Package 'RMTL'

February 27, 2019

Title Regularized Multi-Task Learning

Type Package

Version 0.9

Description Efficient solvers for 10 regularized multi-task learning algorithms applicable for regression, classification, joint feature selection, task clustering, low-rank learning, sparse learning and network incorporation. Based on the accelerated gradient descent method, the algorithms feature a state-of-art computational complexity O(1/k^2). Sparse model structure is induced by the solving the proximal operator. The detail of the package is described in the paper of Han Cao and Emanuel Schwarz (2018) <doi:10.1093/bioinformatics/bty831>.

Depends R (>= 3.5.0)

URL https://github.com/transbioZI/RMTL

BugReports https://github.com/transbioZI/RMTL/issues

Imports MASS (>= 7.3-50), psych (>= 1.8.4), corpcor (>= 1.6.9), doParallel (>= 1.0.14), foreach (>= 1.4.4)

Date 2019-02-15

License GPL-3

Encoding UTF-8

RoxygenNote 6.1.1

Suggests knitr, rmarkdown

VignetteBuilder knitr

NeedsCompilation no

Author Han Cao [cre, aut, cph], Emanuel Schwarz [aut]

Maintainer Han Cao <hank9cao@gmail.com>

Repository CRAN

Date/Publication 2019-02-27 17:00:33 UTC

R topics documented:

RMTL-package 2
calcError 3
Create_simulated_data 4
cvMTL 4
MTL 6
plot.cvMTL 7
plotObj 8
predict.MTL 8
print.MTL 9

Index **10**

RMTL-package *RMTL: Regularized Multi-Task Learning*

Description

This package provides an efficient implementation of regularized multi-task learning (MTL) comprising 10 algorithms applicable for regression, classification, joint feature selection, task clustering, low-rank learning, sparse learning and network incorporation. All algorithms are implemented based on the accelerated gradient descent method and feature a complexity of O(1/k^2). Parallel computing is allowed to improve the efficiency. 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 transferring among tasks. All algorithms share the same framework:

$$\min_{W,C} \sum_{i}^{t} L(W_i, C_i | X_i, Y_i) + \lambda_1 \Omega(W) + \lambda_2 ||W||_F^2$$

where $L(\circ)$ is the loss function (logistic loss for classification or least square loss for linear regression). $\Omega(\circ)$ is the cross-task regularization for knowledge transfer, and $||W||_F^2$ is used for improving the generalization. $X = \{X_i = n_i \times p | i \in \{1, ..., t\}\}$ and $Y = \{Y_i = n_i \times 1 | i \in \{1, ..., t\}\}$ are predictors matrices and responses of t tasks respectively, while each task i contains n_i subjects and p predictors. $W = p \times t$ is the coefficient matrix, where W_i, the ith column of W, refers to the coefficient vector of task i.

The function $\Omega(W)$ jointly modulates multi-task models($\{W_1, W_2, ..., W_t\}$) according to specific prior structure of W. In this package, 5 common regularization methods are implemented to incorporate 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 ($\Omega(W) = tr(W^T W) - tr(F^T W^T W F)$). To call a specific method

correctly, the corresponding "short name" has to be given. Follow the above sequence of methods, the short names are defined: `L21`, `Lasso`, `Trace`, `Graph` and `CMTL`

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.

For the academic references of the implemented algorithms, the readers are referred to the paper (doi:10.1093/bioinformatics/bty831) or the vignettes in the package.

`calcError` *Calculate the prediction error*

Description

Calculate the averaged prediction error across tasks. For classification problem, the miss-classification rate is returned, and for regression problem, the mean square error(MSE) is returned.

Usage

```
calcError(m, newX = NULL, newY = NULL)
```

Arguments

`m`	A MTL model
`newX`	The feature matrices of new individuals
`newY`	The responses of new individuals

Value

The averaged prediction error

Examples

```
#create example data
data<-Create_simulated_data(Regularization="L21", type="Regression")
#train a model
model<-MTL(data$X, data$Y, type="Regression", Regularization="L21",
    Lam1=0.1, Lam2=0, opts=list(init=0,  tol=10^-6, maxIter=1500))
#calculate the training error
calcError(model, newX=data$X, newY=data$Y)
#calculate the test error
calcError(model, newX=data$tX, newY=data$tY)
```

Create_simulated_data *Create an example dataset for testing the MTL algorithm*

Description

Create an example dataset which contains 1), training datasets (X: feature matrices, Y: response vectors); 2), test datasets (tX: feature matrices, tY: response vectors); 3), the ground truth model (W: coefficient matrix) and 4), extra information for some algorithms (i.e. a matrix for encoding the network information is necessary for calling the MTL method with network structure(`Regularization=Graph`)

Usage

```
Create_simulated_data(t = 5, p = 50, n = 20, type = "Regression",
  Regularization = "L21")
```

Arguments

`t`	Number of tasks
`p`	Number of features
`n`	Number of samples of each task. For simplicity, all tasks contain the same number of samples.
`type`	The type of problem, must be "Regression" or "Classification"
`Regularization`	The type of MTL algorithm (cross-task regularizer). The value must be one of {`L21`, `Lasso`, `Trace`, `Graph`, `CMTL` }

Value

The example dataset.

Examples

```
data<-Create_simulated_data(t=5,p=50, n=20, type="Regression", Regularization="L21")
str(data)
```

cvMTL *K-fold cross-validation*

Description

Perform the k-fold cross-validation to estimate the λ_1.

Usage

```
cvMTL(X, Y, type = "Classification", Regularization = "L21",
  Lam1_seq = 10^seq(1, -4, -1), Lam2 = 0, G = NULL, k = 2,
  opts = list(init = 0, tol = 10^-3, maxIter = 1000), stratify = FALSE,
  nfolds = 5, ncores = 2, parallel = FALSE)
```

Arguments

`X`	A set of feature matrices
`Y`	A set of responses, could be binary (classification problem) or continues (regression problem). The valid value of binary outcome $\in \{1, -1\}$
`type`	The type of problem, must be `Regression` or `Classification`
`Regularization`	The type of MTL algorithm (cross-task regularizer). The value must be one of {`L21`, `Lasso`, `Trace`, `Graph`, `CMTL` }
`Lam1_seq`	A positive sequence of `Lam1` which controls the cross-task regularization
`Lam2`	A positive constant λ_2 to improve the generalization performance
`G`	A matrix to encode the network information. This parameter is only used in the MTL with graph structure (`Regularization=Graph`)
`k`	A positive number to modulate the structure of clusters with the default of 2. This parameter is only used in MTL with clustering structure (`Regularization=CMTL`) Note, the larger number is adapted to more complex clustering structure.
`opts`	Options of the optimization procedure. One can set the initial search point, the tolerance and the maximized number of iterations through the parameter. The default value is `list(init=0, tol=10^-3, maxIter=1000)`
`stratify`	`stratify=TRUE` is used for stratified cross-validation
`nfolds`	The number of folds
`ncores`	The number of cores used for parallel computing with the default value of 2
`parallel`	`parallel=TRUE` is used for parallel computing

Value

The estimated λ_1 and related information

Examples

```
#create the example data
data<-Create_simulated_data(Regularization="L21", type="Classification")
#perform the cross validation
cvfit<-cvMTL(data$X, data$Y, type="Classification", Regularization="L21",
    Lam2=0, opts=list(init=0,  tol=10^-6, maxIter=1500), nfolds=5,
    stratify=TRUE, Lam1_seq=10^seq(1,-4, -1))
#show meta-infomration
str(cvfit)
#plot the CV accuracies across lam1 sequence
plot(cvfit)
```

MTL *Train a multi-task learning model.*

Description

Train a multi-task learning model.

Usage

```
MTL(X, Y, type = "Classification", Regularization = "L21",
  Lam1 = 0.1, Lam1_seq = NULL, Lam2 = 0, opts = list(init = 0, tol
  = 10^-3, maxIter = 1000), G = NULL, k = 2)
```

Arguments

X	A set of feature matrices
Y	A set of responses, could be binary (classification problem) or continues (regression problem). The valid value of binary outcome $\in \{1, -1\}$
type	The type of problem, must be Regression or Classification
Regularization	The type of MTL algorithm (cross-task regularizer). The value must be one of {L21, Lasso, Trace, Graph, CMTL }
Lam1	A positive constant λ_1 to control the cross-task regularization
Lam1_seq	A positive sequence of Lam1. If the parameter is given, the model is trained using warm-start technique. Otherwise, the model is trained based on the Lam1 and the initial search point (opts$init).
Lam2	A non-negative constant λ_2 to improve the generalization performance with the default value of 0 (except for Regularization=CMTL)
opts	Options of the optimization procedure. One can set the initial search point, the tolerance and the maximized number of iterations using this parameter. The default value is list(init=0, tol=10^-3, maxIter=1000)
G	A matrix to encode the network information. This parameter is only used in the MTL with graph structure (Regularization=Graph)
k	A positive number to modulate the structure of clusters with the default of 2. This parameter is only used in MTL with clustering structure (Regularization=CMTL) Note, the larger number is adapted to more complex clustering structure.

Value

The trained model including the coefficient matrix W and intercepts C and related meta information

Examples

```
#create the example data
data<-Create_simulated_data(Regularization="L21", type="Regression")
#train a MTL model
#cold-start
model<-MTL(data$X, data$Y, type="Regression", Regularization="L21",
    Lam1=0.1, Lam2=0, opts=list(init=0,  tol=10^-6, maxIter=1500))
#warm-start
model<-MTL(data$X, data$Y, type="Regression", Regularization="L21",
   Lam1=0.1, Lam1_seq=10^seq(1,-4, -1), Lam2=0, opts=list(init=0,  tol=10^-6, maxIter=1500))
#meta-information
str(model)
#plot the historical objective values
plotObj(model)
```

`plot.cvMTL` *Plot the cross-validation curve*

Description

Plot the cross-validation curve

Usage

```
## S3 method for class 'cvMTL'
plot(x, ...)
```

Arguments

`x`	The returned object of function `cvMTL`
`...`	Other parameters

Examples

```
#create the example data
data<-Create_simulated_data(Regularization="L21", type="Classification")
#perform the cv
cvfit<-cvMTL(data$X, data$Y, type="Classification", Regularization="L21",
    Lam2=0, opts=list(init=0,  tol=10^-6, maxIter=1500), nfolds=5,
    stratify=TRUE, Lam1_seq=10^seq(1,-4, -1))
#plot the curve
plot(cvfit)
```

plotObj *Plot the historical values of objective function*

Description

Plot the values of objective function across iterations in the optimization procedure. This function indicates the "inner status" of the solver during the optimization, and could be used for diagnosis of the solver and training procedure.

Usage

```
plotObj(m)
```

Arguments

m	A trained MTL model

Examples

```
#create the example date
data<-Create_simulated_data(Regularization="L21", type="Regression")
#Train a MTL model
model<-MTL(data$X, data$Y, type="Regression", Regularization="L21",
    Lam1=0.1, Lam2=0, opts=list(init=0,  tol=10^-6, maxIter=1500))
#plot the objective values
plotObj(model)
```

predict.MTL *Predict the outcomes of new individuals*

Description

Predict the outcomes of new individuals. For classification, the probability of the individual being assigned to positive label P(y==1) is estimated, and for regression, the prediction score is estimated

Usage

```
## S3 method for class 'MTL'
predict(object, newX = NULL, ...)
```

Arguments

object	A trained MTL model
newX	The feature matrices of new individuals
...	Other parameters

Value

The predictive outcome

Examples

```
#Create data
data<-Create_simulated_data(Regularization="L21", type="Regression")
#Train
model<-MTL(data$X, data$Y, type="Regression", Regularization="L21",
    Lam1=0.1, Lam2=0, opts=list(init=0,  tol=10^-6, maxIter=1500))
predict(model, newX=data$tX)
```

`print.MTL` *Print the meta information of the model*

Description

Print the meta information of the model

Usage

```
## S3 method for class 'MTL'
print(x, ...)
```

Arguments

`x`	A trained MTL model
`...`	Other parameters

Examples

```
#create data
data<-Create_simulated_data(Regularization="L21", type="Regression")
#train a MTL model
model<-MTL(data$X, data$Y, type="Regression", Regularization="L21",
    Lam1=0.1, Lam2=0, opts=list(init=0,  tol=10^-6, maxIter=1500))
#print the information of the model
print(model)
```

Index

calcError, 3
Create_simulated_data, 4
cvMTL, 4

MTL, 6

plot.cvMTL, 7
plotObj, 8
predict.MTL, 8
print.MTL, 9

RMTL-package, 2