Package 'RMKL'

April 25, 2019

Type Package

Title Multiple Kernel Learning for Classification or Regression Problems

Version 1.0

Date 2019-04-11

Author Christopher Wilson, Kaiqiao Li

Maintainer Christopher Wilson <cwilso6@clemson.edu>

Description Provides R and C++ function that enable the user to conduct multiple kernel learning (MKL) and cross validation for support vector machine (SVM) models. Cross validation can be used to identify kernel shapes and hyperparameter combinations that can be used as candidate kernels for MKL. There are three implementations provided in this package, namely SimpleMKL Alain Rakotomamonjy et. al (2008), Simple and Efficient MKL Xu et. al (2010), and Dual augmented Lagrangian MKL Suzuki and Tomioka (2011) <doi:10.1007/s10994-011-5252-9>. These methods identify the convex combination of candidate kernels to construct an optimal hyperplane.

License GPL-3

Depends R (>= 3.5.0)

Imports Rcpp (>= 1.0.0), caret, kernlab, stats, e1071

LinkingTo Rcpp, RcppArmadillo

LazyData true

LazyLoad yes

Encoding UTF-8

RoxygenNote 6.1.1

NeedsCompilation yes

Repository CRAN

Date/Publication 2019-04-25 15:20:03 UTC

R topics documented:

`benchmark.data` *Benchmark.data.*

Description

Datasets two groups, labels -1 and 1, with varying amounts of overlap.

Usage

```
benchmark.data
```

Format

A list of dataframes with 3 columns and 200 samples, where the x and y are generated from 2 multivariate distributions. The mean of the two groups vary, to allow for different amount overlap for the groups.

`C.convert` *Converts Cost from DALMKL to SEMKL or SimpleMKL*

Description

This function estimates an a comparable cost for SEMKL or SimpleMKL from DALMKL.

Usage

```
C.convert(K.train, DALMKL.model, C.DALMKL)
```

Arguments

K.train	Gramm matrix of training data
DALMKL.model	DAL MKL model
C.DALMKL	Cost used in DAMKL model

Value

C cost SEMKL or SimpleMKL

Examples

```
data(benchmark.data)
data.mkl=benchmark.data[[1]]
kernels=rep('radial',2)
sigma=c(2,1/20)
train.samples=sample(1:nrow(data.mkl),floor(0.7*nrow(data.mkl)),replace=FALSE)
degree=sapply(1:length(kernels), function(a) ifelse(kernels[a]=='p',2,0))
#Kernels.gen splts the data into a training and test set, and generates the desired kernel matrices.
#Here we generate two gaussisan kernel matrices with sigma hyperparameter 2 and 0.05
K=kernels.gen(data=data.mkl[,1:2],train.samples=train.samples,kernels=kernels,sigma=sigma,
degree=degree,scale=rep(0,length(kernels)))
C=0.05 #Cost parameter for DALMKL
K.train=K$K.train
K.test=K$K.test

# parameters set up
ytr=data.mkl[train.samples,3]
#Converts list of kernel matrices in to an array with is appropriate for C++ code
k.train=simplify2array(K.train)
k.test=simplify2array(K.test)
 #Implement DALMKL with the hinge loss function
 spicy_svmb1n=SpicyMKL(K=k.train,y=ytr, loss='hinge',C=C)
 #Convert cost from DALMKL to be more compatible withSimpleMKL
 C.SimpleMKL=C.convert(K.train,spicy_svmb1n,C)
```

---

gramm *Gramm Matrices*

Description

This function creates a single gramm matrix for traning set based upon several types of kernels.

Usage

```
gramm(x, kernel, sigma, degree, scale)
```

Arguments

x	Matrix of predictors
kernel	Type of kernel used to compute a gramm matrix
sigma	Hyperparameters for radial kernels
degree, scale	Hyperparameter for polynomial kernel

Value

Gramm matrix

Examples

```
library(kernlab)
data(benchmark.data)
example.data=benchmark.data[[1]]
#Generate linear kernel matrix
gramm(example.data[,1:2],'linear',0,0,0)
#Generate radial kernel matrices with different values for the hyperparameter.
gramm(example.data[,1:2],'radial',2^seq(-3:0),0,0)
```

---

grammpred *Gramm Matrix for Test dataset*

Description

This function creates gramm matrix for test dataset based upon several types of kernel.

Usage

```
grammpred(xtrain, xtest, kernel, sigma, degree, scale)
```

Arguments

xtrain	Matrix of predictors for the training set
xtest	Matrix of predictors for the test set
kernel	Type of kernel used to compute a gramm matrix
sigma	Hyperparameters for radial kernels
degree, scale	Hyperparameter for polynomial kernel

Value

Gramm matrix for test set

Examples

```
library(kernlab)
data(benchmark.data)
example.data=benchmark.data[[1]]
#Create split between training samples and test samples
training.samples=sample(1:dim(example.data)[1],floor(0.7*dim(example.data)[1]),replace=FALSE)
xtrain=example.data[training.samples,1:2]
xtest=example.data[-training.samples,1:2]
#Generate linear kernel
grammpred(xtrain,xtest,'linear',0,0,0)
#Generate radial kernels with different values for the hyperparameter.
grammpred(xtrain,xtest,'radial',2^seq(-3:0),0,0)
```

`kernels.gen` *Generate both training and test kernel matrices*

Description

This function creates gramm matrix for traning set baed upon several types of kernel and specified hyper paremeters. This function is essentially a wrappper functions that combines gramm and grammpred. Additionally this function divides each kernel matrix by it's trace, which is a common transformation used in MKL.

Usage

```
kernels.gen(data, train.samples, kernels, degree, scale, sigma)
```

Arguments

`data`	List of data matrices
`train.samples`	Vector of indices that will be used as training samples
`kernels`	Character vector of kernel types
`degree`	Degree of polynomial kernel matrix
`scale`	Leading coefficient on the polynomial kernel
`sigma`	Hyperparameter for the radial basis kernel

Value

K.train Gramm matricesfor training data

K.test Gramm matrices for test data

Examples

```
library(kernlab)
data(benchmark.data)
example.data=benchmark.data[[1]]
#Dividing the samples into a train set and test set.
training.samples=sample(1:dim(example.data)[1],floor(0.7*dim(example.data)[1]),replace=FALSE)
#Specifying the type and hyperparameters for each kernel.
kernels=c('linear',rep('radial',3))
degree=rep(0,4)
scale=rep(0,4)
sigma=c(0,2^seq(-3:0))
kernels.gen(example.data[,1:2], training.samples, kernels, degree, scale, sigma)
```

---

prediction.Classification

Prediction from MKL model

Description

This function creates gramm matrix for traning set baed upon several types of kernel and specified hyper paremeters. Matrix corresponds to similarity betwween each sample in the training set.

Usage

```
prediction.Classification(model, ktest, train.outcome)
```

Arguments

model	MKL model
ktest	Gramm matrix of training data and test data
train.outcome	Outcome for the training data

Value

yhat Predicted value for each test point

predicted Sign of yhat, which is the final predicted outcome

Examples

```
library(kernlab)
library(caret)
data(benchmark.data)
example.data=benchmark.data[[1]]
training.samples=sample(1:dim(example.data)[1],floor(0.7*dim(example.data)[1]),replace=FALSE)
C=100
kernels=rep('radial',3)
degree=rep(0,3)
scale=rep(0,3)
```

```
sigma=c(0,2^seq(-3:0))
K=kernels.gen(example.data[,1:2], training.samples, kernels, degree, scale, sigma)
K.train=K$K.train
K.test=K$K.test
SEMKL.model=SEMKL.classification(K.train,example.data[training.samples,3], C)
predicted=prediction.Classification(SEMKL.model, K.test, example.data[training.samples,3])
confusionMatrix(factor(predicted$predict, levels=c(-1,1)),
                factor(example.data[-training.samples,3],levels=c(-1,1)))
```

`predict_Spicy` *Predict SpicyMKL*

Description

This function is used to predict SpicyMKL models

Usage

```
predict_Spicy(alpha, b, k0)
```

Arguments

`alpha`	coefficient
`b`	intercept
`k0`	the kernel cube needs prediction

Value

The predicted score

Examples

```
data(benchmark.data)
data.mkl=benchmark.data[[1]]
kernels=rep('radial',2)
sigma=c(2,1/20)
train.samples=sample(1:nrow(data.mkl),floor(0.7*nrow(data.mkl)),replace=FALSE)
degree=sapply(1:length(kernels), function(a) ifelse(kernels[a]=='p',2,0))
#Kernels.gen splts the data into a training and test set, and generates the desired kernel matrices.
#Here we generate two gaussisan kernel matrices with sigma hyperparameter 2 and 0.05
K=kernels.gen(data=data.mkl[,1:2],train.samples=train.samples,
                kernels=kernels,sigma=sigma,degree=degree,scale=rep(0,length(kernels)))
C=0.05 #Cost parameter for DALMKL
K.train=K$K.train
K.test=K$K.test
 ytr=data.mkl[train.samples,3]
 #Converts list of kernel matrices in to an array with is appropriate for C++ code
 k.train=simplify2array(K.train)
 k.test=simplify2array(K.test)
```

```
#Implement DALMKL with the hinge loss function
spicy_svmb1n=SpicyMKL(K=k.train,y=ytr, loss='hinge',C=C)
prediction_logistic=predict_Spicy(spicy_svmb1n$alpha,spicy_svmb1n$b,k.test)
#Implement DALMKL with the hinge loss function
spicy_logistic=SpicyMKL(K=k.train,y=ytr, loss='logistic',C=C)
prediction_logistic=predict_Spicy(spicy_logistic$alpha,spicy_logistic$b,k.test)
```

`SEMKL.classification` *Simple and Efficient MKL*

Description

This function conducts Simple and Efficnent MKL for precomputed gramm matrices

Usage

```
SEMKL.classification(k, outcome, penalty, tol = 1e-04,
  max.iters = 1000)
```

Arguments

`k`	list of Gramm matrices
`outcome`	vector of binary outcome -1 and 1
`penalty`	penalty of the smoothness of the resulting desicion rules
`tol`	change between to iterations is smaller than this, algorithms is considered to have converged
`max.iters`	maximum number of allowed iteratons

Value

gamma weight vector for the importnace of each kernel

alpha coeffiencents of the dual of MKL

time total amount of time to train model

iters Numvber of iterations to reach convergence criteria

gamma_all Kernel weights for each interation of SEMKL

Examples

```
data(benchmark.data)
example.data=benchmark.data[[1]]
#Load data
training.samples=sample(1:dim(example.data)[1],floor(0.7*dim(example.data)[1]),replace=FALSE)
# Split samples into training and test sets
C=1
kernels=c('radial','polynomial')
```

```
degree=c(0,2)
scale=c(0,2)
sigma=c(2,0)
K=kernels.gen(example.data[,1:2], training.samples, kernels, degree, scale, sigma)
K.train=K$K.train
SEMKL.classification(K.train,example.data[training.samples,3], C)
```

SimpleMKL.classification *Simple MKL*

Description

This function conducts Simple MKL for precomputed gramm matrices

Usage

```
SimpleMKL.classification(k, outcome, penalty, tol = 10^(-4),
  max.iters = 1000)
```

Arguments

k	list of Gramm matrices
outcome	vector of binary outcome -1 and 1
penalty	ppenalty of the smoothness of the resulting desicion rules
tol	change between to iterations is smaller than this, algorithms is considered to have converged
max.iters	maximum number of allowed iteratons

Value

gamma weight vector for the importnace of each kernel

alpha coeffiencents of the dual of MKL

time total amount of time to train model

max.iters Numvber of iterations to reach convergence criteria

Examples

```
library(kernlab)
library(caret)
library(RMKL)
#Load data
data(benchmark.data)
example.data=benchmark.data[[1]]
# Split samples into training and test sets
training.samples=sample(1:dim(example.data)[1],floor(0.7*dim(example.data)[1]),replace=FALSE)
```

```
# Set up cost parameters and kernels
C=100
kernels=rep('radial',3)
degree=rep(0,3)
scale=rep(0,3)
sigma=c(0,2^seq(-3:0))
K=kernels.gen(example.data[,1:2], training.samples, kernels, degree, scale, sigma)
K.train=K$K.train
SimpleMKL.classification(K.train,example.data[training.samples,3], C)
```

SpicyMKL *DALMKL*

Description

This function conducts DALMKL for precomputed gramm matrices

Usage

```
SpicyMKL(K, y, loss = "hinge", C = 0.5, tolOuter = 0.01,
  tolInner = 1e-06, OuterMaxiter = 500, InnerMaxiter = 500,
  calpha = 10)
```

Arguments

K	The multiple kernel cube (3-d array)
y	The outome variable, must be -1/1
loss	The loss function to be used, must be either 'hinge' or 'logistic', default to be 'hinge'
C	tuning parameter for block one norm, default to be .5
tolOuter	change between to iterations is smaller than this, algorithms is considered to have converged for outer loop, default to be .01
tolInner	change between to iterations is smaller than this, algorithms is considered to have converged for inner loop, default to be .000001
OuterMaxiter	maximum number of allowed iteratons for outer loop, default to be 500
InnerMaxiter	maximum number of allowed iteratons for inner loop, default to be 500
calpha	Lagrangian parameter, default to be 10

Value

b Estimated Intercept

alpha coeffiencents of the dual of MKL

weight Estimated between kernel weight

rho Estimated within kernel weight

Examples

```
data(benchmark.data)
data.mkl=benchmark.data[[1]]
kernels=rep('radial',2)
sigma=c(2,1/20)
train.samples=sample(1:nrow(data.mkl),floor(0.7*nrow(data.mkl)),replace=FALSE)
degree=sapply(1:length(kernels), function(a) ifelse(kernels[a]=='p',2,0))
#Kernels.gen splts the data into a training and test set, and generates the desired kernel matrices.
#Here we generate two gaussisan kernel matrices with sigma hyperparameter 2 and 0.05
K=kernels.gen(data=data.mkl[,1:2],train.samples=train.samples,kernels=kernels,sigma=sigma,
degree=degree,scale=rep(0,length(kernels)))
C=0.05 #Cost parameter for DALMKL
K.train=K$K.train
K.test=K$K.test
# parameters set up
 ytr=data.mkl[train.samples,3]
#Converts list of kernel matrices in to an array with is appropriate for C++ code
 k.train=simplify2array(K.train)
 k.test=simplify2array(K.test)
#Implement DALMKL with the hinge loss function
 spicy_svmb1n=SpicyMKL(K=k.train,y=ytr, loss='hinge',C=C)
 #Implement DALMKL with the hinge loss function
 spicy_logistic=SpicyMKL(K=k.train,y=ytr, loss='logistic',C=C)#'
```

`SVM.nfoldcv` *Cross validation for SVM*

Description

This function performs cross validation to find best combinatiuon of hyper perameters and cost and uses this model to provide prediction performace results.

Usage

```
SVM.nfoldcv(data, outcome, train.samples, C, kernels, degree, scale, sigma,
  nfold = 10)
```

Arguments

`data`	List of data matrices for each pathways for each pathway
`outcome`	Binary outcome variable for MKL
`train.samples`	Vector of indices that will be used as training samples
`C`	Vector of canduidate cost parameters
`kernels`	vector of kernel types
`degree`	Degree of polynomial kernel matrix
`scale`	Leading coefficient on the polynomial kernel
`sigma`	Hyperparameter for the radial basis kernel
`nfold`	Number of folds used in cross validation

Value

cm Confustion matrix along with a variety of accuracy statistics

best.model Model that had the highest accuracy with nfold cross validation

Examples

```
data(benchmark.data)
example.data=benchmark.data[[1]]
training.samples=sample(1:nrow(example.data), floor(0.7*nrow(example.data)),replace=FALSE)
C=2^c(-1,1)
#Find the best cost parameter within the provided range if a linear kernel is used
SVM.nfoldcv(example.data[,1:2], as.factor(example.data[,3]),training.samples,C,
            'linear',0,0,0,nfold=3)
```

Index