Package 'RMTL'

April 26, 2018

Type Package

Title RMTL: An R library for Multi-Task Learning

Version 1.0

Date 2018-04-20

Author <han.cao@zi-mannheim.de>

Maintainer <han.cao@zi-mannheim.de>

Description This package provides an efficient implementation of regularized multi-task learning comprising 10 algorithms applicable for regression, classification, joint feature selection, task clustering, low-rank learning, sparse learning and network incorporation. All algorithms are implemented basd on the accelerated gradient descent method and feature a complexity of O(1/k^2). Sparse model structure is induced by the solving the proximal operator.

License GPL-2

R topics documented:

RMTL-package *RMTL: An R library for Multi-Task Learning*

Description

This package provides an efficient implementation of regularized multi-task learning comprising 10 algorithms applicable for regression, classification, joint feature selection, task clustering, low-rank learning, sparse learning and network incorporation. All algorithms are implemented basd on the accelerated gradient descent method and feature a complexity of O(1/k^2). Sparse model structure is induced by the solving the proximal operator.

Details

This package provides 10 multi-task learning algorithms (5 classification and 5 regression), which incorporate five regularization strategies for knowledge transfering among tasks. All algorithms share the same framework:

$$\min_{W} \sum_{i}^{t} \frac{1}{n_i} L(W_i|X_i, Y_i) + \Omega(W)$$

where $L(\circ)$ is the loss function (logistic loss for classification or least square loss for regression). X and Y are sets of feature matrixes and responses respectively, W is the coefficient matrix, and t is the number of tasks. Therefore Y_i, X_i, W_i and n_i refer to the data, model parameter vector and the number of subject for task i. Note W_i is the ith column of W.

Knowledge exchange is achieved via the convex term $\Omega(W)$, which jointly modulates models according their specific functionalities. In this package, 5 common regularization methods are implemented to incorperate different priors, i.e. sparse structure ($\Omega(W) = ||W||_1$), joint feature selection ($\Omega(W) = ||W||_{2,1}$), low-rank structure ($\Omega(W) = ||W||_*$), network-based relatedness across tasks ($\Omega(W) = ||WG||_F^2$) and task clustering ($\Omega(W) = \lambda_1\eta(1+\eta)tr(W(\eta I + M)^{-1}W^T)$).

For all algorithms, we implemented an solver based on the accelerated gradient descent method, which takes advantage of information from the previous two iterations to calculate the current gradient and then achieves an improved convergent rate. To solve the non-smooth and convex regularizer, the proximal operator is applied. Moreover, backward line search is used to determine the appropriate step-size in each iteration. Overall, the solver achieves a complexity of $O(\frac{1}{k^2})$ and is optimal among first-order gradient descent methods.

```
opts = list(init = 0, tol = 10^-3, maxIter = 1000)
```

These options are used to control the optimization procedure and can be customized by users. `opts$init` specifies the starting point of the gradient descent algorithm, `opts$tol` controls tolerance of the acceptable precision of solution to terminate the algorithm, and `opts$maxIter` is the maximum number of iterations.

For `opts$init`, two options are provided. `opts$init==0` refers to 0 matrix. And `opts$init==1` refers to the user-specific starting point. If specified, the algorithm will attempt to access `opts$W0` and `opts$C0` as the starting point, which have to be given by users in advance. Otherwise, errors are reported. Particularly, the setting `opts$init==1` is key to warm-start technique for sparse model training.

Only two core functions are necessary for application of each algorithm. The naming of functions follows the structure:

1) Training procedure: MTR_<name_of_prior> or MTC_<name_of_prior>

i.e. MTR_L21 or MTC_L21

2) cross-validation procedure: cv.MTR_<name_of_prior> or cv.MTC_<name_of_prior>

i.e. cv.MTR_L21 or cv.MTC_L21

Here, "MTR" is short for multi-task regression, and "MTC" is short for multi-task classification. Available <name_of_prior> include L21, Lasso, Trace, Graph and CMTL.

Author(s)

<han.cao@zi-mannheim.de> Maintainer: <han.cao@zi-mannheim.de>

References

Beck, A., & Teboulle, M. (2009). A fast iterative shrinkage-thresholding algorithm for linear inverse problems. SIAM journal on imaging sciences, 2(1), 183-202.

Parikh, N., & Boyd, S. (2014). Proximal algorithms. Foundations and Trends® in Optimization, 1(3), 127-239.

Nesterov, Y. (2007). Gradient methods for minimizing composite objective function.

Examples

```
#classification example
#load the data
####
#X, Y: training data
#tX, tY: test data
#W: ground truth
####
load('./RMTL/data/Simulated_Classification_L21.rda')

#specify the options
myopt<-list(init = 0, tol = 10^-3, maxIter = 1000)
lam1=10^seq(1,-6, -0.1)

#perform cross-validation and training using the user-specific options
cv <- cv.MTC_L21(X, Y, opts=myopt, lam1=lam1)
m <- MTC_L21(X, Y, opts=myopt, lam1=cv$lam1.min)

#predict on test data
predict(m, tX)

#------------------------------------------------------------
# regression example
#load the data
####
#X, Y: training data
#tX, tY: test data
#W: ground truth
####
load('./RMTL/data/Simulated_Regression_L21.rda')

#perform cv and train using the default options
cv <- cv.MTC_L21(X, Y)
m <- MTC_L21(X, Y, lam1=cv$lam1.min)
```

```
#predict on test data
predict(m, tX)
```

`cv.MTC_CMTL` *Cross-validation of multi-task classification with clustered structure*

Description

k-fold cross-validation for MTC_CMTL generates a cv plot and estimates two parameters. The default values of the arguments are indicated in the usage section.

Usage

```
cv.MTC_CMTL(X, Y, k = 2, opts = list(init = 0, tol = 10^-3,
maxIter = 1000), stratify = FALSE, nfolds = 5, lam1 = 10^seq(1, -4, -1),
 lam2 = 10^seq(1, -4, -1))
```

Arguments

`X`	a set of feature matrixes
`Y`	a set of binary responses $\in \{-1, 1\}$
`k`	a positive number to modulate the stucture of clusters
`opts`	options of solver
`stratify`	`stratify==TRUE` is used for stratified cross-validation
`nfolds`	number of folds
`lam1`	a positive sequence of λ_1 to balance the clustering and data fitting effect
`lam2`	a positive constant λ_2 to improve the generalization performance

Details

The function first computes `nfolds` solution paths, then calculates the mean error for each pair of λ_1 and λ_2, and finally selects the pair with the minimum error.

Value

`lam1`	λ_1 sequence
`lam2`	λ_2 sequence
`lam1.min`	the selected λ_1 with the lowest cross-validation error
`lam2.min`	the selected λ_2 with the lowest cross-validation error
`cvm`	cross-validation error for each possible choices of λ_1 and λ_2

Author(s)

han.cao@zi-mannheim.de

See Also

`MTC_CMTL` `MTR_CMTL`

Examples

```
#load the data
load('./RMTL/data/Simulated_Classification_CMTL.rda')

#cross-validation
cv <- cv.MTC_CMTL(X, Y, k=2)

#selected parameters
cv$lam1.min
cv$lam2.min

#plot the cross-validation error
plot(cv)
```

---

cv.MTC_Graph *Cross-validation of multi-task classification with network structure*

Description

k-fold cross-validation for MTC_Graph generates a cv plot and estimates one parameter. The default values of the arguments are indicated in the usage section.

Usage

```
cv.MTC_Graph(X, Y, G, opts = list(init = 0, tol = 10^-3,
maxIter = 1000), stratify = FALSE, nfolds = 5, lam1 = 10^seq(1, -4, -1),
lam2=0)
```

Arguments

X	a set of feature matrixes
Y	a set of binary responses $\in \{-1, 1\}$
G	a matrix to encode graph information. For more detail, please refer to MTC_Graph
lam2	a positive constant λ_2 to improve the generalization performance
opts	options of solver
stratify	stratify==TRUE is used for stratified cross-validation
nfolds	number of folds
lam1	a positive sequence of λ_1 to control the network constraint

Details

The function first trains a model using each possible choice of λ_1 in each fold, then calculates the mean error across folds, and finally selects the solution with the minimum error. Please note, λ_2 is pre-defined by users to avoid over-fitting, and not selected by cross-validation.

Value

lam1	λ_1 sequence
lam2	λ_2 value
lam1.min	the selected λ_1 with the lowest cross-validation error
cvm	cross-validation error for each possible choice of λ_1

Author(s)

han.cao@zi-mannheim.de

See Also

MTC_Graph MTR_Graph

Examples

```
#load the data
load('./RMTL/data/Simulated_Classification_Graph.rda')

#specify the parameters
lam1=10^seq(2,-5, -0.05)

#cross-validation
cv <- cv.MTC_Graph(X, Y, G=G, lam1=lam1)

#best parameters
cv$lam1.min

#plot the cv error
plot(cv)
```

---

cv.MTC_L21 *Cross-validation of multi-task classification with joint feature selection*

Description

k-fold cross-validation for MTC_L21 generates a cv plot and estimates one parameter. The default values of the arguments are indicated in the usage section.

Usage

```
cv.MTC_L21(X, Y, lam2 = 0, opts = list(init = 0, tol = 10^-3,
maxIter = 1000), stratify = FALSE, nfolds = 5, lam1 = 10^seq(1, -5, -1))
```

Arguments

X	a set of feature matrixes
Y	a set of binary responses $\in \{-1, 1\}$
lam2	a positive constant λ_2 to improve the generalization performance
opts	options of solver
stratify	stratify==TRUE is used for stratified cross-validation
nfolds	number of folds
lam1	a positive sequence of λ_1 to control the $L_{2,1}$ constraint

Details

The function first computes `nfolds` solution paths, then calculates the mean error for each possible choice of λ_1, and finally selects the solution with the minimum error. Please note, λ_2 is pre-defined by users to avoid over-fitting, and not selected by cross-validation.

Value

`lam1`	λ_1 sequence
`lam2`	λ_2 value
`lam1.min`	the selected λ_1 with the lowest cross-validation error
`cvm`	cross-validation error for each possible choice of λ_1

Author(s)

han.cao@zi-mannheim.de

See Also

`MTC_L21` `MTR_L21`

Examples

```
#load the data
load('./RMTL/data/Simulated_Classification_L21.rda')

#specify the parameters
opts=list(init=0,  tol=10^-6, maxIter=10000)
lam1=10^seq(1,-6, -0.1)

#cross-validation
cv <- cv.MTC_L21(X, Y, lam1=lam1, opts=opts)

#best parameter
cv$lam1.min

#plot cv errors
plot(cv)
```

`cv.MTC_Lasso` *Cross-validation of multi-task classification with L1 regularizer*

Description

k-fold cross-validation for MTC_Lasso generates a cv plot and estimates one parameter. The default values of the arguments are indicated in the usage section.

Usage

```
cv.MTC_Lasso(X, Y, lam2 = 0, opts = list(init = 0, tol = 10^-3,
maxIter = 1000), stratify = FALSE, nfolds = 5, lam1 = 10^seq(1, -5, -1))
```

Arguments

X	a set of feature matrixes
Y	a set of binary responses $\in \{-1, 1\}$
lam2	a positive constant λ_2 to improve the generalization performance
opts	options of solver
stratify	stratify==TRUE is used for stratified cross-validation
nfolds	number of folds
lam1	a positive sequence of λ_1 to control the sparse constraint

Details

The function will compute nfolds solution paths, and calculate the mean error for each candidate λ_1, then selects the solution with the minimum error. Please note, λ_2 is pre-defined by users to avoid over-fitting, and not selected by cross-validation.

Value

lam1	λ_1 sequence
lam2	λ_2 value
lam1.min	the selected λ_1 with the lowest cross-validation error
cvm	cross-validation error for each possible choice of λ_1

Author(s)

han.cao@zi-mannheim.de

See Also

MTC_Lasso MTR_Lasso

Examples

```
#load the data
load('./RMTL/data/Simulated_Classification_Lasso.rda')

#specify the parameters
opts=list(init=0,  tol=10^-6, maxIter=10000)
lam1=10^seq(1,-6, -0.1)

#perform cv
cv <- cv.MTC_Lasso(X, Y, lam1=lam1, opts=opts)

#best parameter
cv$lam1.min

#plot cv errors
plot(cv)
```

`cv.MTC_Trace` *Cross-validation of multi-task classification with low-rank structure*

Description

k-fold cross-validation for MTC_Trace generates a cv plot and estimates one parameter. The default values of the arguments are indicated in the usage section.

Usage

```
cv.MTC_Trace(X, Y, lam2 = 0, opts = list(init = 0, tol = 10^-3,
maxIter = 1000), stratify = FALSE, nfolds = 5, lam1 = 10^seq(2, -4, -1))
```

Arguments

`X`	a set of feature matrixes
`Y`	a set of binary responses $\in \{-1, 1\}$
`lam2`	a positive constant λ_2 to improve the generalization performance
`stratify`	`stratify==TRUE` is used for stratified cross-validation
`opts`	options of solver
`nfolds`	number of folds
`lam1`	a positive sequence of λ_1 to control the low-rank structure

Details

The function first computes `nfolds` solution paths, then calculates the mean error for each possible choice of λ_1, and finally selects the solution with the minimum error. Please note, λ_2 is pre-defined by users to avoid over-fitting, and not selected by cross-validation.

Value

`lam1`	λ_1 sequence
`lam2`	λ_2 value
`lam1.min`	the selected λ_1 with the lowest cross-validation error
`cvm`	cross-validation error for each possible choice of λ_1

Author(s)

han.cao@zi-mannheim.de

See Also

`MTC_Trace` `MTR_Trace`

Examples

```
#load the data
load('./RMTL/data/Simulated_Classification_Trace.rda')

#specify parameters
opts=list(init=0,  tol=10^-6, maxIter=10000)
lam1=10^seq(2,-3, -0.05)

#perform cv
cv <- cv.MTC_Trace(X, Y, lam1=lam1, opts=opts)

#best parameter
cv$lam1.min

#plot results
plot(cv)
```

`cv.MTR_CMTL` *Cross-validation of multi-task regression with clustered structure*

Description

k-fold cross-validation for MTR_CMTL generates a cv plot and estimates two parameters. The default values of the arguments are indicated in the usage section.

Usage

```
cv.MTR_CMTL(X, Y, k = 2, opts = list(init = 0, tol = 10^-3,
maxIter = 1000), nfolds = 5, lam1 = 10^seq(3, -2, -1),
lam2 = 10^seq(3, -2, -1))
```

Arguments

`X`	a set of feature matrixes
`Y`	a set of contiuous responses
`k`	a positive number to modulate the stucture of clusters
`opts`	options of solver
`nfolds`	number of folds
`lam1`	a positive sequence of λ_1 to balance the clustering and data fitting effect
`lam2`	a positive constant λ_2 to improve the generalization performance

Details

The function first computes `nfolds` solution paths, then calculates the mean error for each pair of λ_1 and λ_2, and finally selects the pair with the minimum error.

Value

lam1	λ_1 sequence
lam2	λ_2 sequence
lam1.min	the selected λ_1 with the lowest cross-validation error
lam2.min	the selected λ_2 with the lowest cross-validation error
cvm	cross-validation error for each possible choices of λ_1 and λ_2

Author(s)

han.cao@zi-mannheim.de

See Also

MTR_CMTL MTC_CMTL

Examples

```
#load the data
load('./RMTL/data/Simulated_Regression_CMTL.rda')

#cross-validation
cv <- cv.MTR_CMTL(X,Y, k=2)

#selected parameters
cv$lam1.min
cv$lam2.min

#plot the cross-validation error
plot(cv)
```

---

cv.MTR_Graph *Cross-validation of multi-task regression with network structure*

Description

k-fold cross-validation for MTR_Graph generates a cv plot and estimates one parameter. The default values of the arguments are indicated in the usage section.

Usage

```
cv.MTR_Graph(X, Y, G, opts = list(init = 0, tol = 10^-3,
maxIter = 1000), nfolds = 5, lam1 = 10^seq(3, -2, -1), lam2 = 0)
```

Arguments

X	a set of feature matrixes
Y	a set of contiuous responses
G	a matrix to encode graph information. For more detail, please refer to MTR_Graph
lam2	a positive constant λ_2 to improve the generalization performance
opts	options of solver
nfolds	number of folds
lam1	a positive sequence of λ_1 to control the network constraint

Details

The function first trains a model using each possible choice of λ_1 in each fold, then calculates the mean error across folds, and finally selects the solution with the minimum error. Please note, λ_2 is pre-defined by users to avoid over-fitting, and not selected by cross-validation.

Value

`lam1`	λ_1 sequence
`lam2`	λ_2 value
`lam1.min`	the selected λ_1 with the lowest cross-validation error
`cvm`	cross-validation error for each possible choice of λ_1

Author(s)

han.cao@zi-mannheim.de

See Also

`MTR_Graph` `MTC_Graph`

Examples

```
#load the data
load('./RMTL/data/Simulated_Regression_Graph.rda')

#speficy options
lam1=10^seq(2,-5, -0.05)

#cross-validation
cv <- cv.MTR_Graph(X, Y, G=G, lam1=lam1)

#selected parameter
cv$lam1.min

#plot the cross-validation error
plot(cv)
```

`cv.MTR_L21` *Cross-validation of multi-task regression with joint feature selection*

Description

k-fold cross-validation for MTR_L21 generates a cv plot and estimates one parameter. The default values of the arguments are indicated in the usage section.

Usage

```
cv.MTR_L21(X, Y, lam2 = 0, opts = list(init = 0, tol = 10^-3,
maxIter = 1000), nfolds = 5, lam1 = 10^seq(4, -2, -1))
```

Arguments

`X`	a set of feature matrixes
`Y`	a set of contiuous responses
`lam2`	a positive constant λ_2 to improve the generalization performance
`opts`	options of solver
`nfolds`	number of folds
`lam1`	a positive sequence of λ_1 to control the $L_{2,1}$ constraint

Details

The function first computes `nfolds` solution paths, then calculates the mean error for each possible choice of λ_1, and finally selects the solution with the minimum error. Please note, λ_2 is pre-defined by users to avoid over-fitting, and not selected by cross-validation. .

Value

`lam1`	λ_1 sequence
`lam2`	λ_2 value
`lam1.min`	the selected λ_1 with the lowest cross-validation error
`cvm`	cross-validation error for each possible choice of λ_1

Author(s)

han.cao@zi-mannheim.de

See Also

`MTC_L21` `MTR_L21`

Examples

```
#load the data
load('./RMTL/data/Simulated_Regression_L21.rda')

#speficy options
opts=list(init=0,  tol=10^-6, maxIter=10000)
lam1=10^seq(1,-6, -0.1)

#cross-validation
cv <- cv.MTR_L21(X, Y, lam1=lam1, opts=opts)

#selected parameter
cv$lam1.min

#plot the cross-validation error
plot(cv)
```

`cv.MTR_Lasso` *Cross-validation of multi-task regression with L1 regularizer*

Description

k-fold cross-validation for MTR_Lasso generates a cv plot and estimates one parameter. The default values of the arguments are indicated in the usage section.

Usage

```
cv.MTR_Lasso(X, Y, lam2 = 0, opts = list(init = 0, tol = 10^-3,
maxIter = 1000), nfolds = 5, lam1 = 10^seq(3, -2, -1))
```

Arguments

`X`	a set of feature matrixes
`Y`	a set of contiuous responses
`lam2`	a positive constant λ_2 to improve the generalization performance
`opts`	options of solver
`nfolds`	number of folds
`lam1`	a positive sequence of λ_1 to control the sparse constraint

Details

The function first computes `nfolds` solution paths, then calculates the mean error for each possible choice of λ_1, and finally selects the solution with the minimum error. Please note, λ_2 is pre-defined by users to avoid over-fitting, and not selected by cross-validation.

Value

`lam1`	λ_1 sequence
`lam2`	λ_2 value
`lam1.min`	the selected λ_1 with the lowest cross-validation error
`cvm`	cross-validation error for each possible choice of λ_1

Author(s)

han.cao@zi-mannheim.de

See Also

`MTC_Lasso` `MTR_Lasso`

Examples

```
#load the data
load('./RMTL/data/Simulated_Regression_Lasso.rda')

#speficy options
opts=list(init=0,  tol=10^-6, maxIter=10000)
lam1=10^seq(1,-6, -0.1)

#cross-validation
cv <- cv.MTR_Lasso(X, Y, lam1=lam1, opts=opts)

#selected parameter
cv$lam1.min

#plot the cross-validation error
plot(cv)
```

---

cv.MTR_Trace *Cross-validation of multi-task regression with low-rank structure*

Description

k-fold cross-validation for MTR_Trace generates a cv plot and estimates one parameter. The default values of the arguments are indicated in the usage section.

Usage

```
cv.MTR_Trace(X, Y, lam2 = 0, opts = list(init = 0, tol = 10^-3,
maxIter = 1000), nfolds = 5, lam1 = 10^seq(4, -2, -1))
```

Arguments

X	a set of feature matrixes
Y	a set of contiuous responses
lam2	a positive constant λ_2 to improve the generalization performance
opts	options of solver
nfolds	number of folds
lam1	a positive sequence of λ_1 to control the low-rank structure

Details

The function first computes `nfolds` solution paths, then calculates the mean error for each possible choice of λ_1, and finally selects the solution with the minimum error. Please note, λ_2 is pre-defined by users to avoid over-fitting, and not selected by cross-validation.

Value

lam1	λ_1 sequence
lam2	λ_2 value
lam1.min	the selected λ_1 with the lowest cross-validation error
cvm	cross-validation error for each possible choice of λ_1

Author(s)

han.cao@zi-mannheim.de

See Also

MTC_Trace MTR_Trace

Examples

```
#load the data
load('./RMTL/data/Simulated_Regression_Trace.rda')

#speficy options
opts=list(init=0,  tol=10^-6, maxIter=10000)
lam1=10^seq(2,-3, -0.05)


#cross-validation
cv <- cv.MTR_Trace(X, Y, lam1=lam1, opts=opts)

#selected parameter
cv$lam1.min

#plot the cross-validation error
plot(cv)
```

MTC_CMTL *Multi-task classification with clustered structure*

Description

The formulation combines the objective of logistic loss and convex form of k-means clustering, and therefore is able to detect a clustered structure among tasks. The parameter (λ_1) is used to balance the clustering and data fitting effects, and can be learned via cross-validation.

Usage

```
MTC_CMTL(X, Y, ...)
```

Arguments

X	a set of feature matrixes
Y	a set of binary responses $\in \{-1, 1\}$
k	a positive number to modulate the stucture of clusters with the default of 2
Lam1	a positive constant λ_1 to control the clustering effect with default of 0.5
lam2	a positive constant λ_2 to improve the generalization performance with default of 0.5
opts	options of the solver. The default is `list(init = 0, tol = 10^-3, maxIter = 1000)`

Details

$$\min_{W,C} \sum_{i}^{t} \frac{1}{n_i} sum(log(1 + e^{<-Y_i^T, X_i W_i + C_i>})) + \lambda_1 \eta(1+\eta) tr(W(\eta I + M)^{-1} W^T)$$

$$s.t. \quad tr(M) = k, M \preceq I, M \in S_+^t, \eta = \frac{\lambda_2}{\lambda_1}$$

X and Y are the sets of feature matrixes and binary responses respectively, W is the coefficient matrix, and t is the number of tasks. Accordingly, Y_i, X_i, W_i and n_i refer to the data, model parameter set and the number of subjects for task i. Note W_i is the ith column of W. k is the number of clusters.

Before training, k, λ_1, λ_2 need to be specified by users in advance.

Value

The function will return a trained MTC_CMTL model

`W`	a matrix of features' coefficients
`C`	a constant vector(intercept) of all models
`Obj`	historical record of objective values
`fitted.values`	predictive scores(probability) of the training data.
`residuals`	the residuals of the training data. For each subject i, the residual is $y_i - \hat{y}_i$
`lam1`	λ_1 value
`lam2`	λ_2 value
`k`	the number of clusters
`opts`	options of the solver
`dim`	size of feature matrix of each task
`features`	feature names

Author(s)

han.cao@zi-mannheim.de

References

Zhou, J., Chen, J., & Ye, J. (2011). Clustered multi-task learning via alternating structure optimization. In Advances in neural information processing systems (pp. 702-710).

Jacob, L., Vert, J. P., & Bach, F. R. (2009). Clustered multi-task learning: A convex formulation. In Advances in neural information processing systems (pp. 745-752).

See Also

`MTR_CMTL` `cv.MTC_CMTL`

Examples

```
#load the data
####
#X, Y: training data
#tX, tY: test data
#W: ground truth
#k: a possitive number to modulate the clustered structure
####
load('./RMTL/data/Simulated_Classification_CMTL.rda')

#cross-validation
cv <- cv.MTC_CMTL(X, Y, k=2)

#training
m <- MTC_CMTL(X, Y, lam1=cv$lam1.min, lam2=cv$lam2.min, k=2)

#predict on new dataset
predict(m, tX)

#compare the learnt model with the groud truth
library(fields)
par(mfrow=c(1,2))
image.plot(cor(W), xlab='tasks', ylab='features', main="ground truth")
image.plot(cor(m$W), xlab='tasks', ylab='features', main="CMTL")

#extract more information about the model
print(m)
plotObj(m)
m$fitted.values
m$residuals
m$opts
m$features
m$dim
```

`MTC_Graph` *Multi-task classification with network structure*

Description

This formulation constraints the models' relatedness according to the pre-defined graph `G`. If the penalty is heavy enough, the difference of connected tasks is 0.

Usage

```
MTC_Graph(X, Y, ...)
```

Arguments

`X`	a set of feature matrixes
`Y`	a set of binary responses $\in \{-1, 1\}$
`Lam1`	a positive constant λ_1 to control the network constraint with default of 0.01
`lam2`	a positive constant λ_2 to improve the generalization performance with default of 0

`G`	a matrix to encode the network information. For more information, please refer to the `Details` section
`opts`	options of the solver. The default is `list(init = 0, tol = 10^-3, maxIter = 1000)`

Details

$$\min_{W,C} \sum_{i}^{t} \frac{1}{n_i} sum(log(1 + e^{<-Y_i^T, X_i W_i + C_i>})) + \lambda_2 ||WG||_F^2 + \lambda_1 ||W||_F^2$$

X and Y are the sets of feature matrixes and binary responses respectively, W is the coefficient matrix, and t is the number of tasks. Accordingly, Y_i, X_i, W_i and n_i refer to the data, model parameter set and the number of subjects for task i. Note W_i is the ith column of W, and C_i is the ith term of C. $||\circ||_F$ is the Frobenius norm.

G represents the task-task similarity and needs to be specified by users. Intuitively, $||WG||_F^2$ equals to an accumulation of differences bewteen pairwise tasks, i.e. $||WG||_F^2 = \sum ||W_{\S} - W_{\dagger}||_2^2$, where $\S$ and $\dagger$ are connected tasks. Theoretically, $||WG||_F^2 = tr(WLW^T)$, where L is the graph Laplacian. Therefore, penalizing such term improves the task relatedness.

However, it is nontrivial to design G. Here, we give three common examples to demonstrate the construction of G.

1) Assume your tasks are subject to orders i.e. temporal or spatial order. Such order forces the order-oriented smoothness across tasks such that the adjacent models(tasks) are similar, then your penalty can be designed as:

$$||WG||_F^2 = \sum_{i}^{t-1} ||W_i - W_{i+1}||_2^2$$

where the size of G is $(t+1) \times t$

$$G_{ij} = \begin{cases} 1 & i = j \\ -1 & i = j+1 \\ 0 & otherwise \end{cases}$$

2) The so-called mean-regularized multi-task learning. In this formulation, each model is forced to approximate the mean of all models,

$$||WG||_F^2 = \sum_{i}^{t} ||W_i - \frac{1}{t} \sum_{j}^{t} W_j||_2^2$$

where the size of G is $t \times t$

$$G_{ij} = \begin{cases} \frac{t-1}{t} & i = j \\ -\frac{1}{t} & others \end{cases}$$

3) Assume your tasks are related according to a given graph $g = (N, E)$ where E is the edge set, and N is the node set. Then the penalty is

$$||WG||_F^2 = \sum_{i}^{||E||} ||W_{\alpha^i} - W_{\beta^i}||_2^2$$

where $\alpha \in N$ and $\beta \in N$ are connected tasks in the graph. The size of G is $t \times ||E||$ For each column i of G:

$$G_{ji} = \begin{cases} 1 & j = \alpha^i \\ -1 & j = \beta^i \\ 0 & otherwise \end{cases}$$

For more examples, the users are refered to the rich literatures of graph Laplacian.

Value

The function will return a trained MTC_Graph model

`W`	a matrix of features' coefficients
`C`	a constant vector(intercept) of all models
`Obj`	historical record of objective values
`fitted.values`	predictive scores(probability) of the training data.
`residuals`	the residuals of the training data. For each subject i, the residual is $y_i - \hat{y}_i$
`lam1`	λ_1 value
`lam2`	λ_2 value
`G`	graph information
`opts`	options of the solver
`dim`	size of feature matrix of each task
`features`	feature names

Author(s)

han.cao@zi-mannheim.de

References

Evgeniou, T., & Pontil, M. (2004, August). Regularized multi-task learning. In Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 109-117). ACM.

Widmer, C., Kloft, M., Görnitz, N., & Rätsch, G. (2012). Efficient training of graph-regularized Multitask SVMs. Machine Learning and Knowledge Discovery in Databases, 633-647.

See Also

`MTR_Graph` `cv.MTC_Graph`

Examples

```
#load the data
####
#X, Y: training data
#tX, tY: test data
#W: ground truth
#G: the network information
####
load('./RMTL/data/Simulated_Classification_Graph.rda')

#specify the parameters
lam1=10^seq(2,-5, -0.05)

#cross-validation
cv <- cv.MTC_Graph(X, Y, G=G, lam1=lam1)

#training
r <- MTC_Graph(X, Y, G=G, lam1=cv$lam1.min)
```

```
#predict on new dataset
predict(r, tX)

#compare the learnt model with the groud truth
par(mfrow=c(1,2))
library(fields)
image.plot(cor(W), xlab='tasks', ylab='tasks', main="ground truth")
image.plot(cor(r$W), xlab='tasks', ylab='tasks', main="Graph")

#extract more information about the model
print(r)
plotObj(r)
r$fitted.values
r$residuals
r$opts
r$features
r$dim
```

`MTC_L21` *Multi-task classification with joint feature selection*

Description

This formulation constraints all models to select or reject the same set of features simultaneously. Therefore, the solution only contains features which are consistently important to all tasks.

Usage

```
MTC_L21(X, Y, ...)
```

Arguments

`X`	a set of feature matrixes
`Y`	a set of binary responses $\in \{-1, 1\}$
`Lam1`	a positive constant λ_1 to control the group sparsity. The default is 0.01
`lam2`	a positive constant λ_2 to improve the generalization performance. The default is 0
`opts`	options of the solver. The default is `list(init = 0, tol = 10^-3, maxIter = 1000)`

Details

$$\min_{W,C} \sum_{i}^{t} \frac{1}{n_i} sum(log(1 + e^{<-Y_i^T, X_i W_i + C_i>})) + \lambda_1 ||W||_{2,1} + \lambda_2 ||W||_F^2$$

X and Y are the sets of feature matrixes and binary responses respectively, W is the coefficient matrix, and t is the number of tasks. Accordingly, Y_i, X_i, W_i and n_i refer to the data, model parameter set and the number of subjects for task i. Note W_i is the ith column of W, and C_i is the ith term of C. $||\circ||_F$ is the Frobenius norm.

$||W||_{2,1} = \sum_i ||W[i,]||_2$ aims to create the group sparse structure in the feature space. In the multi-task learning scenario, the same feature of all tasks form a group, such that features in the same group are equally penalized, while results in the group-wise sparsity.

Value

The function will return a trained MTC_L21 model

`W`	a matrix of features' coefficients
`C`	a constant vector(intercept) of all models
`Obj`	historical record of objective values
`fitted.values`	predictive scores(probability) of the training data.
`residuals`	the residuals of the training data. For each subject i, the residual is $y_i - \hat{y}_i$
`lam1`	λ_1 value
`lam2`	λ_2 value
`opts`	options of the solver
`dim`	size of feature matrix of each task
`features`	feature names

Author(s)

han.cao@zi-mannheim.de

References

Beck, A., & Teboulle, M. (2009). A fast iterative shrinkage-thresholding algorithm for linear inverse problems. SIAM journal on imaging sciences, 2(1), 183-202.

Tibshirani, R. (1996). Regression shrinkage and selection lasso. Journal of the Royal Statistical Society. Series B (Methodological), 267-288.

See Also

`MTR_L21` `cv.MTR_L21`

Examples

```
#load the data
####
#X, Y: training data
#tX, tY: test data
#W: ground truth
####
load('./RMTL/data/Simulated_Classification_L21.rda')

#specify the parameters
opts=list(init=0,  tol=10^-6, maxIter=10000)
lam1=10^seq(1,-6, -0.1)

#cross-validation
cv <- cv.MTC_L21(X, Y, lam1=lam1, opts=opts)

#training with warm start
opt <- opts
for (i in 1: length(lam1)){
    r <- MTC_L21(X, Y, lam1=lam1[i], opts=opt)
    opt$init=1;
```

```
    opt$W0=r$W;
    opt$C0=r$C;
    if (lam1[i]==cv$lam1.min) break
}

#predict on new dataset
predict(r, tX)

#show results
par(mfrow=c(1,2))
library(fields)
image.plot(t(W!=0), xlab='tasks', ylab='features', main="ground truth")
image.plot(t(opt$W0!=0), xlab='tasks', ylab='features', main="L21")

#extract more information about the model
print(r)
plotObj(r)
r$fitted.values
r$residuals
r$opts
r$features
r$dim
```

`MTC_Lasso` *Multi-task classification with L1 regularizer*

Description

This formulation extends the lasso to the multi-task scenario such that all models are penalized according to the same L_1 strength, and all unimportant coefficients are shrunken to 0, to achieve the global optimun.

Usage

```
MTC_Lasso(X, Y, ...)
```

Arguments

`X`	a set of feature matrixes
`Y`	a set of binary responses $\in \{-1, 1\}$
`Lam1`	a positive constant λ_1 to control the sparsity. The default is 0.01
`lam2`	a positive constant λ_2 to improve the generalization performance. The default is 0
`opts`	options of the solver. The default is `list(init = 0, tol = 10^-3, maxIter = 1000)`

Details

$$\min_{W,C} \sum_{i}^{t} \frac{1}{n_i} sum(log(1 + e^{<-Y_i^T, X_i W_i + C_i>})) + \lambda_1 ||W||_1 + \lambda_2 ||W||_F^2$$

X and Y are the sets of feature matrixes and binary responses respectively, W is the coefficient matrix, and t is the number of tasks. Accordingly, Y_i, X_i, W_i and n_i refer to the data, model parameter set and the number of subjects for task i. Note W_i is the ith column of W, and C_i is the ith term of C. $||\circ||_F$ is the Frobenius norm. $||\circ||_1$ is the L1 norm.

Value

The function will retrun a trained MTC_Lasso model

`W`	a matrix of features' coefficients
`C`	a constant vector(intercept) of all models
`Obj`	historical record of objective values
`fitted.values`	predictive scores(probability) of the training data.
`residuals`	the residuals of the training data. For each subject i, the residual is $y_i - \hat{y}_i$
`lam1`	λ_1 value
`lam2`	λ_2 value
`opts`	options of the solver
`dim`	size of feature matrix of each task
`features`	feature names

Author(s)

han.cao@zi-mannheim.de

References

Beck, A., & Teboulle, M. (2009). A fast iterative shrinkage-thresholding algorithm for linear inverse problems. SIAM journal on imaging sciences, 2(1), 183-202.

Tibshirani, R. (1996). Regression shrinkage and selection lasso. Journal of the Royal Statistical Society. Series B (Methodological), 267-288.

See Also

`MTC_Lasso` `cv.MTC_Lasso`

Examples

```
#load the data
####
#X, Y: training data
#tX, tY: test data
#W: ground truth
####
load('./RMTL/data/Simulated_Classification_Lasso.rda')

#specify the parameters
opts=list(init=0,  tol=10^-6, maxIter=10000)
lam1=10^seq(1,-6, -0.1)

#perform cv
cv <- cv.MTC_Lasso(X, Y, lam1=lam1, opts=opts)
```

```
#training with warm start
opt <- opts
for (i in 1: length(lam1)){
    r <- MTC_Lasso(X, Y, lam1=lam1[i], opts=opt)
    opt$init=1;
    opt$W0=r$W;
    opt$C0=r$C;
    if (lam1[i]==cv$lam1.min) break
}

#predict on new dataset
predict(r, tX)

# show results
par(mfrow=c(1,2))
library(fields)
image.plot(t(W!=0), xlab='tasks', ylab='features', main="ground truth")
image.plot(t(opt$W0!=0), xlab='tasks', ylab='features', main="Sparse")

#extract more information about the model
print(r)
plotObj(r)
r$fitted.values
r$residuals
r$opts
r$features
r$dim
```

`MTC_Trace` *Multi-task classification with low-rank structure*

Description

This formulation constraints all models to a low-rank subspace. With increasing penalty(λ_1), the correlation between models increases.

Usage

```
MTC_Trace(X, Y, ...)
```

Arguments

`X`	a set of feature matrixes
`Y`	a set of binary responses $\in \{-1, 1\}$
`Lam1`	a positive constant (λ_1) to constraint the rank of W. The default is 0.01
`lam2`	a positive constant (λ_2) to improve the generalization performance. The default is 0
`opts`	options of the solver. The default is `list(init = 0, tol = 10^-3, maxIter = 1000)`

Details

$$\min_{W,C} \sum_{i}^{t} \frac{1}{n_i} sum(log(1 + e^{<-Y_i^T, X_i W_i + C_i>})) + \lambda_1 ||W||_* + \lambda_2 ||W||_F^2$$

X and Y are the sets of feature matrixes and binary responses respectively, W is the coefficient matrix, and t is the number of tasks. Accordingly, Y_i, X_i, W_i and n_i refer to the data, model parameter set and the number of subjects for task i. Note W_i is the ith column of W, and C_i is the ith term of C. $||\circ||_F$ is the Frobenius norm. $||W||_*$ is the trace norm of W.

Value

The function will return a trained MTC_Trace model

`W`	a matrix of features' coefficients
`C`	a constant vector(intercept) of all models
`Obj`	historical record of objective values
`fitted.values`	predictive scores(probability) of the training data.
`residuals`	the residuals of the training data. For each subject i, the residual is $y_i - \hat{y}_i$
`lam1`	λ_1 value
`lam2`	λ_2 value
`opts`	options of the solver
`dim`	size of feature matrix of each task
`features`	feature names

Author(s)

han.cao@zi-mannheim.de

References

Ji, S., & Ye, J. (2009, June). An accelerated gradient method for trace norm minimization. In Proceedings of the 26th annual international conference on machine learning (pp. 457-464). ACM.

Pong, T. K., Tseng, P., Ji, S., & Ye, J. (2010). Trace norm regularization: Reformulations, algorithms, and multi-task learning. SIAM Journal on Optimization, 20(6), 3465-3489.

See Also

`MTR_Trace` `cv.MTC_Trace`

Examples

```
####
#X, Y: training data
#tX, tY: test data
#W: ground truth
####
load('./RMTL/data/Simulated_Classification_Trace.rda')

#specify parameters
```

```
opts=list(init=0,  tol=10^-6, maxIter=10000)
lam1=10^seq(2,-3, -0.05)

#perform cv
cv <- cv.MTC_Trace(X, Y, lam1=lam1, opts=opts)

#training with warm start
opt <- opts
for (i in 1: length(lam1)){
    r <- MTC_Trace(X, Y, lam1=lam1[i], opts=opt)
    opt$init=1;
    opt$W0=r$W;
    opt$C0=r$C;
    if (lam1[i]==cv$lam1.min) break
}

#predict on new dataset
predict(r, tX)

#compare the learnt model with the groud truth
par(mfrow=c(1,2))
library(fields)
image.plot(cor(W), xlab='tasks', ylab='tasks', main="ground truth")
image.plot(cor(opt$W0), xlab='tasks', ylab='tasks', main="Trace")

#extract more information about the model
print(r)
plotObj(r)
r$fitted.values
r$residuals
r$opts
r$features
r$dim
```

`MTR_CMTL` *Multi-task regression with clustered structure*

Description

The formulation combines the objective of logistic loss and convex form of k-means clustering, and therefore is able to detect a clustered structure among tasks. The parameter (λ_1) is used to balance the clustering and data fitting effects, and can be learned via cross-validation.

Usage

```
MTR_CMTL(X, Y, ...)
```

Arguments

`X`	a set of feature matrixes
`Y`	a set of continuous responses
`k`	a positive number to modulate the stucture of clusters with default of 2
`Lam1`	a positive number (λ_1) to control the clustering effect with default of 0.5

`lam2`	a positive number (λ_2) to improve the generalization performance with default of 0.5
`opts`	options of the solver with default of `list(init = 0, tol = 10^-3, maxIter = 1000)`

Details

$$\min_W \sum_i^t \frac{1}{n_i} ||Y_i - X_i W_i||^2 + \lambda_1 \eta(1+\eta) tr(W(\eta I + M)^{-1} W^T)$$

$$s.t. \quad tr(M) = k, M \preceq I, M \in S_+^t, \eta = \frac{\lambda_2}{\lambda_1}$$

X and Y are the sets of feature matrixes and continuous responses respectively, W is the coefficient matrix, and t is the number of tasks. Accordingly, Y_i, X_i, W_i and n_i refer to the data, model parameter set and the number of subjects for task i. Note W_i is the ith column of W. k is the number of clusters.

Before training, k, λ_1, λ_2 need to be specified by users in advance.

Value

The function will return a trained MTR_CMTL model

`W`	a matrix of features' coefficients
`Obj`	historical record of objective values
`fitted.values`	predictive scores of the training data
`residuals`	the residuals of the training data. For each subject i, the residual is $y_i - \hat{y}_i$
`lam1`	λ_1 value
`lam2`	λ_2 value
`k`	the number of clusters
`opts`	options of the solver
`dim`	size of feature matrix of each task
`features`	feature names

Author(s)

han.cao@zi-mannheim.de

References

Zhou, J., Chen, J., & Ye, J. (2011). Clustered multi-task learning via alternating structure optimization. In Advances in neural information processing systems (pp. 702-710).

Jacob, L., Vert, J. P., & Bach, F. R. (2009). Clustered multi-task learning: A convex formulation. In Advances in neural information processing systems (pp. 745-752).

See Also

`MTC_CMTL` `cv.MTR_CMTL`

Examples

```
#load the data
####
#X, Y: training data
#tX, tY: test data
#W: ground truth
#k: a possitive number to modulate the clustered structure
####
load('./RMTL/data/Simulated_Regression_CMTL.rda')

#cross-validation
cv <- cv.MTR_CMTL(X,Y, k=2)


m <- MTR_CMTL(X, Y, lam1=cv$lam1.min, lam2=cv$lam2.min, k=2)

#predict on new dataset
predict(m, tX)

#compare the learnt model with the groud truth
library(fields)
par(mfrow=c(1,2))
image.plot(cor(W), xlab='tasks', ylab='features', main="ground truth")
image.plot(cor(m$W), xlab='tasks', ylab='features', main="CMTL")

#extract more information about the model
print(m)
plotObj(m)
m$fitted.values
m$residuals
m$opts
m$features
m$dim
```

`MTR_Graph` *Multi-task regression with network structure*

Description

This formulation constraints the models' relatedness according to the pre-defined graph `G`. If the penalty is heavy enough, the difference of connected tasks is0.

Usage

```
MTR_Graph(X, Y, G, ...)
```

Arguments

`X`	a set of feature matrixes
`Y`	a set of continuous responses
`Lam1`	a positive number λ_1 to control the network constraint with default of 1
`lam2`	a positive number λ_2 to improve the generalization performance with default of 0

`G` a matrix to encode the network information. For more information, please refer to the `Details` section

`opts` options of the solver. The default is `list(init = 0, tol = 10^-3, maxIter = 1000)`

Details

$$\min_W \sum_i^t \frac{1}{n_i}||Y_i - X_i W_i||^2 + \lambda_2 ||WG||_F^2 + \lambda_1 ||W||_F^2$$

X and Y are the sets of feature matrixes and continuous responses respectively, W is the coefficient matrix, and t is the number of tasks. Accordingly, Y_i, X_i, W_i and n_i refer to the data, model parameter set and the number of subjects for task i. Note W_i is the ith column of W, and C_i is the ith term of C. $||\circ||_F$ is the Frobenius norm.

G represents the task-task similarity and needs to be specified by users. Intuitively, $||WG||_F^2$ equals to an accumulation of differences bewteen pairwise tasks, i.e. $||WG||_F^2 = \sum ||W_§ - W_†||_2^2$, where § and † are connected tasks. Theoretically, $||WG||_F^2 = tr(WLW^T)$, where L is the graph Laplacian. Therefore, penalizing such term improves the task relatedness.

However, it is nontrivial to design G. Here, we give three common examples to demonstrate the construction of G.

1) Assume your tasks are subject to orders i.e. temporal or spatial order. Such order forces the order-oriented smoothness across tasks such that the adjacent models(tasks) are similar, then your penalty can be designed as:

$$||WG||_F^2 = \sum_i^{t-1} ||W_i - W_{i+1}||_2^2$$

where the size of G is $(t+1) \times t$

$$G_{ij} = \begin{cases} 1 & i = j \\ -1 & i = j+1 \\ 0 & otherwise \end{cases}$$

2) The so-called mean-regularized multi-task learning. In this formulation, each model is forced to approximate the mean of all models,

$$||WG||_F^2 = \sum_i^t ||W_i - \frac{1}{t}\sum_j^t W_j||_2^2$$

where the size of G is $t \times t$

$$G_{ij} = \begin{cases} \frac{t-1}{t} & i = j \\ -\frac{1}{t} & others \end{cases}$$

3) Assume your tasks are related according to a given graph $g = (N, E)$ where E is the edge set, and N is the node set. Then the penalty is

$$||WG||_F^2 = \sum_i^{||E||} ||W_{\alpha^i} - W_{\beta^i}||_2^2$$

where $\alpha \in N$ and $\beta \in N$ are connected tasks in the graph. The size of G is $t \times ||E||$ For each column i of G:

$$G_{ji} = \begin{cases} 1 & j = \alpha^i \\ -1 & j = \beta^i \\ 0 & otherwise \end{cases}$$

For more examples, the users are refered to the rich literatures of graph Laplacian.

Value

The function will return a trained MTR_Graph model

`W`	a matrix of features' coefficients
`Obj`	historical record of objective values
`fitted.values`	predictive values of the training data.
`residuals`	the residuals of the training data. For each subject i, the residual is $y_i - \hat{y}_i$
`lam1`	λ_1 value
`lam2`	λ_2 value
`G`	graph information
`opts`	options of the solver
`dim`	size of feature matrix of each task
`features`	feature names

Author(s)

han.cao@zi-mannheim.de

References

Evgeniou, T., & Pontil, M. (2004, August). Regularized multi-task learning. In Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 109-117). ACM.

Widmer, C., Kloft, M., Görnitz, N., & Rätsch, G. (2012). Efficient training of graph-regularized Multitask SVMs. Machine Learning and Knowledge Discovery in Databases, 633-647.

See Also

`MTC_Graph` `cv.MTR_Graph`

Examples

```
#load the data
####
#X, Y: training data
#tX, tY: test data
#W: ground truth
#G: the network information
####
load('./RMTL/data/Simulated_Regression_Graph.rda')

#speficy options
lam1=10^seq(2,-5, -0.05)

#cross-validation
cv <- cv.MTR_Graph(X, Y, G=G, lam1=lam1)

#training
r <- MTR_Graph(X, Y, G=G, lam1=cv$lam1.min)

#predict on new dataset
```

```
predict(r, tX)

#compare the learnt model with the groud truth
par(mfrow=c(1,2))
library(fields)
image.plot(cor(W), xlab='tasks', ylab='tasks', main="ground truth")
image.plot(cor(r$W), xlab='tasks', ylab='tasks', main="Graph")

#extract more information about the model
print(r)
plotObj(r)
r$fitted.values
r$residuals
r$opts
r$features
r$dim
```

`MTR_L21` *Multi-task regression with joint feature selection*

Description

This formulation constraints all models to select or reject the same set of features simultaneously. Therefore, the solution only contains features which are consistently important to all tasks.

Usage

```
MTR_L21(X, Y, ...)
```

Arguments

`X`	a set of feature matrixes
`Y`	a set of continuous responses
`Lam1`	a parameter (λ_1) to control the group sparsity. The default is 1
`lam2`	a parameter (λ_2) to improve the generalization performance. The default is 0
`opts`	options of the solver. The default is `list(init = 0, tol = 10^-3, maxIter = 1000)`

Details

$$\min_W \sum_i^t \frac{1}{n_i}||Y_i - X_i W_i||^2 + \lambda_1 ||W||_{2,1} + \lambda_2 ||W||_F^2$$

X and Y are the sets of feature matrixes and continuous responses respectively, W is the coefficient matrix, and t is the number of tasks. Accordingly, Y_i, X_i, W_i and n_i refer to the data, model parameter set and the number of subjects for task i. Note W_i is the ith column of W, and C_i is the ith term of C. $||\circ||_F$ is the Frobenius norm.

$||W||_{2,1} = \sum_i ||W[i,]||_2$ aims to create the group sparse structure in the feature space. In the multi-task learning scenario, the same feature of all tasks form a group, such that features in the same group are equally penalized, while results in the group-wise sparsity.

Value

The function will return a trained MTR_L21 model

`W`	a matrix of features' coefficients
`Obj`	historical record of objective values
`fitted.values`	predictive values of the training data.
`residuals`	the residuals of the training data. For each subject i, the residual is $y_i - \hat{y}_i$
`lam1`	λ_1 value
`lam2`	λ_2 value
`opts`	options of the solver
`dim`	size of feature matrix of each task
`features`	feature names

Author(s)

han.cao@zi-mannheim.de

References

Liu, J., Ji, S., & Ye, J. (2009, June). Multi-task feature learning via efficient l 2, 1-norm minimization. In Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence (pp. 339-348). AUAIPress.

Liu, J., & Ye, J. (2010). Efficient l1/lq norm regularization. arXiv preprint arXiv:1009.4766.

See Also

`MTC_L21` `cv.MTR_L21`

Examples

```
#load the data
####
#X, Y: training data
#tX, tY: test data
#W: ground truth
####
load('./RMTL/data/Simulated_Regression_L21.rda')

#speficy options
opts=list(init=0,  tol=10^-6, maxIter=10000)
lam1=10^seq(1,-6, -0.1)

#cross-validation
cv <- cv.MTR_L21(X, Y, lam1=lam1, opts=opts)

#training with the warm start
opt <- opts
for (i in 1: length(lam1)){
    r <- MTR_L21(X, Y, lam1=lam1[i], opts=opt)
    opt$init=1;
    opt$W0=r$W;
```

```
    opt$C0=r$C;
    if (lam1[i]==cv$lam1.min) break
}

#predict on new dataset
predict(r, tX)

#show results
par(mfrow=c(1,2))
library(fields)
image.plot(t(W!=0), xlab='tasks', ylab='features', main="ground truth")
image.plot(t(opt$W0!=0), xlab='tasks', ylab='features', main="L21")

#extract more information about the model
print(r)
plotObj(r)
r$fitted.values
r$residuals
r$opts
r$features
r$dim
```

`MTR_Lasso` *Multi-task regression with low-rank structure*

Description

This formulation extends the lasso to the multi-task scenario such that all models are penalized according to the same L_1 strength, and all unimportant coefficients are shrunken to 0, to achieve the global optimun.

Usage

```
MTR_Lasso(X, Y, ...)
```

Arguments

`X`	a set of feature matrixes
`Y`	a set of continuous responses
`Lam1`	a parameter λ_1 to control the sparsity. The default is 1
`lam2`	a parameter λ_2 to improve the generalization performance. The default is 0
`opts`	options of the solver. The default is `list(init = 0, tol = 10^-3, maxIter = 1000)`

Details

$$\min_W \sum_i^t \frac{1}{n_i}||Y_i - X_i W_i||^2 + \lambda_1||W||_1 + \lambda_2||W||_F^2$$

X and Y are the sets of feature matrixes and continuous responses respectively, W is the coefficient matrix, and t is the number of tasks. Accordingly, Y_i, X_i, W_i and n_i refer to the data, model parameter set and the number of subjects for task i. Note W_i is the ith column of W, and C_i is the ith term of C. $||\circ||_F$ is the Frobenius norm. $||\circ||_1$ is the L1 norm.

Value

The function will return a trained MTR_Lasso model

`W`	a matrix of features' coefficients
`Obj`	historical record of objective values
`fitted.values`	predictive value of the training data.
`residuals`	the residuals of the training data. For each subject i, the residual is $y_i - \hat{y}_i$
`lam1`	λ_1 value
`lam2`	λ_2 value
`opts`	options of the solver
`dim`	size of feature matrix of each task
`features`	feature names

References

Beck, A., & Teboulle, M. (2009). A fast iterative shrinkage-thresholding algorithm for linear inverse problems. SIAM journal on imaging sciences, 2(1), 183-202.

Tibshirani, R. (1996). Regression shrinkage and selection lasso. Journal of the Royal Statistical Society. Series B (Methodological), 267-288.

See Also

`MTC_Lasso` `cv.MTR_Lasso`

Examples

```
#load the data
####
#X, Y: training data
#tX, tY: test data
#W: ground truth
####
load('./RMTL/data/Simulated_Regression_Lasso.rda')

#speficy options
opts=list(init=0,  tol=10^-6, maxIter=10000)
lam1=10^seq(1,-6, -0.1)

#cross-validation
cv <- cv.MTR_Lasso(X, Y, lam1=lam1, opts=opts)

#training with the warm start
opt <- opts
for (i in 1: length(lam1)){
    r <- MTR_Lasso(X, Y, lam1=lam1[i], opts=opt)
    opt$init=1;
    opt$W0=r$W;
    opt$C0=r$C;
    if (lam1[i]==cv$lam1.min) break
}
```

```
#predict on new dataset
predict(r, tX)

#show results
par(mfrow=c(1,2))
library(fields)
image.plot(t(W!=0), xlab='tasks', ylab='features', main="ground truth")
image.plot(t(opt$W0!=0), xlab='tasks', ylab='features', main="Sparse")

#extract more information about the model
print(r)
plotObj(r)
r$fitted.values
r$residuals
r$opts
r$features
r$dim
```

`MTR_Trace` *Multi-task regression with trace norm regularizer*

Description

This formulation constraints all models to a low-rank subspace. With increasing penalty(λ_1), the correlation between models increases.

Usage

```
MTR_Trace(X, Y, ...)
```

Arguments

`X`	a set of feature matrixes
`Y`	a set of continuous responses
`Lam1`	a positive number (λ_1) to constraint the rank of W. The default is 1
`lam2`	a positive number (λ_2) to improve the generalization performance. The default is 0
`opts`	options of the solver. The default is `list(init = 0, tol = 10^-3, maxIter = 1000)`

Details

$$\min_W \sum_i^t \frac{1}{n_i}||Y_i - X_i W_i||^2 + \lambda_1||W||_* + \lambda_2||W||_F^2$$

X and Y are the sets of feature matrixes and continuous responses respectively, W is the coefficient matrix, and t is the number of tasks. Accordingly, Y_i, X_i, W_i and n_i refer to the data, model parameter set and the number of subjects for task i. Note W_i is the ith column of W, and C_i is the ith term of C. $||\circ||_F$ is the Frobenius norm. $||W||_*$ is the trace norm of W.

Value

The function will return a trained MTR_Trace model

`W`	a matrix of features' coefficients
`Obj`	historical record of objective values
`fitted.values`	predictive values of the training data.
`residuals`	the residuals of the training data. For each subject i, the residual is $y_i - \hat{y}_i$
`lam1`	λ_1 value
`lam2`	λ_2 value
`opts`	options of the solver
`dim`	size of feature matrix of each task
`features`	feature names

Author(s)

han.cao@zi-mannheim.de

References

Ji, S., & Ye, J. (2009, June). An accelerated gradient method for trace norm minimization. In Proceedings of the 26th annual international conference on machine learning (pp. 457-464). ACM.

Pong, T. K., Tseng, P., Ji, S., & Ye, J. (2010). Trace norm regularization: Reformulations, algorithms, and multi-task learning. SIAM Journal on Optimization, 20(6), 3465-3489.

See Also

`MTC_Trace` `cv.MTR_Trace`

Examples

```
#load the data
####
#X, Y: training data
#tX, tY: test data
#W: ground truth
####
load('./RMTL/data/Simulated_Regression_Trace.rda')

#speficy options
opts=list(init=0,  tol=10^-6, maxIter=10000)
lam1=10^seq(2,-3, -0.05)


#cross-validation
cv <- cv.MTR_Trace(X, Y, lam1=lam1, opts=opts)

#training
opt <- opts
for (i in 1: length(lam1)){
    r <- MTR_Trace(X, Y, lam1=lam1[i], opts=opt)
    opt$init=1;
```

```
    opt$W0=r$W;
    opt$C0=r$C;
    if (lam1[i]==cv$lam1.min) break
}

#predict on new dataset
predict(r, tX)

#compare the learnt model with the groud truth
par(mfrow=c(1,2))
library(fields)
image.plot(cor(W), xlab='tasks', ylab='tasks', main="ground truth")
image.plot(cor(opt$W0), xlab='tasks', ylab='tasks', main="Trace")

#extract more information about the model
print(r)
plotObj(r)
r$fitted.values
r$residuals
r$opts
r$features
r$dim
```

Index