Package 'Rtsne'

April 14, 2017

Type Package

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

Version 0.13

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 BSD_3_clause + file LICENSE

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

Imports Rcpp (>= 0.11.0)

LinkingTo Rcpp

Suggests testthat

RoxygenNote 6.0.1

NeedsCompilation yes

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

Maintainer Jesse Krijthe <jkrijthe@gmail.com>

Repository CRAN

Date/Publication 2017-04-14 16:38:37 UTC

R topics documented:

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, max_iter = 1000,
  verbose = FALSE, is_distance = FALSE, Y_init = NULL,
  pca_center = TRUE, pca_scale = FALSE,
  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, ...)

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

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

Arguments

X	matrix; Data matrix
...	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
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)
max_iter	integer; Number of iterations (default: 1000)

verbose	logical; Whether progress updates should be printed (default: FALSE)
is_distance	logical; Indicate whether X is a distance matrix (experimental, 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)
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)

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_ij.

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

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 assymetry 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
`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)

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

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.seed(42) # Set a seed if you want reproducible results
tsne_out <- Rtsne(iris_matrix) # Run TSNE

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

# Using a dist object
tsne_out <- Rtsne(dist(iris_matrix))
plot(tsne_out$Y,col=iris_unique$Species)

# Use a given initialization of the locations of the points
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=150,Y_init=tsne_part1$Y)
```

Index