Package 'LiblineaR.ACF'

January 4, 2016

Title Linear Classification with Online Adaptation of Coordinate Frequencies

Version 1.94-2

Author Aydin Demircioglu <aydin.demircioglu@ini.rub.de>; Tobias Glasmachers <tobias.glasmachers@ini.rub.de>; Urun Dogan <urundogan@gmail.com>

Maintainer Aydin Demircioglu <aydin.demircioglu@ini.rub.de>

Description Solving the linear SVM problem with coordinate descent is very efficient and is implemented in one of the most often used packages, 'LIBLINEAR' (available at http://www.csie.ntu.edu.tw/~cjlin/liblinear). It has been shown that the uniform selection of coordinates can be accelerated by using an online adaptation of coordinate frequencies (ACF). This package implements ACF and is based on 'LIBLINEAR' as well as the 'LiblineaR' package (<https://cran.r-project.org/package=LiblineaR>). It currently supports L2-regularized L1-loss as well as L2-loss linear SVM. Similar to 'LIBLINEAR' multi-class classification (one-vs-the rest, and Crammer & Singer method) and cross validation for model selection is supported. The training of the models based on ACF is much faster than standard 'LIBLINEAR' on many problems.

License GPL-2

Date 2016-01-04

LazyLoad yes

Suggests SparseM, testthat

URL http://github.com/aydindemircioglu/liblineaR.ACF/

NeedsCompilation yes

Repository CRAN

RoxygenNote 5.0.1

Date/Publication 2016-01-04 12:39:03

R topics documented:

`LiblineaR.ACF` *Linear predictive models estimation with Online Adaptation of Coordinate Frequencies based on the LIBLINEAR C/C++ Library.*

Description

`LiblineaR.ACF` is a modification of the LiblineaR package that uses the idea of adaptive coordinate frequencies (ACF) method. Solving the linear SVM problem with coordinate descent is very efficient and is implemented in one of the most often used packages, LIBLINEAR (available at http://www.csie.ntu.edu.tw/~cjlin/liblinear). It has been shown that the uniform selection of coordinates can be accelerated by using an online adaptation of coordinate frequencies (ACF). This package implements ACF and is based on LIBLINEAR as well as the LiblineaR package (https://cran.r-project.org/package=LiblineaR). It currently supports L2-regularized L1-loss as well as L2-loss linear SVM. Similar to LIBLINEAR multi-class classification (one-vs-the rest, and Crammer & Singer method) and cross validation for model selection is supported. The training of the models based on ACF is much faster than standard LIBLINEAR on many problems.

Usage

```
LiblineaR.ACF(data, target, type = 0, cost = 1, epsilon = 0.01,
  bias = TRUE, wi = NULL, cross = 0, change_rate = 0.2,
  pref_min = 0.05, pref_max = 20, max_iter = 1000, verbose = FALSE, ...)
```

Arguments

`data` a nxp data matrix. Each row stands for an example (sample, point) and each column stands for a dimension (feature, variable). A sparse matrix (from SparseM package) will also work.

`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.

`type` `LiblineaR` can produce several types of (generalized) linear models, by combining several types of loss functions and regularization schemes. The regularization is L2, and the losses can be the regular L2-loss or L1-loss. The default value for `type` is 1. Valid options are:

for multi-class classification

- 1 – L2-regularized L2-loss support vector classification (dual)
- 3 – L2-regularized L1-loss support vector classification (dual)
- 4 – support vector classification by Crammer and Singer

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 in the LiblineaR package.
epsilon	set tolerance of termination criterion for optimization. If NULL, the LIBLINEAR defaults are used, which are: **if** type **is 1, 3 or 4** epsilon=0.1 The meaning of epsilon is as follows: Dual maximal violation $\leq$ epsilon (default 0.1)
bias	if bias is TRUE (default), instances of data becomes [data; 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.
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.
change_rate	learning rate of the preference adaptation, default is 0.2
pref_min	lower bound on the preference adaptation, default is 1/20
pref_max	upper bound on the preference adaptation, default is 20
max_iter	the maximum number of iterations, default (from original LIBLINEAR code) is 1000.
verbose	if TRUE, information are printed. Default is FALSE.
...	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 http://www.csie.ntu.edu.tw/~cjlin/liblinear. The ACF code can be found at http://www.ini.rub.de/PEOPLE/glasmtbl/code/acf-cd.

Value

If cross>0, the average accuracy (classification) 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 is TRUE, 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, 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	TRUE or FALSE, according to the value of bias
ClassNames	A vector containing the class names.

Note

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

Author(s)

Aydin Demircioglu <aydin.demircioglu@ini.rub.de> Based on LiblineaR package by Thibault Helleputte <thibault.helleputte@dnalytics.com> and
Pierre Gramme <pierre.gramme@dnalytics.com>.
Based on C/C++-code by Chih-Chung Chang and Chih-Jen Lin Based on C/C++-code by Tobias Glasmachers and Urun Dogan

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.
http://www.csie.ntu.edu.tw/~cjlin/liblinear

See Also

predict.LiblineaR.ACF, 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 3-fold cross-validations
tryTypes=c(1,3,4)
tryCosts=c(1000,1,0.001)
bestCost=NA
bestAcc=0
bestType=NA

for(ty in tryTypes){
for(co in tryCosts){
acc=LiblineaR.ACF(data=s,target=yTrain,type=ty,cost=co,
bias=TRUE,cross=3,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.ACF(data=s,target=yTrain,type=bestType,cost=bestCost,bias=TRUE,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:

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.ACF(data=xTrain,target=yTrain,type=bestType,cost=bestCost,bias=TRUE,verbose=FALSE)

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

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

`predict.LiblineaR.ACF` *Predictions with LiblineaR.ACF model*

Description

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

Usage

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

Arguments

`object`	Object of class "`LiblineaR.ACF`", created by `LiblineaR.ACF`.
`newx`	An n x p matrix containing the new input data. A vector will be transformed to a n x 1 matrix. A sparse matrix (from SparseM package) will also work.
`decisionValues`	Logical indicating whether model decision values should be computed and returned. 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.

If `decisionValues` is set to `TRUE`, 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 Pierre Gramme <pierre.gramme@dnalytics.com>.
Modified by Aydin Demircioglu.
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.
  http://www.csie.ntu.edu.tw/~cjlin/liblinear

See Also

`LiblineaR.ACF`

Index