Learning the topology of graphs

Convex Group-HKUST

2018-08-17

Contents

---

1 Installation

For installation instructions, please visit https://github.com/dppalomar/spectralGraphTopology

2 Problem Statement

The underlying optimization problem of learning a K-component graph may be expressed as follows

$$\begin{array}{ll} \underset{\mathbf{w},\boldsymbol{\Lambda},\mathbf{U}}{\textsf{minimize}} & -\log\det(\boldsymbol{\Lambda}) + \mathrm{tr}\left(\mathbf{K}\mathcal{L}\mathbf{w}\right) + \frac{\beta}{2}\left\|\mathcal{L}\mathbf{w} - \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T\right\|_F^2 \\ \textsf{subject to} & \mathbf{w} \geq 0, \boldsymbol{\Lambda} \in \mathcal{S}_{\boldsymbol{\Lambda}}, \text{ and } \mathbf{U}^T\mathbf{U} = \mathbf{I} \end{array}$$

in which $\mathcal{S}_{\boldsymbol{\Lambda}}$ is the space of eigenvalues of matrices $\boldsymbol{\Theta}$ which are positive semi-definite, symmetric, and whose sum of the elements of any row or column is equal to zero.

In order to solve this problem, we use a block coordinate descent algorithm to iteratively optimize each variable while helding the others fixed.

2.1 w-update

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \frac{\nabla_{\mathbf{w}} f(\mathbf{w}^{(k)})}{2N} \tag{1}$$

3 Usage of the package

We illustrate the usage of the package with simulated data, as follows:

```
library(spectralGraphTopology)
set.seed(123)

# Number of samples
T <- 200
# Vector to generate the Laplacian matrix of the graph
w <- runif(10)
```

```r
# Laplacian matrix
Theta <- L(w)
# Sample data from a Multivariate Gaussian
N <- ncol(Theta)
Y <- MASS::mvrnorm(T, rep(0, N), MASS::ginv(Theta))
# Number of components of the graph
K <- 1
# Learn the Laplacian matrix
res <- learnGraphTopology(Y, K, beta = 10)
```

Let's visually inspect the true Laplacian and the estimated one:

```r
Theta
#>            [,1]       [,2]       [,3]       [,4]       [,5]
#> [1,]  2.3678770 -0.2875775 -0.7883051 -0.4089769 -0.8830174
#> [2,] -0.2875775  1.8017068 -0.9404673 -0.0455565 -0.5281055
#> [3,] -0.7883051 -0.9404673  3.1726265 -0.8924190 -0.5514350
#> [4,] -0.4089769 -0.0455565 -0.8924190  1.8035672 -0.4566147
#> [5,] -0.8830174 -0.5281055 -0.5514350 -0.4566147  2.4191726
res$Theta
#>            [,1]        [,2]       [,3]        [,4]       [,5]
#> [1,]  2.2953931 -0.27079062 -0.6305267 -0.41598450 -0.9780912
#> [2,] -0.2707906  1.95134562 -1.0254774 -0.06127561 -0.5938020
#> [3,] -0.6305267 -1.02547743  2.9706229 -0.83893311 -0.4756856
#> [4,] -0.4159845 -0.06127561 -0.8389331  1.91751385 -0.6013206
#> [5,] -0.9780912 -0.59380196 -0.4756856 -0.60132063  2.6488995
```

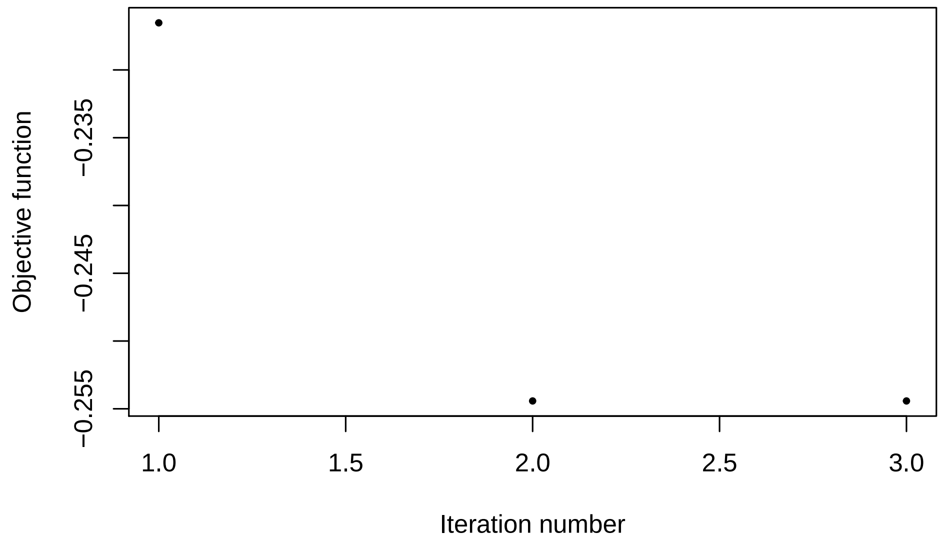
We can evaluate the performance of the learning process in a more objective manner by computing the relative error between the true Laplacian matrix and the estimated one, which can be done as follows:

```r
norm(Theta - res$Theta, type="F") / norm(Theta, type="F")
#> [1] 0.08869914

Theta_naive <- MASS::ginv(cov(Y))
norm(Theta - Theta_naive, type="F") / norm(Theta, type="F")
#> [1] 0.0922406
```

Let's also look at the convergence of the objective function versus iteration:

```r
k <- length(res$fun)
plot(c(1:k), res$fun,  pch=19, cex=.5,
     xlab = "Iteration number", ylab = "Objective function")
```

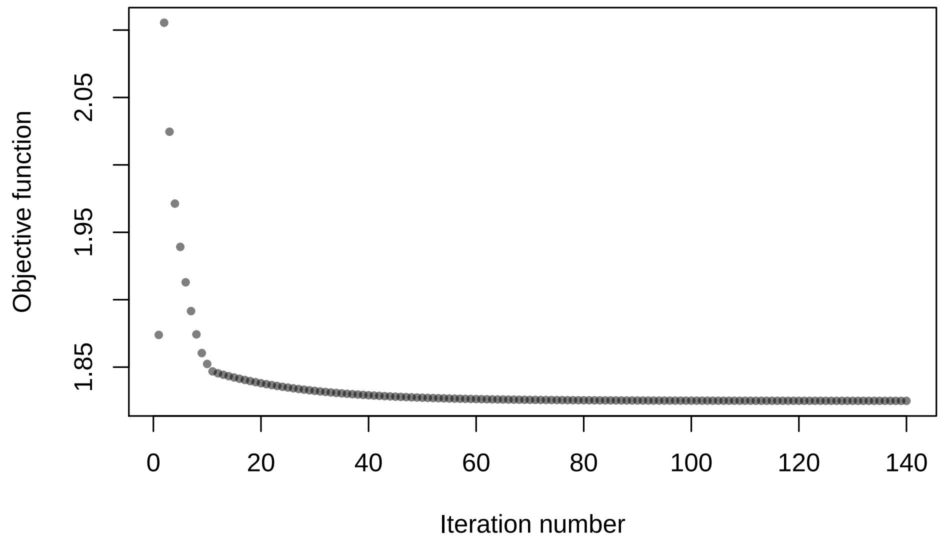


For K > 1, we can generate the Laplacian as a block diagonal matrix, as follows

```r
library(spectralGraphTopology)
T <- 200
w1 <- runif(3)
w2 <- runif(3)
Theta1 <- L(w1)
Theta2 <- L(w2)
N1 <- ncol(Theta1)
N2 <- ncol(Theta2)
Theta <- rbind(cbind(Theta1, matrix(0, N1, N2)),
               cbind(matrix(0, N2, N1), Theta2))
Y <- MASS::mvrnorm(T, rep(0, N1 + N2), MASS::ginv(Theta))
K <- 2
res <- learnGraphTopology(Y, K, beta = 10)
norm(Theta - res$Theta, type="F") / norm(Theta, type="F")
#> [1] 0.1204677
norm(Theta - MASS::ginv(cov(Y)), type="F") / norm(Theta, type="F")
#> [1] 0.1776732
```

```r
Theta
#>            [,1]       [,2]       [,3]       [,4]       [,5]       [,6]
#> [1,]  1.1183094 -0.8509632 -0.2673462  0.0000000  0.0000000  0.0000000
#> [2,] -0.8509632  1.4495635 -0.5986003  0.0000000  0.0000000  0.0000000
#> [3,] -0.2673462 -0.5986003  0.8659465  0.0000000  0.0000000  0.0000000
#> [4,]  0.0000000  0.0000000  0.0000000  1.6007581 -0.6085997 -0.9921584
#> [5,]  0.0000000  0.0000000  0.0000000 -0.6085997  0.7997897 -0.1911900
#> [6,]  0.0000000  0.0000000  0.0000000 -0.9921584 -0.1911900  1.1833484
res$Theta
#>             [,1]       [,2]       [,3]       [,4]       [,5]        [,6]
#> [1,]  1.29785412 -0.9860017 -0.2678456  0.0000000  0.0000000 -0.04400687
#> [2,] -0.98600169  1.5451241 -0.5591224  0.0000000  0.0000000  0.00000000
#> [3,] -0.26784556 -0.5591224  0.8269680  0.0000000  0.0000000  0.00000000
#> [4,]  0.00000000  0.0000000  0.0000000  1.3877236 -0.4591925 -0.92853108
#> [5,]  0.00000000  0.0000000  0.0000000 -0.4591925  0.6864209 -0.22722839
#> [6,] -0.04400687  0.0000000  0.0000000 -0.9285311 -0.2272284  1.19976633
```

```r
k <- length(res$fun)
plot(c(1:k), res$fun, pch=19, cex=.6, col = scales::alpha("black", .5),
     xlab = "Iteration number", ylab = "Objective function")
```



4 Explanation of the algorithms

In this section we describe in detail the algorithms designed to solve the graph topology learning problem.

4.1 `learnGraphTopology`: Learning the topology of graph

The goal of `learnGraphTopology()` is to estimate the Laplacian matrix generated by the weight vector of a graph, $\mathbf{w}$. The algorithm for the function `learnGraphTopology` is stated as follows:

Data: $\mathbf{Y}$ (data matrix), K (#{components}), β (regularization term), $\mathbf{w}^{(0)}$, $\boldsymbol{\lambda}^{(0)}$, $\mathbf{U}^{(0)}$ (initial parameter estimates), α_1, α_2 (lower and upper bound on the eigenvalues of the Laplacian matrix), ρ (how much to increase beta per iteration)

Result: $\boldsymbol{\Theta}$ (Laplacian matrix)

$N \leftarrow$ `ncol`$(\mathbf{Y})$

while *objective function do not converged or max #{iterations} not reached* **do**

  $k \leftarrow 0$

  while *parameters do not converged or max #{iterations} not reached* **do**

    $\mathbf{w}^{(k+1)} \leftarrow$ `w_update`$(\mathbf{w}^{(k)}, \mathbf{U}^{(k)}, \boldsymbol{\lambda}^{(k)}, \beta, N, \mathbf{K})$

    $\mathbf{U}^{(k+1)} \leftarrow$ `U_update`$(\mathbf{w}^{(k+1)}, N)$

    $\boldsymbol{\lambda}^{(k+1)} \leftarrow$ `lambda_update`$(\mathbf{w}^{(k+1)}, \mathbf{U}^{(k+1)}, \alpha_1, \alpha_2, \beta, N, K)$

    $k \leftarrow k + 1$

  end

  $\beta \leftarrow \beta(\rho + 1)$

end

return $\mathcal{L}(\mathbf{w}^{(k+1)})$

Function `w_update`$(\mathbf{w}, \mathbf{U}, \boldsymbol{\lambda}, \beta, N, \mathbf{K})$**:**

  $\nabla_{\mathbf{w}} f \leftarrow \mathcal{L}^{\star}\left(\mathcal{L}(\mathbf{w}) - \mathbf{U}\texttt{diag}(\boldsymbol{\lambda})\mathbf{U}^T + \frac{\mathbf{K}}{\beta}\right)$

  return $\texttt{max}\left(0, \mathbf{w} - \frac{\nabla_{\mathbf{w}} f}{2N}\right)$

Function `U_update`$(\mathbf{w}, N)$**:**

  return `eigenvectors`$(\mathcal{L}(\mathbf{w}))$ # *increasing order w.r.t. eigenvalues*

Function `lambda_update`$(\mathbf{w}, \mathbf{U}, \alpha_1, \alpha_2, \beta, N, K)$**:**

  $\mathbf{d} \leftarrow \texttt{diag}\left(\mathbf{U}^T \mathcal{L}(\mathbf{w}) \mathbf{U}\right)$

  $\boldsymbol{\lambda} \leftarrow \frac{1}{2}\left(\mathbf{d} + \sqrt{\mathbf{d} \odot \mathbf{d} + \frac{4}{\beta}}\right)$ # $\odot$ *means element-wise multiplication*

  if $\boldsymbol{\lambda}$ *has its elements in nondecreasing order **and*** $\min(\boldsymbol{\lambda}) \geq \alpha_1$ ***and*** $\max(\boldsymbol{\lambda}) \leq \alpha_2$ **then**

    return $\boldsymbol{\lambda}$

  else

    set to α_1 the elements of $\boldsymbol{\lambda}$ whose values are less than α_1

    set to α_2 the elements of $\boldsymbol{\lambda}$ whose values are greater than α_2

  end

  if $\boldsymbol{\lambda}$ *has its elements in nondecreasing order* **then**

    return $\boldsymbol{\lambda}$

  else

    raise `Exception`(*"eigenvalues are not in increasing order"*)

  end

References