

Package ‘caret’

November 5, 2009

Version 4.27

Date 2009-11-01

Title Classification and Regression Training

Author Max Kuhn. Contributions from Jed Wing, Steve Weston, Andre Williams, Chris Keefer and Allan Engelhardt

Description Misc functions for training and plotting classification and regression models

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

Depends R (>= 2.5.1), lattice

URL <http://caret.r-forge.r-project.org/>

Suggests gbm, pls, mlbench, rpart, ellipse, ipred, klaR, randomForest, gpls, pamr, kernlab, mda, mgcv, nnet, class, MASS, mboost, earth (>= 2.2-3), party, ada, affy, proxy, e1071, grid, elasticnet, SDDA, caTools, RWeka (>= 0.3-14), superpc, penalized, sparseLDA (>= 0.1-1), spls, sda, glmnet, relaxo, lars, vbmp, nodeHarvest

License GPL-2

Repository CRAN

Repository/R-Forge/Project caret

Repository/R-Forge/Revision 94

Date/Publication 2009-11-05 12:34:33

R topics documented:

Alternate Affy Gene Expression Summary Methods.	3
applyProcessing	4
as.table.confusionMatrix	5
aucRoc	6
bagEarth	7
bagFDA	9

BloodBrain	11
caretFuncs	11
classDist	12
confusionMatrix	14
cox2	17
createDataPartition	18
createGrid	19
dotPlot	20
featurePlot	21
filterVarImp	22
findCorrelation	24
findLinearCombos	25
format.bagEarth	26
histogram.train	27
knn3	29
knnreg	30
lattice.rfe	32
maxDissim	33
mdrr	36
nearZeroVar	36
normalize.AffyBatch.normalize2Reference	38
normalize2Reference	39
oil	40
oneSE	41
panel.needle	43
pcaNNet.default	44
plot.train	46
plot.varImp.train	48
plotClassProbs	49
plotObsVsPred	50
plsda	51
postResample	54
pottery	55
predict.bagEarth	56
predict.knn3	57
predict.knnreg	58
predict.train	58
predictors	61
preProcess	64
print.confusionMatrix	66
print.train	67
resampleHist	68
resampleSummary	69
rfe	70
rfeControl	74
roc	76
sensitivity	77
spatialSign	81

summary.bagEarth	82
tecator	83
train	84
trainControl	91
varImp	92

Index	95
--------------	-----------

Alternate Affy Gene Expression Summary Methods.
Generate Expression Values from Probes

Description

Generate an expression from the probes

Usage

```
generateExprVal.method.trimMean(probes, trim = 0.15)
```

Arguments

probes	a matrix of probe intensities with rows representing probes and columns representing samples. Usually <code>pm(probeset)</code> where <code>probeset</code> is a of class <code>ProbeSet</code>
trim	the fraction (0 to 0.5) of observations to be trimmed from each end of the data before the mean is computed.

Value

A list containing entries:

<code>exprs</code>	The expression values.
<code>se.exprs</code>	The standard error estimate.

See Also

[generateExprSet-methods](#)

Examples

```
## Not run:
# first, let affy/expresso know that the method exists
express.summary.stat.methods <- c(express.summary.stat.methods, "trimMean")

example not run, as it would take a while
RawData <- ReadAffy(celfile.path=FilePath)

expresso(RawData,
```

```

bgcorrect.method="rma",
normalize.method="quantiles",
normalize.param = list(type= "pmonly"),
pmcorrect.method="pmonly",
summary.method="trimMean")

step1 <- bg.correct(RawData, "rma")
step2 <- normalize.AffyBatch.quantiles(step1)
step3 <- computeExprSet(step2, "pmonly", "trimMean")

## End(Not run)

```

applyProcessing *Data Processing on Predictor Variables (Deprecated)*

Description

processData is used to estimate data processing parameters and applyProcessing takes the results and applies them to data objects

Usage

```

processData(x, center = TRUE, scale = TRUE, na.remove = TRUE)
applyProcessing(x, object)

```

Arguments

x	A matrix or data frame with variables in rows and samples in columns
center	A boolean to center the data.
scale	A boolean to scale the data.
na.remove	Should NA values be removed?
object	the results from a previous call to processData.

Details

These functions are deprecated and have been replaced by the [preProcess](#) class. In a few more versions, they will not be around.

Value

x, after it has been processed.

Author(s)

Max Kuhn

Examples

```
set.seed(115)
xData <- matrix(rgamma(6 * 4, 5), 6)

trainValues01 <- apply(xData, 2, processData)
applyProcessing(xData, trainValues01)
```

```
as.table.confusionMatrix
      Save Confusion Table Results
```

Description

Conversion functions for class `confusionMatrix`

Usage

```
## S3 method for class 'confusionMatrix':
as.matrix(x, what = "xtabs", ...)

## S3 method for class 'confusionMatrix':
as.table(x, ...)
```

Arguments

<code>x</code>	an object of class <code>confusionMatrix</code>
<code>what</code>	data to convert to matrix. Either "xtabs", "overall" or "classes"
<code>...</code>	not currently used

Details

For `as.table`, the cross-tabulations are saved. For `as.matrix`, the three object types are saved in matrix format.

Value

A matrix or table

Author(s)

Max Kuhn

See Also

[confusionMatrix](#)

Examples

```
#####
## 2 class example

lvs <- c("normal", "abnormal")
truth <- factor(rep(lvs, times = c(86, 258)),
               levels = rev(lvs))
pred <- factor(
  c(
    rep(lvs, times = c(54, 32)),
    rep(lvs, times = c(27, 231))),
  levels = rev(lvs))

xtab <- table(pred, truth)

results <- confusionMatrix(xtab)
as.table(results)
as.matrix(results)
as.matrix(results, what = "overall")
as.matrix(results, what = "classes")

#####
## 3 class example

library(MASS)

fit <- lda(Species ~ ., data = iris)
model <- predict(fit)$class

irisTabs <- table(model, iris$Species)

results <- confusionMatrix(irisTabs)
as.table(results)
as.matrix(results)
as.matrix(results, what = "overall")
as.matrix(results, what = "classes")
```

aucRoc

Compute the area under an ROC curve

Description

This function uses the trapezoidal rule to calculate the area. If the area is less than .5, then $1 - \text{area}$ is used.

Usage

```
aucRoc(object)
```

Arguments

object object from the `roc` function or a matrix/data frame with columns named "sensitivity" and "specificity"

Value

a scalar number

Author(s)

Max Kuhn

See Also

[sensitivity](#), [specificity](#), [roc](#)

Examples

```
set.seed(6)
testData <- data.frame(
  x = c(rnorm(200), rnorm(200) + 1),
  group = factor(rep(letters[1:2], each = 200)))

densityplot(~testData$x,
  groups = testData$group,
  auto.key = TRUE)

rocValues <- roc(testData$x, testData$group)
aucRoc(rocValues)
```

bagEarth

Bagged Earth

Description

A bagging wrapper for multivariate adaptive regression splines (MARS) via the `earth` function

Usage

```
## S3 method for class 'formula':
bagEarth(formula, data = NULL, B = 50,
  summary = mean, keepX = TRUE,
  ..., subset, weights, na.action = na.omit)
## Default S3 method:
bagEarth(x, y, weights = NULL, B = 50,
  summary = mean, keepX = TRUE, ...)
```

Arguments

<code>formula</code>	A formula of the form $y \sim x_1 + x_2 + \dots$
<code>x</code>	matrix or data frame of 'x' values for examples.
<code>y</code>	matrix or data frame of numeric values outcomes.
<code>weights</code>	(case) weights for each example - if missing defaults to 1.
<code>data</code>	Data frame from which variables specified in 'formula' are preferentially to be taken.
<code>subset</code>	An index vector specifying the cases to be used in the training sample. (NOTE: If given, this argument must be named.)
<code>na.action</code>	A function to specify the action to be taken if 'NA's are found. The default action is for the procedure to fail. An alternative is <code>na.omit</code> , which leads to rejection of cases with missing values on any required variable. (NOTE: If given, this argument must be named.)
<code>B</code>	the numebr of bootstrap samples
<code>summary</code>	a function with a single argument specifying how the bagged predictions should be summarized
<code>keepX</code>	a logical: should the original training data be kept?
<code>...</code>	arguments passed to the <code>earth</code> function

Details

The function computes a Earth model for each bootstrap sample.

Value

A list with elements

<code>fit</code>	a list of B Earth fits
<code>B</code>	the number of bootstrap samples
<code>call</code>	the function call
<code>x</code>	either NULL or the value of <code>x</code> , depending on the value of <code>keepX</code>
<code>oob</code>	a matrix of performance estimates for each bootstrap sample

Author(s)

Max Kuhn (`bagEarth.formula` is based on Ripley's `nnet.formula`)

References

J. Friedman, "Multivariate Adaptive Regression Splines" (with discussion) (1991). *Annals of Statistics*, 19/1, 1-141.

See Also

[earth](#), [predict.bagEarth](#)

Examples

```
library(mda)
library(earth)
data(trees)
fit1 <- earth(trees[,-3], trees[,3])
fit2 <- bagEarth(trees[,-3], trees[,3], B = 10)
```

bagFDA

*Bagged FDA***Description**

A bagging wrapper for flexible discriminant analysis (FDA) using multivariate adaptive regression splines (MARS) basis functions

Usage

```
bagFDA(x, ...)
## S3 method for class 'formula':
bagFDA(formula, data = NULL, B = 50, keepX = TRUE,
        ..., subset, weights, na.action = na.omit)
## Default S3 method:
bagFDA(x, y, weights = NULL, B = 50, keepX = TRUE, ...)
```

Arguments

formula	A formula of the form $y \sim x_1 + x_2 + \dots$
x	matrix or data frame of 'x' values for examples.
y	matrix or data frame of numeric values outcomes.
weights	(case) weights for each example - if missing defaults to 1.
data	Data frame from which variables specified in 'formula' are preferentially to be taken.
subset	An index vector specifying the cases to be used in the training sample. (NOTE: If given, this argument must be named.)
na.action	A function to specify the action to be taken if 'NA's are found. The default action is for the procedure to fail. An alternative is na.omit, which leads to rejection of cases with missing values on any required variable. (NOTE: If given, this argument must be named.)
B	the numebr of bootstrap samples
keepX	a logical: should the original training data be kept?
...	arguments passed to the mars function

Details

The function computes a FDA model for each bootstap sample.

Value

A list with elements

<code>fit</code>	a list of B FDA fits
<code>B</code>	the number of bootstrap samples
<code>call</code>	the function call
<code>x</code>	either <code>NULL</code> or the value of <code>x</code> , depending on the value of <code>keepX</code>
<code>oob</code>	a matrix of performance estimates for each bootstrap sample

Author(s)

Max Kuhn (`bagFDA.formula` is based on Ripley's `nnet.formula`)

References

J. Friedman, "Multivariate Adaptive Regression Splines" (with discussion) (1991). *Annals of Statistics*, 19/1, 1-141.

See Also

[fda](#), [predict.bagFDA](#)

Examples

```
library(mlbench)
library(earth)
data(Glass)

set.seed(36)
inTrain <- sample(1:dim(Glass)[1], 150)

trainData <- Glass[ inTrain, ]
testData  <- Glass[-inTrain, ]

baggedFit <- bagFDA(Type ~ ., trainData)
baggedMat <- table(
  predict(baggedFit, testData[, -10]),
  testData[, 10])

print(baggedMat)

classAgreement(baggedMat)
```

`BloodBrain`*Blood Brain Barrier Data*

Description

Mente and Lombardo (2005) develop models to predict the log of the ratio of the concentration of a compound in the brain and the concentration in blood. For each compound, they computed three sets of molecular descriptors: MOE 2D, rule-of-five and Charge Polar Surface Area (CPSA). In all, 134 descriptors were calculated. Included in this package are 208 non-proprietary literature compounds. The vector `logBBB` contains the concentration ratio and the data frame `bbbDescr` contains the descriptor values.

Usage

```
data(BloodBrain)
```

Value

<code>bbbDescr</code>	data frame of chemical descriptors
<code>logBBB</code>	vector of assay results

Source

Mente, S.R. and Lombardo, F. (2005). A recursive-partitioning model for blood-brain barrier permeation, *Journal of Computer-Aided Molecular Design*, Vol. 19, pg. 465–481.

`caretFuncs`*Backwards Feature Selection Helper Functions*

Description

Ancillary functions for backwards selection

Usage

```
pickSizeTolerance(x, metric, tol = 1.5, maximize)
pickSizeBest(x, metric, maximize)
```

```
pickVars(y, size)
```

```
caretFuncs
lmFuncs
rfFuncs
treebagFuncs
ldaFuncs
nbFuncs
```

Arguments

x	a matrix or data frame with the performance metric of interest
metric	a character string with the name of the performance metric that should be used to choose the appropriate number of variables
maximize	a logical; should the metric be maximized?
tol	a scalar to denote the acceptable difference in optimal performance (see Details below)
y	a list of data frames with variables Overall and var
size	an integer for the number of variables to retain

Details

This page describes the functions that are used in backwards selection (aka recursive feature elimination). The functions described here are passed to the algorithm via the `functions` argument of [rfeControl](#).

See [rfeControl](#) for details on how these functions should be defined.

Author(s)

Max Kuhn

See Also

[rfeControl](#), [rfe](#)

classDist

Compute and predict the distances to class centroids

Description

This function computes the class centroids and covariance matrix for a training set for determining Mahalanobis distances of samples to each class centroid.

Usage

```
classDist(x, ...)

## Default S3 method:
classDist(x, y, groups = 5, pca = FALSE, keep = NULL, ...)

## S3 method for class 'classDist':
predict(object, newdata, trans = log, ...)
```

Arguments

x	a matrix or data frame of predictor variables
y	a numeric or factor vector of class labels
groups	an integer for the number of bins for splitting a numeric outcome
pca	a logical: should principal components analysis be applied to the dataset prior to splitting the data by class?
keep	an integer for the number of PCA components that should be used to predict new samples (NULL uses all within a tolerance of <code>sqrt(.Machine\$double.eps)</code>)
object	an object of class <code>classDist</code>
newdata	a matrix or data frame. If <code>vars</code> was previously specified, these columns should be in <code>newdata</code>
trans	an optional function that can be applied to each class distance. <code>trans = NULL</code> will not apply a function
...	optional arguments to pass (not currently used)

Details

For factor outcomes, the data are split into groups for each class and the mean and covariance matrix are calculated. These are then used to compute Mahalanobis distances to the class centers (using `predict.classDist`). The function will check for non-singular matrices.

For numeric outcomes, the data are split into roughly equal sized bins based on `groups`. Percentiles are used to split the data.

Value

for `classDist`, an object of class `classDist` with elements:

values	a list with elements for each class. Each element contains a mean vector for the class centroid and the inverse of the class covariance matrix
classes	a character vector of class labels
pca	the results of <code>prcomp</code> when <code>pca = TRUE</code>
call	the function call
p	the number of variables
n	a vector of samples sizes per class

For `predict.classDist`, a matrix with columns for each class. The column names are the names of the class with the prefix `dist..` In the case of numeric `y`, the class labels are the percentiles. For example, of `groups = 9`, the variable names would be `dist.11.11`, `dist.22.22`, etc.

Author(s)

Max Kuhn

See Also[mahalanobis](#)**Examples**

```

trainSet <- sample(1:150, 100)

distData <- classDist(iris[trainSet, 1:4],
                      iris$Species[trainSet])

newDist <- predict(distData,
                  iris[-trainSet, 1:4])

sploM(newDist, groups = iris$Species[-trainSet])

```

confusionMatrix *Create a confusion matrix*

Description

Calculates a cross-tabulation of observed and predicted classes with associated statistics.

Usage

```

confusionMatrix(data, ...)

## Default S3 method:
confusionMatrix(data, reference, positive = NULL,
                dnn = c("Prediction", "Reference"),
                prevalence = NULL, ...)

## S3 method for class 'table':
confusionMatrix(data, positive = NULL, prevalence = NULL, ...)

```

Arguments

data	a factor of predicted classes (for the default method) or an object of class table .
reference	a factor of classes to be used as the true results
positive	an optional character string for the factor level that corresponds to a "positive" result (if that makes sense for your data). If there are only two factor levels, the first level will be used as the "positive" result.
dnn	a character vector of dimnames for the table
prevalence	a numeric value or matrix for the rate of the "positive" class of the data. When data has two levels, prevalence should be a single numeric value. Otherwise, it should be a vector of numeric values with elements for each class. The vector should have names corresponding to the classes.
...	options to be passed to table . NOTE: do not include dnn here

Details

The functions requires that the factors have exactly the same levels.

For two class problems, the sensitivity, specificity, positive predictive value and negative predictive value is calculated using the `positive` argument. Also, the prevalence of the "event" is computed from the data (unless passed in as an argument), the detection rate (the rate of true events also predicted to be events) and the detection prevalence (the prevalence of predicted events).

Suppose a 2x2 table with notation

	Reference	
Predicted	Event	No Event
Event	A	B
No Event	C	D

The formulas used here are:

$$Sensitivity = A / (A + C)$$

$$Specificity = D / (B + D)$$

$$Prevalence = (A + C) / (A + B + C + D)$$

$$PPV = (sensitivity * Prevalence) / ((sensitivity * Prevalence) + ((1 - specificity) * (1 - Prevalence)))$$

$$NPV = (specificity * (1 - Prevalence)) / (((1 - sensitivity) * Prevalence) + (specificity * (1 - Prevalence)))$$

$$DetectionRate = A / (A + B + C + D)$$

$$DetectionPrevalence = (A + B) / (A + B + C + D)$$

See the references for discussions of the first five formulas.

For more than two classes, these results are calculated comparing each factor level to the remaining levels (i.e. a "one versus all" approach).

The overall accuracy and unweighted Kappa statistic are calculated.

The overall accuracy rate is computed along with a 95 percent confidence interval for this rate (using `binom.test`) and a one-sided test to see if the accuracy is better than the "no information rate," which is taken to be the largest class percentage in the data.

Value

a list with elements

<code>table</code>	the results of <code>table</code> on <code>data</code> and <code>reference</code>
<code>positive</code>	the positive result level
<code>overall</code>	a numeric vector with overall accuracy and Kappa statistic values
<code>byClass</code>	the sensitivity, specificity, positive predictive value, negative predictive value, prevalence, dection rate and detection prevalence for each class. For two class systems, this is calculated once using the <code>positive</code> argument

Author(s)

Max Kuhn

References

Kuhn, M. (2008), "Building predictive models in R using the caret package," *Journal of Statistical Software*, (<http://www.jstatsoft.org/v28/i05/>).

Altman, D.G., Bland, J.M. (1994) "Diagnostic tests 1: sensitivity and specificity," *British Medical Journal*, vol 308, 1552.

Altman, D.G., Bland, J.M. (1994) "Diagnostic tests 2: predictive values," *British Medical Journal*, vol 309, 102.

See Also

[as.table.confusionMatrix](#), [as.matrix.confusionMatrix](#), [sensitivity](#), [specificity](#), [posPredValue](#), [negPredValue](#), [print.confusionMatrix](#), [binom.test](#)

Examples

```
#####
## 2 class example

lvs <- c("normal", "abnormal")
truth <- factor(rep(lvs, times = c(86, 258)),
               levels = rev(lvs))
pred <- factor(
  c(
    rep(lvs, times = c(54, 32)),
    rep(lvs, times = c(27, 231))),
  levels = rev(lvs))

xtab <- table(pred, truth)

confusionMatrix(xtab)
confusionMatrix(pred, truth)
confusionMatrix(xtab, prevalence = 0.25)

#####
## 3 class example

library(MASS)

fit <- lda(Species ~ ., data = iris)
model <- predict(fit)$class

irisTabs <- table(model, iris$Species)

confusionMatrix(irisTabs)
confusionMatrix(model, iris$Species)

newPrior <- c(.05, .8, .15)
```

```
names(newPrior) <- levels(iris$Species)

confusionMatrix(irisTabs, prevalence = newPrior)
## Need names for prevalence
## Not run: confusionMatrix(irisTabs, prevalence = c(.05, .8, .15))
```

cox2

COX-2 Activity Data

Description

From Sutherland, O'Brien, and Weaver (2003): "A set of 467 cyclooxygenase-2 (COX-2) inhibitors has been assembled from the published work of a single research group, with in vitro activities against human recombinant enzyme expressed as IC50 values ranging from 1 nM to >100 uM (53 compounds have indeterminate IC50 values)."

The data are in the Supplemental Data file for the article.

A set of 255 descriptors (MOE2D and QikProp) were generated. To classify the data, we used a cutoff of $2^{2.5}$ to determine activity

Usage

```
data(cox2)
```

Value

cox2Descr	the descriptors
cox2IC50	the IC50 data used to determine activity
cox2Class	the categorical outcome ("Active" or "Inactive") based on the $2^{2.5}$ cutoff

Source

Sutherland, J. J., O'Brien, L. A. and Weaver, D. F. (2003). Spline-Fitting with a Genetic Algorithm: A Method for Developing Classification Structure-Activity Relationships, *Journal of Chemical Information and Computer Sciences*, Vol. 43, pg. 1906–1915.

```
createDataPartition
```

Data Splitting functions

Description

A series of test/training partitions are created using `createDataPartition` while `createResample` creates one or more bootstrap samples. `createFolds` splits the data into `k` groups.

Usage

```
createDataPartition(y,
                    times = 1,
                    p = 0.5,
                    list = TRUE,
                    groups = min(5, length(y)))
createResample(y, times = 10, list = TRUE)
createFolds(y, k = 10, list = TRUE, returnTrain = FALSE)
```

Arguments

<code>y</code>	a vector of outcomes
<code>times</code>	the number of partitions to create
<code>p</code>	the percentage of data that goes to training
<code>list</code>	logical - should the results be in a list (TRUE) or a matrix with the number of rows equal to <code>floor(p * length(y))</code> and <code>times</code> columns.
<code>groups</code>	for numeric <code>y</code> , the number of breaks in the quantiles (see below)
<code>k</code>	an integer for the number of folds.
<code>returnTrain</code>	a logical. When true, the values returned are the sample positions corresponding to the data used during training. This argument only works in conjunction with <code>list = TRUE</code>

Details

For bootstrap samples, simple random sampling is used.

For other data splitting, the random sampling is done within the levels of `y` when `y` is a factor in an attempt to balance the class distributions within the splits. For numeric `y`, the sample is split into `groups` sections based on quantiles and sampling is done within these subgroups. Also, for very small class sizes (≤ 3) the classes may not show up in both the training and test data

Value

A list or matrix of row position integers corresponding to the training data

Author(s)

Max Kuhn

Examples

```
data(oil)
createDataPartition(oilType, 2)

x <- rgamma(50, 3, .5)
inA <- createDataPartition(x, list = FALSE)

plot(density(x[inA]))
rug(x[inA])

points(density(x[-inA]), type = "l", col = 4)
rug(x[-inA], col = 4)

createResample(oilType, 2)

createFolds(oilType, 10)
createFolds(oilType, 5, FALSE)

createFolds(rnorm(21))
```

`createGrid`*Tuning Parameter Grid*

Description

This function creates a data frame that contains a grid of complexity parameters specific methods.

Usage

```
createGrid(method, len = 3, data = NULL)
```

Arguments

<code>method</code>	a string specifying which classification model to use. See train for a full list.
<code>len</code>	an integer specifying the number of points on the grid for each tuning parameter.
<code>data</code>	the training data (only needed in the case where the method is <code>cforest</code> , <code>earth</code> , <code>bagEarth</code> , <code>fda</code> , <code>bagFDA</code> , <code>rpart</code> , <code>svmRadial</code> , <code>pam</code> , <code>lars2</code> , <code>rf</code> or <code>pls</code>). The outcome should be in a column called <code>.outcome</code> .

Details

A grid is created with rows corresponding to complexity parameter combinations. If the model does not use tuning parameters (like a linear model), values of `NA` are returned. Columns are named the same as the parameter name, but preceded by a period.

For some models (see list above), the data should be passed to the function via the `data` argument. In these cases, the outcome should be included in a column named `.outcome`.

Value

A data frame where the rows are combinations of tuning parameters and columns correspond to the parameters. The column names should be the parameter names preceded by a dot (e.g. `.mtry`)

Author(s)

Max Kuhn

See Also

`train`

Examples

```
createGrid("rda", 4)
createGrid("lm")
createGrid("nnet")

## data needed for SVM with RBF:
tmp <- iris
names(tmp)[5] <- ".outcome"
head(tmp)
createGrid("svmRadial", data = tmp, len = 4)
```

dotPlot

Create a dotplot of variable importance values

Description

A lattice `dotplot` is created from an object of class `varImp.train`.

Usage

```
dotPlot(x, top = min(20, dim(x$importance)[1]), ...)
```

Arguments

`x` an object of class `varImp.train`
`top` the number of predictors to plot
`...` options passed to `dotplot`

Value

an object of class `trellis`.

Author(s)

Max Kuhn

See Also

`varImp`, `dotplot`

Examples

```
data(iris)
TrainData <- iris[,1:4]
TrainClasses <- iris[,5]

knnFit <- train(TrainData, TrainClasses, "knn")

knnImp <- varImp(knnFit)

dotPlot(knnImp)
```

featurePlot

Wrapper for Lattice Plotting of Predictor Variables

Description

A shortcut to produce lattice graphs

Usage

```
featurePlot(x, y,
            plot = if(is.factor(y)) "strip" else "scatter",
            labels = c("Feature", ""),
            ...)
```

Arguments

x	a matrix or data frame of continuous feature/probe/spectra data.
y	a factor indicating class membership.
plot	the type of plot. For classification: box, strip, density, pairs or ellipse. For regression, pairs or scatter
labels	a bad attempt at pre-defined axis labels
...	options passed to lattice calls.

Details

This function “stacks” data to get it into a form compatible with lattice and creates the plots

Value

An object of class “trellis”. The ‘update’ method can be used to update components of the object and the ‘print’ method (usually called by default) will plot it on an appropriate plotting device.

Author(s)

Max Kuhn

Examples

```
x <- matrix(rnorm(50*5), ncol=5)
y <- factor(rep(c("A", "B"), 25))

trellis.par.set(theme = col.whitebg(), warn = FALSE)
featurePlot(x, y, "ellipse")
featurePlot(x, y, "strip", jitter = TRUE)
featurePlot(x, y, "box")
featurePlot(x, y, "pairs")
```

filterVarImp

Calculation of filter-based variable importance

Description

Specific engines for variable importance on a model by model basis.

Usage

```
filterVarImp(x, y, nonpara = FALSE, ...)
```

Arguments

<code>x</code>	A matrix or data frame of predictor data
<code>y</code>	A vector (numeric or factor) of outcomes
<code>nonpara</code>	should nonparametric methods be used to assess the relationship between the features and response
<code>...</code>	options to pass to either <code>lm</code> or <code>loess</code>

Details

The importance of each predictor is evaluated individually using a “filter” approach.

For classification, ROC curve analysis is conducted on each predictor. For two class problems, a series of cutoffs is applied to the predictor data to predict the class. The sensitivity and specificity are computed for each cutoff and the ROC curve is computed. The trapezoidal rule is used to compute the area under the ROC curve. This area is used as the measure of variable importance. For multi-class outcomes, the problem is decomposed into all pair-wise problems and the area under the curve is calculated for each class pair (i.e class 1 vs. class 2, class 2 vs. class 3 etc.). For a specific class, the maximum area under the curve across the relevant pair-wise AUC’s is used as the variable importance measure.

For regression, the relationship between each predictor and the outcome is evaluated. An argument, `nonpara`, is used to pick the model fitting technique. When `nonpara = FALSE`, a linear model is fit and the absolute value of the t -value for the slope of the predictor is used. Otherwise, a loess smoother is fit between the outcome and the predictor. The R^2 statistic is calculated for this model against the intercept only null model.

Value

A data frame with variable importances. Column names depend on the problem type. For regression, the data frame contains one column: "Overall" for the importance values.

Author(s)

Max Kuhn

Examples

```
data(mdr)
filterVarImp(mdrDescr[, 1:5], mdrClass)

data(BloodBrain)

filterVarImp(bbbDescr[, 1:5], logBBB, nonpara = FALSE)
apply(
  bbbDescr[, 1:5],
  2,
  function(x, y) summary(lm(y~x))$coefficients[2,3],
  y = logBBB)

filterVarImp(bbbDescr[, 1:5], logBBB, nonpara = TRUE)
```

findCorrelation *Determine highly correlated variables*

Description

This function searches through a correlation matrix and returns a vector of integers corresponding to columns to remove to reduce pair-wise correlations.

Usage

```
findCorrelation(x, cutoff = .90, verbose = FALSE)
```

Arguments

x	A correlation matrix
cutoff	A numeric value for the pairwise absolute correlation cutoff
verbose	A boolean for printing the details

Details

The absolute values of pair-wise correlations are considered. If two variables have a high correlation, the function looks at the mean absolute correlation of each variable and removes the variable with the largest mean absolute correlation.

Value

A vector of indices denoting the columns to remove. If no correlations meet the criteria, `numeric(0)` is returned.

Author(s)

Original R code by Dong Li, modified by Max Kuhn

Examples

```
corrMatrix <- diag(rep(1, 5))
corrMatrix[2, 3] <- corrMatrix[3, 2] <- .7
corrMatrix[5, 3] <- corrMatrix[3, 5] <- -.7
corrMatrix[4, 1] <- corrMatrix[1, 4] <- -.67

corrDF <- expand.grid(row = 1:5, col = 1:5)
corrDF$correlation <- as.vector(corrMatrix)
levelplot(correlation ~ row+ col, corrDF)

findCorrelation(corrMatrix, cutoff = .65, verbose = TRUE)

findCorrelation(corrMatrix, cutoff = .99, verbose = TRUE)
```

findLinearCombos *Determine linear combinations in a matrix*

Description

Enumerate and resolve the linear combinations in a numeric matrix

Usage

```
findLinearCombos(x)
```

Arguments

x a numeric matrix

Details

The QR decomposition is used to determine if the matrix is full rank and then identify the sets of columns that are involved in the dependencies.

To "resolve" them, columns are iteratively removed and the matrix rank is rechecked.

Value

a list with elements:

linearCombos

If there are linear combinations, this will be a list with elements for each dependency that contains vectors of column numbers.

remove

a list of column numbers that can be removed to counter the linear combinations

Author(s)

Kirk Mettler and Jed Wing (enumLC) and Max Kuhn (findLinearCombos)

Examples

```
testData1 <- matrix(0, nrow=20, ncol=8)
testData1[,1] <- 1
testData1[,2] <- round(rnorm(20), 1)
testData1[,3] <- round(rnorm(20), 1)
testData1[,4] <- round(rnorm(20), 1)
testData1[,5] <- 0.5 * testData1[,2] - 0.25 * testData1[,3] - 0.25 * testData1[,4]
testData1[1:4,6] <- 1
testData1[5:10,7] <- 1
testData1[11:20,8] <- 1

findLinearCombos(testData1)

testData2 <- matrix(0, nrow=6, ncol=6)
```

```

testData2[,1] <- c(1, 1, 1, 1, 1, 1)
testData2[,2] <- c(1, 1, 1, 0, 0, 0)
testData2[,3] <- c(0, 0, 0, 1, 1, 1)
testData2[,4] <- c(1, 0, 0, 1, 0, 0)
testData2[,5] <- c(0, 1, 0, 0, 1, 0)
testData2[,6] <- c(0, 0, 1, 0, 0, 1)

findLinearCombos(testData2)

```

format.bagEarth *Format 'bagEarth' objects*

Description

Return a string representing the 'bagEarth' expression.

Usage

```

## S3 method for class 'bagEarth':
format(x, file = "", cat = TRUE, ...)

```

Arguments

x	An bagEarth object. This is the only required argument.
file	A connection, or a character string naming the file to print to. If "" (the default), the output prints to the standard output connection. See <code>link[base]{cat}</code> .
cat	a logical; should the equation be printed?
...	Arguments to format.earth .

Value

A character representation of the bagged earth object.

See Also

[earth](#)

Examples

```

a <- bagEarth(Volume ~ ., data = trees, B= 3)
cat(format(a))

# yields:
# (
# 31.61075
# + 6.587273 * pmax(0, Girth - 14.2)
# - 3.229363 * pmax(0, 14.2 - Girth)
# - 0.3167140 * pmax(0, 79 - Height)

```

```
# +
# 22.80225
# + 5.309866 * pmax(0, Girth - 12)
# - 2.378658 * pmax(0, 12 - Girth)
# + 0.793045 * pmax(0, Height - 80)
# - 0.3411915 * pmax(0, 80 - Height)
# +
# 31.39772
# + 6.18193 * pmax(0, Girth - 14.2)
# - 3.660456 * pmax(0, 14.2 - Girth)
# + 0.6489774 * pmax(0, Height - 80)
# )/3
```

histogram.train *Lattice functions for plotting resampling results*

Description

A set of lattice functions are provided to plot the resampled performance estimates (e.g. classification accuracy, RMSE) over tuning parameters (if any).

Usage

```
## S3 method for class 'train':
histogram(x, data = NULL, metric = x$metric, ...)

## S3 method for class 'train':
densityplot(x, data = NULL, metric = x$metric, ...)

## S3 method for class 'train':
xyplot(x, data = NULL, metric = x$metric, ...)

## S3 method for class 'train':
stripplot(x, data = NULL, metric = x$metric, ...)
```

Arguments

x	An object produced by <code>train</code>
data	This argument is not used
metric	A character string specifying the single performance metric that will be plotted
...	arguments to pass to either <code>histogram</code> , <code>densityplot</code> , <code>xyplot</code> or <code>stripplot</code>

Details

By default, only the resampling results for the optimal model are saved in the `train` object. The function `trainControl` can be used to save all the results (see the example below).

If leave-one-out or out-of-bag resampling was specified, plots cannot be produced (see the `method` argument of `trainControl`)

For `xyplot` and `stripplot`, the tuning parameter with the most unique values will be plotted on the x-axis. The remaining parameters (if any) will be used as conditioning variables. For `densityplot` and `histogram`, all tuning parameters are used for conditioning.

Using `horizontal = FALSE` in `stripplot` works.

Value

A lattice plot object

Author(s)

Max Kuhn

See Also

`train`, `trainControl`, `histogram`, `densityplot`, `xyplot`, `stripplot`

Examples

```
library(mlbench)
data(BostonHousing)

library(rpart)
rpartFit <- train(medv ~ .,
                  data = BostonHousing,
                  "rpart",
                  tuneLength = 9,
                  trControl = trainControl(
                    method = "boot",
                    returnResamp = "all"))

densityplot(rpartFit,
            adjust = 1.25)

xyplot(rpartFit,
       metric = "Rsquared",
       type = c("p", "a"))

stripplot(rpartFit,
          horizontal = FALSE,
          jitter = TRUE)
```

knn3

*k-Nearest Neighbour Classification***Description**

k -nearest neighbour classification that can return class votes for all classes.

Usage

```
## S3 method for class 'formula':
knn3(formula, data, subset, na.action, k = 5, ...)

## S3 method for class 'matrix':
knn3(x, y, k = 5, ...)

knn3Train(train, test, cl, k=1, l=0, prob = TRUE, use.all=TRUE)
```

Arguments

formula	a formula of the form $lhs \sim rhs$ where lhs is the response variable and rhs a set of predictors.
data	optional data frame containing the variables in the model formula.
subset	optional vector specifying a subset of observations to be used.
na.action	function which indicates what should happen when the data contain NAs.
k	number of neighbours considered.
x	a matrix of training set predictors
y	a factor vector of training set classes
...	additional parameters to pass to <code>knn3Train</code> . However, passing <code>prob = FALSE</code> will be over-ridden.
train	matrix or data frame of training set cases.
test	matrix or data frame of test set cases. A vector will be interpreted as a row vector for a single case.
cl	factor of true classifications of training set
l	minimum vote for definite decision, otherwise doubt. (More precisely, less than $k-1$ dissenting votes are allowed, even if k is increased by ties.)
prob	If this is true, the proportion of the votes for each class are returned as attribute <code>prob</code> .
use.all	controls handling of ties. If true, all distances equal to the k th largest are included. If false, a random selection of distances equal to the k th is chosen to use exactly k neighbours.

Details

knn3 is essentially the same code as [ipredknn](#) and knn3Train is a copy of [knn](#). The underlying C code from the `class` package has been modified to return the vote percentages for each class (previously the percentage for the winning class was returned).

Value

An object of class `knn3`. See [predict.knn3](#).

Author(s)

[knn](#) by W. N. Venables and B. D. Ripley and [ipredknn](#) by Torsten.Hothorn <Torsten.Hothorn@rzmail.uni-erlangen.de>, modifications by Max Kuhn and Andre Williams

Examples

```
irisFit1 <- knn3(Species ~ ., iris)

irisFit2 <- knn3(as.matrix(iris[, -5]), iris[, 5])

data(iris3)
train <- rbind(iris3[1:25,,1], iris3[1:25,,2], iris3[1:25,,3])
test <- rbind(iris3[26:50,,1], iris3[26:50,,2], iris3[26:50,,3])
cl <- factor(c(rep("s",25), rep("c",25), rep("v",25)))
knn3Train(train, test, cl, k = 5, prob = TRUE)
```

knnreg

k-Nearest Neighbour Regression

Description

k -nearest neighbour classification that can return the average value for the neighbours.

Usage

```
## Default S3 method:
knnreg(x, ...)

## S3 method for class 'formula':
knnreg(formula, data, subset, na.action, k = 5, ...)

## S3 method for class 'matrix':
knnreg(x, y, k = 5, ...)

## S3 method for class 'data.frame':
knnreg(x, y, k = 5, ...)
```

```
knnregTrain(train, test, y, k = 5, use.all=TRUE)
```

Arguments

<code>formula</code>	a formula of the form <code>lhs ~ rhs</code> where <code>lhs</code> is the response variable and <code>rhs</code> a set of predictors.
<code>data</code>	optional data frame containing the variables in the model formula.
<code>subset</code>	optional vector specifying a subset of observations to be used.
<code>na.action</code>	function which indicates what should happen when the data contain NAs.
<code>k</code>	number of neighbours considered.
<code>x</code>	a matrix or data frame of training set predictors.
<code>y</code>	a numeric vector of outcomes.
<code>...</code>	additional parameters to pass to <code>knnregTrain</code> .
<code>train</code>	matrix or data frame of training set cases.
<code>test</code>	matrix or data frame of test set cases. A vector will be interpreted as a row vector for a single case.
<code>use.all</code>	controls handling of ties. If true, all distances equal to the <code>k</code> th largest are included. If false, a random selection of distances equal to the <code>k</code> th is chosen to use exactly <code>k</code> neighbours.

Details

`knnreg` is similar to [ipredknn](#) and `knnregTrain` is a modification of [knn](#). The underlying C code from the `class` package has been modified to return average outcome.

Value

An object of class `knnreg`. See [predict.knnreg](#).

Author(s)

[knn](#) by W. N. Venables and B. D. Ripley and [ipredknn](#) by Torsten.Hothorn <Torsten.Hothorn@rzmail.uni-erlangen.de>, modifications by Max Kuhn and Chris Keefer

Examples

```
data(BloodBrain)

inTrain <- createDataPartition(logBBB, p = .8)[[1]]

trainX <- bbbDescr[inTrain,]
trainY <- logBBB[inTrain]

testX <- bbbDescr[-inTrain,]
testY <- logBBB[-inTrain]
```

```
fit <- knnreg(trainX, trainY, k = 3)
plot(testY, predict(fit, testX))
```

lattice.rfe	<i>Lattice functions for plotting resampling results of recursive feature selection</i>
-------------	---

Description

A set of lattice functions are provided to plot the resampled performance estimates (e.g. classification accuracy, RMSE) over different subset sizes.

Usage

```
## S3 method for class 'rfe':
histogram(x, data = NULL, metric = x$metric, ...)

## S3 method for class 'rfe':
densityplot(x, data = NULL, metric = x$metric, ...)

## S3 method for class 'rfe':
xyplot(x, data = NULL, metric = x$metric, ...)

## S3 method for class 'rfe':
stripplot(x, data = NULL, metric = x$metric, ...)
```

Arguments

x	An object produced by rfe
data	This argument is not used
metric	A character string specifying the single performance metric that will be plotted
...	arguments to pass to either histogram , densityplot , xyplot or stripplot

Details

By default, only the resampling results for the optimal model are saved in the `rfe` object. The function [rfeControl](#) can be used to save all the results using the `returnResamp` argument.

If leave-one-out or out-of-bag resampling was specified, plots cannot be produced (see the `method` argument of [rfeControl](#))

Value

A lattice plot object

Author(s)

Max Kuhn

See Also

[rfe](#), [rfeControl](#), [histogram](#), [densityplot](#), [xyplot](#), [stripplot](#)

Examples

```
## Not run:
library(mlbench)
n <- 100
p <- 40
sigma <- 1
set.seed(1)
sim <- mlbench.friedman1(n, sd = sigma)
x <- cbind(sim$x, matrix(rnorm(n * p), nrow = n))
y <- sim$y
colnames(x) <- paste("var", 1:ncol(x), sep = "")

normalization <- preProcess(x)
x <- predict(normalization, x)
x <- as.data.frame(x)
subsets <- c(10, 15, 20, 25)

ctrl <- rfeControl(
  functions = lmFuncs,
  method = "cv",
  verbose = FALSE,
  returnResamp = "all")

lmProfile <- rfe(x, y,
  sizes = subsets,
  rfeControl = ctrl)
xyplot(lmProfile)
stripplot(lmProfile)

histogram(lmProfile)
densityplot(lmProfile)

## End(Not run)
```

Description

Functions to create a sub-sample by maximizing the dissimilarity between new samples and the existing subset.

Usage

```
maxDissim(a, b, n = 2, obj = minDiss, randomFrac = 1, verbose = FALSE, ...)
minDiss(u)
sumDiss(u)
```

Arguments

a	a matrix or data frame of samples to start
b	a matrix or data frame of samples to sample from
n	the size of the sub-sample
obj	an objective function to measure overall dissimilarity
randomFrac	a number in (0, 1] that can be used to sub-sample from the remaining candidate values
verbose	a logical; should each step be printed?
...	optional arguments to pass to dist
u	a vector of dissimilarities

Details

Given an initial set of m samples and a larger pool of n samples, this function iteratively adds points to the smaller set by finding with of the n samples is most dissimilar to the initial set. The argument `obj` measures the overall dissimilarity between the initial set and a candidate point. For example, maximizing the minimum or the sum of the m dissimilarities are two common approaches.

This algorithm tends to select points on the edge of the data mainstream and will reliably select outliers. To select more samples towards the interior of the data set, set `randomFrac` to be small (see the examples below).

Value

a vector of integers corresponding to the rows of `b` that comprise the sub-sample.

Author(s)

Max Kuhn <max.kuhn@pfizer.com>

References

Willett, P. (1999), "Dissimilarity-Based Algorithms for Selecting Structurally Diverse Sets of Compounds," *Journal of Computational Biology*, 6, 447-457.

See Also

[dist](#)

Examples

```

example <- function(pct = 1, obj = minDiss, ...)
{
  tmp <- matrix(rnorm(200 * 2), nrow = 200)

  ## start with 15 data points
  start <- sample(1:dim(tmp)[1], 15)
  base <- tmp[start,]
  pool <- tmp[-start,]

  ## select 9 for addition
  newSamp <- maxDissim(
    base, pool,
    n = 9,
    randomFrac = pct, obj = obj, ...)

  allSamp <- c(start, newSamp)

  plot(
    tmp[-newSamp,],
    xlim = extendrange(tmp[,1]), ylim = extendrange(tmp[,2]),
    col = "darkgrey",
    xlab = "variable 1", ylab = "variable 2")
  points(base, pch = 16, cex = .7)

  for(i in seq(along = newSamp))
    points(
      pool[newSamp[i],1],
      pool[newSamp[i],2],
      pch = paste(i), col = "darkred")
}

par(mfrow=c(2,2))

set.seed(414)
example(1, minDiss)
title("No Random Sampling, Min Score")

set.seed(414)
example(.1, minDiss)
title("10 Pct Random Sampling, Min Score")

set.seed(414)
example(1, sumDiss)
title("No Random Sampling, Sum Score")

set.seed(414)
example(.1, sumDiss)
title("10 Pct Random Sampling, Sum Score")

```

 mdrrr

Multidrug Resistance Reversal (MDRR) Agent Data

Description

Svetnik et al (2003) describe these data: "Bakken and Jurs studied a set of compounds originally discussed by Klopman et al., who were interested in multidrug resistance reversal (MDRR) agents. The original response variable is a ratio measuring the ability of a compound to reverse a leukemia cell's resistance to adriamycin. However, the problem was treated as a classification problem, and compounds with the ratio >4.2 were considered active, and those with the ratio ≤ 2.0 were considered inactive. Compounds with the ratio between these two cutoffs were called moderate and removed from the data for twoclass classification, leaving a set of 528 compounds (298 actives and 230 inactives). (Various other arrangements of these data were examined by Bakken and Jurs, but we will focus on this particular one.) We did not have access to the original descriptors, but we generated a set of 342 descriptors of three different types that should be similar to the original descriptors, using the DRAGON software."

The data and R code are in the Supplemental Data file for the article.

Usage

```
data(mdrrr)
```

Value

```
mdrrDescr    the descriptors
mdrrClass    the categorical outcome ("Active" or "Inactive")
```

Source

Svetnik, V., Liaw, A., Tong, C., Culberson, J. C., Sheridan, R. P. Feuston, B. P (2003). Random Forest: A Classification and Regression Tool for Compound Classification and QSAR Modeling, *Journal of Chemical Information and Computer Sciences*, Vol. 43, pg. 1947-1958.

 nearZeroVar

Identification of near zero variance predictors

Description

This function diagnoses predictors that have one unique value (i.e. are zero variance predictors) or predictors that have both of the following characteristics: they have very few unique values relative to the number of samples and the ratio of the frequency of the most common value to the frequency of the second most common value is large.

Usage

```
nearZeroVar(x, freqCut = 95/5, uniqueCut = 10, saveMetrics = FALSE)
```

Arguments

<code>x</code>	a numeric vector or matrix, or a data frame with all numeric data
<code>freqCut</code>	the cutoff for the ratio of the most common value to the second most common value
<code>uniqueCut</code>	the cutoff for the percentage of distinct values out of the number of total samples
<code>saveMetrics</code>	a logical. If false, the positions of the zero- or near-zero predictors is returned. If true, a data frame with predictor information is returned.

Details

For example, an example of near zero variance predictor is one that, for 1000 samples, has two distinct values and 999 of them are a single value.

To be flagged, first the frequency of the most prevalent value over the second most frequent value (called the “frequency ratio”) must be above `freqCut`. Secondly, the “percent of unique values,” the number of unique values divided by the total number of samples (times 100), must also be below `uniqueCut`.

In the above example, the frequency ratio is 999 and the unique value percentage is 0.0001.

Value

If `saveMetrics = FALSE`, a vector of integers corresponding to the column positions of the problematic predictors. If `saveMetrics = TRUE`, a data frame with columns:

<code>freqRatio</code>	the ratio of frequencies for the most common value over the second most common value
<code>percentUnique</code>	the percentage of unique data points out of the total number of data points
<code>zeroVar</code>	a vector of logicals for whether the predictor has only one distinct value
<code>nzv</code>	a vector of logicals for whether the predictor is a near zero variance predictor

Author(s)

Max Kuhn, speed improvements by Allan Engelhardt

Examples

```
nearZeroVar(iris[, -5], saveMetrics = TRUE)

data(BloodBrain)
nearZeroVar(bbbDescr)
```

`normalize.AffyBatch.normalize2Reference`*Quantile Normalization to a Reference Distribution*

Description

Quantile normalization based upon a reference distribution. This function normalizes a matrix of data (typically Affy probe level intensities).

Usage

```
normalize.AffyBatch.normalize2Reference(  
  abatch,  
  type = c("separate", "pmonly", "mmonly", "together"),  
  ref = NULL)
```

Arguments

<code>abatch</code>	An {AffyBatch}
<code>type</code>	A string specifying how the normalization should be applied. See details for more.
<code>ref</code>	A vector of reference values. See details for more.

Details

This method is based upon the concept of a quantile-quantile plot extended to n dimensions. No special allowances are made for outliers. If you make use of quantile normalization either through `rma` or `expresso` please cite Bolstad et al, Bioinformatics (2003).

The `type` argument should be one of "separate", "pmonly", "mmonly", "together" which indicates whether to normalize only one probe type (PM,MM) or both together or separately.

The function uses the data supplied in `ref` to use as the reference distribution. In other words, the PMs in `abatch` will be normalized to have the same distribution as the data in `ref`. If `ref` is `NULL`, the normalizing takes place using the average quantiles of the PM values in `abatch` (just as in `normalize.AffyBatch.quantile`).

Value

A normalized `AffyBatch`.

Author(s)

Max Kuhn, adapted from Ben Bolstad, <bolstad@stat.berkeley.edu>

References

Bolstad, B (2001) *Probe Level Quantile Normalization of High Density Oligonucleotide Array Data*. Unpublished manuscript

Bolstad, B. M., Irizarry R. A., Astrand, M, and Speed, T. P. (2003) *A Comparison of Normalization Methods for High Density Oligonucleotide Array Data Based on Bias and Variance*. *Bioinformatics* 19(2) ,pp 185-193.

See Also

normalize

Examples

```
# first, let affy/expresso know that the method exists
# normalize.AffyBatch.methods <- c(normalize.AffyBatch.methods, "normalize2Reference")

# example not run, as it would take a while
# RawData <- ReadAffy(celfile.path=FilePath)

# Batch1Step1 <- bg.correct(RawData, "rma")
# Batch1Step2 <- normalize.AffyBatch.quantiles(Batch1Step1)
# referencePM <- pm(Batch1Step2)[,1]
# Batch1Step3 <- computeExprSet(Batch1Step2, "pmonly", "medianpolish")

# Batch2Step1 <- bg.correct(RawData2, "rma")
# Batch2Step2 <- normalize.AffyBatch.normalize2Reference(Batch2Step1, ref = referencePM)
# Batch2Step3 <- computeExprSet(Batch2Step2, "pmonly", "medianpolish")
```

normalize2Reference

Quantile Normalize Columns of a Matrix Based on a Reference Distribution

Description

Normalize the columns of a matrix to have the same quantiles, allowing for missing values. Users do not normally need to call this function directly - use `normalize.AffyBatch.normalize2Reference` instead.

Usage

```
normalize2Reference(data, refData = NULL, ties = TRUE)
```

Arguments

<code>data</code>	numeric matrix. Missing values are allowed.
<code>refData</code>	A vector of reference values.
<code>ties</code>	logical. If TRUE, ties in each column of A are treated in careful way. Tied values will be normalized to the mean of the corresponding pooled quantiles.

Details

This function is intended to normalize single channel or A-value microarray intensities between arrays. Each quantile of each column is set to either: quantiles of the reference distribution (`refData` supplied) or the mean of that quantile across arrays (`refData` is `NULL`). The intention is to make all the normalized columns have the same empirical distribution. This will be exactly true if there are no missing values and no ties within the columns: the normalized columns are then simply permutations of one another.

If there are ties amongst the intensities for a particular array, then with `ties=FALSE` the ties are broken in an unpredictable order. If `ties=TRUE`, all the tied values for that array will be normalized to the same value, the average of the quantiles for the tied values.

Value

A matrix of the same dimensions as `A` containing the normalized values.

Author(s)

Max Kuhn, adapted from Gordon Smyth

References

Bolstad, B. M., Irizarry R. A., Astrand, M., and Speed, T. P. (2003), A comparison of normalization methods for high density oligonucleotide array data based on bias and variance. *Bioinformatics* **19**, 185-193.

See Also

`normalize.AffyBatch.normalize2Reference`

oil

Fatty acid composition of commercial oils

Description

Fatty acid concentrations of commercial oils were measured using gas chromatography. The data is used to predict the type of oil. Note that only the known oils are in the data set. Also, the authors state that there are 95 samples of known oils. However, we count 96 in Table 1 (pgs. 33-35).

Usage

```
data(oil)
```

Value

fattyAcids	data frame of fatty acid compositions: Palmitic, Stearic, Oleic, Linoleic, Linolenic, Eicosanoic and Eicosenoic. When values fell below the lower limit of the assay (denoted as <X in the paper), the limit was used.
oilType	factor of oil types: pumpkin (A), sunflower (B), peanut (C), olive (D), soybean (E), rapeseed (F) and corn (G).

Source

Brodnjak-Voncina et al. (2005). Multivariate data analysis in classification of vegetable oils characterized by the content of fatty acids, *Chemometrics and Intelligent Laboratory Systems*, Vol. 75:31-45.

 oneSE

Selecting tuning Parameters

Description

Various functions for setting tuning parameters

Usage

```
best(x, metric, maximize)
oneSE(x, metric, num, maximize)
tolerance(x, metric, tol = 1.5, maximize)
```

Arguments

x	a data frame of tuning parameters and model results, sorted from least complex models to the most complex
metric	a string that specifies what summary metric will be used to select the optimal model. By default, possible values are "RMSE" and "Rsquared" for regression and "Accuracy" and "Kappa" for classification. If custom performance metrics are used (via the <code>summaryFunction</code> argument in <code>trainControl</code>), the value of <code>metric</code> should match one of the arguments. If it does not, a warning is issued and the first metric given by the <code>summaryFunction</code> is used.
maximize	a logical: should the metric be maximized or minimized?
num	the number of resamples (for <code>oneSE</code> only)
tol	the acceptable percent tolerance (for <code>tolerance</code> only)

Details

These functions can be used by `train` to select the "optimal" model form a series of models. Each requires the user to select a metric that will be used to judge performance. For regression models, values of "RMSE" and "Rsquared" are applicable. Classification models use either "Accuracy" or "Kappa" (for unbalanced class distributions).

By default, `train` uses `best`.

`best` simply chooses the tuning parameter associated with the largest (or lowest for "RMSE") performance.

`oneSE` is a rule in the spirit of the "one standard error" rule of Breiman et al (1984), who suggest that the tuning parameter associated with the best performance may over fit. They suggest that the simplest model within one standard error of the empirically optimal model is the better choice. This assumes that the models can be easily ordered from simplest to most complex (see the Details section below).

`tolerance` takes the simplest model that is within a percent tolerance of the empirically optimal model. For example, if the largest Kappa value is 0.5 and a simpler model within 3 percent is acceptable, we score the other models using $(x - 0.5) / 0.5 * 100$. The simplest model whose score is not less than 3 is chosen (in this case, a model with a Kappa value of 0.35 is acceptable).

User-defined functions can also be used. The argument `selectionFunction` in `trainControl` can be used to pass the function directly or to pass the function by name.

Value

an row index

Note

In many cases, it is not very clear how to order the models on simplicity. For simple trees and other models (such as PLS), this is straightforward. However, for others it is not.

For example, many of the boosting models used by `caret` have parameters for the number of boosting iterations and the tree complexity (others may also have a learning rate parameter). In this implementation, we order models on number of iterations, then tree depth. Clearly, this is arguable (please email the author for suggestions though).

For MARS models, they are ordered on the degree of the features, then the number of retained terms.

RBF SVM models are ordered first by the cost parameter, then by the kernel parameter while polynomial models are ordered first on polynomial degree, then cost and scale.

Neural networks are ordered by the number of hidden units and then the amount of weight decay.

k -nearest neighbor models are ordered from most neighbors to least (i.e. smoothest to model jagged decision boundaries).

Elastic net models are ordered first on the L1 penalty, then by the L2 penalty.

Author(s)

Max Kuhn

References

Breiman, Friedman, Olshen, and Stone. (1984) *Classification and Regression Trees*. Wadsworth.

See Also

[train](#), [trainControl](#)

Examples

```
# simulate a PLS regression model
test <- data.frame(
  ncomp = 1:5,
  RMSE = c(3, 1.1, 1.02, 1, 2),
  RMSESD = .4)

best(test, "RMSE", maximize = FALSE)
oneSE(test, "RMSE", maximize = FALSE, num = 10)
tolerance(test, "RMSE", tol = 3, maximize = FALSE)

### usage example

data(BloodBrain)

marsGrid <- data.frame(
  .degree = 1,
  .nprune = (1:10) * 3)

set.seed(1)
marsFit <- train(
  bbbDescr, logBBB,
  "earth",
  tuneGrid = marsGrid,
  trControl = trainControl(
    method = "cv",
    number = 10,
    selectionFunction = "tolerance"))

# around 18 terms should yield the smallest CV RMSE
```

panel.needle

Needle Plot Lattice Panel

Description

A variation of `panel.dotplot` that plots horizontal lines from zero to the data point.

Usage

```
panel.needle(x, y, horizontal = TRUE,
             pch, col, lty, lwd,
             col.line, levels.fos,
             groups = NULL,
             ...)
```

Arguments

<code>x, y</code>	variables to be plotted in the panel. Typically <code>y</code> is the 'factor'
<code>horizontal</code>	logical. If <code>FALSE</code> , the plot is 'transposed' in the sense that the behaviours of <code>x</code> and <code>y</code> are switched. <code>x</code> is now the 'factor'. Interpretation of other arguments change accordingly. See documentation of <code>bwplot</code> for a fuller explanation.
<code>pch, col, lty, lwd, col.line</code>	graphical parameters
<code>levels.fos</code>	locations where reference lines will be drawn
<code>groups</code>	grouping variable (affects graphical parameters)
<code>...</code>	extra parameters, passed to <code>panel.xyplot</code> which is responsible for drawing the foreground points (<code>panel.dotplot</code> only draws the background reference lines).

Details

Creates (possibly grouped) needleplot of `x` against `y` or vice versa

Author(s)

Max Kuhn, based on `panel.dotplot` by Deepayan Sarkar

See Also

[dotplot](#)

pcaNNet.default *Neural Networks with a Principal Component Step*

Description

Run PCA on a dataset, then use it in a neural network model

Usage

```
## Default S3 method:
pcaNNet(x, y, thresh = 0.99, ...)
## S3 method for class 'formula':
pcaNNet(formula, data, weights, ...,
         thresh = .99, subset, na.action, contrasts = NULL)

## S3 method for class 'pcaNNet':
predict(object, newdata, type = c("raw", "class"), ...)
```

Arguments

<code>formula</code>	A formula of the form <code>class ~ x1 + x2 + ...</code>
<code>x</code>	matrix or data frame of <code>x</code> values for examples.
<code>y</code>	matrix or data frame of target values for examples.
<code>weights</code>	(case) weights for each example – if missing defaults to 1.
<code>thresh</code>	a threshold for the cumulative proportion of variance to capture from the PCA analysis. For example, to retain enough PCA components to capture 95 percent of variation, set <code>thresh = .95</code>
<code>data</code>	Data frame from which variables specified in <code>formula</code> are preferentially to be taken.
<code>subset</code>	An index vector specifying the cases to be used in the training sample. (NOTE: If given, this argument must be named.)
<code>na.action</code>	A function to specify the action to be taken if NAs are found. The default action is for the procedure to fail. An alternative is <code>na.omit</code> , which leads to rejection of cases with missing values on any required variable. (NOTE: If given, this argument must be named.)
<code>contrasts</code>	a list of contrasts to be used for some or all of the factors appearing as variables in the model formula.
<code>object</code>	an object of class <code>nnet</code> as returned by <code>nnet</code> .
<code>newdata</code>	matrix or data frame of test examples. A vector is considered to be a row vector comprising a single case.
<code>type</code>	Type of output
<code>...</code>	arguments passed to <code>nnet</code>

Details

The function first will run principal component analysis on the data. The cumulative percentage of variance is computed for each principal component. The function uses the `thresh` argument to determine how many components must be retained to capture this amount of variance in the predictors.

The principal components are then used in a neural network model.

When predicting samples, the new data are similarly transformed using the information from the PCA analysis on the training data and then predicted.

Because the variance of each predictor is used in the PCA analysis, the code does a quick check to make sure that each predictor has at least two distinct values. If a predictor has one unique value, it is removed prior to the analysis.

Value

For `pcaNNet`, an object of `"pcaNNet"` or `"pcaNNet.formula"`. Items of interest in the output are:

<code>pc</code>	the output from <code>preProcess</code>
<code>model</code>	the model generated from <code>nnet</code>
<code>names</code>	if any predictors had only one distinct value, this is a character string of the remaining columns. Otherwise a value of <code>NULL</code>

Author(s)

These are heavily based on the `nnet` code from Brian Ripley.

References

Ripley, B. D. (1996) *Pattern Recognition and Neural Networks*. Cambridge.

See Also

`nnet`, `preProcess`

Examples

```
data(BloodBrain)
modelFit <- pcaNNet(bbbDescr, logBBB, size = 5, linout = TRUE, trace = FALSE)
modelFit

predict(modelFit, bbbDescr)
```

`plot.train`

Plot Method for the train Class

Description

This function takes the output of a `train` object and creates a line or level plot using the `lattice` library.

Usage

```
## S3 method for class 'train':
plot(x,
      plotType = "scatter",
      metric = x$perfNames[1],
      digits = getOption("digits") - 5,
      xTrans = NULL,
      ...)
```

Arguments

<code>x</code>	an object of class <code>train</code> .
<code>metric</code>	What measure of performance to plot. Possible values are "RMSE", "Rsquared", "Accuracy" or "Kappa"
<code>plotType</code>	a string describing the type of plot ("scatter", "level" or "line")
<code>digits</code>	an integer specifying the number of significant digits used to label the parameter value.
<code>xTrans</code>	a function that will be used to scale the x-axis in scatter plots.
<code>...</code>	specifications to be passed to <code>levelplot</code> or <code>bwplot</code> (for line plots). These values should not be axis labels, panel functions, or titles.

Details

If there are no tuning parameters, or none were varied, a plot of the resampling distribution is produced via `resampleHist`.

If the model has one tuning parameter with multiple candidate values, a plot is produced showing the profile of the results over the parameter. Also, a plot can be produced if there are multiple tuning parameters but only one is varied.

If there are two tuning parameters with different values, a plot can be produced where a different line is shown for each value of the other parameter. For three parameters, the same line plot is created within conditioning panels of the other parameter.

Also, with two tuning parameters (with different values), a `levelplot` (i.e. un-clustered heatmap) can be created. For more than two parameters, this plot is created inside conditioning panels.

Author(s)

Max Kuhn

References

Kuhn (2008), "Building Predictive Models in R Using the caret" (<http://www.jstatsoft.org/v28/i05/>)

See Also

`train`

Examples

```

data(iris)
TrainData <- iris[,1:4]
TrainClasses <- iris[,5]

library(e1071)
rpartFit <- train(TrainData, TrainClasses, "rpart",
                 tuneLength=15)
plot(rpartFit, scales = list(x = list(rot = 90)))

library(klaR)
rdaFit <- train(TrainData, TrainClasses, "rda",
               control = trainControl(method = "cv"))
plot(rdaFit, plotType = "line", auto.key = TRUE)

```

plot.varImp.train *Plotting variable importance measures*

Description

This function produces lattice plots of objects with class "varImp.train". More info will be forthcoming.

Usage

```

## S3 method for class 'varImp.train':
plot(x, top = dim(x$importance)[1], ...)

```

Arguments

x	an object with class varImp.
top	a scalar numeric that specifies the number of variables to be displayed (in order of importance)
...	arguments to pass to the lattice plot function (dotplot and panel.needle)

Details

For models where there is only one importance value, such as regression models, a "Pareto-type" plot is produced where the variables are ranked by their importance and a needle-plot is used to show the top variables.

When there is more than one importance value per predictor, the same plot is produced within conditioning panels for each class. The top predictors are sorted by their average importance.

Value

a lattice plot object

Author(s)

Max Kuhn

`plotClassProbs`*Plot Predicted Probabilities in Classification Models*

Description

This function takes an object (preferably from the function `extractProb`) and creates a lattice plot.

If the call to `extractProb` included test data, these data are shown, but if unknowns were also included, these are not plotted

Usage

```
plotClassProbs(object, ...)
```

Arguments

<code>object</code>	an object (preferably from the function <code>extractProb</code> . There should be columns for each level of the class factor and columns named <code>obs</code> , <code>pred</code> , <code>model</code> (e.g. "rpart", "nnet" etc) and <code>dataType</code> (e.g. "Training", "Test" etc)
<code>...</code>	parameters to pass to <code>histogram</code>

Value

A lattice object. Note that the plot has to be printed to be displayed (especially in a loop).

Author(s)

Max Kuhn

Examples

```
data(iris)
set.seed(90)
inTrain <- sample(1:dim(iris)[1], 100)

trainData <- iris[inTrain,]
testData <- iris[-inTrain,]

rpartFit <- train(trainData[, -5], trainData[, 5], "rpart",
                 tuneLength = 15)
ldaFit <- train(trainData[, -5], trainData[, 5], "lda")
predProbs <- extractProb(list(ldaFit, rpartFit),
                        testX = testData[, -5],
                        testY = testData[, 5])
```

```
plotClassProbs(predProbs)
plotClassProbs(predProbs[predProbs$model == "lda",])
plotClassProbs(predProbs[predProbs$model == "lda" &
                          predProbs$dataType == "Test",])
```

plotObsVsPred	<i>Plot Observed versus Predicted Results in Regression and Classification Models</i>
---------------	---

Description

This function takes an object (preferably from the function `extractPrediction`) and creates a lattice plot. For numeric outcomes, the observed and predicted data are plotted with a 45 degree reference line and a smoothed fit. For factor outcomes, a dotplot plot is produced with the accuracies for the different models.

If the call to `extractPrediction` included test data, these data are shown, but if unknowns were also included, they are not plotted

Usage

```
plotObsVsPred(object, equalRanges = TRUE, ...)
```

Arguments

object	an object (preferably from the function <code>extractPrediction</code> . There should be columns named <code>obs</code> , <code>pred</code> , <code>model</code> (e.g. "rpart", "nnet" etc) and <code>dataType</code> (e.g. "Training", "Test" etc)
equalRanges	a logical; should the x- and y-axis ranges be the same?
...	parameters to pass to <code>xyplot</code> or <code>dotplot</code> , such as <code>auto.key</code>

Value

A lattice object. Note that the plot has to be printed to be displayed (especially in a loop).

Author(s)

Max Kuhn

Examples

```
## Not run:
# regression example
data(BostonHousing)
rpartFit <- train(BostonHousing[1:100, -c(4, 14)],
                  BostonHousing$medv[1:100],
                  "rpart", tuneLength = 9)
```

```
plsFit <- train(BostonHousing[1:100, -c(4, 14)],
               BostonHousing$medv[1:100],
               "pls")

predVals <- extractPrediction(list(rpartFit, plsFit),
                              testX = BostonHousing[101:200, -c(4, 14)],
                              testY = BostonHousing$medv[101:200],
                              unkX = BostonHousing[201:300, -c(4, 14)])

plotObsVsPred(predVals)

#classification example
data(Satellite)
numSamples <- dim(Satellite)[1]
set.seed(716)

varIndex <- 1:numSamples

trainSamples <- sample(varIndex, 150)

varIndex <- (1:numSamples)[-trainSamples]
testSamples <- sample(varIndex, 100)

varIndex <- (1:numSamples)[-c(testSamples, trainSamples)]
unkSamples <- sample(varIndex, 50)

trainX <- Satellite[trainSamples, -37]
trainY <- Satellite[trainSamples, 37]

testX <- Satellite[testSamples, -37]
testY <- Satellite[testSamples, 37]

unkX <- Satellite[unkSamples, -37]

knnFit <- train(trainX, trainY, "knn")
rpartFit <- train(trainX, trainY, "rpart")

predTargets <- extractPrediction(list(knnFit, rpartFit),
                                 testX = testX,
                                 testY = testY,
                                 unkX = unkX)

plotObsVsPred(predTargets)

## End(Not run)
```

Description

`plsda` is used to fit standard PLS models for classification while `splsda` performs sparse PLS that embeds feature selection and regularization for the same purpose.

Usage

```
plsda(x, ...)

## Default S3 method:
plsda(x, y, ncomp = 2, probMethod = "softmax", prior = NULL, ...)

## S3 method for class 'plsda':
predict(object, newdata = NULL, ncomp = NULL, type = "class", ...)

splsda(x, ...)

## Default S3 method:
splsda(x, y, probMethod = "softmax", prior = NULL, ...)

## S3 method for class 'splsda':
predict(object, newdata = NULL, type = "class", ...)
```

Arguments

<code>x</code>	a matrix or data frame of predictors
<code>y</code>	a factor or indicator matrix for the discrete outcome. If a matrix, the entries must be either 0 or 1 and rows must sum to one
<code>ncomp</code>	the number of components to include in the model. Predictions can be made for models with values less than <code>ncomp</code> .
<code>probMethod</code>	either "softmax" or "Bayes" (see Details)
<code>prior</code>	a vector or prior probabilities for the classes (only used for <code>probMethod = "Bayes"</code>)
<code>...</code>	arguments to pass to <code>plsr</code> or <code>spls</code> . For <code>splsda</code> , this is the method for passing tuning parameters specifications (e.g. <code>K</code> , <code>eta</code> or <code>kappa</code>)
<code>object</code>	an object produced by <code>plsda</code>
<code>newdata</code>	a matrix or data frame of predictors
<code>type</code>	either "class", "prob" or "raw" to produce the predicted class, class probabilities or the raw model scores, respectively.

Details

If a factor is supplied, the appropriate indicator matrix is created.

A multivariate PLS model is fit to the indicator matrix using the `plsr` or `spls` function.

Two prediction methods can be used.

The **softmax function** transforms the model predictions to "probability-like" values (e.g. on [0, 1] and sum to 1). The class with the largest class probability is the predicted class.

Also, **Bayes rule** can be applied to the model predictions to form posterior probabilities. Here, the model predictions for the training set are used along with the training set outcomes to create conditional distributions for each class. When new samples are predicted, the raw model predictions are run through these conditional distributions to produce a posterior probability for each class (along with the prior). This process is repeated `ncomp` times for every possible PLS model. The `NaiveBayes` function is used with `usekernel = TRUE` for the posterior probability calculations.

Value

For `plsda`, an object of class "plsda" and "mvr". For `splsda`, an object of class `splsda`.

The predict methods produce either a vector, matrix or three-dimensional array, depending on the values of `type` of `ncomp`. For example, specifying more than one value of `ncomp` with `type = "class"` will produce a three dimensional array but the default specification would produce a factor vector.

See Also

`pls`, `spls`

Examples

```
data(mdr)
set.seed(1)
inTrain <- sample(seq(along = mdrClass), 450)

nzv <- nearZeroVar(mdrDescr)
filteredDescr <- mdrDescr[, -nzv]

training <- filteredDescr[inTrain,]
test <- filteredDescr[-inTrain,]
trainMDRR <- mdrClass[inTrain]
testMDRR <- mdrClass[-inTrain]

preProcValues <- preProcess(training)

trainDescr <- predict(preProcValues, training)
testDescr <- predict(preProcValues, test)

useBayes <- plsda(trainDescr, trainMDRR, ncomp = 5,
                  probMethod = "Bayes")
useSoftmax <- plsda(trainDescr, trainMDRR, ncomp = 5)

confusionMatrix(
  predict(useBayes, testDescr),
  testMDRR)

confusionMatrix(
  predict(useSoftmax, testDescr),
  testMDRR)
```

```

histogram(
  ~predict(useBayes, testDescr, type = "prob")[, "Active"]
  | testMDRR, xlab = "Active Prob", xlim = c(-.1, 1.1))
histogram(
  ~predict(useSoftmax, testDescr, type = "prob")[, "Active", ]
  | testMDRR, xlab = "Active Prob", xlim = c(-.1, 1.1))

## different sized objects are returned
length(predict(useBayes, testDescr))
dim(predict(useBayes, testDescr, ncomp = 1:3))
dim(predict(useBayes, testDescr, type = "prob"))
dim(predict(useBayes, testDescr, type = "prob", ncomp = 1:3))

## using spls

splsFit <- splsda(trainDescr, trainMDRR,
                 K = 5, eta = .9,
                 probMethod = "Bayes")

confusionMatrix(
  predict(splsFit, testDescr),
  testMDRR)

```

postResample

Calculates performance across resamples

Description

Given two numeric vectors of data, the mean squared error and R-squared are calculated. For two factors, the overall agreement rate and Kappa are determined.

Usage

```

postResample(pred, obs)
defaultSummary(data, lev = NULL, model = NULL)

```

Arguments

pred	A vector of numeric data (could be a factor)
obs	A vector of numeric data (could be a factor)
data	a data frame or matrix with columns obs and pred for the observed and predicted outcomes
lev	a character vector of factors levels for the response. In regression cases, this would be NULL.
model	a character string for the model name (as taken from the method argument of train).

Details

`postResample` is meant to be used with `apply` across a matrix. For numeric data the code checks to see if the standard deviation of either vector is zero. If so, the correlation between those samples is assigned a value of zero. NA values are ignored everywhere.

Note that many models have more predictors (or parameters) than data points, so the typical mean squared error denominator ($n - p$) does not apply. Root mean squared error is calculated using `sqrt(mean((pred - obs)^2))`. Also, R-squared is calculated as the square of the correlation between the observed and predicted outcomes.

For `defaultSummary` is the default function to compute performance metrics in `train`. It is a wrapper around `postResample`.

Other functions can be used via the `summaryFunction` argument of `trainControl`. Custom functions must have the same arguments as `defaultSummary`.

Value

A vector of performance estimates.

Author(s)

Max Kuhn

See Also

`trainControl`

Examples

```
predicted <- matrix(rnorm(50), ncol = 5)
observed <- rnorm(10)
apply(predicted, 2, postResample, obs = observed)
```

pottery

Pottery from Pre-Classical Sites in Italy

Description

Measurements of 58 pottery samples.

Usage

```
data(pottery)
```

Value

`pottery` 11 elemental composition measurements
`potteryClass` factor of pottery type: black carbon containing bulks (A) and clayey (B)

Source

R. G. Brereton (2003). *Chemometrics: Data Analysis for the Laboratory and Chemical Plant*, pg. 261.

predict.bagEarth *Predicted values based on bagged Earth and FDA models*

Description

Predicted values based on bagged Earth and FDA models

Usage

```
predict.bagEarth(object, newdata = NULL, type = "response", ...)
predict.bagFDA(object, newdata = NULL, type = "class", ...)
```

Arguments

object	Object of class inheriting from bagEarth
newdata	An optional data frame or matrix in which to look for variables with which to predict. If omitted, the fitted values are used (see note below).
type	The type of prediction. For bagged <code>earth</code> regression model, <code>type = "response"</code> will produce a numeric vector of the usual model predictions. <code>earth</code> also allows the user to fit generalized linear models. In this case, <code>type = "response"</code> produces the inverse link results as a vector. In the case of a binomial generalized linear model, <code>type = "response"</code> produces a vector of probabilities, <code>type = "class"</code> generates a factor vector and <code>type = "prob"</code> produces a 2 column matrix with probabilities for both classes (averaged across the individual models). Similarly, for bagged <code>fda</code> models, <code>type = "class"</code> generates a factor vector and <code>type = "probs"</code> outputs a matrix of class probabilities.
...	not used

Value

a vector of predictions

Note

If the predictions for the original training set are needed, there are two ways to calculate them. First, the original data set can be predicted by each bagged earth model. Secondly, the predictions from each bootstrap sample could be used (but are more likely to overfit). If the original call to `bagEarth` or `bagFDA` had `keepX = TRUE`, the first method is used, otherwise the values are calculated via the second method.

Author(s)

Max Kuhn

See Also[bagEarth](#)**Examples**

```
data(trees)
fit1 <- bagEarth(Volume ~ ., data = trees, keepX = TRUE)
fit2 <- bagEarth(Volume ~ ., data = trees, keepX = FALSE)
hist(predict(fit1) - predict(fit2))
```

`predict.knn3`*Predictions from k-Nearest Neighbors*

Description

Predict the class of a new observation based on k-NN.

Usage

```
## S3 method for class 'knn3':
predict(object, newdata, type=c("prob", "class"), ...)
```

Arguments

<code>object</code>	object of class <code>knn3</code> .
<code>newdata</code>	a data frame of new observations.
<code>type</code>	return either the predicted class or the the proportion of the votes for the winning class.
<code>...</code>	additional arguments.

Details

This function is a method for the generic function [predict](#) for class `knn3`. For the details see [knn3](#). This is essentially a copy of [predict.ipredknn](#).

Value

Either the predicted class or the the proportion of the votes for each class.

Author(s)

[predict.ipredknn](#) by Torsten.Hothorn <Torsten.Hothorn@rzmail.uni-erlangen.de>

predict.knnreg *Predictions from k-Nearest Neighbors Regression Model*

Description

Predict the outcome of a new observation based on k-NN.

Usage

```
## S3 method for class 'knnreg':  
predict(object, newdata, ...)
```

Arguments

object	object of class knnreg.
newdata	a data frame or matrix of new observations.
...	additional arguments.

Details

This function is a method for the generic function [predict](#) for class knnreg. For the details see [knnreg](#). This is essentially a copy of [predict.ipredknn](#).

Value

a numeric vector

Author(s)

Max Kuhn, Chris Keefer, adapted from [knn](#) and [predict.ipredknn](#)

predict.train *Extract predictions and class probabilities from train objects*

Description

These functions can be used for a single `train` object or to loop through a number of `train` objects to calculate the training and test data predictions and class probabilities.

Usage

```
## S3 method for class 'list':
predict(object, ...)

## S3 method for class 'train':
predict(object, newdata = NULL, type = "raw", ...)

extractPrediction(models,
                  testX = NULL, testY = NULL,
                  unkX = NULL,
                  unkOnly = !is.null(unkX) & is.null(testX),
                  verbose = FALSE)

extractProb(models,
            testX = NULL, testY = NULL,
            unkX = NULL,
            unkOnly = !is.null(unkX) & is.null(testX),
            verbose = FALSE)
```

Arguments

object	For <code>predict.train</code> , an object of class <code>train</code> . For <code>predict.list</code> , a list of objects of class <code>train</code> .
newdata	an optional set of data to predict on. If <code>NULL</code> , then the original training data are used
type	either "raw" or "prob", for the number/class predictions or class probabilities, respectively. Class probabilities are not available for all classification models
models	a list of objects of the class <code>train</code> . The objects must have been generated with <code>fitBest = FALSE</code> and <code>returnData = TRUE</code> .
testX	an optional set of data to predict
testY	an optional outcome corresponding to the data given in <code>testX</code>
unkX	another optional set of data to predict without known outcomes
unkOnly	a logical to bypass training and test set predictions. This is useful if speed is needed for unknown samples.
verbose	a logical for printing messages
...	additional arguments to be passed to other methods

Details

These functions are wrappers for the specific prediction functions in each modeling package. In each case, the optimal tuning values given in the `tuneValue` slot of the `finalModel` object are used to predict.

To get simple predictions for a new data set, the `predict` function can be used.

To get predictions for a series of models at once, a list of `train` objects can be passed to the `predict` function and a list of model predictions will be returned.

The two extraction functions can be used to get the predictions and observed outcomes at once for the training, test and/or unknown samples at once in a single data frame (instead of a list of just the predictions). These objects can then be passed to `plotObsVsPred` or `plotClassProbs`.

Value

For `predict.train`, a vector of predictions if `type = "raw"` or a data frame of class probabilities for `type = "probs"`. In the latter case, there are columns for each class.

For `predict.list`, a list results. Each element is produced by `predict.train`.

For `extractPrediction`, a data frame with columns:

<code>obs</code>	the observed training and test data
<code>pred</code>	predicted values
<code>model</code>	the type of model used to predict
<code>dataType</code>	"Training", "Test" or "Unknown" depending on what was specified

For `extractProb`, a data frame. There is a column for each class containing the probabilities. The remaining columns are the same as above (although the `pred` column is the predicted class)

Author(s)

Max Kuhn

References

Kuhn (2008), "Building Predictive Models in R Using the caret" (<http://www.jstatsoft.org/v28/i05/>)

See Also

`plotObsVsPred`, `plotClassProbs`

Examples

```
## Not run:
library(mlbench)
data(Satellite)
numSamples <- dim(Satellite)[1]
set.seed(716)

varIndex <- 1:numSamples

trainSamples <- sample(varIndex, 150)

varIndex <- (1:numSamples)[-trainSamples]
testSamples <- sample(varIndex, 100)

varIndex <- (1:numSamples)[-c(testSamples, trainSamples)]
```

```

unkSamples <- sample(varIndex, 50)

trainX <- Satellite[trainSamples, -37]
trainY <- Satellite[trainSamples, 37]

testX <- Satellite[testSamples, -37]
testY <- Satellite[testSamples, 37]

unkX <- Satellite[unkSamples, -37]

knnFit <- train(trainX, trainY, "knn")
rpartFit <- train(trainX, trainY, "rpart")

predict(knnFit)
predict(knnFit, newdata = testX)
predict(knnFit, type = "prob")

bothModels <- list(
  knn = knnFit,
  tree = rpartFit)

predict(bothModels)

predTargets <- extractPrediction(
  bothModels,
  testX = testX,
  testY = testY,
  unkX = unkX)

## End(Not run)

```

predictors

List predictors used in the model

Description

This class uses a model fit to determine which predictors were used in the final model.

Usage

```

predictors(x, ...)

## S3 method for class 'train':
predictors(x, ...)

## S3 method for class 'terms':
predictors(x, ...)

```

```
## S3 method for class 'formula':
predictors(x, ...)

## S3 method for class 'list':
predictors(x, ...)

## S3 method for class 'mvr':
predictors(x, ...)

## S3 method for class 'gbm':
predictors(x, ...)

## S3 method for class 'Weka\_classifier':
predictors(x, ...)

## S3 method for class 'fda':
predictors(x, ...)

## S3 method for class 'earth':
predictors(x, ...)

## S3 method for class 'gausspr':
predictors(x, ...)

## S3 method for class 'ksvm':
predictors(x, ...)

## S3 method for class 'lssvm':
predictors(x, ...)

## S3 method for class 'rvm':
predictors(x, ...)

## S3 method for class 'train':
predictors(x, ...)

## S3 method for class 'gpls':
predictors(x, ...)

## S3 method for class 'knn3':
predictors(x, ...)

## S3 method for class 'LogitBoost':
predictors(x, ...)

## S3 method for class 'lda':
predictors(x, ...)
```

```
## S3 method for class 'rda':
predictors(x, ...)

## S3 method for class 'multinom':
predictors(x, ...)

## S3 method for class 'nnet':
predictors(x, ...)

## S3 method for class 'pcaNNet':
predictors(x, ...)

## S3 method for class 'NaiveBayes':
predictors(x, ...)

## S3 method for class 'randomForest':
predictors(x, ...)

## S3 method for class 'pamrtrained':
predictors(x, newdata = NULL, threshold = NULL, ...)

## S3 method for class 'superpc':
predictors(x, newdata = NULL, threshold = NULL, n.components = NULL, ...)

## S3 method for class 'slda':
predictors(x, ...)

## S3 method for class 'rpart':
predictors(x, surrogate = TRUE, ...)

## S3 method for class 'regbagg':
predictors(x, surrogate = TRUE, ...)

## S3 method for class 'classbagg':
predictors(x, surrogate = TRUE, ...)

## S3 method for class 'glmboost':
predictors(x, ...)

## S3 method for class 'gamboost':
predictors(x, ...)

## S3 method for class 'blackboost':
predictors(x, ...)

## S3 method for class 'BinaryTree':
predictors(x, surrogate = TRUE, ...)
```

```
## S3 method for class 'RandomForest':
predictors(x, surrogate = TRUE, ...)

## S3 method for class 'bagEarth':
predictors(x, ...)

## S3 method for class 'bagFDA':
predictors(x, ...)

## S3 method for class 'ppr':
predictors(x, ...)

## S3 method for class 'rfe':
predictors(x, ...)
```

Arguments

<code>x</code>	a model object, list or terms
<code>newdata</code>	for <code>pamr.train</code> and <code>superpc.train</code> : the training data
<code>threshold</code>	for <code>pamr.train</code> and <code>superpc.train</code> : the feature selection threshold
<code>n.components</code>	for <code>superpc.train</code> : the number of PCA components used
<code>surrogate</code>	a logical for <code>rpart</code> , <code>ipredbagg</code> , <code>ctree</code> and <code>cforest</code> : should variables used as surrogate splits also be returned?
<code>...</code>	not currently used

Details

For `randomForest`, `cforest`, `ctree`, `rpart`, `ipredbagg`, `bagging`, `earth`, `fda`, `pamr.train`, `superpc.train`, `bagEarth` and `bagFDA`, an attempt was made to report the predictors that were actually used in the final model.

In cases where the predictors cannot be determined, NA is returned. For example, `nnet` may return missing values from `predictors`.

Value

a character string of predictors or NA.

```
preProcess
```

```
Pre-Processing of Predictors
```

Description

Pre-processing transformation (centering, scaling etc) can be estimated from the training data and applied to any data set with the same variables.

Usage

```
preProcess(x, ...)

## Default S3 method:
preProcess(x, method = c("center", "scale"),
           thresh = 0.95, na.remove = TRUE, ...)

## S3 method for class 'preProcess':
predict(object, newdata, ...)
```

Arguments

<code>x</code>	a matrix or data frame
<code>method</code>	a character vector specifying the type of processing. Possible values are "center", "scale", "pca" and "spatialSign"
<code>thresh</code>	a cutoff for the cumulative percent of variance to be retained by PCA
<code>na.remove</code>	a logical; should missing values be removed from the calculations?
<code>object</code>	an object of class <code>preProcess</code>
<code>newdata</code>	a matrix or data frame of new data to be pre-processed
<code>...</code>	Additional arguments (currently this argument is not used)

Details

The operations are applied in this order: centering, scaling, PCA and spatial sign. If PCA is requested but scaling is not, the values will still be scaled.

The function will throw an error if any variables in `x` has less than two unique values.

Value

`preProcess` results in a list with elements

<code>call</code>	the function call
<code>dim</code>	the dimensions of <code>x</code>
<code>mean</code>	a vector of means (if centering was requested)
<code>std</code>	a vector of standard deviations (if scaling or PCA was requested)
<code>rotation</code>	a matrix of eigenvectors if PCA was requested
<code>method</code>	the value of <code>method</code>
<code>thresh</code>	the value of <code>thresh</code>
<code>numComp</code>	the number of principal components required of capture the specified amount of variance

Author(s)

Max Kuhn

References

Kuhn (2008), “Building Predictive Models in R Using the caret” (<http://www.jstatsoft.org/v28/i05/>)

See Also

`prcomp`, `spatialSign`

Examples

```
data(BloodBrain)
# one variable has one unique value
## Not run: preProc <- preProcess(bbbDescr[1:100,])

preProc <- preProcess(bbbDescr[1:100,-3])
training <- predict(preProc, bbbDescr[1:100,-3])
test <- predict(preProc, bbbDescr[101:208,-3])
```

```
print.confusionMatrix
```

Print method for confusionMatrix

Description

a print method for `confusionMatrix`

Usage

```
## S3 method for class 'confusionMatrix':
print(x, digits = max(3, getOption("digits") - 3),
      printStats = TRUE, ...)
```

Arguments

<code>x</code>	an object of class <code>confusionMatrix</code>
<code>digits</code>	number of significant digits when printed
<code>printStats</code>	a logical: if TRUE then table statistics are also printed
<code>...</code>	optional arguments to pass to <code>print.table</code>

Value

`x` is invisibly returned

Author(s)

Max Kuhn

See Also[confusionMatrix](#)

print.train	<i>Print Method for the train Class</i>
-------------	---

Description

Print the results of a [train](#) object.

Usage

```
## S3 method for class 'train':
print(x,
      digits = min(3, getOption("digits") - 3),
      printCall = TRUE,
      details = FALSE,
      ...)
```

Arguments

x	an object of class train .
digits	an integer specifying the number of significant digits to print.
printCall	a logical to print the call at the top of the output
details	a logical to show print or summary methods for the final model. In some cases (such as gbm , knn , lvq , naive Bayes and bagged tree models), no information will be printed even if <code>details = TRUE</code>
...	options passed to the generic print method

Details

The table of complexity parameters used, their resampled performance and a flag for which rows are optimal.

Value

A data frame with the complexity parameter(s) and performance (invisibly).

Author(s)

Max Kuhn

See Also[train](#)

Examples

```
data(iris)
TrainData <- iris[,1:4]
TrainClasses <- iris[,5]

library(klaR)
rdaFit <- train(TrainData, TrainClasses, "rda",
  control = trainControl(method = "cv"))
print(rdaFit)
```

resampleHist

*Plot the resampling distribution of the model statistics***Description**

Create a lattice histogram or densityplot from the resampled outcomes from a `train` object.

Usage

```
resampleHist(object, type = "density", ...)
```

Arguments

<code>object</code>	an object resulting from a call to <code>train</code>
<code>type</code>	a character string. Either "hist" or "density"
<code>...</code>	options to pass to histogram or densityplot

Details

All the metrics from the object are plotted, but only for the final model. For more comprehensive plots functions, see `histogram.train`, `densityplot.train`, `xyplot.train`, `stripplot.train`.

For the the plot to be made, the `returnResamp` argument in `trainControl` should be either "final" or "all".

Value

a object of class `trellis`

Author(s)

Max Kuhn

See Also

`train`, `histogram`, `densityplot`, `histogram.train`, `densityplot.train`, `xyplot.train`, `stripplot.train`

Examples

```
data(iris)
TrainData <- iris[,1:4]
TrainClasses <- iris[,5]

knnFit <- train(TrainData, TrainClasses, "knn")

resampleHist(knnFit)
```

resampleSummary *Summary of resampled performance estimates*

Description

This function uses the out-of-bag predictions to calculate overall performance metrics and returns the observed and predicted data.

Usage

```
resampleSummary(obs, resampled, index = NULL, keepData = TRUE)
```

Arguments

obs	A vector (numeric or factor) of the outcome data
resampled	For bootstrapping, this is either a matrix (for numeric outcomes) or a data frame (for factors). For cross-validation, a vector is produced.
index	The list to index of samples in each cross-validation fold (only used for cross-validation).
keepData	A logical for returning the observed and predicted data.

Details

The mean and standard deviation of the values produced by `postResample` are calculated.

Value

A list with:

metrics	A vector of values describing the bootstrap distribution.
data	A data frame or NULL. Columns include <code>obs</code> , <code>pred</code> and <code>group</code> (for tracking cross-validation folds or bootstrap samples)

Author(s)

Max Kuhn

See Also

[postResample](#)

Examples

```
resampleSummary(rnorm(10), matrix(rnorm(50), ncol = 5))
```

rfe

Backwards Feature Selection

Description

A simple backwards selection, a.k.a. recursive feature selection (RFE), algorithm

Usage

```
rfe(x, ...)

## Default S3 method:
rfe(x, y,
    sizes = 2^(2:4),
    metric = ifelse(is.factor(y), "Accuracy", "RMSE"),
    maximize = ifelse(metric == "RMSE", FALSE, TRUE),
    rfeControl = rfeControl(),
    ...)

rfeIter(x, y,
        testX, testY,
        sizes,
        rfeControl = rfeControl(),
        ...)
```

Arguments

x	a matrix or data frame of predictors for model training. This object must have unique column names.
y	a vector of training set outcomes (either numeric or factor)
testX	a matrix or data frame of test set predictors. This must have the same column names as x
testY	a vector of test set outcomes
sizes	a numeric vector of integers corresponding to the number of features that should be retained
metric	a string that specifies what summary metric will be used to select the optimal model. By default, possible values are "RMSE" and "Rsquared" for regression and "Accuracy" and "Kappa" for classification. If custom performance metrics are used (via the <code>functions</code> argument in rfeControl , the value of <code>metric</code> should match one of the arguments.

<code>maximize</code>	a logical: should the metric be maximized or minimized?
<code>rfeControl</code>	a list of options, including functions for fitting and prediction. See the package vignette or rfeControl for examples
<code>...</code>	options to pass to the model fitting function

Details

This function implements backwards selection of predictors based on predictor importance ranking. The predictors are ranked and the less important ones are sequentially eliminated. The package vignette for feature selection has detailed descriptions of the algorithms.

`rfeIter` is the basic algorithm while `rfe` wraps these operations inside of resampling. To avoid selection bias, it is better to use the function `rfe` than `rfeIter`.

Value

A list with elements

<code>finalVariables</code>	a list of size <code>length(sizes) + 1</code> containing the column names of the “surviving” predictors at each stage of selection. The first element corresponds to all the predictors (i.e. <code>size = ncol(x)</code>)
<code>pred</code>	a data frame with columns for the test set outcome, the predicted outcome and the subset size.

Author(s)

Max Kuhn

See Also

[rfeControl](#)

Examples

```
## Not run:
data(BloodBrain)

x <- scale(bbbDescr[, -nearZeroVar(bbbDescr)])
x <- x[, -findCorrelation(cor(x), .8)]
x <- as.data.frame(x)

set.seed(1)
lmProfile <- rfe(x, logBBB,
  sizes = c(2:25, 30, 35, 40, 45, 50, 55, 60, 65),
  rfeControl = rfeControl(functions = lmFuncs,
    number = 200))

set.seed(1)
lmProfile2 <- rfe(x, logBBB,
  sizes = c(2:25, 30, 35, 40, 45, 50, 55, 60, 65),
  rfeControl = rfeControl(functions = lmFuncs,
```

```

rerank = TRUE,
number = 200))

xyplot(lmProfile$results$RMSE + lmProfile2$results$RMSE +
       rfProfile$results$RMSE + rfProfile2$results$RMSE ~
       lmProfile$results$Variables,
       type = c("g", "p", "l"),
       auto.key = TRUE)

rfProfile <- rfe(x, logBBB,
               sizes = c(2, 5, 10, 20),
               rfeControl = rfeControl(functions = rfFuncs))

bagProfile <- rfe(x, logBBB,
               sizes = c(2, 5, 10, 20),
               rfeControl = rfeControl(functions = treebagFuncs))

set.seed(1)
svmProfile <- rfe(x, logBBB,
               sizes = c(5, 20, 65),
               rfeControl = rfeControl(functions = caretFuncs,
                                       number = 200),
               ## pass options to train()
               method = "svmRadial",
               fit = FALSE)

## classification with no resampling

data(mdr)
mdrDescr <- scale(mdrDescr[, -nearZeroVar(mdrDescr)])
mdrDescr <- mdrDescr[, -findCorrelation(cor(mdrDescr), .8)]

set.seed(1)
inTrain <- createDataPartition(mdrClass, p = .75, list = FALSE)[,1]

train <- mdrDescr[ inTrain, ]
test  <- mdrDescr[-inTrain, ]
trainClass <- mdrClass[ inTrain]
testClass  <- mdrClass[-inTrain]

preProc <- preProcess(train)
train <- predict(preProc, train)
test  <- predict(preProc, test)

nbProfile <- rfeIter(train, trainClass,
                   test, testClass,
                   sizes = c(1:10, 15, 30),
                   rfeControl = rfeControl(functions = nbFuncs))

splitUp <- split(nbProfile$pred,
                factor(nbProfile$pred$subset))
testResults <- lapply(splitUp,
                    function(u) postResample(u$pred, u$obs))

```

```

Variables <- as.numeric(names(testResults))

testResults <- do.call("rbind", testResults)
testResults <- cbind(testResults, Variables)
plot(testResults[,3], testResults[,1])

## End(Not run)

#####
## Parallel Processing Example via MPI

## Not run:

## A function to emulate lapply in parallel
mpiClacs <- function(X, FUN, ...)
{
  theDots <- list(...)
  parLapply(theDots$cl, X, FUN)
}

library(snow)
cl <- makeCluster(5, "MPI")

set.seed(1)
lmProfile <- rfe(x, logBBB,
                 sizes = c(2:25, 30, 35, 40, 45, 50, 55, 60, 65),
                 rfeControl = rfeControl(functions = lmFuncs,
                                          number = 200,
                                          workers = 5,
                                          computeFunction = mpiClacs,
                                          computeArgs = list(cl = cl)))

stopCluster(cl)

## End(Not run)

#####
## Parallel Processing Example via NWS
## Not run:

nwsClacs <- function(X, FUN, ...)
{
  theDots <- list(...)
  eachElem(theDots$sObj,
           fun = FUN,
           elementArgs = list(X))
}

library(nws)
sObj <- sleigh(workerCount = 5)

set.seed(1)
lmProfile <- rfe(x, logBBB,

```

```

        sizes = c(2:25, 30, 35, 40, 45, 50, 55, 60, 65),
        rfeControl = rfeControl(functions = lmFuncs,
                                number = 200,
                                workers = 5,
                                computeFunction = nwsClacs,
                                computeArgs = list(sObj = sObj))

close(sObj)

## End(Not run)

```

rfeControl

Controlling the Feature Selection Algorithms

Description

This function generates a control object that can be used to specify the details of the feature selection algorithms used in this package.

Usage

```

rfeControl(functions = NULL,
            rerank = FALSE,
            method = "boot",
            saveDetails = FALSE,
            number = ifelse(method == "cv", 10, 25),
            verbose = TRUE,
            returnResamp = "all",
            p = .75,
            index = NULL,
            workers = 1,
            computeFunction = lapply,
            computeArgs = NULL)

```

Arguments

functions	a list of functions for model fitting, prediction and variable importance (see Details below)
rerank	a logical: should variable importance be re-calculated each time features are removed?
method	The external resampling method: <i>boot</i> , <i>cv</i> , <i>LOOCV</i> or <i>LGOCV</i> (for repeated training/test splits)
number	Either the number of folds or number of resampling iterations
saveDetails	a logical to save the predictions and variable importances from the selection process
verbose	a logical to print a log for each external resampling iteration

<code>returnResamp</code>	A character string indicating how much of the resampled summary metrics should be saved. Values can be “final”, “all” or “none”
<code>p</code>	For leave-group out cross-validation: the training percentage
<code>index</code>	a list with elements for each external resampling iteration. Each list element is the sample rows used for training at that iteration.
<code>workers</code>	an integer that specifies how many machines/processors will be used
<code>computeFunction</code>	a function that is <code>lapply</code> or emulates <code>lapply</code> . It must have arguments <code>X</code> , <code>FUN</code> and <code>...</code> <code>computeFunction</code> can be used to build models in parallel. See the examples in <code>link{rfe}</code> .
<code>computeArgs</code>	Extra arguments to pass into the <code>...</code> store in <code>computeFunction</code> . See the examples in <code>link{rfe}</code> .

Details

Backwards selection requires function to be specified for some operations.

The `fit` function builds the model based on the current data set. The arguments for the function must be:

- `x` the current training set of predictor data with the appropriate subset of variables
- `y` the current outcome data (either a numeric or factor vector)
- `first` a single logical value for whether the current predictor set has all possible variables
- `last` similar to `first`, but `TRUE` when the last model is fit with the final subset size and predictors.
- `...` optional arguments to pass to the fit function in the call to `rfe`

The function should return a model object that can be used to generate predictions.

The `pred` function returns a vector of predictions (numeric or factors) from the current model. The arguments are:

- `object` the model generated by the `fit` function
- `x` the current set of predictor set for the held-back samples

The `rank` function is used to return the predictors in the order of the most important to the least important. Inputs are:

- `object` the model generated by the `fit` function
- `x` the current set of predictor set for the held-back samples
- `y` the current training outcomes

The function should return a data frame with a column called `vars` that has the current variable names. The first row should be the most important predictor etc. Other columns can be included in the output and will be returned in the final `rfe` object.

The `selectSize` function determines the optimal number of predictors based on the resampling output. Inputs for the function are:

- `xa` matrix with columns for the performance metrics and the number of variables, called "Variables"
- `metrica` character string of the performance measure to optimize (e.g. "RMSE", "Rsquared", "Accuracy" or "Kappa")
- `maximizea` single logical for whether the metric should be maximized

This function should return an integer corresponding to the optimal subset size. `caret` comes with two examples functions for this purpose: `pickSizeBest` and `pickSizeTolerance`.

After the optimal subset size is determined, the `selectVar` function will be used to calculate the best rankings for each variable across all the resampling iterations. Inputs for the function are:

- `ya` list of variables importance for each resampling iteration and each subset size (generated by the user-defined `rank` function). In the example, each each of the cross-validation groups the output of the `rank` function is saved for each of the subset sizes (including the original subset). If the rankings are not recomputed at each iteration, the values will be the same within each cross-validation iteration.
- `size` the integer returned by the `selectSize` function

This function should return a character string of predictor names (of length `size`) in the order of most important to least important

Examples of these functions are included in the package: `lmFuncs`, `rfFuncs`, `treebagFuncs` and `nbFuncs`.

Model details about these functions, including examples, are in the package vignette for feature selection.

Value

A list

Author(s)

Max Kuhn

See Also

`rfe`, `lmFuncs`, `rfFuncs`, `treebagFuncs`, `nbFuncs`, `pickSizeBest`, `pickSizeTolerance`

roc

Compute the points for an ROC curve

Description

Computes sensitivity and specificity for a variety of cutoffs

Usage

```
roc(data, class, dataGrid = TRUE, gridLength = 100, positive = levels(class)[1])
```

Arguments

<code>data</code>	a numeric variable to cut along
<code>class</code>	a factor with class memberships. There must be only two classes.
<code>dataGrid</code>	should the data define the grid of cut-points? If not a sequence of evenly spaced intervals is used.
<code>gridLength</code>	number of intervals to use if the data do not define the grid.
<code>positive</code>	a character string for the level of the class variable that defines a "positive" event

Value

A matrix of results with columns "cutoff", "sensitivity" and "specificity"

Note

The first row in the output has a cutoff of NA, zero sensitivity and specificity of one.

Author(s)

Max Kuhn

See Also

[sensitivity](#), [specificity](#), [aucRoc](#)

Examples

```
set.seed(6)
testData <- data.frame(
  x = c(rnorm(200), rnorm(200) + 1),
  group = factor(rep(letters[1:2], each = 200)))

densityplot(~testData$x, groups = testData$group, auto.key = TRUE)

roc(testData$x, testData$group)
```

sensitivity

Calculate sensitivity, specificity and predictive values

Description

These functions calculate the sensitivity, specificity or predictive values of a measurement system compared to a reference results (the truth or a gold standard). The measurement and "truth" data must have the same two possible outcomes and one of the outcomes must be thought of as a "positive" results.

The sensitivity is defined as the proportion of positive results out of the number of samples which were actually positive. When there are no positive results, sensitivity is not defined and a value of

NA is returned. Similarly, when there are no negative results, specificity is not defined and a value of NA is returned. Similar statements are true for predictive values.

The positive predictive value is defined as the percent of predicted positives that are actually positive while the negative predictive value is defined as the percent of negative positives that are actually negative.

Usage

```
sensitivity(data, ...)
## Default S3 method:
sensitivity(data, reference, positive = levels(reference)[1], ...)
## S3 method for class 'table':
sensitivity(data, positive = rownames(data)[1], ...)
## S3 method for class 'matrix':
sensitivity(data, positive = rownames(data)[1], ...)

specificity(data, ...)
## Default S3 method:
specificity(data, reference, negative = levels(reference)[-1], ...)
## S3 method for class 'table':
specificity(data, negative = rownames(data)[-1], ...)
## S3 method for class 'matrix':
specificity(data, negative = rownames(data)[-1], ...)

posPredValue(data, ...)
## Default S3 method:
posPredValue(data, reference, positive = levels(reference)[1],
              prevalence = NULL, ...)
## S3 method for class 'table':
posPredValue(data, positive = rownames(data)[1], prevalence = NULL, ...)
## S3 method for class 'matrix':
posPredValue(data, positive = rownames(data)[1], prevalence = NULL, ...)

negPredValue(data, ...)
## Default S3 method:
negPredValue(data, reference, negative = levels(reference)[2],
              prevalence = NULL, ...)
## S3 method for class 'table':
negPredValue(data, negative = rownames(data)[-1], prevalence = NULL, ...)
## S3 method for class 'matrix':
negPredValue(data, negative = rownames(data)[-1], prevalence = NULL, ...)
```

Arguments

<code>data</code>	for the default functions, a factor containing the discrete measurements. For the <code>table</code> or <code>matrix</code> functions, a table or matrix object, respectively.
<code>reference</code>	a factor containing the reference values

positive	a character string that defines the factor level corresponding to the "positive" results
negative	a character string that defines the factor level corresponding to the "negative" results
prevalence	a numeric value for the rate of the "positive" class of the data
...	not currently used

Details

Suppose a 2x2 table with notation

	Reference	
Predicted	Event	No Event
Event	A	B
No Event	C	D

The formulas used here are:

$$\text{Sensitivity} = A / (A + C)$$

$$\text{Specificity} = D / (B + D)$$

$$\text{Prevalence} = (A + C) / (A + B + C + D)$$

$$\text{PPV} = (\text{sensitivity} * \text{Prevalence}) / ((\text{sensitivity} * \text{Prevalence}) + ((1 - \text{specificity}) * (1 - \text{Prevalence})))$$

$$\text{NPV} = (\text{specificity} * (1 - \text{Prevalence})) / (((1 - \text{sensitivity}) * \text{Prevalence}) + (\text{specificity} * (1 - \text{Prevalence})))$$

See the references for discussions of the statistics.

Value

A number between 0 and 1 (or NA).

Author(s)

Max Kuhn

References

Kuhn, M. (2008), "Building predictive models in R using the caret package," *Journal of Statistical Software*, (<http://www.jstatsoft.org/v28/i05/>).

Altman, D.G., Bland, J.M. (1994) "Diagnostic tests 1: sensitivity and specificity," *British Medical Journal*, vol 308, 1552.

Altman, D.G., Bland, J.M. (1994) "Diagnostic tests 2: predictive values," *British Medical Journal*, vol 309, 102.

See Also

[confusionMatrix](#)

Examples

```
#####
## 2 class example

lvs <- c("normal", "abnormal")
truth <- factor(rep(lvs, times = c(86, 258)),
               levels = rev(lvs))
pred <- factor(
  c(
    rep(lvs, times = c(54, 32)),
    rep(lvs, times = c(27, 231))),
  levels = rev(lvs))

xtab <- table(pred, truth)

sensitivity(pred, truth)
sensitivity(xtab)
posPredValue(pred, truth)
posPredValue(pred, truth, prevalence = 0.25)

specificity(pred, truth)
negPredValue(pred, truth)
negPredValue(xtab)
negPredValue(pred, truth, prevalence = 0.25)

prev <- seq(0.001, .99, length = 20)
npvVals <- ppvVals <- prev * NA
for(i in seq(along = prev))
{
  ppvVals[i] <- posPredValue(pred, truth, prevalence = prev[i])
  npvVals[i] <- negPredValue(pred, truth, prevalence = prev[i])
}

plot(prev, ppvVals,
     ylim = c(0, 1),
     type = "l",
     ylab = "",
     xlab = "Prevalence (i.e. prior)")
points(prev, npvVals, type = "l", col = "red")
abline(h=sensitivity(pred, truth), lty = 2)
abline(h=specificity(pred, truth), lty = 2, col = "red")
legend(.5, .5,
      c("ppv", "npv", "sens", "spec"),
      col = c("black", "red", "black", "red"),
      lty = c(1, 1, 2, 2))

#####
## 3 class example

library(MASS)
```

```

fit <- lda(Species ~ ., data = iris)
model <- predict(fit)$class

irisTabs <- table(model, iris$Species)

## When passing factors, an error occurs with more
## than two levels
## Not run: sensitivity(model, iris$Species)

## When passing a table, more than two levels can
## be used
sensitivity(irisTabs, "versicolor")
specificity(irisTabs, c("setosa", "virginica"))

```

spatialSign *Compute the multivariate spatial sign*

Description

Compute the spatial sign (a projection of a data vector to a unit length circle). The spatial sign of a vector w is $w / \text{norm}(w)$.

Usage

```

spatialSign.default(x)
spatialSign.matrix(x)
spatialSign.data.frame(x)

```

Arguments

x an object full of numeric data (which should probably be scaled). Factors are not allowed. This could be a vector, matrix or data frame.

Value

A vector, matrix or data frame with the same dim names of the original data.

Author(s)

Max Kuhn

Examples

```

spatialSign(rnorm(5))

spatialSign(matrix(rnorm(12), ncol = 3))

# should fail since the fifth column is a factor

```

```
try(spatialSign(iris), silent = TRUE)

spatialSign(iris[,-5])

trellis.par.set(caretTheme())
featurePlot(iris[,-5], iris[,5], "pairs")
featurePlot(spatialSign(scale(iris[,-5])), iris[,5], "pairs")
```

summary.bagEarth *Summarize a bagged earth or FDA fit*

Description

The function shows a summary of the results from a bagged earth model

Usage

```
summary.bagEarth(object, ...)
summary.bagFDA(object, ...)
```

Arguments

object	an object of class "bagEarth" or "bagFDA"
...	optional arguments (not used)

Details

The out-of-bag statistics are summarized, as well as the distribution of the number of model terms and number of variables used across all of the bootstrap samples.

Value

a list with elements	
modelInfo	a matrix with the number of model terms and variables used
oobStat	a summary of the out-of-bag statistics
bmarsCall	the original call to bagEarth

Author(s)

Max Kuhn

Examples

```
data(trees)
fit <- bagEarth(trees[,-3], trees[3])
summary(fit)
```

tecator

Fat, Water and Protein Content of Meat Samples

Description

"These data are recorded on a Tecator Infratec Food and Feed Analyzer working in the wavelength range 850 - 1050 nm by the Near Infrared Transmission (NIT) principle. Each sample contains finely chopped pure meat with different moisture, fat and protein contents.

If results from these data are used in a publication we want you to mention the instrument and company name (Tecator) in the publication. In addition, please send a preprint of your article to Karin Thente, Tecator AB, Box 70, S-263 21 Hoganas, Sweden

The data are available in the public domain with no responsibility from the original data source. The data can be redistributed as long as this permission note is attached."

"For each meat sample the data consists of a 100 channel spectrum of absorbances and the contents of moisture (water), fat and protein. The absorbance is $-\log_{10}$ of the transmittance measured by the spectrometer. The three contents, measured in percent, are determined by analytic chemistry."

Included here are the training, monitoring and test sets.

Usage

```
data(tecator)
```

Value

absorp	absorbance data for 215 samples. The first 129 were originally used as a training set
endpoints	the percentages of water, fat and protein

Examples

```
data(tecator)

splom(~endpoints)

# plot 10 random spectra
set.seed(1)
inSubset <- sample(1:dim(endpoints)[1], 10)

absorpSubset <- absorp[inSubset,]
endpointSubset <- endpoints[inSubset, 3]

newOrder <- order(absorpSubset[,1])
absorpSubset <- absorpSubset[newOrder,]
endpointSubset <- endpointSubset[newOrder]

plotColors <- rainbow(10)
```

```

plot(absorpSubset[1,],
     type = "n",
     ylim = range(absorpSubset),
     xlim = c(0, 105),
     xlab = "Wavelength Index",
     ylab = "Absorption")

for(i in 1:10)
{
  points(absorpSubset[i,], type = "l", col = plotColors[i], lwd = 2)
  text(105, absorpSubset[i,100], endpointSubset[i], col = plotColors[i])
}
title("Predictor Profiles for 10 Random Samples")

```

train

Fit Predictive Models over Different Tuning Parameters

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

```

train(x, ...)

## Default S3 method:
train(x, y,
      method = "rf",
      ...,
      weights = NULL,
      metric = ifelse(is.factor(y), "Accuracy", "RMSE"),
      maximize = ifelse(metric == "RMSE", FALSE, TRUE),
      trControl = trainControl(),
      tuneGrid = NULL,
      tuneLength = 3)

## S3 method for class 'formula':
train(form, data, ..., weights, subset, na.action, contrasts = NULL)

```

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.
form	A formula of the form $y \sim x_1 + x_2 + \dots$
data	Data frame from which variables specified in <code>formula</code> are preferentially to be taken.

<code>weights</code>	a numeric vector of case weights. This argument will only affect models that allow case weights.
<code>subset</code>	An index vector specifying the cases to be used in the training sample. (NOTE: If given, this argument must be named.)
<code>na.action</code>	A function to specify the action to be taken if NAs are found. The default action is for the procedure to fail. An alternative is <code>na.omit</code> , which leads to rejection of cases with missing values on any required variable. (NOTE: If given, this argument must be named.)
<code>contrasts</code>	a list of contrasts to be used for some or all of the factors appearing as variables in the model formula.
<code>method</code>	a string specifying which classification or regression model to use. Possible values are: <code>ada</code> , <code>bagEarth</code> , <code>bagFDA</code> , <code>blackboost</code> , <code>cforest</code> , <code>ctree</code> , <code>ctree2</code> , <code>earth</code> , <code>enet</code> , <code>fda</code> , <code>gamboost</code> , <code>gaussprPoly</code> , <code>gaussprRadial</code> , <code>gaussprLinear</code> , <code>gbm</code> , <code>glm</code> , <code>glmboost</code> , <code>glmnet</code> , <code>gpls</code> , <code>J48</code> , <code>JRip</code> , <code>knn</code> , <code>lars</code> , <code>lasso</code> , <code>lda</code> , <code>lm</code> , <code>lmStepAIC</code> , <code>LMT</code> , <code>logitBoost</code> , <code>lssvmPoly</code> , <code>lssvmRadial</code> , <code>lvq</code> , <code>M5Rules</code> , <code>mda</code> , <code>multinom</code> , <code>nb</code> , <code>nnet</code> , <code>nodeHarvest</code> , <code>OneR</code> , <code>pam</code> , <code>pcaNNet</code> , <code>pcr</code> , <code>pda</code> , <code>pda2</code> , <code>penalized</code> , <code>pls</code> , <code>ppr</code> , <code>qda</code> , <code>rda</code> , <code>rf</code> , <code>rlm</code> , <code>rpart</code> , <code>rvmLinear</code> , <code>rvmPoly</code> , <code>rvmRadial</code> , <code>sda</code> , <code>sddaLDA</code> , <code>sddaQDA</code> , <code>slda</code> , <code>smda</code> , <code>sparseLDA</code> , <code>sppls</code> , <code>superpc</code> , <code>svmPoly</code> , <code>svmRadial</code> , <code>svmLinear</code> , <code>treebag</code> and <code>vbmpRadial</code> . See the Details section below.
<code>...</code>	arguments passed to the classification or regression routine (such as <code>randomForest</code>). Errors will occur if values for tuning parameters are passed here.
<code>metric</code>	a string that specifies what summary metric will be used to select the optimal model. By default, possible values are "RMSE" and "Rsquared" for regression and "Accuracy" and "Kappa" for classification. If custom performance metrics are used (via the <code>summaryFunction</code> argument in <code>trainControl</code> , the value of <code>metric</code> should match one of the arguments. If it does not, a warning is issued and the first metric given by the <code>summaryFunction</code> is used. (NOTE: If given, this argument must be named.)
<code>maximize</code>	a logical: should the metric be maximized or minimized?
<code>trControl</code>	a list of values that define how this function acts. See <code>trainControl</code> . (NOTE: If given, this argument must be named.)
<code>tuneGrid</code>	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. <code>.decay</code> , <code>.lambda</code>). See the function <code>createGrid</code> in this package for more details. (NOTE: If given, this argument must be named.)
<code>tuneLength</code>	an integer denoting the number of levels for each tuning parameters that should be generated by <code>createGrid</code> . (NOTE: If given, this argument must be named.)

Details

`train` 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.

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 `train`.

Model	method	Value	Package	Tuning Parameter(s)
Generalized linear model	glm		stats	none
Recursive partitioning	rpart		rpart	maxdepth
	ctree		party	mincriterion
	ctree2		party	maxdepth
Boosted trees	gbm		gbm	interaction depth, n.trees, shrinkage
	blackboost		mboost	maxdepth, mstop
Boosted regression models	ada		ada	maxdepth, iter, nu
	glmboost		mboost	mstop
	gamboost		mboost	mstop
	logitBoost		caTools	nIter
Random forests	rf		randomForest	mtry
	cforest		party	mtry
Bagged trees	treebag		ipred	None
Node Harvest	nodeHarvest		nodeHarvest	maxinter, node
Elastic net (glm)	glmnet		glmnet	alpha, lambda
Neural networks	nnet		nnet	decay, size
	pcaNNet		caret	decay, size
Projection pursuit regression	ppr		stats	nterms
Principal component regression	pcr		pls	ncomp
Partial least squares	pls		pls, caret	ncomp
Sparse partial least squares	spls		spls, caret	K, eta, kappa
Support vector machines	svmLinear		kernlab	C
	svmRadial		kernlab	sigma, C
	svmPoly		kernlab	scale, degree, C
	rvmLinear		kernlab	none
Relevance vector machines	rvmRadial		kernlab	sigma
	rvmPoly		kernlab	scale, degree
	lssvmRadial		kernlab	sigma
Least squares support vector machines	guassprLinear1		kernlab	none
	guassprRadial		kernlab	sigma
	guassprPoly		kernlab	scale, degree
Linear least squares	lm		stats	None
Robust linear regression	rlm		MASS	None
Multivariate adaptive regression splines	earth		earth	degree, nprune
Bagged MARS	bagEarth		caret, earth	degree, nprune
Rule Based Regression	M5Rules		RWeka	pruned
Elastic net	enet		elasticnet	lambda, fraction
Least Angle Regression	lars		lars	fraction
	lars2		lars	steps
The Lasso	enet		elasticnet	fraction
Penalized linear models	penalized		penalized	lambda1, lambda2
Supervised principal components	superpc		superpc	n.components, threshold
Linear discriminant analysis	lda		MASS	None
Quadratic discriminant analysis	qda		MASS	None

Stabilized linear discriminant analysis	slda	ipred	None
Stepwise diagonal discriminant analysis	sddaLDA	SDDA	None
	sddaQDA	SDDA	None
Shrinkage discriminant analysis	sda	sda	diagonal
Sparse linear discriminant analysis	sparseLDA	sparseLDA	NumVars, lambda
Regularized discriminant analysis	rda	klaR	lambda, gamma
Mixture discriminant analysis	mda	mda	subclasses
Sparse mixture discriminant analysis	smda	sparseLDA	NumVars, R, lambda
Penalized discriminant analysis	pda	mda	lambda
	pda2	mda	df
Stabilised linear discriminant analysis	slda	ipred	None
Flexible discriminant analysis (MARS)	fda	mda, earth	degree, nprune
Bagged FDA	bagFDA	caret, earth	degree, nprune
Logistic/multinomial regression	multinom	nnet	decay
Rule-based classification	J48	RWeka	C
	OneR	RWeka	None
	PART	RWeka	threshold, pruned
	JRip	RWeka	NumOpt
Bayesian multinomial probit model	vbmpRadial	vbmp	estimateTheta
k nearest neighbors	knn3	caret	k
Nearest shrunken centroids	pam	pamr	threshold
Naive Bayes	nb	klaR	usekernel
Generalized partial least squares	gpls	gpls	K.prov
Learned vector quantization	lvq	class	k

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" has more details and examples related to this function.

`train` can be used with "explicit parallelism", where different resamples (e.g. cross-validation group) can be split up and run on multiple machines or processors. By default, `train` will use a single processor on the host machine. To use more, the `computeFunction` and `computeArgs` arguments in `trainControl` can be used. `computeFunction` is used to pass a function that takes arguments named `X` and `FUN`. Internally, `train` will pass the data and modeling functions through using these arguments. By default, `train` uses `lapply`. Alternatively, any function that emulates `lapply` but distributes jobs across multiple machines/processors can be used. Arguments to such a function can be passed (if needed) via the `computeArgs` argument in `trainControl`. Examples are given below using the **Rmpi** package (via `snow`) and `NetworkSpaces` (via the `nws` package).

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>resample</code>	A data frame with columns for each performance metric. Each row corresponds to each resample. If leave-one-out cross-validation or out-of-bag estimation methods are requested, this will be <code>NULL</code> . The <code>returnResamp</code> argument of <code>trainControl</code> controls how much of the resampled results are saved.
<code>perfNames</code>	a character vector of performance metrics that are produced by the summary function
<code>maximize</code>	a logical recycled from the function arguments.

Author(s)

Max Kuhn (the guts of `train.formula` were based on Ripley's `nnet.formula`)

References

Kuhn (2008), "Building Predictive Models in R Using the caret" (<http://www.jstatsoft.org/v28/i05/>)

See Also

`trainControl`, `createGrid`, `createFolds`

Examples

```
#####
## Classification Example

data(iris)
TrainData <- iris[,1:4]
TrainClasses <- iris[,5]

knnFit1 <- train(TrainData, TrainClasses,
                 "knn",
                 tuneLength = 10,
                 trControl = trainControl(method = "cv"))
```

```

knnFit2 <- train(TrainData, TrainClasses,
                "knn", tuneLength = 10,
                trControl = trainControl(method = "boot"))

library(MASS)
nnetFit <- train(TrainData, TrainClasses,
                "nnet",
                tuneLength = 2,
                trace = FALSE,
                maxit = 100)

#####
## Regression Example

library(mlbench)
data(BostonHousing)

lmFit <- train(medv ~ . + rm:lstat,
               data = BostonHousing,
               "lm")

library(rpart)
rpartFit <- train(medv ~ .,
                  data = BostonHousing,
                  "rpart",
                  tuneLength = 9)

#####
## Example with a custom metric

madSummary <- function (data,
                        lev = NULL,
                        model = NULL)
{
  out <- mad(data$obs - data$pred,
             na.rm = TRUE)
  names(out) <- "MAD"
  out
}

robustControl <- trainControl(summaryFunction = madSummary)
marsGrid <- expand.grid(.degree = 1,
                      .nprune = (1:10) * 2)

earthFit <- train(medv ~ .,
                  data = BostonHousing,
                  "earth",
                  tuneGrid = marsGrid,
                  metric = "MAD",
                  maximize = FALSE,
                  trControl = robustControl)

```

```
#####
## Parallel Processing Example via MPI

## Not run:

## A function to emulate lapply in parallel
mpiCalcs <- function(X, FUN, ...)
{
  theDots <- list(...)
  parLapply(theDots$cl, X, FUN)
}

library(snow)
cl <- makeCluster(5, "MPI")

## 50 bootstrap models distributed across 5 workers
mpiControl <- trainControl(workers = 5,
                           number = 50,
                           computeFunction = mpiCalcs,
                           computeArgs = list(cl = cl))

set.seed(1)
usingMPI <- train(medv ~ .,
                 data = BostonHousing,
                 "glmboost",
                 trControl = mpiControl)

stopCluster(cl)

## End(Not run)

#####
## Parallel Processing Example via NWS
## Not run:

nwsCalcs <- function(X, FUN, ...)
{
  theDots <- list(...)
  eachElem(theDots$sObj,
           fun = FUN,
           elementArgs = list(X))
}

library(nws)
sObj <- sleigh(workerCount = 5)

nwsControl <- trainControl(workers = 5,
                           number = 50,
                           computeFunction = nwsCalcs,
                           computeArgs = list(sObj = sObj))

set.seed(1)
usingNWS <- train(medv ~ .,
                 data = BostonHousing,
                 "glmboost",
```

```

trControl = nwsControl)

close(sObj)

## End(Not run)

```

trainControl	<i>Control parameters for train</i>
--------------	-------------------------------------

Description

Control of printing and resampling for train

Usage

```

trainControl(
  method = "boot",
  number = ifelse(method == "cv", 10, 25),
  verboseIter = TRUE,
  returnData = TRUE,
  returnResamp = "final",
  p = 0.75,
  summaryFunction = defaultSummary,
  selectionFunction = "best",
  index = NULL,
  workers = 1,
  computeFunction = lapply,
  computeArgs = NULL)

```

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
returnResamp	A character string indicating how much of the resampled summary metrics should be saved. Values can be “final”, “all” or “none”
p	For leave-group out cross-validation: the training percentage
summaryFunction	a function to compute performance metrics across resamples. The arguments to the function should be the same as those in <code>defaultSummary</code> .

selectionFunction	the function used to select the optimal tuning parameter. This can be a name of the function or the function itself. See best for details and other options.
index	a list with elements for each resampling iteration. Each list element is the sample rows used for training at that iteration.
workers	an integer that specifies how many machines/processors will be used
computeFunction	a function that is lapply or emulates lapply . It must have arguments X, FUN and <code>computeFunction</code> can be used to build models in parallel. See the examples in <code>link{train}</code> .
computeArgs	Extra arguments to pass into the ... store in <code>computeFunction</code> . See the examples in <code>link{train}</code> .

Value

An echo of the parameters specified

Author(s)

max Kuhn

varImp	<i>Calculation of variable importance for regression and classification models</i>
--------	--

Description

A generic method for calculating variable importance for objects produced by `train` and method specific methods

Usage

```
## S3 method for class 'train':
varImp(object, useModel = TRUE, nonpara = TRUE, scale = TRUE, ...)
## S3 method for class 'earth':
varImp(object, value = "gcv", ...)
## S3 method for class 'rpart':
varImp(object, ...)
## S3 method for class 'randomForest':
varImp(object, ...)
## S3 method for class 'gbm':
varImp(object, numTrees, ...)
## S3 method for class 'classbagg':
varImp(object, ...)
## S3 method for class 'regbagg':
varImp(object, ...)
## S3 method for class 'pamrtrained':
```

```

varImp(object, threshold, data, ...)
## S3 method for class 'lm':
varImp(object, ...)
## S3 method for class 'mvr':
varImp(object, ...)
## S3 method for class 'bagEarth':
varImp(object, ...)
## S3 method for class 'RandomForest':
varImp(object, normalize = TRUE, ...)
## S3 method for class 'rfe':
varImp(object, drop = FALSE, ...)

```

Arguments

object	an object corresponding to a fitted model
useModel	use a model based technique for measuring variable importance? This is only used for some models (lm, pls, rf, rpart, gbm, pam and mars)
nonpara	should nonparametric methods be used to assess the relationship between the features and response (only used with useModel = FALSE and only passed to filterVarImp).
scale	should the importance values be scaled to 0 and 100?
...	parameters to pass to the specific varImp methods
numTrees	the number of iterations (trees) to use in a boosted tree model
threshold	the shrinkage threshold (pamr models only)
data	the training set predictors (pamr models only)
value	the statistic that will be used to calculate importance: either gcv, nsubsets, or rss
normalize	a logical: should the importance values be divided by their standard deviations?
drop	a logical: should variables not included in the final set be calculated?

Details

For models that do not have corresponding varImp methods, see filterVarImp.

Otherwise:

Linear Models: the absolute value of the t–statistic for each model parameter is used.

Random Forest: varImp.randomForest and varImp.RandomForest are wrappers around the importance functions from the **randomForest** and **party** packages, respectively.

Partial Least Squares: the variable importance measure here is based on weighted sums of the absolute regression coefficients. The weights are a function of the reduction of the sums of squares across the number of PLS components and are computed separately for each outcome. Therefore, the contribution of the coefficients are weighted proportionally to the reduction in the sums of squares.

Recursive Partitioning: The reduction in the loss function (e.g. mean squared error) attributed to each variable at each split is tabulated and the sum is returned. Also, since there may be candidate

variables that are important but are not used in a split, the top competing variables are also tabulated at each split. This can be turned off using the `maxcompete` argument in `rpart.control`. This method does not currently provide class-specific measures of importance when the response is a factor.

Bagged Trees: The same methodology as a single tree is applied to all bootstrapped trees and the total importance is returned

Boosted Trees: `varImp.gbm` is a wrapper around the function from that package (see the **gbm** package vignette)

Multivariate Adaptive Regression Splines: MARS models include a backwards elimination feature selection routine that looks at reductions in the generalized cross-validation (GCV) estimate of error. The `varImp` function tracks the changes in model statistics, such as the GCV, for each predictor and accumulates the reduction in the statistic when each predictor's feature is added to the model. This total reduction is used as the variable importance measure. If a predictor was never used in any of the MARS basis functions in the final model (after pruning), it has an importance value of zero. Prior to June 2008, the package used an internal function for these calculations. Currently, the `varImp` is a wrapper to the `evimp` function in the `earth` package. There are three statistics that can be used to estimate variable importance in MARS models. Using `varImp(object, value = "gcv")` tracks the reduction in the generalized cross-validation statistic as terms are added. However, there are some cases when terms are retained in the model that result in an increase in GCV. Negative variable importance values for MARS are set to zero. Alternatively, using `varImp(object, value = "rss")` monitors the change in the residual sums of squares (RSS) as terms are added, which will never be negative. Also, the option `varImp(object, value = "nsubsets")`, which counts the number of subsets where the variable is used (in the final, pruned model).

Nearest shrunken centroids: The difference between the class centroids and the overall centroid is used to measure the variable influence (see `pamr.predict`). The larger the difference between the class centroid and the overall center of the data, the larger the separation between the classes. The training set predictions must be supplied when an object of class `pamrtrained` is given to `varImp`.

Value

A data frame with class `c("varImp.train", "data.frame")` for `varImp.train` or a matrix for other models.

Author(s)

Max Kuhn

Index

*Topic **datasets**

- BloodBrain, 9
- cox2, 15
- mdrr, 34
- oil, 38
- pottery, 53
- tecator, 81

*Topic **graphs**

- panel.needle, 41

*Topic **hplot**

- dotPlot, 19
- featurePlot, 20
- histogram.train, 25
- lattice.rfe, 30
- plot.train, 44
- plot.varImp.train, 46
- plotClassProbs, 47
- plotObsVsPred, 48
- resampleHist, 66

*Topic **manip**

- Alternate Affy Gene
 - Expression Summary
 - Methods., 2
- applyProcessing, 3
- aucRoc, 5
- classDist, 11
- findCorrelation, 22
- findLinearCombos, 23
- normalize.AffyBatch.normalize2Reference, 36
- oneSE, 39
- predict.train, 56
- roc, 74
- sensitivity, 75
- spatialSign, 79
- summary.bagEarth, 80

*Topic **models**

- caretFuncs, 10
- filterVarImp, 21

- format.bagEarth, 24
- normalize2Reference, 37
- plsda, 49
- predictors, 59
- rfe, 68
- train, 82
- varImp, 90

*Topic **multivariate**

- knn3, 27
- knnreg, 28
- predict.knn3, 55
- predict.knnreg, 56

*Topic **neural**

- pcaNNet.default, 42

*Topic **print**

- print.train, 65

*Topic **regression**

- bagEarth, 6
- bagFDA, 8
- predict.bagEarth, 54

*Topic **utilities**

- as.table.confusionMatrix, 4
- confusionMatrix, 13
- createDataPartition, 16
- createGrid, 18
- maxDissim, 31
- nearZeroVar, 34
- postResample, 52
- preProcess, 62
- print.confusionMatrix, 64
- resampleSummary, 67
- rfeControl, 72
- trainControl, 89

- absorp(*tecator*), 81

- Alternate Affy Gene Expression
 - Summary Methods., 2

- applyProcessing, 3

- as.matrix.confusionMatrix, 15

- as.matrix.confusionMatrix
(*as.table.confusionMatrix*),
4
- as.table.confusionMatrix, 4, 15
- aucRoc, 5, 75
- bagEarth, 6, 24, 55, 62
- bagFDA, 8, 62
- bagging, 62
- bbbDescr (*BloodBrain*), 9
- best, 90
- best (*oneSE*), 39
- binom.test, 14, 15
- BloodBrain, 9
- bwplot, 45
- caretFuncs, 10
- cforest, 62
- classDist, 11
- confusionMatrix, 4, 13, 65, 77
- cox2, 15
- cox2Class (*cox2*), 15
- cox2Descr (*cox2*), 15
- cox2IC50 (*cox2*), 15
- createDataPartition, 16
- createFolds, 86
- createFolds
(*createDataPartition*), 16
- createGrid, 18, 83, 85, 86
- createResample
(*createDataPartition*), 16
- ctree, 62
- defaultSummary, 89
- defaultSummary (*postResample*), 52
- densityplot, 26, 30, 31, 66
- densityplot.rfe (*lattice.rfe*), 30
- densityplot.train, 66
- densityplot.train
(*histogram.train*), 25
- dist, 32
- dotPlot, 19
- dotplot, 19, 42, 46, 48
- earth, 7, 25, 54, 62
- endpoints (*tecator*), 81
- evimp, 92
- extractPrediction, 48
- extractPrediction
(*predict.train*), 56
- extractProb, 47
- extractProb (*predict.train*), 56
- fattyAcids (*oil*), 38
- fda, 9, 54, 62
- featurePlot, 20
- filterVarImp, 21
- findCorrelation, 22
- findLinearCombos, 23
- format.bagEarth, 24
- format.earth, 24
- generateExprSet-methods, 2
- generateExprVal.method.trimMean
(*Alternate Affy Gene
Expression Summary
Methods.*), 2
- histogram, 26, 30, 31, 47, 66
- histogram.rfe (*lattice.rfe*), 30
- histogram.train, 25, 66
- ipredbagg, 62
- ipredknn, 28, 29
- knn, 28, 29, 56
- knn3, 27, 55
- knn3Train (*knn3*), 27
- knnreg, 28, 56
- knnregTrain (*knnreg*), 28
- lapply, 73, 85, 90
- lattice.rfe, 30
- ldaFuncs (*caretFuncs*), 10
- levelplot, 45
- lm, 21
- lmFuncs, 74
- lmFuncs (*caretFuncs*), 10
- loess, 21
- logBBB (*BloodBrain*), 9
- mahalanobis, 12
- maxDissim, 31
- mdrr, 34
- mdrrClass (*mdrr*), 34
- mdrrDescr (*mdrr*), 34
- minDiss (*maxDissim*), 31
- NaiveBayes, 51
- nbFuncs, 74

- nbFuncs (*caretFuncs*), 10
- nearZeroVar, 34
- negPredValue, 15
- negPredValue (*sensitivity*), 75
- nnet, 43, 44, 62
- normalize.AffyBatch.normalize2Reference, 36
- normalize2Reference, 37
- oil, 38
- oilType (*oil*), 38
- oneSE, 39
- pamr.train, 62
- panel.dotplot, 42
- panel.needle, 41, 46
- pcaNNet (*pcaNNet.default*), 42
- pcaNNet.default, 42
- pickSizeBest, 74
- pickSizeBest (*caretFuncs*), 10
- pickSizeTolerance, 74
- pickSizeTolerance (*caretFuncs*), 10
- pickVars (*caretFuncs*), 10
- plot.train, 44
- plot.varImp.train, 46
- plotClassProbs, 47, 58
- plotObsVsPred, 48, 58
- plsda, 49
- plsr, 50, 51
- posPredValue, 15
- posPredValue (*sensitivity*), 75
- postResample, 52, 67, 68
- pottery, 53
- potteryClass (*pottery*), 53
- prcomp, 12, 64
- predict, 55, 56
- predict.bagEarth, 7, 54
- predict.bagFDA, 9
- predict.bagFDA (*predict.bagEarth*), 54
- predict.classDist (*classDist*), 11
- predict.ipredknn, 55, 56
- predict.knn3, 28, 55
- predict.knnreg, 29, 56
- predict.list (*predict.train*), 56
- predict.pcaNNet (*pcaNNet.default*), 42
- predict.plsda (*plsda*), 49
- predict.preProcess (*preProcess*), 62
- predict.splsda (*plsda*), 49
- predict.train, 56
- predictors, 59
- preProcess, 3, 44, 62
- print.bagEarth (*bagEarth*), 6
- print.bagFDA (*bagFDA*), 8
- print.confusionMatrix, 15, 64
- print.train, 65
- ProbeSet, 2
- processData (*applyProcessing*), 3
- randomForest, 62, 83
- resampleHist, 45, 66
- resampleSummary, 67
- rfe, 11, 30, 31, 68, 74
- rfeControl, 11, 30, 31, 68, 69, 72
- rfeIter (*rfe*), 68
- rfFuncs, 74
- rfFuncs (*caretFuncs*), 10
- roc, 6, 74
- rpart, 62
- sensitivity, 6, 15, 75, 75
- spatialSign, 64, 79
- specificity, 6, 15, 75
- specificity (*sensitivity*), 75
- spls, 50, 51
- splsda (*plsda*), 49
- stripplot, 26, 30, 31
- stripplot.rfe (*lattice.rfe*), 30
- stripplot.train, 66
- stripplot.train (*histogram.train*), 25
- sumDiss (*maxDissim*), 31
- summary.bagEarth, 80
- summary.bagFDA (*summary.bagEarth*), 80
- superpc.train, 62
- table, 13
- teclator, 81
- tolerance (*oneSE*), 39
- train, 18, 26, 40, 41, 44, 45, 52, 53, 57, 58, 65, 66, 82
- trainControl, 26, 39–41, 53, 66, 83, 85, 86, 89
- treebagFuncs, 74

`treebagFuncs` (*caretFuncs*), 10

`varImp`, 19, 90

`xyplot`, 26, 30, 31, 48

`xyplot.rfe` (*lattice.rfe*), 30

`xyplot.train`, 66

`xyplot.train` (*histogram.train*), 25