical: show progress messages (default: TRUE)
`num_cores`	Integer: number of cores for parallel processing (default: 1)

Value

A CSFNMF object containing processed data and annotations

Examples

```
data(ref_matrix_toy, qry_matrix_toy, ref_celltype_toy, package = "CellMentor")
obj <- CreateCSFNMFobject(ref_matrix_toy, ref_celltype_toy, qry_matrix_toy,
                          norm = FALSE, most.variable = FALSE, scale = FALSE,
                          verbose = FALSE, num_cores = 1)
inherits(obj, "csfnmf")
```

`data_matrix` *Access the query (data) matrix from a RefDataList object*

Description

Retrieve the single-cell expression matrix for the query dataset.

Usage

```
data_matrix(x)
```

Arguments

`x`	A RefDataList object.

Value

A sparse `Matrix::Matrix` object representing query data.

Examples

```
data(obj_toy, package = "CellMentor")
data_matrix(matrices(obj_toy))
```

`H` *Access the H matrix*

Description

Retrieve the H matrix (cell embeddings) from a CellMentor object.

Usage

```
H(x)
```

Arguments

`x`	A CellMentor object.

Value

A numeric matrix representing the H (cell embeddings) matrix.

Examples

```
data(obj_toy, package = "CellMentor")
H(obj_toy)
```

`hBaronDataset` *Load Baron Human Pancreas Dataset*

Description

Loads and processes the Baron et al. human pancreas single-cell RNA-seq dataset

Usage

```
hBaronDataset()
```

Value

A list containing:

`data`	Expression matrix with genes as rows and cells as columns
`celltypes`	Named vector of cell type annotations

Examples

```
# Load Baron human pancreas dataset
baron <- hBaronDataset()

# Check dimensions
dim(baron$data)

# View cell type distribution
table(baron$celltypes)
```

`matrices` *Access the RefDataList from a CSFNMF object*

Description

Retrieve the RefDataList structure containing both reference and query matrices.

Usage

```
matrices(x)
```

Arguments

`x`	A `csfnmf` or `traincsfnmf` object.

Value

A RefDataList object containing the reference and query matrices.

Examples

```
data(obj_toy, package = "CellMentor")
matrices(obj_toy)
```

`muraro_dataset` *Load Muraro Pancreas Dataset*

Description

Loads and processes the Muraro et al. pancreas single-cell RNA-seq dataset

Usage

```
muraro_dataset()
```

Value

A list containing:

`data`	Expression matrix with genes as rows and cells as columns
`celltypes`	Named vector of cell type annotations

Examples

```
# Load Muraro pancreas dataset
muraro <- muraro_dataset()

# Check dataset dimensions
dim(muraro$data)

# View available cell types
table(muraro$celltypes)

# Check number of cells per type
sort(table(muraro$celltypes), decreasing = TRUE)
```

`obj_toy` *Tiny prebuilt CSFNMF object for accessors (optional)*

Description

Tiny prebuilt CSFNMF object for accessors (optional)

Usage

```
obj_toy
```

Format

An object of class `csfnmf` built on the toy matrices.

Details

Only provided to make accessor examples immediate. For real analyses, construct objects from your own data.

Examples

```
data(obj_toy, package = "CellMentor")
inherits(obj_toy, "csfnmf")
```

`project_data` *Project Data onto NMF Basis Matrix*

Description

Projects new data onto the learned basis matrix (W) using non-negative least squares (NNLS). This function is used to obtain cell-type signatures (H matrix) for new query data using the gene weights (W matrix) learned during training. The projection is performed in chunks to manage memory efficiently, with optional parallel processing.

Usage

```
project_data(W, X, seed = 1, num_cores = 1, chunk_size = 1000, verbose = TRUE)
```

Arguments

`W`	Basis matrix (genes × rank) containing learned gene weights
`X`	Data matrix (genes × cells) to be projected. Must have same number of genes (rows) as W
`seed`	Random seed for reproducibility (default: 1)
`num_cores`	Number of cores for parallel processing (default: 1). If > 1, processing is parallelized across chunks
`chunk_size`	Number of cells to process in each chunk (default: 1000). Smaller chunks use less memory but may be slower
`verbose`	Logical; whether to show progress bar (default: TRUE)

Details

The projection is performed using non-negative least squares (NNLS) to solve the optimization problem: min $\|X - WH\|^2$ subject to $H >= 0$, for each cell in the input matrix X. The resulting H matrix contains the cell-type signatures for the query data.

For memory efficiency, cells are processed in chunks. The chunk_size parameter can be adjusted based on available memory. Parallel processing can be enabled by setting num_cores > 1.

Value

A Matrix object (rank × cells) containing the projection coefficients. The rows correspond to factors (rank) and columns to cells. Additional processing information is stored in attributes: - num_chunks: Number of chunks processed - chunk_size: Size of chunks used - num_cores: Number of cores used

Examples

```
# Minimal, fast example (no external data)
set.seed(1)

# Dimensions
genes <- paste0("Gene", seq_len(50))
k     <- 3    # rank
cells <- 10

# Non-negative basis W (genes x k)
W_ex <- matrix(abs(rnorm(length(genes) * k, sd = 0.5)),
               nrow = length(genes), ncol = k,
               dimnames = list(genes, paste0("k", seq_len(k))))

# Generate a non-negative H_true and synthetic data X = W * H + noise
H_true <- matrix(abs(rnorm(k * cells, sd = 0.5)), nrow = k, ncol = cells)
X_ex   <- W_ex %*% H_true + matrix(rexp(length(genes) * cells, rate = 20),
                                   nrow = length(genes), ncol = cells,
                                 dimnames = list(genes, paste0("cell", seq_len(cells))))

# Project (rank x cells)
H_est <- project_data(
  W = W_ex,
  X = X_ex,
  num_cores = 1,     # keep examples fast & deterministic
  chunk_size = 5,
  verbose = FALSE
)

dim(H_est)           # should be k x cells
```

`ref_matrix` *Access the reference matrix from a RefDataList object*

Description

Retrieve the single-cell expression matrix for the reference dataset.

Usage

```
ref_matrix(x)
```

Arguments

`x`	A RefDataList object.

Value

A sparse `Matrix::Matrix` object representing reference data.

Examples

```
data(obj_toy, package = "CellMentor")
ref_matrix(matrices(obj_toy))
```

`ref_matrix_toy` *Tiny toy matrices and labels for runnable examples*

Description

Tiny toy matrices and labels for runnable examples

Usage

```
ref_matrix_toy

qry_matrix_toy

ref_celltype_toy
```

Format

- `ref_matrix_toy`: numeric matrix (50 genes x 12 cells), dimnames set.
- `qry_matrix_toy`: numeric matrix (50 genes x 8 cells), dimnames set.
- `ref_celltype_toy`: character vector of length 12, names match `colnames(ref_matrix_toy)`.

An object of class `matrix` (inherits from `array`) with 50 rows and 8 columns.

An object of class `character` of length 12.

Details

These are minimal, non-biological toy data with shared gene IDs across reference and query for fast runnable examples and tests.

Examples

```
data(ref_matrix_toy, package = "CellMentor")
data(qry_matrix_toy, package = "CellMentor")
data(ref_celltype_toy, package = "CellMentor")
dim(ref_matrix_toy); dim(qry_matrix_toy)
head(ref_celltype_toy)
```

`W` *Access the W matrix*

Description

Retrieve the W matrix (gene loadings) from a CellMentor object.

Usage

```
W(x)
```

Arguments

`x`	A CellMentor object (e.g., `traincsfnmf` or `csfnmf`).

Value

A numeric matrix representing the W (gene loadings) matrix.

Examples

```
data(obj_toy, package = "CellMentor")
W(obj_toy)
```

Index