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
\textbf{R topics documented:}

- \texttt{benchmark.data} .................................................. 2
- \texttt{C.convert} .......................................................... 2
- \texttt{gramm} .............................................................. 3
- \texttt{grammpred} ......................................................... 4
- \texttt{kernels.gen} ......................................................... 5
- \texttt{prediction.Classification} ......................................... 6
- \texttt{predict_Spicy} ..................................................... 7
- \texttt{SEMKL.classification} ............................................. 8
- \texttt{SimpleMKL.classification} ........................................ 9
- \texttt{SpicyMKL} .......................................................... 10
- \texttt{SVM.nfoldcv} ...................................................... 11

\textbf{Index} \hfil 13

\begin{tabular}{ll}
\texttt{benchmark.data} & \textit{Benchmark.data.} \\
\end{tabular}

\textbf{Description}

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

\textbf{Usage}

\texttt{benchmark.data}

\textbf{Format}

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

\begin{tabular}{ll}
\texttt{C.convert} & \textit{Converting Cost from DALMKL to SEMKL or SimpleMKL} \\
\end{tabular}

\textbf{Description}

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

\textbf{Usage}

\texttt{C.convert(K.train, DALMKL.model, C.DALMKL)}
**gramm**

**Description**

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

**Usage**

```r
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**

```r
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)
```

---

**Description**

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

**Usage**

```r
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
kernels.gen

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 training set based upon several types of kernel and specified hyper parameters. This function is essentially a wrapper functions that combines gramm and grammpred. Additionally this function divides each kernel matrix by its 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 matrices for 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=rep(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 gram matrix for training set based upon several types of kernel and specified hyper parameters. Matrix corresponds to similarity between each sample in the training set.

Usage

prediction.classification(model, ktest, train.outcome)

Arguments

model MKL model
ktest Gram 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)
predict_Spicy

```r
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)))
```

**Description**

This function is used to predict SpicyMKL models.

**Usage**

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

**Arguments**

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

**Value**

The predicted score

**Examples**

```r
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 splits the data into a training and test set, and generates the desired kernel matrices.
# Here we generate two gaussian 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 into 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_svm1n=SpicyMKL(K=k.train,y=ytr, loss='hinge', C=C)
prediction_logistic=predict_Spicy(spicy_svm1n$alpha, spicy_svm1n$b, k.test)

---

### SEMKL.classification

**Simple and Efficient MKL**

**Description**

This function conducts Simple and Efficient MKL for precomputed gramm matrices

**Usage**

```r
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 decision rules
- `tol` change between two iterations is smaller than this, algorithms is considered to have converged
- `max.iters` maximum number of allowed iterations

**Value**

- `gamma` weight vector for the importance of each kernel
- `alpha` coefficients of the dual of MKL
- `time` total amount of time to train model
- `iters` number of iterations to reach convergence criteria
- `gamma_all` Kernel weights for each iteration of SEMKL

**Examples**

```r
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')
```
SimpleMKL.classification

This function conducts Simple MKL for precomputed gram matrices.

**Usage**

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

**Arguments**

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

**Value**

- `gamma`: weight vector for the importance of each kernel
- `alpha`: coefficients of the dual of MKL
- `time`: total amount of time to train model
- `max.iters`: number of iterations to reach convergence criteria

**Examples**

```r
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 gram 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 outcome 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 iterations for outer loop, default to be 500
- **InnerMaxiter**  
  Maximum number of allowed iterations for inner loop, default to be 500
- **calpha**  
  Lagrangian parameter, default to be 10

## Value

- **b** Estimated Intercept
- **alpha coefficients 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 splits the data into a training and test set, and generates the desired kernel matrices.
#Here we generate two gaussian 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)
#Implement DALMKL with the hinge loss function
#spicy_svmh1=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 combination of hyper parameters and cost and uses this model to provide prediction performance 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 candidate 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 Confusion 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

*Topic datasets
  benchmark.data, 2

benchmark.data, 2
C.convert, 2
gramm, 3
grammpred, 4
kernels.gen, 5
predict_Spicy, 7
prediction.Classification, 6
SEMKL.classification, 8
SimpleMKL.classification, 9
SpicyMKL, 10
SVM.nfoldcv, 11