Package 'DDRTree'

April 30, 2017

Type Package

Title Learning Principal Graphs with DDRTree

Version 0.1.5

Date 2017-4-14

Author Xiaojie Qiu, Cole Trapnell, Qi Mao, Li Wang

Depends irlba

Imports Rcpp

LinkingTo Rcpp, RcppEigen, BH

Maintainer Xiaojie Qiu <xqiu@uw.edu>

Description Provides an implementation of the framework of reversed graph embedding (RGE) which projects data into a reduced dimensional space while constructs a principal tree which passes through the middle of the data simultaneously. DDRTree shows superiority to alternatives (Wishbone, DPT) for inferring the ordering as well as the intrinsic structure of the single cell genomics data. In general, it could be used to reconstruct the temporal progression as well as bifurcation structure of any datatype.

License Artistic License 2.0

RoxygenNote 6.0.1

SystemRequirements C++11

NeedsCompilation yes

Repository CRAN

Date/Publication 2017-04-30 20:54:17 UTC

R topics documented:

`DDRTree` *Perform DDRTree construction*

Description

Perform DDRTree construction

This is an R and C code implementation of the DDRTree algorithm from Qi Mao, Li Wang et al.

Qi Mao, Li Wang, Steve Goodison, and Yijun Sun. Dimensionality Reduction via Graph Structure Learning. The 21st ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD'15), 2015
http://dl.acm.org/citation.cfm?id=2783309

to perform dimension reduction and principal graph learning simultaneously. Please cite this package and KDD'15 paper if you found DDRTree is useful for your research.

Usage

```
DDRTree(X, dimensions = 2, initial_method = NULL, maxIter = 20,
  sigma = 0.001, lambda = NULL, ncenter = NULL, param.gamma = 10,
  tol = 0.001, verbose = F, ...)
```

Arguments

`X`	a matrix with $\mathbf{D} \times \mathbf{N}$ dimension which is needed to perform DDRTree construction
`dimensions`	reduced dimension
`initial_method`	a function to take the data transpose of X as input and then output the reduced dimension, row number should not larger than observation and column number should not be larger than variables (like isomap may only return matrix on valid sample sets). Sample names of returned reduced dimension should be preserved.
`maxIter`	maximum iterations
`sigma`	bandwidth parameter
`lambda`	regularization parameter for inverse graph embedding
`ncenter`	number of nodes allowed in the regularization graph
`param.gamma`	regularization parameter for k-means (the prefix of 'param' is used to avoid name collision with gamma)
`tol`	relative objective difference
`verbose`	emit extensive debug output
`...`	additional arguments passed to DDRTree

Value

a list with W, Z, stree, Y, history W is the orthogonal set of d (dimensions) linear basis vector Z is the reduced dimension space stree is the smooth tree graph embedded in the low dimension space Y represents latent points as the center of Z

Introduction

The unprecedented increase in big-data causes a huge difficulty in data visualization and downstream analysis. Conventional dimension reduction approaches (for example, PCA, ICA, Isomap, LLE, etc.) are limited in their ability to explictly recover the intrinisic structure from the data as well as the discriminative feature representation, both are important for scientific discovery. The DDRTree algorithm is a new algorithm to perform the following three tasks in one setting:

1. Reduce high dimension data into a low dimension space

2. Recover an explicit smooth graph structure with local geometry only captured by distances of data points in the low dimension space.

3. Obtain clustering structures of data points in reduced dimension

Dimensionality reduction via graph structure learning

Reverse graph embedding is previously applied to learn the intrinisic graph structure in the original dimension. The optimization of graph inference can be represented as:

$$\underset{f_g \in \mathcal{F}}{min} \underset{\{\mathbf{z}_1,...,\mathbf{z}_M\}}{min} \sum_{(V_i,V_j)\in\mathcal{E}} b_{i,j}||f_g(\mathbf{z}_i) - f_g(\mathbf{z}_j)||^2$$

where f_g is a function to map the instrinsic data space $\mathcal{Z} = \{\mathbf{z}_1, ..., \mathbf{z}_M\}$ back to the input data space (reverse embedding) $\mathcal{X} = \{\mathbf{x}_1, ..., \mathbf{x}_N\}$. V_i is the the vertex of the instrinsic undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. b_{ij} is the edge weight associates with the edge set $\mathcal{E}$. In order to learn the intrinsic structure from a reduced dimension, we need also to consider a term which includes the error during the learning of the instrinsic structure. This strategy is incorporated as the following:

$$\underset{\mathcal{G}\in\hat{\mathcal{G}}_b}{min} \underset{f_g \in \mathcal{F}}{min} \underset{\{\mathbf{z}_1,...,\mathbf{z}_M\}}{min} \sum_{i=1}^{N} ||\mathbf{x}_i - f_g(\mathbf{z}_i)||^2 + \frac{\lambda}{2} \sum_{(V_i,V_j)\in\mathcal{E}} b_{i,j}||f_g(\mathbf{z}_i) - f_g(\mathbf{z}_j)||^2$$

where λ is a non-negative parameter which controls the tradeoff between the data reconstruction error and the reverse graph embedding.

Dimensionality reduction via learning a tree

The general framework for reducing dimension by learning an intrinsic structure in a low dimension requires a feasible set $\hat{\mathcal{G}}_b$ of graph and a mapping function $f_{\mathcal{G}}$. The algorithm uses minimum spanning tree as the feasible tree graph structure, which can be solved by Kruskal' algoritm. A linear projection model $f_g(\mathbf{z}) = \mathbf{W}\mathbf{z}$ is used as the mapping function. Those setting results in the following specific form for the previous framework:

$$\underset{\mathbf{W},\mathbf{Z},\mathbf{B}}{min}\sum_{i=1}^{N}||\mathbf{x}_i-\mathbf{W}\mathbf{z}_i||^2+\frac{\lambda}{2}\sum_{i,j}b_{i,j}||\mathbf{W}\mathbf{z}_i-\mathbf{W}\mathbf{z}_j||^2$$

where $\mathbf{W}=[\mathbf{w}_1,...,\mathbf{w}_d]\in\mathcal{R}^{D\times d}$ is an orthogonal set of d linear basis vectors. We can group tree graph $\mathbf{B}$, the orthogonal set of linear basis vectors and projected points in reduced dimension $\mathbf{W},\mathbf{Z}$ as two groups and apply alternative structure optimization to optimize the tree graph. This method is defined as DRtree (Dimension Reduction tree) as discussed by the authors.

Discriminative dimensionality reduction via learning a tree

In order to avoid the issues where data points scattered into different branches (which leads to lose of cluster information) and to incorporate the discriminative information,another set of points $\{\mathbf{y}_k\}_{k=1}^{K}$ as the centers of $\{\mathbf{z}_i\}_{i=1}^{N}$ can be also introduced. By so doing, the objective functions of K-means and the DRtree can be simulatenously minimized. The author further proposed a soft partition method to account for the limits from K-means and proposed the following objective function:

$$\underset{\mathbf{W},\mathbf{Z},\mathbf{B},\mathbf{Y},\mathbf{R}}{min}\sum_{i=1}^{N}||\mathbf{x}_i-\mathbf{W}\mathbf{z}_i||^2+\frac{\lambda}{2}\sum_{k,k'}b_{k,k'}||\mathbf{W}\mathbf{y}_k-\mathbf{W}\mathbf{y}'_k||^2+\gamma\Big[\sum_{k=1}^{K}\sum_{i=1}^{N}r_{i,k}||\mathbf{z}_i-\mathbf{y}_k||^2+\sigma\Omega(\mathbf{R})\Big]$$

$$s.t.\ \mathbf{W}^T\mathbf{W}=\mathbf{I},\mathbf{B}\in\mathcal{B},\sum_{k=1}^{K}r_{i,k}=1,r_{i,k}\leq 0,\forall i,\forall k$$

where $\mathbf{R}\in\mathcal{R}^{N\times N},\Omega(\mathbf{R})=\sum_{i=1}^{N}\sum_{k=1}^{k}r_{i,k}log\ r_{i,k}$ is the negative entropy regularization which transforms the hard assignments used in K-means into soft assignments and $\sigma>0$ is the regulization parameter. Alternative structure optimization is again used to solve the above problem by separately optimize each group $\mathbf{W},\mathbf{Z},\mathbf{Y},\mathbf{B},\mathbf{R}$ until convergence.

The actual algorithm of DDRTree

```
1. $\mathbf{Input}$: Data matrix $\mathbf{X}$, parameters $\lambda,\sigma,\gamma$
2. Initialize $\mathbf{Z}$ by PCA
3. $K=N,\mathbf{Y}=\mathbf{Z}$
4. $\mathbf{repeat}$:
5. $d_{k,k'}=||\mathbf{y}_k-\mathbf{y}_{k'}||^2,\forall k,\forall k'$
6. Obtain $\mathbf{B}$ via Kruskal's algorithm
7. $\mathbf{L}=diag(\mathbf{B1})-\mathbf{B}$
8. Compute $\mathbf{R}$ with each element
9. $\tau=diag(\mathbf{1}^T\mathbf{R})$
10. $\mathbf{Q}=\frac{1}{1+\gamma}\Big[\mathbf{I}+\mathbf{R}(\frac{1+\gamma}{\gamma}(\frac{\lambda}{\gamma}\mathbf{L}+\tau)-\mathbf{R}^T\mathbf{R})^{-1}\mathbf{R}^T\Big]$
11. $\mathbf{C}=\mathbf{XQX}^T$
12. Perform eigen-decomposition on $\mathbf{C}$ such that $\mathbf{C}=\mathbf{U}\wedge\mathbf{U}^T$ and $diag(\wedge)$ is sorted in a descending order
13. $\mathbf{W}=\mathbf{U}(:,1:d)$
14. $\mathbf{Z}=\mathbf{W}^T\mathbf{XQ}$
15. $\mathbf{Y}=\mathbf{ZR}(\frac{\lambda}{\gamma}\mathbf{L}+\tau)^{-1}$
16. $\mathbf{Until}$ Convergence
```

Implementation of DDRTree algorithm

We implemented the algorithm mostly in Rcpp for the purpose of efficiency. It also has extensive optimization for sparse input data. This implementation is originally based on the matlab code provided from the author of DDRTree paper.

Examples

```
data('iris')
subset_iris_mat <- as.matrix(t(iris[c(1, 2, 52, 103), 1:4])) #subset the data
#run DDRTree with ncenters equal to species number
DDRTree_res <- DDRTree(subset_iris_mat, dimensions = 2, maxIter = 5, sigma = 1e-2,
lambda = 1, ncenter = 3, param.gamma = 10, tol = 1e-2, verbose = FALSE)
Z <- DDRTree_res$Z #obatain matrix
Y <- DDRTree_res$Y
stree <- DDRTree_res$stree
plot(Z[1, ], Z[2, ], col = iris[c(1, 2, 52, 103), 'Species']) #reduced dimension
legend("center", legend = unique(iris[c(1, 2, 52, 103), 'Species']), cex=0.8,
col=unique(iris[c(1, 2, 52, 103), 'Species']), pch = 1) #legend
title(main="DDRTree reduced dimension", col.main="red", font.main=4)
dev.off()
plot(Y[1, ], Y[2, ], col = 'blue', pch = 17) #center of the Z
title(main="DDRTree smooth principal curves", col.main="red", font.main=4)

#run DDRTree with ncenters equal to species number
DDRTree_res <- DDRTree(subset_iris_mat, dimensions = 2, maxIter = 5, sigma = 1e-3,
lambda = 1, ncenter = NULL,param.gamma = 10, tol = 1e-2, verbose = FALSE)
Z <- DDRTree_res$Z #obatain matrix
Y <- DDRTree_res$Y
stree <- DDRTree_res$stree
plot(Z[1, ], Z[2, ], col = iris[c(1, 2, 52, 103), 'Species']) #reduced dimension
legend("center", legend = unique(iris[c(1, 2, 52, 103), 'Species']), cex=0.8,
col=unique(iris[c(1, 2, 52, 103), 'Species']), pch = 1) #legend
title(main="DDRTree reduced dimension", col.main="red", font.main=4)
dev.off()
plot(Y[1, ], Y[2, ], col = 'blue', pch = 2) #center of the Z
title(main="DDRTree smooth principal graphs", col.main="red", font.main=4)
```

`get_major_eigenvalue` *Get the top L eigenvalues*

Description

Get the top L eigenvalues

Usage

```
get_major_eigenvalue(C, L)
```

Arguments

`C`	data matrix used for eigendecomposition
`L`	number for the top eigenvalues

`pca_projection_R` *Compute the PCA projection*

Description

Compute the PCA projection

Usage

```
pca_projection_R(C, L)
```

Arguments

`C`	data matrix used for PCA projection
`L`	number for the top principal components

`sqdist_R` *calculate the square distance between a, b*

Description

calculate the square distance between a, b

Usage

```
sqdist_R(a, b)
```

Arguments

`a`	a matrix with $D \times N$ dimension
`b`	a matrix with $D \times N$ dimension

Value

a numeric value for the different between a and b

Index