

# The caretLSF Package

June 11, 2008

**Version** 1.16

**Date** 2008-06-10

**Title** Classification and Regression Training LSF Style

**Author** Max Kuhn, Nathan Coulter

**Description** Augment some caret functions for parallel processing

**Maintainer** Max Kuhn <Max.Kuhn@pfizer.com>

**Depends** caret (>= 3.21), Rlsf (>= 1.1.0)

**Suggests** randomForest

**License** GPL-2

## R topics documented:

rfLSF . . . . .	1
trainLSF . . . . .	3
trainLSFControl . . . . .	6

<b>Index</b>	<b>8</b>
--------------	----------

---

rfLSF *Parallel RandomForest*

---

## Description

RandomForest model building in parallel using LSF

## Usage

```
rfLSF(x, y, workers = 10, control = lsf.ctrl(), ...)
```

## Arguments

<code>x</code>	a data frame or a matrix of predictor
<code>y</code>	A response vector. If a factor, classification is assumed, otherwise regression is assumed. If omitted, <code>randomForest</code> will run in unsupervised mode.
<code>workers</code>	the number of compute nodes. Note that the value of <code>ntree</code> will be used for each compute node.
<code>control</code>	an optional control object for starting parallel jobs
<code>...</code>	various options to pass to <code>randomForest.default</code>

## Details

The function calls `randomForest.default` on several nodes and uses `combine` put all of the models back together.

Note that the call of the output object will mirror the values of `x` and `y` passed to `rfLSF`.

## Value

a `randomForest` object

## Author(s)

Max Kuhn

## See Also

objects to See Also as [help](#),

## Examples

```
## Not run:
## Classification:
##data(iris)
set.seed(71)
iris.rf <- rfLSF(
  iris[, 1:4],
  iris$Species,
  importance=TRUE,
  proximity=TRUE)

## End(Not run)
```

**Description**

This function sets up a grid of tuning parameters for a number of classification and regression routines, fits each model and calculates a resampling based performance measure.

**Usage**

```
trainLSF(
  x, y,
  method = "rf", ...,
  metric = ifelse(is.factor(y), "Accuracy", "RMSE"),
  trControl = trainLSFControl(),
  tuneGrid = NULL,
  tuneLength = 3)
```

**Arguments**

x	a data frame containing training data where samples are in rows and features are in columns.
y	a numeric or factor vector containing the outcome for each sample.
method	a string specifying which classification or regression model to use. Possible values are: lm, rda, lda, gbm, rf, nnet, multinom, gpls, lvq, rpart, knn, pls, pam, nb, earth, treebag, svmpoly, svmradial, fda, bagEarth, bagFDA, glmboost, gamboost, blackboost, ada, ctree and cforest. See the Details section below.
...	arguments passed to the classification or regression routine (such as <a href="#">randomForest</a> ). Errors will occur if values for tuning parameters are passed here.
metric	a string that specifies what summary metric will be used to select the optimal model. Possible values are "RMSE" and "Rsquared" for regression and "Accuracy" and "Kappa" for classification.(NOTE: If given, this argument must be named.)
trControl	a list of values that define how this function acts. See <a href="#">trainLSFControl</a> . (NOTE: If given, this argument must be named.)
tuneGrid	a data frame with possible tuning values. The columns are named the same as the tuning parameters in each method preceded by a period (e.g. .decay, .lambda). See the function <a href="#">createGrid</a> in this package for more details. (NOTE: If given, this argument must be named.)
tuneLength	an integer denoting the number of levels for each tuning parameters that should be generated by <a href="#">createGrid</a> . (NOTE: If given, this argument must be named.)

## Details

`trainLSF` can be used to tune models by picking the complexity parameters that are associated with the optimal resampling statistics. For particular model, a grid of parameters (if any) is created and the model is trained on slightly different data for each candidate combination of tuning parameters. Across each data set, the performance of held-out samples is calculated and the mean and standard deviation is summarized for each combination. The combination with the optimal resampling statistic is chosen as the final model and the entire training set is used to fit a final model.

Currently, the `trainLSF` function does not support model specification via a formula. It assumes that all of the predictors are numeric (perhaps generated by `model.matrix`).

A variety of models are currently available. The table below enumerates the models and the values of the `method` argument, as well as the complexity parameters used by `trainLSF`.

Model	method	Value	Package	Tuning Parameter(s)
Recursive partitioning	<code>rpart</code>		<b>rpart</b>	<code>maxdepth</code>
	<code>ctree</code>		<b>party</b>	<code>mincriterion</code>
Boosted Trees	<code>gbm</code>		<b>gbm</b>	<code>interaction depth, n.trees, shrinkage</code>
	<code>blackboost</code>		<b>mboost</b>	<code>maxdepth, mstop</code>
Boosted regression models	<code>ada</code>		<b>ada</b>	<code>maxdepth, iter, nu</code>
	<code>glmboost</code>		<b>mboost</b>	<code>mstop</code>
Random forests	<code>gamboost</code>		<b>mboost</b>	<code>mstop</code>
	<code>rf</code>		<b>randomForest</b>	<code>mtry</code>
Bagged Trees	<code>cforest</code>		<b>party</b>	<code>mtry</code>
	<code>treebag</code>		<b>ipred</b>	None
Neural networks	<code>nnet</code>		<b>nnet</b>	<code>decay, size</code>
Partial least squares	<code>pls</code>		<b>pls, caret</b>	<code>ncomp</code>
Support Vector Machines (RBF)	<code>svmradial</code>		<b>kernlab</b>	<code>sigma, C</code>
Support Vector Machines (polynomial)	<code>svmpoly</code>		<b>kernlab</b>	<code>scale, degree, C</code>
Linear least squares	<code>lm</code>		<b>stats</b>	None
Multivariate adaptive regression splines	<code>earth</code>		<b>earth</b>	<code>degree, nprune</code>
Bagged MARS	<code>bagEarth</code>		<b>caret, earth</b>	<code>degree, nprune</code>
Elastic Net	<code>enet</code>		<b>elasticnet</b>	<code>lambda, fraction</code>
The Lasso	<code>enet</code>		<b>elasticnet</b>	<code>fraction</code>
Linear discriminant analysis	<code>lda</code>		<b>MASS</b>	None
Logistic/multinomial regression	<code>multinom</code>		<b>nnet</b>	<code>decay</code>
Regularized discriminant analysis	<code>rda</code>		<b>klaR</b>	<code>lambda, gamma</code>
Flexible discriminant analysis (MARS)	<code>fda</code>		<b>mda, earth</b>	<code>degree, nprune</code>
Bagged FDA	<code>bagFDA</code>		<b>caret, earth</b>	<code>degree, nprune</code>
k nearest neighbors	<code>knn3</code>		<b>caret</b>	<code>k</code>
Nearest shrunken centroids	<code>pam</code>		<b>pamr</b>	<code>threshold</code>
Naive Bayes	<code>nb</code>		<b>klaR</b>	<code>usekernel</code>
Generalized partial least squares	<code>gpls</code>		<b>gpls</b>	<code>K.prov</code>
Learned vector quantization	<code>lvq</code>		<b>class</b>	<code>k</code>

By default, the function `createGrid` is used to define the candidate values of the tuning parameters. The user can also specify their own. To do this, a data frame is created with columns for each

tuning parameter in the model. The column names must be the same as those listed in the table above with a leading dot. For example, `ncomp` would have the column heading `.ncomp`. This data frame can then be passed to `createGrid`.

In some cases, models may require control arguments. These can be passed via the three dots argument. Note that some models can specify tuning parameters in the control objects. If specified, these values will be superseded by those given in the `createGrid` argument.

The vignette entitled "caret Manual – Model Building" in the caret package has more details and examples related to this function.

`caretLSF` is a parallel version of the `train` function in the `caret` package. This function implements a combination of sequential and parallel jobs. For example, if a PLS model with 10 candidate values for the number of retained components is requested along with 10-fold cross-validation, the 10 jobs for each candidate tuning parameter are submitted in parallel. Once finished, the process is started again with the second candidate tuning parameter etc.

The function also has a fault-tolerance function where any jobs that are abnormally long (say 20x the longest job to date) can be killed automatically. In this way, a problem with a worker node may not destroy the results to date. For example, if the models are tuned using 25 bootstrap results, the failure of one node will cause the function to tune the model with the remainder and keep going.

## Value

A list is returned of class `train` containing:

<code>modelType</code>	an identifier of the model type.
<code>results</code>	a data frame the training error rate and values of the tuning parameters.
<code>call</code>	the (matched) function call with dots expanded
<code>dots</code>	a list containing any ... values passed to the original call
<code>metric</code>	a string that specifies what summary metric will be used to select the optimal model.
<code>trControl</code>	the list of control parameters.
<code>finalModel</code>	an fit object using the best parameters
<code>trainingData</code>	a data frame
<code>resamples</code>	A data frame with columns <code>obs</code> , <code>pred</code> and <code>group</code> which are produced during the resampling process for the optimal model

## Author(s)

Max Kuhn

## See Also

[createGrid](#), [createFolds](#)

**Examples**

```
## Not run:
data(iris)
TrainData <- iris[,1:4]
TrainClasses <- iris[,5]

knnFit <- trainLSF(
  TrainData, TrainClasses,
  "knn",
  tuneLength = 10,
  trControl = trainLSFControl(method = "boot"))

## End(Not run)
```

---

trainLSFControl      *Control parameters for train*

---

**Description**

Control of printing and resampling for [trainLSF](#)

**Usage**

```
trainLSFControl(
  method = "boot",
  number = ifelse(method == "cv", 10, 25),
  verboseIter = TRUE,
  returnData = TRUE,
  p = 0.5,
  selectionFunction = "best",
  index = NULL,
  numWorkers = 5,
  buffer = 20,
  pause = 10,
  lsf = lsf.ctrl())
```

**Arguments**

method	The resampling method: <code>boot</code> , <code>cv</code> , <code>LOOCV</code> , <code>LGOCV</code> (for repeated training/test splits), or <code>oob</code> (only for random forest, bagged trees, bagged earth, bagged flexible discriminant analysis, or conditional tree forest models)
number	Either the number of folds or number of resampling iterations
verboseIter	A logical for printing a training log.
returnData	A logical for saving the data
p	For leave-group out cross-validation: the training percentage
selectionFunction	the function used to select the optimal tuning parameter. This can be a name of the function or the function itself. See <a href="#">best</a> for details and other options.

<code>index</code>	a list with elements for each resampling iteration. Each list element is the sample rows used for training at that iteration.
<code>numWorkers</code>	the number of nodes that the job should be split across
<code>buffer</code>	a multiplier for the job kill threshold: how many times slower than the slowest job does a task have to be before it is stopped
<code>pause</code>	number of seconds between checking the current jobs
<code>lsf</code>	a list of optional control parameters for submitting jobs. See <a href="#">lsf.ctrl</a> for more details

**Value**

An echo of the parameters specified

**Author(s)**

max Kuhn

# Index

\*Topic **models**

rfLSF, 1

trainLSF, 3

\*Topic **utilities**

trainLSFControl, 6

best, 6

combine, 2

createFolds, 5

createGrid, 3-5

help, 2

lsf.ctrl, 7

model.matrix, 4

randomForest, 2, 3

randomForest.default, 2

rfLSF, 1

train, 5

trainLSF, 3, 6

trainLSFControl, 3, 6