

Package ‘ADMMsigma’

March 26, 2018

Type Package

Title Penalized Precision Matrix Estimation via ADMM

Version 1.0

Date 2018-02-23

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:10.1561/22000000016> 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

Depends Rcpp (>= 0.12.10),
doParallel

LinkingTo Rcpp,
RcppArmadillo

Suggests testthat

SystemRequirements GNU make

R topics documented:

| | |
|---------------------------|---|
| ADMMsigma | 2 |
| plot.ADMMsigma | 4 |
| plot.RIDGEsigma | 5 |
| RIDGEsigma | 5 |

Description

Penalized precision matrix estimation using the ADMM algorithm. Consider the case where X_1, \dots, 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:

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

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, rho = 2, mu = 10, tau1 = 2, tau2 = 2,
  crit = "ADMM", tol1 = 1e-04, tol2 = 1e-04, maxit = 1000, K = 5,
  cores = 1, quiet = TRUE)
```

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 | tuning parameter for elastic net penalty. Defaults to grid of values <code>10^seq(-5, 5, 0.5)</code> . |
| alpha | elastic net mixing parameter contained in [0, 1]. 0 = ridge, 1 = lasso. Defaults to grid of values <code>seq(-1, 1, 0.1)</code> . |
| diagonal | option to penalize the diagonal elements of the estimated precision matrix (Ω). 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. |
| tau1 | factor in which to increase step size rho |
| tau2 | factor in which to decrease step size rho |
| crit | criterion for convergence (ADMM, grad, or loglik). If <code>crit != ADMM</code> then <code>tol1</code> will be used as the convergence tolerance. Default is ADMM. |
| tol1 | absolute convergence tolerance. Defaults to 1e-4. |
| tol2 | relative convergence tolerance. Defaults to 1e-4. |
| maxit | maximum number of iterations. |
| K | specify the number of folds for cross validation. |
| cores | option to run CV in parallel. Defaults to <code>cores = 1</code> . |
| quiet | specify whether the function returns progress of CV or not. |

Details

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

Value

returns class object ADMMsigma which includes:

| | |
|------------|-------------------------------------------------------------------------------------|
| 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). |
| Gradient | gradient of optimization function (penalized gaussian likelihood). |
| CV.error | cross validation errors. |

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.ADMMsigma](#), [RIDGESigma](#)

Examples

```
# generate data from a dense matrix
# first compute covariance matrix
S = matrix(0.9, nrow = 5, ncol = 5)
diag(S) = 1

# 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)

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

| | |
|----------------|------------------------------|
| plot.ADMMSigma | <i>Plot ADMMsigma object</i> |
|----------------|------------------------------|

Description

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

Usage

```
## S3 method for class 'ADMMsigma'
plot(x, footnote = TRUE, ...)
```

Arguments

| | |
|----------|---------------------------------------------------------------|
| x | class object ADMMsigma. |
| footnote | option to print footnote of optimal values. Defaults to TRUE. |
| ... | additional arguments. |

Examples

```
# generate data from a dense matrix
# first compute covariance matrix
S = matrix(0.9, nrow = 5, ncol = 5)
diag(S) = 1

# 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))
```

| | |
|-----------------|-------------------------------|
| plot.RIDGESigma | <i>Plot RIDGESigma object</i> |
|-----------------|-------------------------------|

Description

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

Usage

```
## S3 method for class 'RIDGESigma'
plot(x, footnote = TRUE, ...)
```

Arguments

| | |
|----------|---------------------------------------------------------------|
| x | class object RIDGESigma |
| footnote | option to print footnote of optimal values. Defaults to TRUE. |
| ... | additional arguments. |

Examples

```
# generate data from a dense matrix
# first compute covariance matrix
S = matrix(0.9, nrow = 5, ncol = 5)
diag(S) = 1

# 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(-8, 8, 0.01)))
```

| | |
|------------|----------------------------------------------------|
| RIDGESigma | <i>Ridge penalized precision matrix estimation</i> |
|------------|----------------------------------------------------|

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

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

Usage

```
RIDGEsigma(X = NULL, S = NULL, lam = 10^seq(-5, 5, 0.5), K = 3,
  quiet = TRUE)
```

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 | tuning parameter for ridge penalty. Defaults to grid of values $10^{\text{seq}(-5, 5, 0.5)}$. |
| K | specify the number of folds for cross validation. |
| quiet | specify whether the function returns progress of CV or not. |

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). |
| Gradient | gradient of optimization function (penalized gaussian likelihood). |
| CV.error | cross validation errors. |

Author(s)

Matt Galloway <gall0441@umn.edu>

See Also

[plot.RIDGESigma](#), [ADMMsigma](#)

Examples

```
# generate data from a dense matrix
# first compute covariance matrix
S = matrix(0.9, nrow = 5, ncol = 5)
diag(S) = 1

# 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(-8, 8, 0.01))

# produce CV heat map for RIDGESigma
plot(RIDGEsigma(X, lam = 10^seq(-8, 8, 0.01)))
```

Index

ADMMsigma, [2](#), [6](#)

plot.ADMMsigma, [3](#), [4](#)

plot.RIDGESigma, [5](#), [6](#)

RIDGESigma, [3](#), [5](#)