Package 'MatrixLDA'

August 6, 2018

Type Package

Title Penalized Matrix-Normal Linear Discriminant Analysis

Version 0.2

Date 2018-08-02

Maintainer Aaron J. Molstad <amolstad@fredhutch.org>

Description Fits the penalized matrix-normal model to be used for linear discriminant analysis with matrix-valued predictors. For a description of the method, see Molstad and Rothman (2018) <doi:10.1080/10618600.2018.1476249>.

License GPL-2

URL ajmolstad@github.io

LazyData TRUE

Imports Rcpp (>= 0.12.1), plyr, glasso

LinkingTo Rcpp, RcppArmadillo

RoxygenNote 5.0.1

NeedsCompilation yes

Author Aaron J. Molstad [aut, cre]

Repository CRAN

Date/Publication 2018-08-06 21:00:06 UTC

R topics documented:

`MatLDA` *Fits the J-class penalized matrix-normal model for a single pair of tuning parameters.*

Description

A function for fitting the J-class penalized matrix-normal model based on a single set of tuning parameters (λ_1, λ_2). Returns an object of class "MN", which can be used for prediction using the `PredictMN` function.

Usage

```
MatLDA(X, class, lambda1, lambda2, quiet = TRUE, Xval = NULL,
  classval = NULL, k.iter = 100, cov.tol = 1e-05, m.tol = 1e-05,
  full.tol = 1e-06)
```

Arguments

`X`	An $r \times c \times N$ array of training set predictors.
`class`	N-vector of training set class labels; should be numeric from $\{1, ..., J\}$.
`lambda1`	A non-negative tuning parameter for the mean penalty.
`lambda2`	A non-negative tuning parameter for the Kronecker penalty.
`quiet`	Logical. Should the objective function value be printed at each update? Default is `TRUE`. Note that `quiet=FALSE` will increase computational time.
`Xval`	An $r \times c \times N_{\text{val}}$ array of validation set predictors. Default is `NULL`.
`classval`	N_{val}-vector of validation set class labels; should be numeric from $\{1, ..., J\}$. Default is `NULL`.
`k.iter`	Maximum number of iterations for full blockwise coordinate descent algorithm.
`cov.tol`	Convergence tolerance for graphical lasso sub-algorithms; passed to `glasso`. Default is $1e^{-5}$.
`m.tol`	Convergence tolerance for mean update alternating minimization algorithm. Default is $1e^{-5}$. It is recommended to track the objective function value using `quiet = FALSE` and adjust if necessary.
`full.tol`	Convergence tolerance for full blockwise coordinate descent algorithm; based on decrease in objective function value. Default is $1e^{-6}$. It is recommended to track the objective function value using `quiet = FALSE` and adjust if necessary.

Value

Returns of list of class "MN", which contains the following elements:

`Val`	The misclassification rate on the validation set, if provided.
`Mean`	$\hat{M}$; an $r \times c \times J$ array of estimated class means.
`U`	$\hat{U}$; the $r \times r$ estimated precision matrix for the row variables.
`V`	$\hat{V}$; the $c \times c$ estimated precision matrix for the column variables.
`pi.list`	$\hat{\pi}$; J-vector with marginal class probabilities from training set.

References

- Molstad, A. J., and Rothman, A. J. (2018). A penalized likelihood method for classification with matrix-valued predictors. *Journal of Computational and Graphical Statistics*.

Examples

```
## Generate realizations of matrix-normal random variables
## set sample size, dimensionality, number of classes,
## and marginal class probabilities

N = 75
N.test = 150
N.val = 75

N.total = N + N.test + N.val

r = 16
p = 16
C = 3

pi.list = rep(1/C, C)

## create class means
M.array = array(0, dim=c(r, p, C))
M.array[3:4, 3:4, 1] = 1
M.array[5:6, 5:6, 2] = .5
M.array[3:4, 3:4, 3] = -2
M.array[5:6, 5:6, 3] = -.5


## create covariance matrices U and V
Uinv = matrix(0, nrow=r, ncol=r)
for (i in 1:r) {
for (j in 1:r) {
Uinv[i,j] = .5^abs(i-j)
}
}

eoU = eigen(Uinv)
Uinv.sqrt = tcrossprod(tcrossprod(eoU$vec,
diag(eoU$val^(1/2))),eoU$vec)

Vinv = matrix(.5, nrow=p, ncol=p)
diag(Vinv) = 1
eoV = eigen(Vinv)
Vinv.sqrt = tcrossprod(tcrossprod(eoV$vec,
diag(eoV$val^(1/2))),eoV$vec)

## generate N.total realizations of matrix-variate normal data
set.seed(10)
dat.array = array(0, dim=c(r,p,N.total))
class.total = numeric(length=N.total)
```

```
for(jj in 1:N.total){
class.total[jj] = sample(1:C, 1, prob=pi.list)
dat.array[,,jj] = tcrossprod(crossprod(Uinv.sqrt,
matrix(rnorm(r*p), nrow=r)),
Vinv.sqrt) + M.array[,,class.total[jj]]
}

## store generated data
X = dat.array[,,1:N]
X.val = dat.array[,,(N+1):(N+N.val)]
X.test = dat.array[,,(N+N.val+1):N.total]

class = class.total[1:N]
class.val = class.total[(N+1):(N+N.val)]
class.test = class.total[(N+N.val+1):N.total]

## fit two-dimensional grid of tuning parameters;
## measure classification accuracy on validation set
lambda1 = c(2^seq(-5, 0, by=1))
lambda2 = c(2^seq(-8, -4, by=1))
fit.grid = MatLDA_Grid(X=X, class=class, lambda1=lambda1,
lambda2=lambda2, quiet=TRUE,
Xval=X.val, classval= class.val,
k.iter = 100, cov.tol=1e-5, m.tol=1e-5, full.tol=1e-6)

## identify minimum misclassification proportion;
## select tuning parameters corresponding to
## smallest model at minimum misclassification proportion
CV.mat = fit.grid$Val.mat
G.mat = fit.grid$G.mat*(CV.mat==min(CV.mat))
ind1 = (which(G.mat==max(G.mat), arr.ind=TRUE))[,2]
ind2 = (which(G.mat==max(G.mat), arr.ind=TRUE))[,1]
out = unique(ind2[which(ind2==max(ind2))])
lambda1.cv = lambda1[out]
out2 = unique(max(ind1[ind2==out]))
lambda2.cv = lambda2[out2]

## refit model with sinlge tuning parameter pair
out = MatLDA(X=X, class=class, lambda1=lambda1.cv,
lambda2=lambda2.cv, quiet=FALSE,
Xval=X.test, classval= class.test,
k.iter = 100, cov.tol=1e-5, m.tol=1e-5, full.tol=1e-6)

## print misclassification proportion on test set
out$Val

## print images of estimated mean differences
dev.new(width=10, height=3)
par(mfrow=c(1,3))
image(t(abs(out$M[,,1] - out$M[,,2]))[,r:1],
main=expression(paste("|", hat(mu)[1], "-", hat(mu)[2], "|")),
 col = grey(seq(1, 0, length = 100)))
image(t(abs(out$M[,,1] - out$M[,,3]))[,r:1],
```

```
main=expression(paste("|", hat(mu)[1], "-", hat(mu)[3], "|")),
 col = grey(seq(1, 0, length = 100)))
image(t(abs(out$M[,2] - out$M[,3]))[,r:1],
main=expression(paste("|", hat(mu)[2], "-", hat(mu)[3], "|")),
 col = grey(seq(1, 0, length = 100)))
```

`MatLDA_Grid` *Fits the J-class penalized matrix-normal model for a two-dimensional grid of tuning parameters. Used for tuning parameter selection.*

Description

A function for fitting the penalized matrix normal model for a two-dimensional grid of tuning parameters. Meant to be used with a validation set to select tuning parameters. Can also be used inside a k-fold cross-validation function where the training set is the data outside the kth fold and the validation set is comprised of the kth fold sample data.

Usage

```
MatLDA_Grid(X, class, lambda1, lambda2, quiet = TRUE, Xval = NULL,
  classval = NULL, k.iter = 100, cov.tol = 1e-05, m.tol = 1e-05,
  full.tol = 1e-06)
```

Arguments

`X`	An $r \times c \times N$ array of training set predictors.
`class`	N-vector of training set class labels; should be numeric from $\{1, ..., J\}$.
`lambda1`	A vector of non-negative candidate tuning parameters for the mean penalty.
`lambda2`	A vector of non-negative candidate tuning parameters for the Kronecker penalty.
`quiet`	Logical. Should the objective function value be printed at each update? Default is `TRUE`. Note that `quiet=FALSE` will increase computational time.
`Xval`	An $r \times c \times N_{\text{val}}$ array of validation set predictors. Default is `NULL`.
`classval`	N_{val}-vector of validation set class labels; should be numeric from $\{1, ..., J\}$.Default is `NULL`.
`k.iter`	Maximum number of iterations for full blockwise coordinate descent algorithm. Default is 100.
`cov.tol`	Convergence tolerance for graphical lasso subalgorithms; passed to `glasso`. Default is $1e^{-5}$.
`m.tol`	Convergence tolerance for mean update alternating minimization algorithm. Default is $1e^{-5}$. It is recommended to track the objective function value using `quiet = FALSE` and adjust tolerance if necessary.
`full.tol`	Convergence tolerance for full blockwise coordinate descent algorithm; based on decrease in objective function value. Default is $1e^{-6}$. It is recommended to track the objective function value using `quiet = FALSE` and adjust tolerance if necessary.

Value

`Val.mat`	A matrix of dimension `length`(lambda1) $\times$`length`(lambda2) with validation set misclassification propotions.
`G.mat`	A matrix of dimension `length`(lambda1) $\times$`length`(lambda2) with the number of pairwise mean differences that are zero, i.e., a larger entry corresponds to a more sparse model.
`U.mat`	A matrix of dimension `length`(lambda1) $\times$`length`(lambda2) with the number of zeros in $\hat{U}$.
`V.mat`	A matrix of dimension `length`(lambda1) $\times$`length`(lambda2) with the number of zeros in $\hat{V}$.

References

- Molstad, A. J., and Rothman, A. J. (2018). A penalized likelihood method for classification with matrix-valued predictors. *Journal of Computational and Graphical Statistics*.

Examples

```
## Generate realizations of matrix-normal random variables
## set sample size, dimensionality, number of classes,
## and marginal class probabilities

N = 75
N.test = 150
N.val = 75

N.total = N + N.test + N.val

r = 16
p = 16
C = 3

pi.list = rep(1/C, C)

## create class means
M.array = array(0, dim=c(r, p, C))
M.array[3:4, 3:4, 1] = 1
M.array[5:6, 5:6, 2] = .5
M.array[3:4, 3:4, 3] = -2
M.array[5:6, 5:6, 3] = -.5


## create covariance matrices U and V
Uinv = matrix(0, nrow=r, ncol=r)
for (i in 1:r) {
for (j in 1:r) {
Uinv[i,j] = .5^abs(i-j)
}
}
```

```
eoU = eigen(Uinv)
Uinv.sqrt = tcrossprod(tcrossprod(eoU$vec,
diag(eoU$val^(1/2))),eoU$vec)

Vinv = matrix(.5, nrow=p, ncol=p)
diag(Vinv) = 1
eoV = eigen(Vinv)
Vinv.sqrt = tcrossprod(tcrossprod(eoV$vec,
diag(eoV$val^(1/2))),eoV$vec)

## generate N.total realizations of matrix-variate normal data
set.seed(10)
dat.array = array(0, dim=c(r,p,N.total))
class.total = numeric(length=N.total)
for(jj in 1:N.total){
class.total[jj] = sample(1:C, 1, prob=pi.list)
dat.array[,,jj] = tcrossprod(crossprod(Uinv.sqrt,
matrix(rnorm(r*p), nrow=r)),
Vinv.sqrt) + M.array[,,class.total[jj]]
}

## store generated data
X = dat.array[,,1:N]
X.val = dat.array[,,(N+1):(N+N.val)]
X.test = dat.array[,,(N+N.val+1):N.total]

class = class.total[1:N]
class.val = class.total[(N+1):(N+N.val)]
class.test = class.total[(N+N.val+1):N.total]

## fit two-dimensional grid of tuning parameters;
## measure classification accuracy on validation set
lambda1 = c(2^seq(-5, 0, by=1))
lambda2 = c(2^seq(-8, -4, by=1))
fit.grid = MatLDA_Grid(X=X, class=class, lambda1=lambda1,
lambda2=lambda2, quiet=TRUE,
Xval=X.val, classval= class.val,
k.iter = 100, cov.tol=1e-5, m.tol=1e-5, full.tol=1e-6)

## identify minimum misclassification proportion;
## select tuning parameters corresponding to
## smallest model at minimum misclassification proportion
CV.mat = fit.grid$Val.mat
G.mat = fit.grid$G.mat*(CV.mat==min(CV.mat))
ind1 = (which(G.mat==max(G.mat), arr.ind=TRUE))[,2]
ind2 = (which(G.mat==max(G.mat), arr.ind=TRUE))[,1]
out = unique(ind2[which(ind2==max(ind2))])
lambda1.cv = lambda1[out]
out2 = unique(max(ind1[ind2==out]))
lambda2.cv = lambda2[out2]

## refit model with sinlge tuning parameter pair
```

```
out = MatLDA(X=X, class=class, lambda1=lambda1.cv,
lambda2=lambda2.cv, quiet=FALSE,
Xval=X.test, classval= class.test,
k.iter = 100, cov.tol=1e-5, m.tol=1e-5, full.tol=1e-6)

## print misclassification proportion on test set
out$Val

## print images of estimated mean differences
dev.new(width=10, height=3)
par(mfrow=c(1,3))
image(t(abs(out$M[,,1] - out$M[,,2]))[,r:1],
main=expression(paste("|", hat(mu)[1], "-", hat(mu)[2], "|")),
col = grey(seq(1, 0, length = 100)))
image(t(abs(out$M[,,1] - out$M[,,3]))[,r:1],
main=expression(paste("|", hat(mu)[1], "-", hat(mu)[3], "|")),
col = grey(seq(1, 0, length = 100)))
image(t(abs(out$M[,,2] - out$M[,,3]))[,r:1],
main=expression(paste("|", hat(mu)[2], "-", hat(mu)[3], "|")),
col = grey(seq(1, 0, length = 100)))
```

MN_MLE *Matrix-normal maximum likelihood estimator*

Description

A function for fitting the J-class matrix-normal model using maximum likelihood. Uses the so-called "flip-flop" algorithm after initializing at $U = I_r$.

Usage

```
MN_MLE(X, class, max.iter = 1000, tol = 1e-06, quiet = TRUE)
```

Arguments

X	An $r \times c \times N$ array of training set predictors.
class	N-vector of training set class labels; should be numeric from $\{1, ..., J\}$.
max.iter	Maximum number of iterations for "flip-flop" algorithm.
tol	Convergence tolerance for the "flip flop" algorithm; based on decrease in negative log-likelihood.
quiet	Logical. Should the objective function value be printed at each update? Default is TRUE. Note that quiet=FALSE will increase computational time.

Value

Returns of list of class "MN", which contains the following elements:

`Mean`	$\bar{X}$; An $r \times c \times C$ array of sample class means.
`U`	$\hat{U}^{\mathrm{MLE}}$; the $r \times r$ estimated precision matrix for the row variables.
`V`	$\hat{V}^{\mathrm{MLE}}$; the $c \times c$ estimated precision matrix for the column variables.
`pi.list`	$\hat{\pi}$; J-vector with marginal class probabilities from training set.

References

- Molstad, A. J., and Rothman, A. J. (2018). A penalized likelihood method for classification with matrix-valued predictors. *Journal of Computational and Graphical Statistics*.

Examples

```
## Generate realizations of matrix-normal random variables
## set sample size, dimensionality, number of classes,
## and marginal class probabilities

N = 75
N.test = 150

N.total = N + N.test

r = 16
p = 16
C = 3

pi.list = rep(1/C, C)

## create class means
M.array = array(0, dim=c(r, p, C))
M.array[3:4, 3:4, 1] = 1
M.array[5:6, 5:6, 2] = .5
M.array[3:4, 3:4, 3] = -2
M.array[5:6, 5:6, 3] = -.5


## create covariance matrices U and V
Uinv = matrix(0, nrow=r, ncol=r)
for (i in 1:r) {
for (j in 1:r) {
Uinv[i,j] = .5^abs(i-j)
}
}

eoU = eigen(Uinv)
Uinv.sqrt = tcrossprod(tcrossprod(eoU$vec,
diag(eoU$val^(1/2))),eoU$vec)

Vinv = matrix(.5, nrow=p, ncol=p)
```

```
diag(Vinv) = 1
eoV = eigen(Vinv)
Vinv.sqrt = tcrossprod(tcrossprod(eoV$vec,
diag(eoV$val^(1/2))),eoV$vec)

## generate N.total realizations of matrix-variate normal data
set.seed(1)
X = array(0, dim=c(r,p,N.total))
class = numeric(length=N.total)
for(jj in 1:N.total){
class[jj] = sample(1:C, 1, prob=pi.list)
X[,,jj] = tcrossprod(crossprod(Uinv.sqrt,
matrix(rnorm(r*p), nrow=r)),
Vinv.sqrt) + M.array[,,class[jj]]
}

## fit matrix-normal model using maximum likelihood
out = MN_MLE(X = X, class = class)
```

`PredictMN` *Classify matrix-valued data based on a matrix-normal linear discriminant; an object of class "MN".*

Description

A function for prediction based on an object of class "MN"; models fit by `MatLDA` or `MN_MLE`.

Usage

```
PredictMN(object, newdata, newclass = NULL)
```

Arguments

`object`	An object of class "MN"; output from `MatLDA` or `MN_MLE`.
`newdata`	New data to be classified; an $r \times c \times N_{\text{test}}$ array.
`newclass`	Class labels for new data; should be $in \{1, \ldots, J\}$. Default is `NULL`.

Value

`pred.class`	An N_{test}-vector of predicted class membership based on the inputed object.
`misclass`	If `newclass` is non-`NULL`, returns the misclassification proportion on the test data set.
`prob.mat`	A $N_{\text{test}} \times J$ matrix with the value of discriminant evaluated at each test data point.

References

- Molstad, A. J., and Rothman, A. J. (2018). A penalized likelihood method for classification with matrix-valued predictors. *Journal of Computational and Graphical Statistics*.

Examples

```
## Generate realizations of matrix-normal random variables
## set sample size, dimensionality, number of classes,
## and marginal class probabilities

N = 75
N.test = 150

N.total = N + N.test

r = 16
p = 16
C = 3

pi.list = rep(1/C, C)

## create class means
M.array = array(0, dim=c(r, p, C))
M.array[3:4, 3:4, 1] = 1
M.array[5:6, 5:6, 2] = .5
M.array[3:4, 3:4, 3] = -2
M.array[5:6, 5:6, 3] = -.5


## create covariance matrices U and V
Uinv = matrix(0, nrow=r, ncol=r)
for (i in 1:r) {
for (j in 1:r) {
Uinv[i,j] = .5^abs(i-j)
}
}


eoU = eigen(Uinv)
Uinv.sqrt = tcrossprod(tcrossprod(eoU$vec, diag(eoU$val^(1/2))),eoU$vec)

Vinv = matrix(.5, nrow=p, ncol=p)
diag(Vinv) = 1
eoV = eigen(Vinv)
Vinv.sqrt = tcrossprod(tcrossprod(eoV$vec, diag(eoV$val^(1/2))),eoV$vec)

## generate N.total realizations of matrix-variate normal data
set.seed(1)
dat.array = array(0, dim=c(r,p,N.total))
class.total = numeric(length=N.total)
for(jj in 1:N.total){
class.total[jj] = sample(1:C, 1, prob=pi.list)
dat.array[,,jj] = tcrossprod(crossprod(Uinv.sqrt, matrix(rnorm(r*p), nrow=r)),
Vinv.sqrt) + M.array[,,class.total[jj]]
}

## store generated data
```

```
X = dat.array[,,1:N]
X.test = dat.array[,,(N+1):N.total]

class = class.total[1:N]
class.test = class.total[(N+1):N.total]

## fit matrix-normal model using maximum likelihood
out = MN_MLE(X = X, class = class)

## use output to classify test set
check = PredictMN(out, newdata = X.test, newclass = class.test)

## print misclassification proportion
check$misclass
```

Index