Package: Rtsne (via r-universe)

October 28, 2026

Type Package

Title T-Distributed Stochastic Neighbor Embedding using a Barnes-Hut Implementation

Version 0.17

Description An R wrapper around the fast T-distributed Stochastic Neighbor Embedding implementation by Van der Maaten (see <https://github.com/lvdmaaten/bhtsne/> for more information on the original implementation).

License file LICENSE

URL https://github.com/jkrijthe/Rtsne

Encoding UTF-8

Imports Rcpp (>= 0.11.0), stats

LinkingTo Rcpp

Suggests irlba, testthat

RoxygenNote 7.2.3

NeedsCompilation yes

Author Jesse Krijthe [aut, cre], Laurens van der Maaten [cph] (Author of original C++ code)

Maintainer Jesse Krijthe <jkrijthe@gmail.com>

License_is_FOSS yes

Repository https://cran.r-universe.dev

Date/Publication 2023-12-08 02:39:39 UTC

RemoteUrl https://github.com/cran/Rtsne

RemoteRef HEAD

RemoteSha 1f8864eb33f2cc5d011f63ba333dee9078b315d1

Contents

normalize_input 2
Rtsne 3

Index **9**

`normalize_input` *Normalize input data matrix*

Description

Mean centers each column of an input data matrix so that it has a mean of zero. Scales the entire matrix so that the largest absolute of the centered matrix is equal to unity.

Usage

```
normalize_input(X)
```

Arguments

`X`	matrix; Input data matrix with rows as observations and columns as variables/dimensions.

Details

Normalization avoids numerical problems when the coordinates (and thus the distances between observations) are very large. Directly computing distances on this scale may lead to underflow when computing the probabilities in the t-SNE algorithm. Rescaling the input values mitigates these problems to some extent.

Value

A numeric matrix of the same dimensions as `X` but centred by column and scaled to have a maximum deviation of 1.

Author(s)

Aaron Lun

Examples

```
iris_unique <- unique(iris) # Remove duplicates
iris_matrix <- as.matrix(iris_unique[,1:4])
X <- normalize_input(iris_matrix)
colMeans(X)
range(X)
```

Rtsne *Barnes-Hut implementation of t-Distributed Stochastic Neighbor Embedding*

Description

Wrapper for the C++ implementation of Barnes-Hut t-Distributed Stochastic Neighbor Embedding. t-SNE is a method for constructing a low dimensional embedding of high-dimensional data, distances or similarities. Exact t-SNE can be computed by setting theta=0.0.

Usage

```
Rtsne(X, ...)

## Default S3 method:
Rtsne(
  X,
  dims = 2,
  initial_dims = 50,
  perplexity = 30,
  theta = 0.5,
  check_duplicates = TRUE,
  pca = TRUE,
  partial_pca = FALSE,
  max_iter = 1000,
  verbose = getOption("verbose", FALSE),
  is_distance = FALSE,
  Y_init = NULL,
  pca_center = TRUE,
  pca_scale = FALSE,
  normalize = TRUE,
  stop_lying_iter = ifelse(is.null(Y_init), 250L, 0L),
  mom_switch_iter = ifelse(is.null(Y_init), 250L, 0L),
  momentum = 0.5,
  final_momentum = 0.8,
  eta = 200,
  exaggeration_factor = 12,
  num_threads = 1,
  ...
)

## S3 method for class 'dist'
Rtsne(X, ..., is_distance = TRUE)

## S3 method for class 'data.frame'
Rtsne(X, ...)
```

```
Rtsne_neighbors(
  index,
  distance,
  dims = 2,
  perplexity = 30,
  theta = 0.5,
  max_iter = 1000,
  verbose = getOption("verbose", FALSE),
  Y_init = NULL,
  stop_lying_iter = ifelse(is.null(Y_init), 250L, 0L),
  mom_switch_iter = ifelse(is.null(Y_init), 250L, 0L),
  momentum = 0.5,
  final_momentum = 0.8,
  eta = 200,
  exaggeration_factor = 12,
  num_threads = 1,
  ...
)
```

Arguments

X	matrix; Data matrix (each row is an observation, each column is a variable)
...	Other arguments that can be passed to Rtsne
dims	integer; Output dimensionality (default: 2)
initial_dims	integer; the number of dimensions that should be retained in the initial PCA step (default: 50)
perplexity	numeric; Perplexity parameter (should not be bigger than 3 * perplexity < nrow(X) - 1, see details for interpretation)
theta	numeric; Speed/accuracy trade-off (increase for less accuracy), set to 0.0 for exact TSNE (default: 0.5)
check_duplicates	logical; Checks whether duplicates are present. It is best to make sure there are no duplicates present and set this option to FALSE, especially for large datasets (default: TRUE)
pca	logical; Whether an initial PCA step should be performed (default: TRUE)
partial_pca	logical; Whether truncated PCA should be used to calculate principal components (requires the irlba package). This is faster for large input matrices (default: FALSE)
max_iter	integer; Number of iterations (default: 1000)
verbose	logical; Whether progress updates should be printed (default: global "verbose" option, or FALSE if that is not set)
is_distance	logical; Indicate whether X is a distance matrix (default: FALSE)
Y_init	matrix; Initial locations of the objects. If NULL, random initialization will be used (default: NULL). Note that when using this, the initial stage with exaggerated perplexity values and a larger momentum term will be skipped.

pca_center	logical; Should data be centered before pca is applied? (default: TRUE)
pca_scale	logical; Should data be scaled before pca is applied? (default: FALSE)
normalize	logical; Should data be normalized internally prior to distance calculations with normalize_input? (default: TRUE)
stop_lying_iter	integer; Iteration after which the perplexities are no longer exaggerated (default: 250, except when Y_init is used, then 0)
mom_switch_iter	integer; Iteration after which the final momentum is used (default: 250, except when Y_init is used, then 0)
momentum	numeric; Momentum used in the first part of the optimization (default: 0.5)
final_momentum	numeric; Momentum used in the final part of the optimization (default: 0.8)
eta	numeric; Learning rate (default: 200.0)
exaggeration_factor	numeric; Exaggeration factor used to multiply the P matrix in the first part of the optimization (default: 12.0)
num_threads	integer; Number of threads to use when using OpenMP, default is 1. Setting to 0 corresponds to detecting and using all available cores
index	integer matrix; Each row contains the identity of the nearest neighbors for each observation
distance	numeric matrix; Each row contains the distance to the nearest neighbors in index for each observation

Details

Given a distance matrix D between input objects (which by default, is the euclidean distances between two objects), we calculate a similarity score in the original space:

$$p_{j|i} = \frac{\exp(-\|D_{ij}\|^2/2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|D_{ij}\|^2/2\sigma_i^2)}$$

which is then symmetrized using:

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}.$$

The σ for each object is chosen in such a way that the perplexity of $p_{j|i}$ has a value that is close to the user defined perplexity. This value effectively controls how many nearest neighbours are taken into account when constructing the embedding in the low-dimensional space. For the low-dimensional space we use the Cauchy distribution (t-distribution with one degree of freedom) as the distribution of the distances to neighbouring objects:

$$q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k \neq l} 1 + \|y_k - y_l\|^2)^{-1}}.$$

By changing the location of the objects y in the embedding to minimize the Kullback-Leibler divergence between these two distributions q_{ij} and p_{ij}, we create a map that focusses on small-scale structure, due to the asymmetry of the KL-divergence. The t-distribution is chosen to avoid the

crowding problem: in the original high dimensional space, there are potentially many equidistant objects with moderate distance from a particular object, more than can be accounted for in the low dimensional representation. The t-distribution makes sure that these objects are more spread out in the new representation.

For larger datasets, a problem with the a simple gradient descent to minimize the Kullback-Leibler divergence is the computational complexity of each gradient step (which is $O(n^2)$). The Barnes-Hut implementation of the algorithm attempts to mitigate this problem using two tricks: (1) approximating small similarities by 0 in the p_{ij} distribution, where the non-zero entries are computed by finding 3*perplexity nearest neighbours using an efficient tree search. (2) Using the Barnes-Hut algorithm in the computation of the gradient which approximates large distance similarities using a quadtree. This approximation is controlled by the `theta` parameter, with smaller values leading to more exact approximations. When `theta=0.0`, the implementation uses a standard t-SNE implementation. The Barnes-Hut approximation leads to a $O(nlog(n))$ computational complexity for each iteration.

During the minimization of the KL-divergence, the implementation uses a trick known as early exaggeration, which multiplies the p_{ij}'s by 12 during the first 250 iterations. This leads to tighter clustering and more distance between clusters of objects. This early exaggeration is not used when the user gives an initialization of the objects in the embedding by setting `Y_init`. During the early exaggeration phase, a momentum term of 0.5 is used while this is changed to 0.8 after the first 250 iterations. All these default parameters can be changed by the user.

After checking the correctness of the input, the `Rtsne` function (optionally) does an initial reduction of the feature space using `prcomp`, before calling the C++ TSNE implementation. Since R's random number generator is used, use `set.seed` before the function call to get reproducible results.

If `X` is a data.frame, it is transformed into a matrix using `model.matrix`. If `X` is a `dist` object, it is currently first expanded into a full distance matrix.

Value

List with the following elements:

`Y`	Matrix containing the new representations for the objects
`N`	Number of objects
`origD`	Original Dimensionality before TSNE (only when `X` is a data matrix)
`perplexity`	See above
`theta`	See above
`costs`	The cost for every object after the final iteration
`itercosts`	The total costs (KL-divergence) for all objects in every 50th + the last iteration
`stop_lying_iter`	Iteration after which the perplexities are no longer exaggerated
`mom_switch_iter`	Iteration after which the final momentum is used
`momentum`	Momentum used in the first part of the optimization
`final_momentum`	Momentum used in the final part of the optimization
`eta`	Learning rate
`exaggeration_factor`	Exaggeration factor used to multiply the P matrix in the first part of the optimization

Methods (by class)

- `Rtsne(default)`: Default Interface
- `Rtsne(dist)`: tsne on given dist object
- `Rtsne(data.frame)`: tsne on data.frame

Supplying precomputed distances

If a distance matrix is already available, this can be directly supplied to `Rtsne` by setting `is_distance=TRUE`. This improves efficiency by avoiding recalculation of distances, but requires some work to get the same results as running default `Rtsne` on a data matrix. Specifically, Euclidean distances should be computed from a normalized data matrix - see `normalize_input` for details. PCA arguments will also be ignored if `is_distance=TRUE`.

NN search results can be directly supplied to `Rtsne_neighbors` to avoid repeating the (possibly time-consuming) search. To achieve the same results as `Rtsne` on the data matrix, the search should be conducted on the normalized data matrix. The number of nearest neighbors should also be equal to three-fold the `perplexity`, rounded down to the nearest integer. Note that pre-supplied NN results cannot be used when `theta=0` as they are only relevant for the approximate algorithm.

Any kind of distance metric can be used as input. In contrast, running `Rtsne` on a data matrix will always use Euclidean distances.

References

Maaten, L. Van Der, 2014. Accelerating t-SNE using Tree-Based Algorithms. Journal of Machine Learning Research, 15, p.3221-3245.

van der Maaten, L.J.P. & Hinton, G.E., 2008. Visualizing High-Dimensional Data Using t-SNE. Journal of Machine Learning Research, 9, pp.2579-2605.

Examples

```
iris_unique <- unique(iris) # Remove duplicates
iris_matrix <- as.matrix(iris_unique[,1:4])

# Set a seed if you want reproducible results
set.seed(42)
tsne_out <- Rtsne(iris_matrix,pca=FALSE,perplexity=30,theta=0.0) # Run TSNE

# Show the objects in the 2D tsne representation
plot(tsne_out$Y,col=iris_unique$Species, asp=1)

# data.frame as input
tsne_out <- Rtsne(iris_unique,pca=FALSE, theta=0.0)

# Using a dist object
set.seed(42)
tsne_out <- Rtsne(dist(normalize_input(iris_matrix)), theta=0.0)
plot(tsne_out$Y,col=iris_unique$Species, asp=1)

set.seed(42)
tsne_out <- Rtsne(as.matrix(dist(normalize_input(iris_matrix))),theta=0.0)
```

```
plot(tsne_out$Y,col=iris_unique$Species, asp=1)

# Supplying starting positions (example: continue from earlier embedding)
set.seed(42)
tsne_part1 <- Rtsne(iris_unique[,1:4], theta=0.0, pca=FALSE, max_iter=350)
tsne_part2 <- Rtsne(iris_unique[,1:4], theta=0.0, pca=FALSE, max_iter=650, Y_init=tsne_part1$Y)
plot(tsne_part2$Y,col=iris_unique$Species, asp=1)
## Not run:
# Fast PCA and multicore

tsne_out <- Rtsne(iris_matrix, theta=0.1, partial_pca = TRUE, initial_dims=3)
tsne_out <- Rtsne(iris_matrix, theta=0.1, num_threads = 2)

## End(Not run)
```

Index

dist, *6*

model.matrix, *6*

normalize_input, 2, *5*, *7*

prcomp, *6*

Rtsne, 3
Rtsne.data.frame *(*Rtsne*)*, 3
Rtsne.default *(*Rtsne*)*, 3
Rtsne.dist *(*Rtsne*)*, 3
Rtsne_neighbors *(*Rtsne*)*, 3

set.seed, *6*