Package ‘classyfire’

November 22, 2014

Type Package

Title Robust multivariate classification using highly optimised SVM ensembles

Version 0.1-1

Date 2014-11-22

Author Eleni Chatzimichali <ea.chatzimichali@gmail.com> and Conrad Bessant <c.bessant@qmul.ac.uk>

Maintainer Eleni Chatzimichali <ea.chatzimichali@gmail.com>

Description A collection of functions for the creation and application of highly optimised, robustly evaluated ensembles of support vector machines (SVMs). The package takes care of training individual SVM classifiers using a fast parallel heuristic algorithm, and combines individual classifiers into ensembles. Robust metrics of classification performance are offered by bootstrap resampling and permutation testing.

License GPL (>= 2)

Depends R (>= 3.0.0), snowfall (>= 1.84-6), e1071 (>= 1.6-3), boot (>= 1.3-11), neldermead (>= 1.0-9)

Imports ggplot2 (>= 1.0-0), optimbase (>= 1.0-9)

R topics documented:

`classyfire-package` *Robust multivariate classification using highly optimised SVM ensembles*

Description

The aim of the `classyfire` package is to improve the quality of multivariate classification projects by making a state-of-the-art multivariate classification workflow available to everyone. Classyfire achieves this by providing powerful functions which automate as much of the classifier building and testing as possible. However, to avoid these functions becoming impenetrable black boxes, detailed information is provided about how these functions work, and full access is provided to the internals of all classifiers that are produced.

Details

Package:	classyfire
Type:	Package
Version:	0.1-1
Date:	2014-11-22
License:	GPL (>= 2)

Author(s)

Adapted functionality by Eleni Chatzimichali (ea.chatzimichali@gmail.com)

Author of the SVM functions: David Meyer (<David.Meyer@R-project.org>)
(based on C/C++-code by Chih-Chung Chang and Chih-Jen Lin)
Author of Scilab neldermead module: Michael Baudin (INRIA - Digiteo)
Author of Scilab R adaptation: Sebastien Bihorel (<sb.pmlab@gmail.com>)
Authors of bootstrap functions: Angelo Canty and Brian Ripley (originally by Angelo Canty for S)

References

There are many references explaining the concepts behind the functionality of this package. Among them are :

Chang, Chih-Chung and Lin, Chih-Jen:
LIBSVM: a library for Support Vector Machines
`http://www.csie.ntu.edu.tw/~cjlin/libsvm`

Exact formulations of models, algorithms, etc. can be found in the document:
Chang, Chih-Chung and Lin, Chih-Jen:
LIBSVM: a library for Support Vector Machines
`http://www.csie.ntu.edu.tw/~cjlin/papers/libsvm.ps.gz`

More implementation details and speed benchmarks can be found on:
Rong-En Fan and Pai-Hsune Chen and Chih-Jen Lin:
Working Set Selection Using the Second Order Information for Training SVM
`http://www.csie.ntu.edu.tw/~cjlin/papers/quadworkset.pdf`

Spendley, W. and Hext, G. R. and Himsworth, F. R.
Sequential Application of Simplex Designs in Optimisation and Evolutionary Operation
American Statistical Association and American Society for Quality, 1962

Nelder, J. A. and Mead, R.
A Simplex Method for Function Minimization
The Computer Journal, 1965

C. T. Kelley
Iterative Methods for Optimization
SIAM Frontiers in Applied Mathematics, 1999

A. C. Davison and D. V. Hinkley
Bootstrap Methods and Their Applications
CUP, 1997

Booth, J.G., Hall, P. and Wood, A.T.A.
Balanced importance resampling for the bootstrap.
Annals of Statistics, 21, 286-298, 1993

Davison, A.C. and Hinkley, D.V.
Bootstrap Methods and Their Application
Cambridge University Press, 1997

Efron, B. and Tibshirani, R.
An Introduction to the Bootstrap
Chapman & Hall, 1993

`cfBuild` *Create a highly optimised ensemble of RBF SVM classifiers*

Description

The `cfBuild` function creates a highly optimised ensemble of radial basis function (RBF) support vector machines (SVMs). The `cfBuild` function takes care of all aspects of the SVM optimisation, internally splitting the supplied data into separate training and testing subsets using a bootstrapping approach coupled with a heuristic optimisation algorithm and parallel processing to minimise computation time. The ensemble object can then be used to classify newly acquired data using the `cfPredict` function.

Usage

```
cfBuild(inputData, inputClass, ...)

## Default S3 method:
cfBuild(inputData, inputClass, bootNum = 100, ensNum = 100, parallel=TRUE,
        cpus = NULL, type = "SOCK", socketHosts = NULL, ...)
```

Arguments

`inputData`	The input data matrix as provided by the user (mandatory field).
`inputClass`	The input class vector as provided by the user (mandatory field).
`bootNum`	The number of bootstrap iterations in the optimisation process. By default, the bootNum value is set to 100.
`ensNum`	The number of classifiers that form the classification ensemble. By default, the ensNum value is set to 100.
`parallel`	Boolean value that determines parallel or sequential execution. By default set to `TRUE`. For more details, see sfInit.
`cpus`	Numeric value that provides the number of CPUs requested for the cluster. For more details, see sfInit.
`type`	The type of cluster. It can take the values 'SOCK', 'MPI', 'PVM' or 'NWS'. By default, type is equal to 'SOCK'. For more details, see sfInit.
`socketHosts`	Host list for socket clusters. Only needed for socketmode (SOCK) and if using more than one machines (if using only your local machine (localhost) no list is needed). For more details, see sfInit.
`...`	The remaining optional fields.

Details

For a given input dataset D, a random fraction of samples is removed and kept aside as an independent test set during the training process of the model. This selection of samples forms the dataset D_{test}. This test set typically comprises a third of the original samples. Using a stratified holdout approach, the test set consists of the same balance of sample classes as the initial dataset D. The remaining samples that are not selected, form the training set D_{train}. Since the test set is kept aside during the whole training process, the risk of overfitting is minimised.

In the case of bootstrapping, a bootstrap training set $D_{bootTrain}$ is created by randomly picking n samples with replacement from the training dataset D_{train}. The total size of $D_{bootTrain}$ is equal to the size of D_{train}. Since bootstrapping is based on sampling with replacement, any given sample could be present multiple times within the same bootstrap training set. The remaining samples not found in the bootstrap training set make up the bootstrap test set $D_{bootTest}$. To avoid reliance on one specific bootstrapping split, bootstrapping is repeated at least $bootNum$ times until a clear winning parameter combination emerges.

Ultimately, the optimal parameters are used to train a new classifier with the full D_{train} dataset and test it on the independent test set D_{test}, which has been left aside during the entire optimisation process. Even though the approach described thus far generates an excellent classifier, the random selection of test samples in the initial split may have been fortunate. For a more accurate and reliable overview, the whole process should be repeated a minimum of 100 times (default value of $ensNum$) or until a stable average classification rate emerges. The output of this repetition consists of at least $ensNum$ individual classification models built using the optimum parameter settings. Rather than isolating a single classifier, all individual classification models are fused into a classification ensemble.

Value

The `cfBuild` function returns an object in the form of an R list. The attributes of the list can be accessed by executing the attributes command. More specifically, the list of attributes includes:

`testAcc`	A vector of the test accuracies (%CC) of the single classifiers in the ensemble.

`trainAcc`	A vector of the train accuracies of the single classifiers in the ensemble.
`optGamma`	A vector of the optimal gamma values.
`optCost`	A vector of the optimal cost values.
`totalTime`	The overall execution time of the ensemble in seconds.
`runTime`	The individual execution times of the classifiers within the ensemble in seconds.
`confMatr`	A list featuring the confusion matrices of the classifiers in the ensemble.
`predClasses`	A list with the vectors of predicted classes as predicted by each classifier.
`testClasses`	A list with the vectors of true classes of the test class.
`missNames`	In case that the names of the samples (rows) are supplied in the input data matrix, the `missNames` attribute returns the names of the missclassified samples.
`accNames`	In case that the names of the samples (rows) are supplied in the input data matrix, the `accNames` attribute returns the names of the correctly classified samples.
`testIndx`	The randomly selected samples (rows) that were used in this instance to create the train and test data.
`svmModel`	A list containing the generated SVM models in the ensemble.

Note

- Data are scaled internally, usually yielding better results. The parameters of SVM-models usually *must* be tuned to yield sensible results! For more information, see function svm.
- The `cfBuild` function does not force an upper limit for the bootNum, ensNum and cpus parameters to the users. However, it is advisable not to use extremely high values for these parameters.

Author(s)

Adapted functionality by Eleni Chatzimichali (ea.chatzimichali@gmail.com)
Author of the SVM functions: David Meyer (<David.Meyer@R-project.org>)
(based on C/C++-code by Chih-Chung Chang and Chih-Jen Lin)
Author of Scilab neldermead module: Michael Baudin (INRIA - Digiteo)
Author of Scilab R adaptation: Sebastien Bihorel (<sb.pmlab@gmail.com>)
Authors of bootstrap functions: Angelo Canty and Brian Ripley (originally by Angelo Canty for S)

References

There are many references explaining the concepts behind the functionality of this function. Among them are :

Chang, Chih-Chung and Lin, Chih-Jen:
LIBSVM: a library for Support Vector Machines
`http://www.csie.ntu.edu.tw/~cjlin/libsvm`

Exact formulations of models, algorithms, etc. can be found in the document:
Chang, Chih-Chung and Lin, Chih-Jen:
LIBSVM: a library for Support Vector Machines
`http://www.csie.ntu.edu.tw/~cjlin/papers/libsvm.ps.gz`

More implementation details and speed benchmarks can be found on:
Rong-En Fan and Pai-Hsune Chen and Chih-Jen Lin:
Working Set Selection Using the Second Order Information for Training SVM
`http://www.csie.ntu.edu.tw/~cjlin/papers/quadworkset.pdf`

Spendley, W. and Hext, G. R. and Himsworth, F. R.
Sequential Application of Simplex Designs in Optimisation and Evolutionary Operation
American Statistical Association and American Society for Quality, 1962

Nelder, J. A. and Mead, R.
A Simplex Method for Function Minimization
The Computer Journal, 1965

C. T. Kelley
Iterative Methods for Optimization
SIAM Frontiers in Applied Mathematics, 1999

A. C. Davison and D. V. Hinkley
Bootstrap Methods and Their Applications
CUP, 1997

Booth, J.G., Hall, P. and Wood, A.T.A.
Balanced importance resampling for the bootstrap.
Annals of Statistics, 21, 286-298, 1993

Davison, A.C. and Hinkley, D.V.
Bootstrap Methods and Their Application
Cambridge University Press, 1997

Efron, B. and Tibshirani, R.
An Introduction to the Bootstrap
Chapman & Hall, 1993

See Also

cfPredict, cfPermute

Examples

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

irisClass <- iris[,5]
irisData  <- iris[,-5]

# Construct a classification ensemble with 100 classifiers and 100 bootstrap
# iterations during optimisation

ens <- cfBuild(inputData = irisData, inputClass = irisClass, bootNum = 100,
               ensNum = 100, parallel = TRUE, cpus = 4, type = "SOCK")

# List of attributes available for each classifier in the ensemble
attributes(ens)

# Get the overall average Test and Train accuracy
getAvgAcc(ens)$Test
getAvgAcc(ens)$Train
```

```
# Get all the test accuracies and the average test accuracy in the ensemble
ens$testAcc    # alternatively, getAcc(ens)$Test
ens$trainAcc   # alternatively, getAcc(ens)$rain

# Randomly generate test data to find out their classes using the generated ensemble
# 400 points are selected at random, which results in 100 samples (rows).
# Predict the classes of the data using the classifiers in the constructed ensemble

testMatr <- matrix(runif(400)*100, ncol = ncol(irisData))
predRes  <- cfPredict(ens, testMatr)

## End(Not run)
```

---

cfPermute *Permutation testing to indicate statistical significance of performance*

Description

The `cfPermute` function performs permutation testing on a classification ensemble produced by `cfBuild`. This is essentially a comparison between the classification performance achieved for a given dataset and the performance that would be achieved by random chance. It therefore provides an indication of significance of the performance of a classifier.

Usage

```
cfPermute(inputData, inputClass, bootNum = 100, ensNum = 100, permNum = 100,
          parallel = TRUE, cpus = NULL, type = "SOCK", socketHosts = NULL,
          progressBar = TRUE)
```

Arguments

`inputData`	The input data matrix as provided by the user (mandatory field).
`inputClass`	The input class vector as provided by the user (mandatory field).
`bootNum`	The number of bootstrap iterations during the optimisation process. By default, the value is set to 100.
`ensNum`	The number of classifiers that constitute the ensemble for each permutation. By default, the value is set to 100.
`permNum`	The number of permutations to be executed. By default, the value is set to 100.
`parallel`	Boolean value that determines parallel or sequential execution. By default set to `TRUE`. For more details, see sfInit.
`cpus`	Numeric value that provides the number of CPUs requested for the cluster. For more details, see sfInit.
`type`	The type of cluster. It can take the values ‘SOCK’, ‘MPI’, ‘PVM’ or ‘NWS’. By default, type is equal to ‘SOCK’. For more details, see sfInit.
`socketHosts`	Host list for socket clusters. Only needed for socketmode (SOCK) and if using more than one machines (if using only your local machine (localhost) no list is needed). For more details, see sfInit.
`progressBar`	Boolean value that determines whether a progress bar should be displayed. By default set to `TRUE`.

Details

Permutation testing is a widely-applied process used in order to provide an indication of the statistical significance of the classification results. In a permutation test, the entries of the original class vector (`inputClass`) are randomly shuffled, while the class distribution is preserved. This approach destroys all the sample membership information since the samples of a permuted dataset correspond to randomly assigned classes. The whole model building process as described in `cfBuild` is once more repeated for the "false" (permuted) classes. In general, permutation testing should be performed at least 100 times (default value of `permNum`) until a stable distribution of results is obtained.

Value

The `cfPermute` function returns an object in the form of an R list. The attributes of the list can be accessed by executing the attributes command. More specifically, the list of attributes includes:

`avgAcc`	The average test accuracy across all ensembles within each permutation iteration.
`totalTime`	The overall execution time of permutation testing.
`execTime`	The individual execution times for each permutation round.
`permList`	For each permutation iteration, a new object (list) is generated by the function `cfBuild` using as input the initial data and the permuted class. This attribute will have the same length - the same number of elements - as the `permNum` attribute specified in the `cfPermute` function. For more information on the arguments of the object, see `cfBuild`

References

Good, P. I.
Permutation, Parametric and Bootstrap Tests of Hypotheses
3rd ed, Springer-Verlag New York Inc, Dordrecht, 2006

Hesterberg, T., Moore, D. S., Monaghan, S., Clipson, A. and Epstein, R. *Bootstrap methods and permutation tests*
Introduction to the Practice of Statistics, vol. 5, pp. 1-70, 2005

See Also

`getPerm5Num`, `ggPermHist`

Examples

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

irisClass <- iris[,5]
irisData  <- iris[,-5]

ens <- cfBuild(irisData, irisClass, bootNum = 100, ensNum = 100, parallel = TRUE,
               cpus = 4, type = "SOCK")

# Execute 5 permutation rounds; in each permutation test, an ensemble of 20 classifiers
# is constructed, each running 10 bootstrap iterations during the optimization process
# The default values for permutation testing are ensNum = bootNum = permNum = 100
```

```
permObj <- cfPermute(irisData, irisClass, bootNum = 10, ensNum = 20, permNum = 5, parallel = TRUE,
                     cpus = 4, type = "SOCK")

# List of attributes for each permutation
attributes(permObj)

# Get the vector of averaged accuracies, one for each permutation
# (each permutation is an independent classification ensemble)
permObj$avgAcc

# Get the overall elapsed time for the permutation process
permObj$totalTime[3]

# Get the vector of individual execution times for each permutation
permObj$execTime

# Access the first ensemble in the permutation list
permObj$permList[[1]]

## End(Not run)
```

`cfPredict` *Predict the class of new data using an existing ensemble*

Description

The `cfPredict` function uses a classification ensemble object created by `cfBuild` to predict the class(es) of one or more samples described by a given data matrix. The function returns the predicted classes for each sample, together with a confidence score (between 0 and 100) which equates to the percentage of SVMs within the classifier that voted for the reported class.

Usage

```
cfPredict(ensObj, newInputData)
```

Arguments

`ensObj`	The classification ensemble (in the form of an R list) as generated by `cfBuild`
`newInputData`	A new independent dataset with unknown classes. The new dataset must have exactly the same number of columns as the `inputData`, passed as an argument in `cfBuild`.

Value

The `cfPredict` function returs a matrix of the predicted classes as generated by a majority vote between the classifiers in the ensemble along with their confidence scores (the % percentage of the predicted class in the majority vote) for each sample.

See Also

`cfBuild`

Examples

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

irisClass <- iris[,5]
irisData  <- iris[,-5]

# Construct a classification ensemble with 100 classifiers and 100 bootstrap
# iterations during optimisation

ens <- cfBuild(irisData, irisClass, bootNum = 100, ensNum = 100, parallel = TRUE,
               cpus = 4, type = "SOCK")

# Randomly generate test data to find out their classes using the generated ensemble
# 400 points are selected at random, which results in 100 samples (rows).
# Predict the classes of the data using the classifiers in the constructed ensemble

testMatr <- matrix(runif(400)*100, ncol=4)
predRes  <- cfPredict(ens, testMatr)

## End(Not run)
```

`getAcc` *Get the accuracies of a classification ensemble*

Description

The `getAcc` function returns the test and train accuracies for all the classifiers within a classification ensemble as generated by `cfBuild`.

Usage

```
getAcc(ensObj)
```

Arguments

`ensObj`	The classification ensemble (in the form of an R list) as generated by `cfBuild`

Value

The `getAcc` function returns a list with two named (`Test` and `Train`) vectors, equal to the overall test accuracies (%CC) and overall train accuracies of the classifiers within the ensemble. The attributes of the list can be accessed by executing the attributes command.

See Also

`getAvgAcc`

Examples

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

irisClass <- iris[,5]
irisData  <- iris[,-5]

ens <- cfBuild(inputData = irisData, inputClass = irisClass, bootNum = 100,
               ensNum = 100, parallel = TRUE, cpus = 4, type = "SOCK")

# Get the attributes provided by the getAcc function
attributes(getAcc(ens))

# Get both the vectors of test and train accuracies from the classifiers in the ensemble
getAcc(ens)

# Get the vector of test accuracies from the classifiers in the ensemble
getAcc(ens)$Test

# Get the vector of train accuracies from the classifiers in the ensemble
getAcc(ens)$Train

## End(Not run)
```

---

getAvgAcc — *Get the average accuracies of a classification ensemble*

Description

The `avgTestAcc` function returns the average test accuracy (%CC) and average train accuracy of an ensemble of classifiers as generated by the function `cfBuild`.

Usage

```
getAvgAcc(ensObj)
```

Arguments

`ensObj`	The classification ensemble (in the form of an R list) as generated by `cfBuild`

Value

The `avgTestAcc` function returns a list with two named (`Test` and `Train`) numerical values, equal to the average overall test accuracy (%CC) and the average overall train accuracy of the ensemble. The attributes of the list can be accessed by executing the attributes command.

See Also

`getAcc`

Examples

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

irisClass <- iris[,5]
irisData  <- iris[,-5]

ens <- cfBuild(inputData = irisData, inputClass = irisClass, bootNum = 100,
               ensNum = 100, parallel = TRUE, cpus = 4, type = "SOCK")

# Get the attributes provided by the getAvgAcc function
attributes(getAvgAcc(ens))

# Get the average test and train accuracies within the ensemble
getAvgAcc(ens)

# Get the average test accuracy
getAvgAcc(ens)$Test

# Get the average train accuracy
getAvgAcc(ens)$Train

## End(Not run)
```

`getConfMatr` *Confusion matrix summarising the performance of an ensemble*

Description

The `getConfMatr` function returns a confusion matrix from an ensemble created by `cfBuild`. The matrix is populated using class predictions for test data, predictions which were obtained during the building and optimisation of the classifier. Each column of the confusion matrix represents the instances in a predicted class, while each row represents the instances in an actual class. Cells in the matrix indicate the percentage of samples predicted to belong to each class. In a perfect ensemble the diagonal elements would all be 100%.

Usage

```
getConfMatr(ensObj)
```

Arguments

`ensObj`	The classification ensemble (in the form of an R list) as generated by `cfBuild`

Note

A graphical representation of this information is provided by `ggClassPred`.

Examples

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

irisClass <- iris[,5]
irisData  <- iris[,-5]

ens <- cfBuild(irisData, irisClass, bootNum = 100, ensNum = 100, parallel = TRUE,
               cpus = 4, type = "SOCK")

# Get the confusion matrix summarising the performance of the ensemble
getConfMatr(ens)

## End(Not run)
```

---

getOptParam *Get the optimal SVM hyperparameters of a classification ensemble*

Description

To allow detailed reporting of the methods used to create classifiers, the getOptParam function retrieves the optimum hyperparameters for each SVM within an ensemble classifier, built using cfBuild.

Usage

```
getOptParam(ensObj)
```

Arguments

ensObj	The classification ensemble (in the form of an R list) as generated by cfBuild

Value

Returns a matrix containing the optimal gamma and optimal cost for each SVM in the classification ensemble. For information about what these hyperparameters are and how they are determined, see the documentation for cfBuild.

Examples

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

irisClass <- iris[,5]
irisData  <- iris[,-5]

ens <- cfBuild(irisData, irisClass, bootNum = 100, ensNum = 100, parallel = TRUE,
               cpus = 4, type = "SOCK")

# Get the optimal SVM hyperparameters of the classification ensemble
optParam <- getOptParam(ens)
optParam

## End(Not run)
```

getPerm5Num *Get descriptive statistics from a permutation object*

Description

The getPerm5Num function returns the "five number summary", a descriptive statistic that consists of the minimum, first (lower) quartile, median, third (upper) quartile and maximum value of a given distribution. In this case, the function is applied directly on the output of permutation testing, generated by the cfPermute function.

Usage

```
getPerm5Num(permObj)
```

Arguments

permObj	The permutation object as generated by cfPermute

Value

The getPerm5Num function returns an R object in the form of a list that contains the five number summary. The names of the returned statistics can be viewed by using the function attributes.

Examples

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

irisClass <- iris[,5]
irisData  <- iris[,-5]

permObj <- cfPermute(irisData, irisClass, bootNum = 10, ensNum = 20, permNum = 5,
                     parallel = TRUE, cpus = 4, type = "SOCK")

getPerm5Num(permObj)
getPerm5Num(permObj)$median
getPerm5Num(permObj)$minimum
getPerm5Num(permObj)$maximum
getPerm5Num(permObj)$upperQ
getPerm5Num(permObj)$lowerQ

## End(Not run)
```

ggClassPred *Barplot of the per class accuracies.*

Description

The ggClasPred function generates a barplot with the per class accuracies (%) for all the correctly classified and misclassified samples in the classification ensemble.

Usage

```
ggClassPred(ensObj, position = "stack", displayAll = FALSE, showText = FALSE,
            xlabel = NULL, ylabel = NULL, cbPalette = FALSE, fillBrewer = FALSE)
```

Arguments

ensObj	The classification ensemble (in the form of an R list) as generated by cfBuild
position	The position may be equal to either "stack" or "dodge".
displayAll	Boolean value, by default set to FALSE. When displayAll= FALSE, only the percentages of correctly classified samples are displayed in the barplot. If displayAll = TRUE, the percentages of all classified and missclassified samples are depicted in the barplot.
showText	Boolean value, by default set to FALSE. If showText=TRUE, then the per class accuracies (%) for all classifiers in the ensemble are displayed in the plot.
xlabel	A sub title for the x axis (optional field).
ylabel	A sub title for the y axis (optional field).
cbPalette	If TRUE, enable a color-blind-friendly palette.
fillBrewer	If TRUE, enable a color scale taken from the RColorBrewer package.

Examples

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

irisClass <- iris[,5]
irisData  <- iris[,-5]

ens <- cfBuild(irisData, irisClass, bootNum = 100, ensNum = 100, parallel = TRUE,
               cpus = 4, type = "SOCK")

# Show the percentages of correctly classified samples in
# a barplot with or without text respectively

ggClassPred(ens)
ggClassPred(ens, showText = TRUE)

# Show the percentages of classified and missclassified samples
# in a barplot simultaneously with and without text

ggClassPred(ens, displayAll = TRUE)
ggClassPred(ens, position="stack", displayAll = TRUE)
ggClassPred(ens, position="stack", displayAll = TRUE, showText = TRUE)

# Alernatively, using a dodge position
ggClassPred(ens, position = "dodge", displayAll = TRUE)
ggClassPred(ens, position = "dodge", displayAll = TRUE, showText = TRUE)

## End(Not run)
```

ggEnsHist *Ensemble Histograms*

Description

The ggEnsHist function generates a histogram of the ensemble results as generated by cfBuild.

Usage

```
ggEnsHist(ensObj, density = FALSE, percentiles = FALSE, mean = FALSE, median = FALSE)
```

Arguments

ensObj	The classification ensemble (in the form of an R list) as generated by cfBuild
density	Boolean value, by default equal to FALSE. If density = FALSE, the histogram depicts frequencies, the counts component of the result. Instead, for density = TRUE, probability densities are plotted.
percentiles	Boolean value, by default equal to FALSE. If percentiles = TRUE, the upper and lower percentiles of the distribution are depicted in the plot.
mean	Boolean value, by default equal to FALSE. If mean = TRUE, the mean of the distribution is depicted in the plot.
median	Boolean value, by default equal to FALSE. If median = TRUE, the median of the distribution is depicted in the plot.

See Also

ggEnsHist

Examples

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

irisClass <- iris[,5]
irisData  <- iris[,-5]

ens <- cfBuild(irisData, irisClass, bootNum = 100, ensNum = 100, parallel = TRUE,
               cpus = 4, type = "SOCK")

# Basic histogram of the test accuracies in the ensemble
ggEnsHist(ens)

# Density plot of the test accuracies in the ensemble
ggEnsHist(ens, density = TRUE)

# Density plot that highlights additional descriptive statistics
ggEnsHist(ens, density = TRUE, percentiles=TRUE)
ggEnsHist(ens, density = TRUE, percentiles=TRUE, mean=TRUE)
ggEnsHist(ens, density = TRUE, percentiles=TRUE, median=TRUE)

## End(Not run)
```

ggEnsTrend *Trend of the test accuracies*

Description

The ggEnsTrend function displays the average test accuracies for every new classifier added to the ensemble, as constructed by the cfBuild function.

Usage

```
ggEnsTrend(ensObj, xlabel = NULL, ylabel = NULL, showText = FALSE, xlims = NULL,
             ylims = NULL)
```

Arguments

ensObj	The R object as generated by cfBuild
xlabel	A sub title for the x axis (optional field).
ylabel	A sub title for the y axis (optional field).
showText	Boolean value, by default set to FALSE. If showText=TRUE, then the values of all test accuracies in the ensemble are displayed in the plot.
xlims	A vector of numeric values that specifies the minimum and maximum values in the x axis.
ylims	A vector of numeric values that specifies the minimum and maximum values in the y axis.

Examples

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

irisClass <- iris[,5]
irisData  <- iris[,-5]

ens <- cfBuild(irisData, irisClass, bootNum = 100, ensNum = 100, parallel = TRUE,
               cpus = 4, type = "SOCK")

# Plot the trend of the test accuracies in the ensemble
ggEnsTrend(ens)

# Plot with text
ggEnsTrend(ens, showText  = TRUE)

# Plot with text; set different limits on y axis
ggEnsTrend(ens, showText  = TRUE, ylims=c(90, 100))

## End(Not run)
```

ggPermHist *Permutation Histograms*

Description

The ggPermHist function generates a histogram of the permutation results as generated by cfPermute.

Usage

```
ggPermHist(permObj, density = FALSE, percentiles = FALSE, mean = FALSE,
            median = FALSE)
```

Arguments

permObj	The permutation object as generated by cfPermute
density	Boolean value, by default equal to FALSE. If density = FALSE, the histogram depicts frequencies, the counts component of the result. Instead, for density = TRUE, probability densities are plotted.
percentiles	Boolean value, by default equal to FALSE. If percentiles = TRUE, the upper and lower percentiles of the distribution are depicted in the plot.
mean	Boolean value, by default equal to FALSE. If mean = TRUE, the mean of the distribution is depicted in the plot.
median	Boolean value, by default equal to FALSE. If median = TRUE, the median of the distribution is depicted in the plot.

See Also

ggEnsHist

Examples

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

irisClass <- iris[,5]
irisData  <- iris[,-5]

permObj <- cfPermute(irisData, irisClass, bootNum = 10, ensNum = 20, permNum = 5,
                     parallel = TRUE, cpus = 4, type = "SOCK")

# Basic histogram of the averaged test accuracies during permutation
ggPermHist(permObj)

# Density plot
ggPermHist(permObj, density=TRUE)

# Density plot that highlights additional descriptive statistics
ggPermHist(permObj, density=TRUE, percentiles = TRUE, mean = TRUE)
ggPermHist(permObj, density=TRUE, percentiles = TRUE, median = TRUE)

## End(Not run)
```

Index