Learning the topology of graphs

Convex Group-HKUST

2018-08-17

Contents

---

1 Problem Statement

The problem of learning a K-component graph may be expressed mathematically as

$$\texttt{minimize}_{\mathbf{w},\mathbf{\Lambda},\mathbf{U}} - \log\det(\mathbf{\Lambda}) + \operatorname{tr}(\mathbf{K}\mathcal{L}\mathbf{w}) + \frac{\beta}{2}||\mathcal{L}\mathbf{w} - \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T||_F^2 \tag{1}$$

$$\texttt{subject to } \mathbf{w} \geq 0, \mathbf{\Lambda} \in \mathcal{S}_{\mathbf{\Lambda}}, \mathbf{U}^T\mathbf{U} = \mathbf{I}, \tag{2}$$

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

We use a block coordinate descent to optimize each variable while helding the others fixed. Therefore, problem (**??**) can be divided in the following problems:

2 Usage of the package

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

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

# Number of samples
T <- 20
# Vector to generate the Laplacian matrix of the graph
w <- runif(10)
# 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:

```
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,]  3.2548374 -0.4536832 -0.4571741 -0.4580733 -1.8859068
#> [2,] -0.4536832  1.9060008 -1.0409410  0.0000000 -0.4113766
#> [3,] -0.4571741 -1.0409410  3.7007733 -1.6002919 -0.6023664
#> [4,] -0.4580733  0.0000000 -1.6002919  2.5778784 -0.5195132
#> [5,] -1.8859068 -0.4113766 -0.6023664 -0.5195132  3.4191631
```

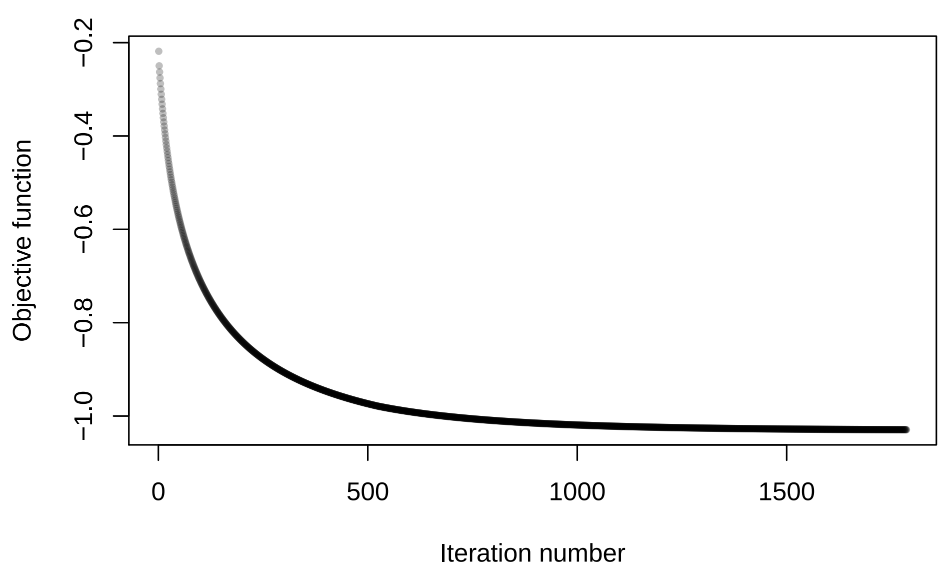
We can evaluate the performance of the learning process in a more objective manner by computing the relative error between the true Laplacian matrix and the estimated one, which can be done as follows:

```
norm(Theta - res$Theta, type="F") / norm(Theta, type="F")
#> [1] 0.407842

Theta_naive <- ginv(cov(Y))
norm(Theta - Theta_naive, type="F") / norm(Theta, type="F")
#> [1] 0.4477575
```

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

```
k <- length(res$fun)
plot(c(1:k), res$fun,  pch=19, cex=.5, col = scales::alpha("black", .25),
     xlab = "Iteration number", ylab = "Objective function")
```



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

```
library(spectralGraphTopology)
T <- 20
```

```r
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
beta <- 5
res <- learnGraphTopology(Y, K, beta = beta, ftol = 1e-3)
norm(Theta - res$Theta, type="F") / norm(Theta, type="F")
#> [1] 0.7435981
norm(Theta - ginv(cov(Y)), type="F") / norm(Theta, type="F")
#> [1] 0.8690037
```

```r
Theta
#>            [,1]       [,2]       [,3]        [,4]       [,5]        [,6]
#> [1,]  1.2436331 -0.2825283 -0.9611048  0.00000000  0.0000000  0.00000000
#> [2,] -0.2825283  1.0109228 -0.7283944  0.00000000  0.0000000  0.00000000
#> [3,] -0.9611048 -0.7283944  1.6894992  0.00000000  0.0000000  0.00000000
#> [4,]  0.0000000  0.0000000  0.0000000  0.73921902 -0.6863751 -0.05284394
#> [5,]  0.0000000  0.0000000  0.0000000 -0.68637508  1.0815952 -0.39522013
#> [6,]  0.0000000  0.0000000  0.0000000 -0.05284394 -0.3952201  0.44806408
res$Theta
#>            [,1]       [,2]       [,3]       [,4]       [,5]       [,6]
#> [1,]  1.4927989  0.0000000 -0.9343098  0.0000000 -0.5584891  0.0000000
#> [2,]  0.0000000  0.8696941 -0.2299241 -0.1676904 -0.4720797  0.0000000
#> [3,] -0.9343098 -0.2299241  2.1503082  0.0000000 -0.9860742  0.0000000
#> [4,]  0.0000000 -0.1676904  0.0000000  0.5936755  0.0000000 -0.4259851
#> [5,] -0.5584891 -0.4720797 -0.9860742  0.0000000  2.0166430  0.0000000
#> [6,]  0.0000000  0.0000000  0.0000000 -0.4259851  0.0000000  0.4259851
```

As we can observe, the matrices' structure do not quite match.
I suspect they might be similar matrices. Let's check some properties:

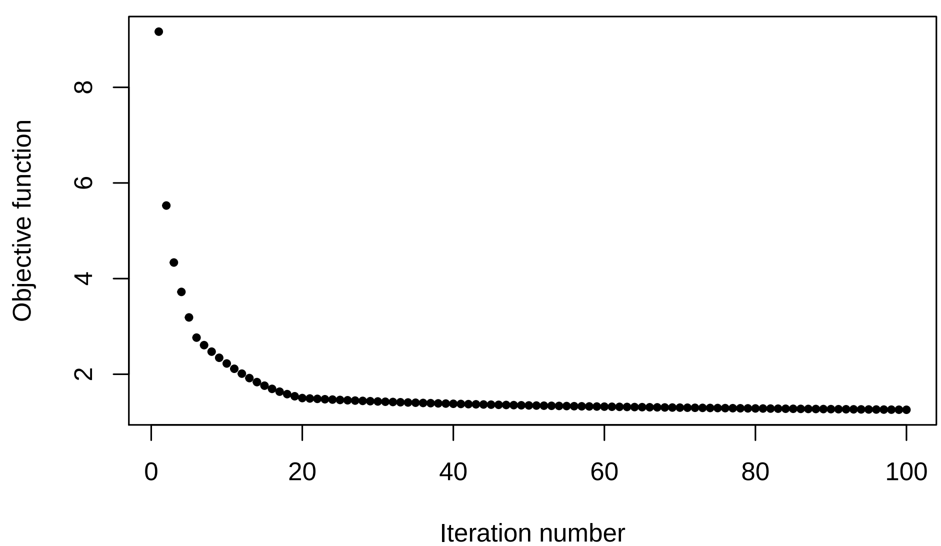
```r
eigen(Theta, only.values = TRUE)
#> $values
#> [1]  2.569278e+00  1.683691e+00  1.374777e+00  5.851877e-01  6.054185e-16
#> [6] -5.237570e-18
#>
#> $vectors
#> NULL
eigen(res$Theta, only.values = TRUE)
#> $values
#> [1] 3.155595e+00 2.381399e+00 1.068947e+00 8.444937e-01 9.866976e-02
#> [6] 3.091021e-16
#>
#> $vectors
#> NULL
```

```r
sum(diag(Theta))
#> [1] 6.212933
sum(diag(res$Theta))
#> [1] 7.549105
```

```r
det(Theta)
#> [1] 2.383136e-33
det(res$Theta)
#> [1] 1.08372e-17
```

```r
N <- N1 + N2
evd_true <- eigen(Theta)
vec_true <- evd_true$vectors[, N:1]
vec_true
#>            [,1]       [,2]       [,3]       [,4]       [,5]       [,6]
#> [1,]  0.0000000 -0.5773503  0.0000000  0.6267502  0.0000000  0.5233076
#> [2,]  0.0000000 -0.5773503  0.0000000 -0.7665728  0.0000000  0.2811278
#> [3,]  0.0000000 -0.5773503  0.0000000  0.1398226  0.0000000 -0.8044354
#> [4,] -0.5773503  0.0000000 -0.5906573  0.0000000  0.5637292  0.0000000
#> [5,] -0.5773503  0.0000000 -0.1928751  0.0000000 -0.7933888  0.0000000
#> [6,] -0.5773503  0.0000000  0.7835325  0.0000000  0.2296597  0.0000000
vals_true <- evd_true$values[N:1]
vals_true
#> [1] 1.110223e-15 1.332268e-15 5.851877e-01 1.374777e+00 1.683691e+00
#> [6] 2.569278e+00
res$lambda
#> [1] 0.000000 0.000000 1.037302 1.231368 2.462613 3.217750
objFunction(Theta, vec_true, vals_true, res$Km, beta, N, K)
#> [1] 2.460337
objFunction(res$Theta, vec_true, vals_true, res$Km, beta, N, K)
#> [1] 18.24495
```

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



3 Explanation of the algorithms

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

3.1 `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:

Algorithm 1: Blah, blah, blah
Data: asdljflasdf**
begin**
    asdkjhfaksljdf
    asdfsdfasd
end

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)
$N \leftarrow \texttt{ncol}(\mathbf{Y})$
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**
        $\mathbf{w}^{(k+1)} \leftarrow \texttt{w\_update}(\mathbf{w}^{(k)}, \mathbf{U}^{(k)}, \boldsymbol{\lambda}^{(k)}, \beta, N, \mathbf{K})$
        $\mathbf{U}^{(k+1)} \leftarrow \texttt{U\_update}(\mathbf{w}^{(k+1)}, N)$
        $\boldsymbol{\lambda}^{(k+1)} \leftarrow \texttt{lambda\_update}(\mathbf{w}^{(k+1)}, \mathbf{U}^{(k+1)}, \alpha_1, \alpha_2, \beta, N, K)$
        $k \leftarrow k+1$
    end
    $\beta \leftarrow \beta(\rho+1)$
end
return $\mathcal{L}(\mathbf{w}^{(k+1)})$

Function $\texttt{w\_update}(\mathbf{w}, \mathbf{U}, \boldsymbol{\lambda}, \beta, N, \mathbf{K})$**:**
    $\nabla_{\mathbf{w}} f \leftarrow \mathcal{L}^{\star}\left(\mathcal{L}\left(\mathbf{w}\right) - \mathbf{U}\texttt{diag}(\boldsymbol{\lambda})\mathbf{U}^T + \dfrac{\mathbf{K}}{\beta}\right)$
    return $\texttt{max}\left(0, \mathbf{w} - \dfrac{\nabla_{\mathbf{w}} f}{2N}\right)$

Function $\texttt{U\_update}(\mathbf{w}, N)$**:**
    return $\texttt{eigen}(\mathcal{L}(\mathbf{w}))\$\texttt{vectors}[, N:1]$

Function $\texttt{lambda\_update}(\mathbf{w}, \mathbf{U}, \alpha_1, \alpha_2, \beta, N, K)$**:**
    $\mathbf{d} \leftarrow \texttt{diag}\left(\mathbf{U}^T \mathcal{L}(\mathbf{w}) \mathbf{U}\right)$
    $\boldsymbol{\lambda} \leftarrow \frac{1}{2}\left(\mathbf{d} + \sqrt{\mathbf{d} \odot \mathbf{d} + \frac{4}{\beta}}\right)$
    if $\boldsymbol{\lambda}$ *has its elements in increasing order* **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 increasing order* **then**
        return $\boldsymbol{\lambda}$
    else
        raise `Exception`(*"eigenvalues are not in increasing order"*)
    end

References