Package 'ADMMsigma'

May 19, 2018

Type Package

Title Penalized Precision Matrix Estimation via ADMM

Version 1.0

Date 2018-03-29

Description Estimates a penalized precision matrix via the alternating direction method of multipliers (ADMM) algorithm. It currently supports a general elastic-net penalty that allows for both ridge and lasso-type penalties as special cases. This package is an alternative to the 'glasso' package.
See Boyd et al (2010) <doi:10.1561/2200000016> for details regarding the estimation method.

URL https://github.com/MGallow/ADMMsigma

BugReports https://github.com/MGallow/ADMMsigma/issues

License GPL (>= 2)

ByteCompile TRUE

NeedsCompilation yes

Encoding UTF-8

LazyData true

RoxygenNote 6.0.1

Imports stats,
parallel,
foreach,
ggplot2,
dplyr

Depends Rcpp (>= 0.12.10),
RcppProgress (>= 0.1),
doParallel

LinkingTo Rcpp,
RcppArmadillo,
RcppProgress

Suggests testthat

SystemRequirements GNU make

R topics documented:

`ADMMsigma` *Penalized precision matrix estimation via ADMM*

Description

Penalized precision matrix estimation using the ADMM algorithm. Consider the case where $X_1, ..., X_n$ are iid $N_p(\mu, \Sigma)$ and we are tasked with estimating the precision matrix, denoted $\Omega \equiv \Sigma^{-1}$. This function solves the following optimization problem:

Objective: $\hat{\Omega}_\lambda = \arg\min_{\Omega \in S_+^p} \left\{ Tr\left(S\Omega\right) - \log\det\left(\Omega\right) + \lambda \left[\frac{1-\alpha}{2} \left\|\Omega\right|_F^2 + \alpha \left\|\Omega\right\|_1 \right] \right\}$

where $0 \leq \alpha \leq 1$, $\lambda > 0$, $\|\cdot\|_F^2$ is the Frobenius norm and we define $\|A\|_1 = \sum_{i,j} |A_{ij}|$. This elastic net penalty is identical to the penalty used in the popular penalized regression package `glmnet`. Clearly, when $\alpha = 0$ the elastic-net reduces to a ridge-type penalty and when $\alpha = 1$ it reduces to a lasso-type penalty.

Usage

```
ADMMsigma(X = NULL, S = NULL, lam = 10^seq(-5, 5, 0.5), alpha = seq(0,
  1, 0.1), diagonal = FALSE, path = FALSE, rho = 2, mu = 10,
  tau.inc = 2, tau.dec = 2, crit = c("ADMM", "loglik"), tol.abs = 1e-04,
  tol.rel = 1e-04, maxit = 10000, adjmaxit = NULL, K = 5,
  start = c("warm", "cold"), cores = 1, trace = c("progress", "print",
  "none"))
```

Arguments

`X`	option to provide a nxp data matrix. Each row corresponds to a single observation and each column contains n observations of a single feature/variable.
`S`	option to provide a pxp sample covariance matrix (denominator n). If argument is `NULL` and `X` is provided instead then `S` will be computed automatically.
`lam`	positive tuning parameters for elastic net penalty. If a vector of parameters is provided, they should be in increasing order. Defaults to grid of values `10^seq(-5, 5, 0.5)`.
`alpha`	elastic net mixing parameter contained in [0, 1]. `0 = ridge, 1 = lasso`. If a vector of parameters is provided, they should be in increasing order. Defaults to grid of values `seq(-1, 1, 0.1)`.
`diagonal`	option to penalize the diagonal elements of the estimated precision matrix (Ω). Defaults to `FALSE`.
`path`	option to return the regularization path. This option should be used with extreme care if the dimension is large. If set to TRUE, cores must be set to 1 and errors and optimal tuning parameters will based on the full sample. Defaults to FALSE.

rho	initial step size for ADMM algorithm.
mu	factor for primal and residual norms in the ADMM algorithm. This will be used to adjust the step size rho after each iteration.
tau.inc	factor in which to increase step size rho
tau.dec	factor in which to decrease step size rho
crit	criterion for convergence (ADMM or loglik). If crit != ADMM then tol.abs will be used as the convergence tolerance. Default is ADMM and follows the procedure outlined in Boyd, et al.
tol.abs	absolute convergence tolerance. Defaults to 1e-4.
tol.rel	relative convergence tolerance. Defaults to 1e-4.
maxit	maximum number of iterations. Defaults to 1e4.
adjmaxit	adjusted maximum number of iterations. During cross validation this option allows the user to adjust the maximum number of iterations after the first lam tuning parameter has converged (for each alpha). This option is intended to be paired with warm starts and allows for 'one-step' estimators. Defaults to NULL.
K	specify the number of folds for cross validation.
start	specify warm or cold start for cross validation. Default is warm.
cores	option to run CV in parallel. Defaults to cores = 1.
trace	option to display progress of CV. Choose one of progress to print a progress bar, print to print completed tuning parameters, or none.

Details

For details on the implementation of 'ADMMsigma', see the vignette https://mgallow.github.io/ADMMsigma/.

Value

returns class object ADMMsigma which includes:

Call	function call.
Iterations	number of iterations.
Tuning	optimal tuning parameters (lam and alpha).
Lambdas	grid of lambda values for CV.
Alphas	grid of alpha values for CV.
maxit	maximum number of iterations.
Omega	estimated penalized precision matrix.
Sigma	estimated covariance matrix from the penalized precision matrix (inverse of Omega).
Path	array containing the solution path. Solutions will be ordered in ascending alpha values for each lambda.
Z	final sparse update of estimated penalized precision matrix.
Y	final dual update.
rho	final step size.
Loglik	penalized log-likelihood for Omega
MIN.error	minimum average cross validation error for optimal parameters.
AVG.error	average cross validation error across all folds.
CV.error	cross validation errors (negative validation likelihood).

Author(s)

Matt Galloway <gall0441@umn.edu>

References

- For more information on the ADMM algorithm, see:
  Boyd, Stephen, Neal Parikh, Eric Chu, Borja Peleato, Jonathan Eckstein, and others. 2011. 'Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers.' *Foundations and Trends in Machine Learning* 3 (1). Now Publishers, Inc.: 1-122.
  https://web.stanford.edu/~boyd/papers/pdf/admm_distr_stats.pdf

See Also

plot.ADMM, RIDGEsigma

Examples

```
# generate data from a sparse matrix
# first compute covariance matrix
S = matrix(0.7, nrow = 5, ncol = 5)
for (i in 1:5){
 for (j in 1:5){
   S[i, j] = S[i, j]^abs(i - j)
 }
 }

# generate 100 x 5 matrix with rows drawn from iid N_p(0, S)
Z = matrix(rnorm(100*5), nrow = 100, ncol = 5)
out = eigen(S, symmetric = TRUE)
S.sqrt = out$vectors %*% diag(out$values^0.5)
S.sqrt = S.sqrt %*% t(out$vectors)
X = Z %*% S.sqrt

# elastic-net type penalty (use CV for optimal lambda and alpha)
ADMMsigma(X)

# ridge penalty (use CV for optimal lambda)
ADMMsigma(X, alpha = 0)

# lasso penalty (lam = 0.1)
ADMMsigma(X, lam = 0.1, alpha = 1)
```

---

plot.ADMM *Plot ADMM object*

Description

Produces a heat plot for the cross validation errors, if available.

Usage

```
## S3 method for class 'ADMM'
plot(x, type = c("heatmap", "line"), footnote = TRUE, ...)
```

Arguments

x	class object ADMM.
type	produce either 'heatmap' or 'line' graph
footnote	option to print footnote of optimal values. Defaults to TRUE.
...	additional arguments.

Examples

```
# generate data from a sparse matrix
# first compute covariance matrix
S = matrix(0.7, nrow = 5, ncol = 5)
for (i in 1:5){
 for (j in 1:5){
   S[i, j] = S[i, j]^abs(i - j)
 }
 }

# generate 100 x 5 matrix with rows drawn from iid N_p(0, S)
Z = matrix(rnorm(100*5), nrow = 100, ncol = 5)
out = eigen(S, symmetric = TRUE)
S.sqrt = out$vectors %*% diag(out$values^0.5)
S.sqrt = S.sqrt %*% t(out$vectors)
X = Z %*% S.sqrt

# produce CV heat map for ADMMsigma
plot(ADMMsigma(X))

# produce line graph for ADMMsigma
plot(ADMMsigma(X), type = 'line')
```

---

plot.RIDGE *Plot RIDGE object*

Description

Produces a heat plot for the cross validation errors, if available.

Usage

```
## S3 method for class 'RIDGE'
plot(x, type = c("heatmap", "line"), footnote = TRUE, ...)
```

Arguments

x	class object RIDGE
type	produce either 'heatmap' or 'line' graph
footnote	option to print footnote of optimal values. Defaults to TRUE.
...	additional arguments.

Examples

```
# generate data from a sparse matrix
# first compute covariance matrix
S = matrix(0.7, nrow = 5, ncol = 5)
for (i in 1:5){
 for (j in 1:5){
   S[i, j] = S[i, j]^abs(i - j)
 }
 }

# generate 100 x 5 matrix with rows drawn from iid N_p(0, S)
Z = matrix(rnorm(100*5), nrow = 100, ncol = 5)
out = eigen(S, symmetric = TRUE)
S.sqrt = out$vectors %*% diag(out$values^0.5)
S.sqrt = S.sqrt %*% t(out$vectors)
X = Z %*% S.sqrt

# produce CV heat map for RIDGEsigma
plot(RIDGEsigma(X, lam = 10^seq(-5, 5, 0.5)))

# produce line graph for RIDGEsigma
plot(RIDGEsigma(X), type = 'line')
```

`RIDGEsigma` *Ridge penalized precision matrix estimation*

Description

Ridge penalized matrix estimation via closed-form solution. If one is only interested in the ridge penalty, this function will be faster and provide a more precise estimate than using `ADMMsigma`. Consider the case where $X_1, ..., X_n$ are iid $N_p(\mu, \Sigma)$ and we are tasked with estimating the precision matrix, denoted $\Omega \equiv \Sigma^{-1}$. This function solves the following optimization problem:

Objective: $\hat{\Omega}_\lambda = \arg\min_{\Omega \in S_+^p} \left\{ Tr\left(S\Omega\right) - \log\det\left(\Omega\right) + \frac{\lambda}{2} \left\|\Omega\right\|_F^2 \right\}$

where $\lambda > 0$ and $\|\cdot\|_F^2$ is the Frobenius norm.

Usage

```
RIDGEsigma(X = NULL, S = NULL, lam = 10^seq(-5, 5, 0.5), path = FALSE,
  K = 5, cores = 1, trace = c("none", "progress", "print"))
```

Arguments

`X`	option to provide a nxp data matrix. Each row corresponds to a single observation and each column contains n observations of a single feature/variable.
`S`	option to provide a pxp sample covariance matrix (denominator n). If argument is `NULL` and `X` is provided instead then `S` will be computed automatically.
`lam`	positive tuning parameters for ridge penalty. If a vector of parameters is provided, they should be in increasing order. Defaults to grid of values `10^seq(-5, 5, 0.5)`.

path	option to return the regularization path. This option should be used with extreme care if the dimension is large. If set to TRUE, cores will be set to 1 and errors and optimal tuning parameters will based on the full sample. Defaults to FALSE.
K	specify the number of folds for cross validation.
cores	option to run CV in parallel. Defaults to `cores = 1`.
trace	option to display progress of CV. Choose one of `progress` to print a progress bar, `print` to print completed tuning parameters, or `none`.

Value

returns class object `RIDGEsigma` which includes:

Lambda	optimal tuning parameter.
Lambdas	grid of lambda values for CV.
Omega	estimated penalized precision matrix.
Sigma	estimated covariance matrix from the penalized precision matrix (inverse of Omega).
Path	array containing the solution path. Solutions are ordered dense to sparse.
Gradient	gradient of optimization function (penalized gaussian likelihood).
MIN.error	minimum average cross validation error for optimal parameters.
AVG.error	average cross validation error across all folds.
CV.error	cross validation errors (negative validation likelihood).

Author(s)

Matt Galloway <gall0441@umn.edu>

See Also

plot.RIDGE, ADMMsigma

Examples

```
# generate data from a sparse matrix
# first compute covariance matrix
S = matrix(0.7, nrow = 5, ncol = 5)
for (i in 1:5){
 for (j in 1:5){
   S[i, j] = S[i, j]^abs(i - j)
 }
 }

# generate 100 x 5 matrix with rows drawn from iid N_p(0, S)
Z = matrix(rnorm(100*5), nrow = 100, ncol = 5)
out = eigen(S, symmetric = TRUE)
S.sqrt = out$vectors %*% diag(out$values^0.5)
S.sqrt = S.sqrt %*% t(out$vectors)
X = Z %*% S.sqrt

# ridge penalty no ADMM
RIDGEsigma(X, lam = 10^seq(-5, 5, 0.5))
```

Index