Using the hdInference package

Renxiong Liu and Yunzhang Zhu

2021-08-16

The `hdInference` package contains the implementations of likelihood ratio test for a subset of parameters in Gaussian graphical model. The testing problem can be formalized as:

$$\hat{\theta}^{(0)} = \arg\max_{\theta} \; L_n(\theta) \quad \text{subject to:} \sum_{i \notin B} p_\tau(|\theta_i|) \leq K \text{ and } \theta_B = 0$$
$$\hat{\theta}^{(1)} = \arg\max_{\theta} \; L_n(\theta) \quad \text{subject to:} \sum_{i \notin B} p_\tau(|\theta_i|) \leq K,$$

where $L_n(\theta) = \sum_{i=1}^n \log p_\theta(X_i)$ is the log-likelihood for Gaussian graphical model, $p_\tau(x) = \min(x/\tau, 1)$ is the truncated L_1 function as the surrogate function of L_0 function, and (K, τ) are nonnegative tuning parameters. The details of proposal can be found in Zhu, Y., Shen, X., Pan, W. (2020). *On High-Dimensional Constrained Maximum Likelihood Inference*.

Using `hdInference` package

1. `glasso_nonconvex_constrained_cv` function

To use the `glasso_nonconvex_constrained_cv` function in our `hdInference` package, you need to specify

1. **sim**: a list containing a $n \times d$ data matrix;
2. **bound**: a vector of upper bounds for the constraint (K in above testing problem);
3. **tau**: tuning parameter for the truncated L_1 penalty (τ in above testing problem);
4. **num.fold**: fold number for cross validation.

2. `inference_constrained` function

Similarly, to apply `inference_constrained` function, you need to specify

1. **sim**: a list containing a $n \times d$ data matrix;
2. **para_index**: indices of the parameters of interest (B in above testing problem);
2. **bound**: a vector of upper bounds for the constraint (K in above testing problem);
3. **tau**: tuning parameter for the truncated L_1 penalty (τ in above testing problem);
4. **inference_type**: either 'LR' or 'LR_gen' with the former based on chi-square test and the later based on normal approximation.

Real data example

We next elaborate the details of using `hdInference` package with a real data example. We consider the ADNI-1 baseline data (adni.loni.usc.edu) for brain network analysis. To load ADNI-1 baseline data, type in R console

```
library(hdInference);
path_to_data=system.file("extdata", "ADNILongiBaseLine2.csv", package = "hdInference");
```

After loading the data, we can extract the cortical thicknesses for $p = 68$ regions of interest (ROIs). Since some previous studies have identified default mode network (DMN) to be associated with Alzheimer's disease (AD), we will pay particular attention to this sub-network which includes 12 ROIs. To this end, we first regress the cortical thickness on 5 covariates, then use the residuals to estimate precision matrices. After obtaing the residuals, we apply our `glasso_nonconvex_constrained_cv` function to get the cross-validation score.

```
ROI.index = 2:69
feature.index = 70:74
sr_data = read.table(file=path_to_data,header=TRUE,sep=",")
n = dim(sr_data)[1]
p = length(ROI.index)
condition = c("LMCI","AD","CN")

res.mat = matrix(0,n,length(ROI.index))
for (i in ROI.index)
{
  data.tmp = sr_data[,c(i,feature.index)]
  tmp.fit = lm(data.tmp)
  res.mat[,i-1] = tmp.fit$residuals
}

sim = list(data = res.mat,sigmahat = cor(res.mat))
bound = p*c(9,10,11)
cv.score = glasso_nonconvex_constrained_cv(sim,bound)
## bound.best.all = 10*p using all the data ##
bound.best.all = bound[which.min(apply(cv.score,2,mean))]
```

Similar idea can be used to model three groups "LMCI","AD","CN" separately and obtain the best bound parameters.

```
cv.score.separate = list()
bound.best.separate = list()
for (j in seq_along(condition))
{
  desease = condition[j]
  desease.index = which(sr_data$Disease==desease)
  data_desease = sr_data[desease.index,]
  n.desease = dim(data_desease)[1]
  res.mat = matrix(0,n.desease,length(ROI.index))
  for (i in ROI.index)
  {
    data.tmp = data_desease[,c(i,feature.index)]
    tmp.fit = lm(data.tmp)
    res.mat[,i-1] = tmp.fit$residuals
  }
  sim = list(data = res.mat,sigmahat = cor(res.mat))
  bound = p*c(2,3,4,5,6,7,8,9,10,11,12)
  cv.score.separate[[j]] = glasso_nonconvex_constrained_cv(sim,bound)
  bound.best.separate[[j]] = bound[which.min(apply(cv.score.separate[[j]],2,mean))]
}
```

Now, we can use above best bound parameters to test dependence between the first DMN nodes and the rest of the nodes by our `inference_constrained` function.

```
LR.DMN.rest = list()
for (j in seq_along(condition))
{
  generate.index.pairs = function(A,B)
  {
    index.pairs = rep(0,2*length(A)*length(B))
    counter = 1
    for (i in A){
      for (j in B){
        index.pairs[counter:(counter+1)] = c(i,j)
        counter = counter + 2
      }
    }
    return (index.pairs)
  }
  desease = condition[j]
  desease.index = which(sr_data$Disease==desease)
  data_desease = sr_data[desease.index,]
  n.desease = dim(data_desease)[1]
  res.mat = matrix(0,n.desease,length(ROI.index))
  for (i in ROI.index)
  {
    data.tmp = data_desease[,c(i,feature.index)]
    tmp.fit = lm(data.tmp)
    res.mat[,i-1] = tmp.fit$residuals
  }
  sim = list(data = res.mat,sigmahat = cor(res.mat))
  bound = bound.best.separate[[j]]
  LR = rep(0,12)
  for (k in 1:12){
    para_index = generate.index.pairs(k,13:length(ROI.index))
    LR[k] = inference_constrained(sim, para_index, bound=bound, inference_type="LR_gen")$LR
  }
  LR.DMN.rest[[j]] = LR
}
```