Package 'LiblineaR'

March 2, 2021

Encoding UTF-8

Title Linear Predictive Models Based on the LIBLINEAR C/C++ Library

Version 2.10-12

Date 2021-03-01

Description A wrapper around the LIBLINEAR C/C++ library for machine learning (available at <https://www.csie.ntu.edu.tw/~cjlin/liblinear/>). LIBLINEAR is a simple library for solving large-scale regularized linear classification and regression. It currently supports L2-regularized classification (such as logistic regression, L2-loss linear SVM and L1-loss linear SVM) as well as L1-regularized classification (such as L2-loss linear SVM and logistic regression) and L2-regularized support vector regression (with L1- or L2-loss). The main features of LiblineaR include multi-class classification (one-vs-the rest, and Crammer & Singer method), cross validation for model selection, probability estimates (logistic regression only) or weights for unbalanced data. The estimation of the models is particularly fast as compared to other libraries.

License GPL-2

LazyLoad yes

Suggests SparseM, Matrix

Imports methods

URL <https://www.dnalytics.com/software/liblinear/>

RoxygenNote 7.1.1

NeedsCompilation yes

Author Thibault Helleputte [cre, aut, cph],
Jérôme Paul [aut],
Pierre Gramme [aut]

Maintainer Thibault Helleputte <thibault.helleputte@dnalytics.com>

Repository CRAN

Date/Publication 2021-03-02 06:20:13 UTC

R topics documented:

`heuristicC` *Fast Heuristics For The Estimation Of the C Constant Of A Support Vector Machine.*

Description

`heuristicC` implements a heuristics proposed by Thorsten Joachims in order to make fast estimates of a convenient value for the C constant used by support vector machines. This implementation only works for linear support vector machines.

Usage

```
heuristicC(data)
```

Arguments

`data`	a nxp data matrix. Each row stands for an example (sample, point) and each column stands for a dimension (feature, variable)

Value

A value for the C constant is returned, computed as follows:
$\frac{1}{\frac{1}{n}\sum_{i=1}^{n}\sqrt{G[i,i]}}$ where $G = \texttt{data}\% * \%t(\texttt{data})$

Note

Classification models usually perform better if each dimension of the data is first centered and scaled. If data are scaled, it is better to compute the heuristics on the scaled data as well.

Author(s)

Thibault Helleputte <thibault.helleputte@dnalytics.com>

References

- T. Joachims
SVM light (2002)
http://svmlight.joachims.org

See Also

LiblineaR

Examples

```
data(iris)

x=iris[,1:4]
y=factor(iris[,5])
train=sample(1:dim(iris)[1],100)

xTrain=x[train,]
xTest=x[-train,]
yTrain=y[train]
yTest=y[-train]

# Center and scale data
s=scale(xTrain,center=TRUE,scale=TRUE)

# Sparse Logistic Regression
t=6

co=heuristicC(s)
m=LiblineaR(data=s,labels=yTrain,type=t,cost=co,bias=TRUE,verbose=FALSE)

# Scale the test data
s2=scale(xTest,attr(s,"scaled:center"),attr(s,"scaled:scale"))

# Make prediction
p=predict(m,s2)

# Display confusion matrix
res=table(p$predictions,yTest)
print(res)

# Compute Balanced Classification Rate
BCR=mean(c(res[1,1]/sum(res[,1]),res[2,2]/sum(res[,2]),res[3,3]/sum(res[,3])))
print(BCR)
```

`LiblineaR` *Linear predictive models estimation based on the LIBLINEAR C/C++ Library.*

Description

`LiblineaR` allows the estimation of predictive linear models for classification and regression, such as L1- or L2-regularized logistic regression, L1- or L2-regularized L2-loss support vector classification, L2-regularized L1-loss support vector classification and multi-class support vector classification. It also supports L2-regularized support vector regression (with L1- or L2-loss). The estimation of the models is particularly fast as compared to other libraries. The implementation is based on the LIBLINEAR C/C++ library for machine learning.

Usage

```
LiblineaR(
  data,
  target,
  type = 0,
  cost = 1,
  epsilon = 0.01,
  svr_eps = NULL,
  bias = 1,
  wi = NULL,
  cross = 0,
  verbose = FALSE,
  findC = FALSE,
  useInitC = TRUE,
  ...
)
```

Arguments

data	a nxp data matrix. Each row stands for an example (sample, point) and each column stands for a dimension (feature, variable). Sparse matrices of class matrix.csr from package SparseM or sparse matrices of class dgCMatrix or dgRMatrix from package Matrix are also accepted. Note that C code at the core of LiblineaR package corresponds to a row-based sparse format. Hence, dgCMatrix inputs are first transformed into dgRMatrix format, which requires small extra computation time.
target	a response vector for prediction tasks with one value for each of the n rows of `data`. For classification, the values correspond to class labels and can be a 1xn matrix, a simple vector or a factor. For regression, the values correspond to the values to predict, and can be a 1xn matrix or a simple vector.
type	`LiblineaR` can produce 10 types of (generalized) linear models, by combining several types of loss functions and regularization schemes. The regularization can be L1 or L2, and the losses can be the regular L2-loss for SVM (hinge loss), L1-loss for SVM, or the logistic loss for logistic regression. The default value for `type` is 0. See details below. Valid options are:
for multi-class classification	• 0 – L2-regularized logistic regression (primal) • 1 – L2-regularized L2-loss support vector classification (dual) • 2 – L2-regularized L2-loss support vector classification (primal) • 3 – L2-regularized L1-loss support vector classification (dual) • 4 – support vector classification by Crammer and Singer • 5 – L1-regularized L2-loss support vector classification • 6 – L1-regularized logistic regression • 7 – L2-regularized logistic regression (dual)
for regression	• 11 – L2-regularized L2-loss support vector regression (primal)

- 12 – L2-regularized L2-loss support vector regression (dual)
- 13 – L2-regularized L1-loss support vector regression (dual)

`cost` cost of constraints violation (default: 1). Rules the trade-off between regularization and correct classification on `data`. It can be seen as the inverse of a regularization constant. See information on the 'C' constant in details below. A usually good baseline heuristics to tune this constant is provided by the `heuristicC` function of this package.

`epsilon` set tolerance of termination criterion for optimization. If `NULL`, the LIBLINEAR defaults are used, which are:

if `type` **is 0, 2, 5 or 6** `epsilon`=0.01

if `type` **is 1, 3, 4, 7, 12 or 13** `epsilon`=0.1

The meaning of `epsilon` is as follows:

if `type` **is 0 or 2:** $|f'(w)|_2 \leq$ `epsilon` $\times \min(pos, neg)/l \times |f'(w_0)|_2$, where f is the primal function and pos/neg are # of positive/negative data (default 0.01)

if `type` **is 11:** $|f'(w)|_2 \leq$ `epsilon` $\times |f'(w_0)|_2$, where f is the primal function (default 0.001)

if `type` **is 1, 3, 4 or 7:** Dual maximal violation $\leq$ `epsilon` (default 0.1)

if `type` **is 5 or 6:** $|f'(w)|_\infty \leq$ `epsilon` $\times \min(pos, neg)/l \, |f'(w_0)|_\infty$, where f is the primal function (default 0.01)

if `type` **is 12 or 13:** $|f'(\alpha)|_1 \leq$ `epsilon` $\times |f'(\alpha_0)|_1$, where f is the dual function (default 0.1)

`svr_eps` set tolerance margin (epsilon) in regression loss function of SVR. Not used for classification methods.

`bias` if bias > 0, instance `data` becomes [`data`; `bias`]; if <= 0, no bias term added (default 1).

`wi` a named vector of weights for the different classes, used for asymmetric class sizes. Not all factor levels have to be supplied (default weight: 1). All components have to be named according to the corresponding class label. Not used in regression mode.

`cross` if an integer value k>0 is specified, a k-fold cross validation on `data` is performed to assess the quality of the model via a measure of the accuracy. Note that this metric might not be appropriate if classes are largely unbalanced. Default is 0.

`verbose` if `TRUE`, information are printed. Default is `FALSE`.

`findC` if `findC` is `TRUE` runs a cross-validation of `cross` folds to find the best cost (C) value (works only for type 0 and 2). Cross validation is conducted many times under parameters C = start_C, 2*start_C, 4*start_C, 8*start_C, ..., and finds the best one with the highest cross validation accuracy. The procedure stops when the models of all folds become stable or C reaches the maximal value of 1024.

`useInitC` if `useInitC` is `TRUE` (default) `cost` is used as the smallest start_C value of the search range (`findC` has to be `TRUE`). If `useInitC` is `FALSE`, then the procedure calculates a small enough start_C.

`...` for backwards compatibility, parameter `labels` may be provided instead of `target`. A warning will then be issued, or an error if both are present. Other extra parameters are ignored.

Details

For details for the implementation of LIBLINEAR, see the README file of the original c/c++ LIBLINEAR library at https://www.csie.ntu.edu.tw/~cjlin/liblinear/.

Value

If `cross`>0, the average accuracy (classification) or mean square error (regression) computed over `cross` runs of cross-validation is returned.

Otherwise, an object of class `"LiblineaR"` containing the fitted model is returned, including:

`TypeDetail`	A string decsribing the type of model fitted, as determined by `type`.
`Type`	An integer corresponding to `type`.
`W`	A matrix with the model weights. If `bias` >0, `W` contains p+1 columns, the last being the bias term. The columns are named according to the names of `data`, if provided, or `"Wx"` where `"x"` ranges from 1 to the number of dimensions. The bias term is named `"Bias"`.If the number of classes is 2, or if in regression mode rather than classification, the matrix only has one row. If the number of classes is k>2 (classification), it has k rows. Each row i corresponds then to a linear model discriminating between class i and all the other classes. If there are more than 2 classes, rows are named according to the class i which is opposed to the other classes.
`Bias`	The value of `bias`
`ClassNames`	A vector containing the class names. This entry is not returned in case of regression models.

Note

Classification models usually perform better if each dimension of the data is first centered and scaled.

Author(s)

Thibault Helleputte <thibault.helleputte@dnalytics.com> and
Jerome Paul <jerome.paul@dnalytics.com> and Pierre Gramme.
Based on C/C++-code by Chih-Chung Chang and Chih-Jen Lin

References

- For more information on LIBLINEAR itself, refer to:
R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin.
LIBLINEAR: A Library for Large Linear Classification,
Journal of Machine Learning Research 9(2008), 1871-1874.
https://www.csie.ntu.edu.tw/~cjlin/liblinear/

See Also

`predict.LiblineaR`, `heuristicC`

Examples

```
data(iris)
attach(iris)

x=iris[,1:4]
y=factor(iris[,5])
train=sample(1:dim(iris)[1],100)

xTrain=x[train,]
xTest=x[-train,]
yTrain=y[train]
yTest=y[-train]

# Center and scale data
s=scale(xTrain,center=TRUE,scale=TRUE)

# Find the best model with the best cost parameter via 10-fold cross-validations
tryTypes=c(0:7)
tryCosts=c(1000,1,0.001)
bestCost=NA
bestAcc=0
bestType=NA

for(ty in tryTypes){
for(co in tryCosts){
acc=LiblineaR(data=s,target=yTrain,type=ty,cost=co,bias=1,cross=5,verbose=FALSE)
cat("Results for C=",co," : ",acc," accuracy.\n",sep="")
if(acc>bestAcc){
bestCost=co
bestAcc=acc
bestType=ty
}
}
}

cat("Best model type is:",bestType,"\n")
cat("Best cost is:",bestCost,"\n")
cat("Best accuracy is:",bestAcc,"\n")

# Re-train best model with best cost value.
m=LiblineaR(data=s,target=yTrain,type=bestType,cost=bestCost,bias=1,verbose=FALSE)

# Scale the test data
s2=scale(xTest,attr(s,"scaled:center"),attr(s,"scaled:scale"))

# Make prediction
pr=FALSE
if(bestType==0 || bestType==7) pr=TRUE

p=predict(m,s2,proba=pr,decisionValues=TRUE)

# Display confusion matrix
```

```
res=table(p$predictions,yTest)
print(res)

# Compute Balanced Classification Rate
BCR=mean(c(res[1,1]/sum(res[,1]),res[2,2]/sum(res[,2]),res[3,3]/sum(res[,3])))
print(BCR)

#' ##########################################

# Example of the use of a sparse matrix of class matrix.csr :

if(require(SparseM)){

 # Sparsifying the iris dataset:
 iS=apply(iris[,1:4],2,function(a){a[a<quantile(a,probs=c(0.25))]=0;return(a)})
 irisSparse<-as.matrix.csr(iS)

 # Applying a similar methodology as above:
 xTrain=irisSparse[train,]
 xTest=irisSparse[-train,]

 # Re-train best model with best cost value.
 m=LiblineaR(data=xTrain,target=yTrain,type=bestType,cost=bestCost,bias=1,verbose=FALSE)

 # Make prediction
 p=predict(m,xTest,proba=pr,decisionValues=TRUE)

 # Display confusion matrix
 res=table(p$predictions,yTest)
 print(res)
}

#' ##########################################

# Example of the use of a sparse matrix of class dgCMatrix :

if(require(Matrix)){

 # Sparsifying the iris dataset:
 iS=apply(iris[,1:4],2,function(a){a[a<quantile(a,probs=c(0.25))]=0;return(a)})
 irisSparse<-as(iS,"sparseMatrix")

 # Applying a similar methodology as above:
 xTrain=irisSparse[train,]
 xTest=irisSparse[-train,]

 # Re-train best model with best cost value.
 m=LiblineaR(data=xTrain,target=yTrain,type=bestType,cost=bestCost,bias=1,verbose=FALSE)

 # Make prediction
 p=predict(m,xTest,proba=pr,decisionValues=TRUE)

 # Display confusion matrix
```

```
 res=table(p$predictions,yTest)
 print(res)
}

##############################################

# Try regression instead, to predict sepal length on the basis of sepal width and petal width:

xTrain=iris[c(1:25,51:75,101:125),2:3]
yTrain=iris[c(1:25,51:75,101:125),1]
xTest=iris[c(26:50,76:100,126:150),2:3]
yTest=iris[c(26:50,76:100,126:150),1]

# Center and scale data
s=scale(xTrain,center=TRUE,scale=TRUE)

# Estimate MSE in cross-vaidation on a train set
MSECross=LiblineaR(data = s, target = yTrain, type = 13, cross = 10, svr_eps=.01)

# Build the model
m=LiblineaR(data = s, target = yTrain, type = 13, cross=0, svr_eps=.01)

# Test it, after test data scaling:
s2=scale(xTest,attr(s,"scaled:center"),attr(s,"scaled:scale"))
pred=predict(m,s2)$predictions
MSETest=mean((yTest-pred)^2)

# Was MSE well estimated?
print(MSETest-MSECross)

# Distribution of errors
print(summary(yTest-pred))
```

`predict.LiblineaR` *Predictions with LiblineaR model*

Description

The function applies a model (classification or regression) produced by the `LiblineaR` function to every row of a data matrix and returns the model predictions.

Usage

```
## S3 method for class 'LiblineaR'
predict(object, newx, proba = FALSE, decisionValues = FALSE, ...)
```

Arguments

object	Object of class "LiblineaR", created by LiblineaR.
newx	An n x p matrix containing the new input data. A vector will be transformed to a n x 1 matrix. Sparse matrices of class matrix.csr from package SparseM or sparse matrices of class dgCMatrix or dgRMatrix from package Matrix are also accepted. Note that C code at the core of LiblineaR package corresponds to a row-based sparse format. Hence, dgCMatrix inputs are first transformed into dgRMatrix format, which requires small extra computation time.
proba	Logical indicating whether class probabilities should be computed and returned. Only possible if the model was fitted with type=0, type=6 or type=7, i.e. a Logistic Regression. Default is FALSE.
decisionValues	Logical indicating whether model decision values should be computed and returned. Only possible for classification models (type<10). Default is FALSE.
...	Currently not used

Value

By default, the returned value is a list with a single entry:

predictions	A vector of predicted labels (or values for regression).

If proba is set to TRUE, and the model is a logistic regression, an additional entry is returned:

probabilities	An n x k matrix (k number of classes) of the class probabilities. The columns of this matrix are named after class labels.

If decisionValues is set to TRUE, and the model is not a regression model, an additional entry is returned:

decisionValues	An n x k matrix (k number of classes) of the model decision values. The columns of this matrix are named after class labels.

Note

If the data on which the model has been fitted have been centered and/or scaled, it is very important to apply the same process on the newx data as well, with the scale and center values of the training data.

Author(s)

Thibault Helleputte <thibault.helleputte@dnalytics.com> and
Jerome Paul <jerome.paul@dnalytics.com> and Pierre Gramme.
Based on C/C++-code by Chih-Chung Chang and Chih-Jen Lin

References

- For more information on LIBLINEAR itself, refer to:
R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin.
LIBLINEAR: A Library for Large Linear Classification,
Journal of Machine Learning Research 9(2008), 1871-1874.
https://www.csie.ntu.edu.tw/~cjlin/liblinear/

See Also

LiblineaR

Index