Package 'ssc'

July 23, 2025

Type Package

Title Semi-Supervised Classification Methods

Version 2.1-0

Description Provides a collection of self-labeled techniques for semi-supervised classification. In semi-supervised classification, both labeled and unlabeled data are used to train a classifier. This learning paradigm has obtained promising results, specifically in the presence of a reduced set of labeled examples. This package implements a collection of self-labeled techniques to construct a classification model. This family of techniques enlarges the original labeled set using the most confident predictions to classify unlabeled data. The techniques implemented can be applied to classification problems in several domains by the specification of a supervised base classifier. At low ratios of labeled data, it can be shown to perform better than classical supervised classifiers.

Depends R (>= 3.2.3)

Imports stats, proxy

Suggests caret, e1071, C50, kernlab, testthat, timeDate, stringi, R.rsp

VignetteBuilder R.rsp

License GPL (>= 3)

URL https://github.com/mabelc/SSC

BugReports https://github.com/mabelc/SSC/issues

Encoding UTF-8

RoxygenNote 6.1.0

NeedsCompilation no

Author Mabel González [aut] (ORCID: <https://orcid.org/0000-0003-0152-444X>), Osmani Rosado-Falcón [aut] (ORCID: <https://orcid.org/0000-0002-2639-3354>), José Daniel Rodríguez [aut] (ORCID: <https://orcid.org/0000-0002-8489-4106>), Christoph Bergmeir [ths, cre] (ORCID: <https://orcid.org/0000-0002-3665-9021>),

Isaac Triguero [ctb] (ORCID: <https://orcid.org/0000-0002-0150-0651>),
José Manuel Benítez [ths] (ORCID:
<https://orcid.org/0000-0002-2346-0793>)

Maintainer Christoph Bergmeir <c.bergmeir@decsai.ugr.es>

Repository CRAN

Date/Publication 2019-12-15 21:30:02 UTC

Contents

`coBC` *CoBC method*

Description

Co-Training by Committee (CoBC) is a semi-supervised learning algorithm with a co-training style. This algorithm trains `N` classifiers with the learning scheme defined in the `learner` argument using a reduced set of labeled examples. For each iteration, an unlabeled example is labeled for a classifier if the most confident classifications assigned by the other `N-1` classifiers agree on the labeling proposed. The unlabeled examples candidates are selected randomly from a pool of size `u`.

Usage

```
coBC(x, y, x.inst = TRUE, learner, learner.pars = NULL,
  pred = "predict", pred.pars = NULL, N = 3, perc.full = 0.7,
  u = 100, max.iter = 50)
```

Arguments

x	An object that can be coerced to a matrix. This object has two possible interpretations according to the value set in the x.inst argument: a matrix with the training instances where each row represents a single instance or a precomputed (distance or kernel) matrix between the training examples.
y	A vector with the labels of the training instances. In this vector the unlabeled instances are specified with the value NA.
x.inst	A boolean value that indicates if x is or not an instance matrix. Default is TRUE.
learner	either a function or a string naming the function for training a supervised base classifier, using a set of instances (or optionally a distance matrix) and it's corresponding classes.
learner.pars	A list with additional parameters for the learner function if necessary. Default is NULL.
pred	either a function or a string naming the function for predicting the probabilities per classes, using the base classifiers trained with the learner function. Default is "predict".
pred.pars	A list with additional parameters for the pred function if necessary. Default is NULL.
N	The number of classifiers used as committee members. All these classifiers are trained using the gen.learner function. Default is 3.
perc.full	A number between 0 and 1. If the percentage of new labeled examples reaches this value the self-labeling process is stopped. Default is 0.7.
u	Number of unlabeled instances in the pool. Default is 100.
max.iter	Maximum number of iterations to execute in the self-labeling process. Default is 50.

Details

This method trains an ensemble of diverse classifiers. To promote the initial diversity the classifiers are trained from the reduced set of labeled examples by Bagging. The stopping criterion is defined through the fulfillment of one of the following criteria: the algorithm reaches the number of iterations defined in the max.iter parameter or the portion of unlabeled set, defined in the perc.full parameter, is moved to the enlarged labeled set of the classifiers.

Value

A list object of class "coBC" containing:

model The final N base classifiers trained using the enlarged labeled set.

model.index List of N vectors of indexes related to the training instances used per each classifier. These indexes are relative to the y argument.

instances.index The indexes of all training instances used to train the N models. These indexes include the initial labeled instances and the newly labeled instances. These indexes are relative to the y argument.

model.index.map List of three vectors with the same information in `model.index` but the indexes are relative to `instances.index` vector.

classes The levels of y factor.

pred The function provided in the pred argument.

pred.pars The list provided in the `pred.pars` argument.

x.inst The value provided in the `x.inst` argument.

References

Avrim Blum and Tom Mitchell.
Combining labeled and unlabeled data with co-training.
In Eleventh Annual Conference on Computational Learning Theory, COLT' 98, pages 92-100, New York, NY, USA, 1998. ACM. ISBN 1-58113-057-0. doi: 10.1145/279943.279962.

Examples

```
library(ssc)

## Load Wine data set
data(wine)

cls <- which(colnames(wine) == "Wine")
x <- wine[, -cls] # instances without classes
y <- wine[, cls] # the classes
x <- scale(x) # scale the attributes

## Prepare data
set.seed(20)
# Use 50% of instances for training
tra.idx <- sample(x = length(y), size = ceiling(length(y) * 0.5))
xtrain <- x[tra.idx,] # training instances
ytrain <- y[tra.idx]  # classes of training instances
# Use 70% of train instances as unlabeled set
tra.na.idx <- sample(x = length(tra.idx), size = ceiling(length(tra.idx) * 0.7))
ytrain[tra.na.idx] <- NA # remove class information of unlabeled instances

# Use the other 50% of instances for inductive testing
tst.idx <- setdiff(1:length(y), tra.idx)
xitest <- x[tst.idx,] # testing instances
yitest <- y[tst.idx] # classes of testing instances

## Example: Training from a set of instances with 1-NN as base classifier.
set.seed(1)
m1 <- coBC(x = xtrain, y = ytrain,
           learner = caret::knn3,
```

```
           learner.pars = list(k = 1),
           pred = "predict")
pred1 <- predict(m1, xitest)
table(pred1, yitest)

## Example: Training from a distance matrix with 1-NN as base classifier.
dtrain <- proxy::dist(x = xtrain, method = "euclidean", by_rows = TRUE)
set.seed(1)
m2 <- coBC(x = dtrain, y = ytrain, x.inst = FALSE,
           learner = ssc::oneNN,
           pred = "predict",
           pred.pars = list(distance.weighting = "none"))
ditest <- proxy::dist(x = xitest, y = xtrain[m2$instances.index,],
                      method = "euclidean", by_rows = TRUE)
pred2 <- predict(m2, ditest)
table(pred2, yitest)

## Example: Training from a set of instances with SVM as base classifier.
learner <- e1071::svm
learner.pars <- list(type = "C-classification", kernel="radial",
                     probability = TRUE, scale = TRUE)
pred <- function(m, x){
  r <- predict(m, x, probability = TRUE)
  prob <- attr(r, "probabilities")
  prob
}
set.seed(1)
m3 <- coBC(x = xtrain, y = ytrain,
           learner = learner,
           learner.pars = learner.pars,
           pred = pred)
pred3 <- predict(m3, xitest)
table(pred3, yitest)

## Example: Training from a set of instances with Naive-Bayes as base classifier.
set.seed(1)
m4 <- coBC(x = xtrain, y = ytrain,
           learner = function(x, y) e1071::naiveBayes(x, y),
           pred = "predict",
           pred.pars = list(type = "raw"))
pred4 <- predict(m4, xitest)
table(pred4, yitest)

## Example: Training from a set of instances with C5.0 as base classifier.
set.seed(1)
m5 <- coBC(x = xtrain, y = ytrain,
           learner = C50::C5.0,
           pred = "predict",
           pred.pars = list(type = "prob"))

pred5 <- predict(m5, xitest)
table(pred5, yitest)
```

coBCCombine *Combining the hypothesis*

Description

This function combines the probabilities predicted by the committee of classifiers.

Usage

```
coBCCombine(h.prob, classes)
```

Arguments

h.prob	A list of probability matrices.
classes	The classes in the same order that appear in the columns of each matrix in h.prob.

Value

A probability matrix

coBCG *CoBC generic method*

Description

CoBC is a semi-supervised learning algorithm with a co-training style. This algorithm trains N classifiers with the learning scheme defined in gen.learner using a reduced set of labeled examples. For each iteration, an unlabeled example is labeled for a classifier if the most confident classifications assigned by the other N-1 classifiers agree on the labeling proposed. The unlabeled examples candidates are selected randomly from a pool of size u.

Usage

```
coBCG(y, gen.learner, gen.pred, N = 3, perc.full = 0.7, u = 100,
  max.iter = 50)
```

Arguments

y	A vector with the labels of training instances. In this vector the unlabeled instances are specified with the value NA.
gen.learner	A function for training N supervised base classifiers. This function needs two parameters, indexes and cls, where indexes indicates the instances to use and cls specifies the classes of those instances.
gen.pred	A function for predicting the probabilities per classes. This function must be two parameters, model and indexes, where the model is a classifier trained with gen.learner function and indexes indicates the instances to predict.
N	The number of classifiers used as committee members. All these classifiers are trained using the gen.learner function. Default is 3.
perc.full	A number between 0 and 1. If the percentage of new labeled examples reaches this value the self-labeling process is stopped. Default is 0.7.
u	Number of unlabeled instances in the pool. Default is 100.
max.iter	Maximum number of iterations to execute in the self-labeling process. Default is 50.

Details

coBCG can be helpful in those cases where the method selected as base classifier needs a learner and pred functions with other specifications. For more information about the general coBC method, please see coBC function. Essentially, coBC function is a wrapper of coBCG function.

Value

A list object of class "coBCG" containing:

model The final N base classifiers trained using the enlarged labeled set.

model.index List of N vectors of indexes related to the training instances used per each classifier. These indexes are relative to the y argument.

instances.index The indexes of all training instances used to train the N models. These indexes include the initial labeled instances and the newly labeled instances. These indexes are relative to the y argument.

model.index.map List of three vectors with the same information in model.index but the indexes are relative to instances.index vector.

classes The levels of y factor.

Examples

```
library(ssc)

## Load Wine data set
data(wine)

cls <- which(colnames(wine) == "Wine")
x <- wine[, -cls] # instances without classes
```

```
y <- wine[, cls] # the classes
x <- scale(x) # scale the attributes

## Prepare data
set.seed(20)
# Use 50% of instances for training
tra.idx <- sample(x = length(y), size = ceiling(length(y) * 0.5))
xtrain <- x[tra.idx,] # training instances
ytrain <- y[tra.idx]  # classes of training instances
# Use 70% of train instances as unlabeled set
tra.na.idx <- sample(x = length(tra.idx), size = ceiling(length(tra.idx) * 0.7))
ytrain[tra.na.idx] <- NA # remove class information of unlabeled instances

# Use the other 50% of instances for inductive testing
tst.idx <- setdiff(1:length(y), tra.idx)
xitest <- x[tst.idx,] # testing instances
yitest <- y[tst.idx] # classes of testing instances

## Example: Training from a set of instances with 1-NN (knn3) as base classifier.
gen.learner1 <- function(indexes, cls)
  caret::knn3(x = xtrain[indexes, ], y = cls, k = 1)
gen.pred1 <- function(model, indexes)
  predict(model, xtrain[indexes, ])

set.seed(1)
md1 <- coBCG(y = ytrain, gen.learner1, gen.pred1)

# Predict probabilities per instances using each model
h.prob <- lapply(
  X = md1$model,
  FUN = function(m) predict(m, xitest)
)
# Combine the predictions
cls1 <- coBCCombine(h.prob, md1$classes)
table(cls1, yitest)

## Example: Training from a distance matrix with 1-NN (oneNN) as base classifier.
dtrain <- as.matrix(proxy::dist(x = xtrain, method = "euclidean", by_rows = TRUE))
gen.learner2 <- function(indexes, cls) {
  m <- ssc::oneNN(y = cls)
  attr(m, "tra.idxs") <- indexes
  m
}

gen.pred2 <- function(model, indexes)  {
  tra.idxs <- attr(model, "tra.idxs")
  d <- dtrain[indexes, tra.idxs]
  prob <- predict(model, d, distance.weighting = "none")
  prob
}

set.seed(1)
md2 <- coBCG(y = ytrain, gen.learner2, gen.pred2)
```

```
# Predict probabilities per instances using each model
ditest <- proxy::dist(x = xitest, y = xtrain[md2$instances.index,],
                      method = "euclidean", by_rows = TRUE)

h.prob <- list()
ninstances <- nrow(dtrain)
for(i in 1:length(md2$model)){
  m <- md2$model[[i]]
  D <- ditest[, md2$model.index.map[[i]]]
  h.prob[[i]] <- predict(m, D)
}
# Combine the predictions
cls2 <- coBCCombine(h.prob, md2$classes)
table(cls2, yitest)
```

`coffee` *Time series data set*

Description

A dataset containing 56 times series z-normalized. Time series length is 286.

Usage

```
data(coffee)
```

Format

A data frame with 56 rows and 287 variables including the class.

`democratic` *Democratic method*

Description

Democratic Co-Learning is a semi-supervised learning algorithm with a co-training style. This algorithm trains N classifiers with different learning schemes defined in list `gen.learners`. During the iterative process, the multiple classifiers with different inductive biases label data for each other.

Usage

```
democratic(x, y, x.inst = TRUE, learners, learners.pars = NULL,
  preds = rep("predict", length(learners)), preds.pars = NULL)
```

Arguments

x	A object that can be coerced as matrix. This object has two possible interpretations according to the value set in the x.inst argument: a matrix with the training instances where each row represents a single instance or a precomputed (distance or kernel) matrix between the training examples.
y	A vector with the labels of the training instances. In this vector the unlabeled instances are specified with the value NA.
x.inst	A boolean value that indicates if x is or not an instance matrix. Default is TRUE.
learners	A list of functions or strings naming the functions for training the different supervised base classifiers.
learners.pars	A list with the set of additional parameters for each learner functions if necessary. Default is NULL.
preds	A list of functions or strings naming the functions for predicting the probabilities per classes, using the base classifiers trained with the functions defined in learners. Default is "predict" function for each learner in learners.
preds.pars	A list with the set of additional parameters for each function in preds if necessary. Default is NULL.

Details

This method trains an ensemble of diverse classifiers. To promote the initial diversity the classifiers must represent different learning schemes. When x.inst is FALSE all learners defined must be able to learn a classifier from the precomputed matrix in x. The iteration process of the algorithm ends when no changes occurs in any model during a complete iteration. The generation of the final hypothesis is produced via a weigthed majority voting.

Value

A list object of class "democratic" containing:

W A vector with the confidence-weighted vote assigned to each classifier.

model A list with the final N base classifiers trained using the enlarged labeled set.

model.index List of N vectors of indexes related to the training instances used per each classifier. These indexes are relative to the y argument.

instances.index The indexes of all training instances used to train the N models. These indexes include the initial labeled instances and the newly labeled instances. These indexes are relative to the y argument.

model.index.map List of three vectors with the same information in model.index but the indexes are relative to instances.index vector.

classes The levels of y factor.

preds The functions provided in the preds argument.

preds.pars The set of lists provided in the preds.pars argument.

x.inst The value provided in the x.inst argument.

Examples

```
## Not run:

library(ssc)

## Load Wine data set
data(wine)

cls <- which(colnames(wine) == "Wine")
x <- wine[, -cls] # instances without classes
y <- wine[, cls] # the classes
x <- scale(x) # scale the attributes

## Prepare data
set.seed(20)
# Use 50% of instances for training
tra.idx <- sample(x = length(y), size = ceiling(length(y) * 0.5))
xtrain <- x[tra.idx,] # training instances
ytrain <- y[tra.idx]  # classes of training instances
# Use 70% of train instances as unlabeled set
tra.na.idx <- sample(x = length(tra.idx), size = ceiling(length(tra.idx) * 0.7))
ytrain[tra.na.idx] <- NA # remove class information of unlabeled instances

# Use the other 50% of instances for inductive testing
tst.idx <- setdiff(1:length(y), tra.idx)
xitest <- x[tst.idx,] # testing instances
yitest <- y[tst.idx] # classes of testing instances

## Example: Training from a set of instances with
# 1-NN and C-svc (SVM) as base classifiers.
# knn3 learner
library(caret)
knn <- knn3             # learner function
knn.pars <- list(k = 1) # parameters for learner function
knn.prob <- predict     # function to predict probabilities
knn.prob.pars <- NULL   # parameters for prediction function

# ksvm learner
library(kernlab)
svm <- ksvm             # learner function
svm.pars <- list(       # parameters for learner function
  type = "C-svc",  C = 1,
  kernel = "rbfdot", kpar = list(sigma = 0.048),
  prob.model = TRUE,
  scaled = FALSE
)
svm.prob <- predict     # function to predict probabilities
svm.prob.pars <- list(  # parameters for prediction function
  type = "probabilities"
)

# train a model
```

```
m <- democratic(x = xtrain, y = ytrain,
                learners = list(knn, svm),
                learners.pars = list(knn.pars, svm.pars),
                preds = list(knn.prob, svm.prob),
                preds.pars = list(knn.prob.pars, svm.prob.pars))
# predict classes
m.pred <- predict(m, xitest)
table(m.pred, yitest)


## End(Not run)
```

`democraticCombine` *Combining the hypothesis of the classifiers*

Description

This function combines the probabilities predicted by the set of classifiers.

Usage

```
democraticCombine(pred, W, classes)
```

Arguments

`pred`	A list with the prediction for each classifier.
`W`	A vector with the confidence-weighted vote assigned to each classifier during the training process.
`classes`	the classes.

Value

The classification proposed.

`democraticG` *Democratic generic method*

Description

Democratic is a semi-supervised learning algorithm with a co-training style. This algorithm trains N classifiers with different learning schemes defined in list `gen.learners`. During the iterative process, the multiple classifiers with different inductive biases label data for each other.

Usage

```
democraticG(y, gen.learners, gen.preds)
```

Arguments

y	A vector with the labels of training instances. In this vector the unlabeled instances are specified with the value NA.
gen.learners	A list of functions for training N different supervised base classifiers. Each function needs two parameters, indexes and cls, where indexes indicates the instances to use and cls specifies the classes of those instances.
gen.preds	A list of functions for predicting the probabilities per classes. Each function must be two parameters, model and indexes, where the model is a classifier trained with gen.learner function and indexes indicates the instances to predict.

Details

democraticG can be helpful in those cases where the method selected as base classifier needs a learner and pred functions with other specifications. For more information about the general democratic method, please see democratic function. Essentially, democratic function is a wrapper of democraticG function.

Value

A list object of class "democraticG" containing:

W A vector with the confidence-weighted vote assigned to each classifier.

model A list with the final N base classifiers trained using the enlarged labeled set.

model.index List of N vectors of indexes related to the training instances used per each classifier. These indexes are relative to the y argument.

instances.index The indexes of all training instances used to train the N models. These indexes include the initial labeled instances and the newly labeled instances. These indexes are relative to the y argument.

model.index.map List of three vectors with the same information in model.index but the indexes are relative to instances.index vector.

classes The levels of y factor.

References

Yan Zhou and Sally Goldman.
Democratic co-learning.
In IEEE 16th International Conference on Tools with Artificial Intelligence (ICTAI), pages 594-602. IEEE, Nov 2004. doi: 10.1109/ICTAI.2004.48.

Examples

```
## Not run:
# this is a long running example

library(ssc)
```

```
## Load Wine data set
data(wine)

cls <- which(colnames(wine) == "Wine")
x <- wine[, -cls] # instances without classes
y <- wine[, cls] # the classes
x <- scale(x) # scale the attributes

## Prepare data
set.seed(20)
# Use 50% of instances for training
tra.idx <- sample(x = length(y), size = ceiling(length(y) * 0.5))
xtrain <- x[tra.idx,] # training instances
ytrain <- y[tra.idx]  # classes of training instances
# Use 70% of train instances as unlabeled set
tra.na.idx <- sample(x = length(tra.idx), size = ceiling(length(tra.idx) * 0.7))
ytrain[tra.na.idx] <- NA # remove class information of unlabeled instances

# Use the other 50% of instances for inductive testing
tst.idx <- setdiff(1:length(y), tra.idx)
xitest <- x[tst.idx,] # testing instances
yitest <- y[tst.idx] # classes of testing instances

## Example A:
# Training from a set of instances with
# 1-NN and C-svc (SVM) as base classifiers.

### Define knn base classifier using knn3 from caret package
library(caret)
# learner function
knn <- function(indexes, cls) {
  knn3(x = xtrain[indexes, ], y = cls, k = 1)
}
# function to predict probabilities
knn.prob <- function(model, indexes) {
  predict(model, xtrain[indexes, ])
}

### Define svm base classifier using ksvm from kernlab package
library(kernlab)
library(proxy)
# learner function
svm <- function(indexes, cls) {
  rbf <- function(x, y) {
    sigma <- 0.048
    d <- dist(x, y, method = "Euclidean", by_rows = FALSE)
    exp(-sigma *  d * d)
  }
  class(rbf) <- "kernel"
  ksvm(x = xtrain[indexes, ], y = cls, scaled = FALSE,
       type = "C-svc",  C = 1,
       kernel = rbf, prob.model = TRUE)
}
```

```
# function to predict probabilities
svm.prob <- function(model, indexes) {
  predict(model, xtrain[indexes, ], type = "probabilities")
}

### Train
m1 <- democraticG(y = ytrain,
                  gen.learners = list(knn, svm),
                  gen.preds = list(knn.prob, svm.prob))
### Predict
# predict labels using each classifier
m1.pred1 <- predict(m1$model[[1]], xitest, type = "class")
m1.pred2 <- predict(m1$model[[2]], xitest)
# combine predictions
m1.pred <- list(m1.pred1, m1.pred2)
cls1 <- democraticCombine(m1.pred, m1$W, m1$classes)
table(cls1, yitest)

## Example B:
# Training from a distance matrix and a kernel matrix with
# 1-NN and C-svc (SVM) as base classifiers.

### Define knn2 base classifier using oneNN from ssc package
library(ssc)
# Compute distance matrix D
# D is used in knn2.prob
D <- as.matrix(dist(x = xtrain, method = "euclidean", by_rows = TRUE))
# learner function
knn2 <- function(indexes, cls) {
  model <- oneNN(y = cls)
  attr(model, "tra.idxs") <- indexes
  model
}
# function to predict probabilities
knn2.prob <- function(model, indexes)  {
  tra.idxs <- attr(model, "tra.idxs")
  predict(model, D[indexes, tra.idxs], distance.weighting = "none")
}

### Define svm2 base classifier using ksvm from kernlab package
library(kernlab)

# Compute kernel matrix K
# K is used in svm2 and svm2.prob functions
sigma <- 0.048
K <- exp(- sigma * D * D)

# learner function
svm2 <- function(indexes, cls) {
  model <- ksvm(K[indexes, indexes], y = cls,
                type = "C-svc", C = 1,
                kernel = "matrix",
                prob.model = TRUE)
```

```
  attr(model, "tra.idxs") <- indexes
  model
}
# function to predict probabilities
svm2.prob <- function(model, indexes)  {
  tra.idxs <- attr(model, "tra.idxs")
  sv.idxs <- tra.idxs[SVindex(model)]
  predict(model,
          as.kernelMatrix(K[indexes, sv.idxs]),
          type = "probabilities")
}


## End(Not run)
```

---

oneNN *1-NN supervised classifier builder*

Description

Build a model using the given data to be able to predict the label or the probabilities of other instances, according to 1-NN algorithm.

Usage

```
oneNN(x = NULL, y)
```

Arguments

x	This argument is not used, the reason why he gets is to fulfill an agreement
y	a vector with the labels of training instances

Value

A model with the data needed to use 1-NN

---

predict.coBC *Predictions of the coBC method*

Description

Predicts the label of instances according to the coBC model.

Usage

```
## S3 method for class 'coBC'
predict(object, x, ...)
```

Arguments

object	coBC model built with the coBC function.
x	An object that can be coerced to a matrix. Depending on how the model was built, x is interpreted as a matrix with the distances between the unseen instances and the selected training instances, or a matrix of instances.
...	This parameter is included for compatibility reasons.

Details

For additional help see coBC examples.

Value

Vector with the labels assigned.

---

predict.democratic *Predictions of the Democratic method*

Description

Predicts the label of instances according to the democratic model.

Usage

```
## S3 method for class 'democratic'
predict(object, x, ...)
```

Arguments

object	Democratic model built with the democratic function.
x	A object that can be coerced as matrix. Depending on how was the model built, x is interpreted as a matrix with the distances between the unseen instances and the selected training instances, or a matrix of instances.
...	This parameter is included for compatibility reasons.

Details

For additional help see democratic examples.

Value

Vector with the labels assigned.

predict.OneNN *Model Predictions*

Description

This function predicts the class label of instances or its probability of pertaining to each class based on the distance matrix.

Usage

```
## S3 method for class 'OneNN'
predict(object, dists, type = "prob", ...)
```

Arguments

object	A model of class OneNN built with oneNN
dists	A matrix of distances between the instances to classify (by rows) and the instances used to train the model (by column)
type	A string that can take two values: "class" for computing the class of the instances or "prob" for computing the probabilities of belonging to each class.
...	Currently not used.

Value

If type is equal to "class" a vector of length equal to the rows number of matrix dists, containing the predicted labels. If type is equal to "prob" it returns a matrix which has nrow(dists) rows and a column for every class, where each cell represents the probability that the instance belongs to the class, according to 1NN.

predict.selfTraining *Predictions of the Self-training method*

Description

Predicts the label of instances according to the selfTraining model.

Usage

```
## S3 method for class 'selfTraining'
predict(object, x, ...)
```

Arguments

object	Self-training model built with the selfTraining function.
x	A object that can be coerced as matrix. Depending on how was the model built, x is interpreted as a matrix with the distances between the unseen instances and the selected training instances, or a matrix of instances.
...	This parameter is included for compatibility reasons.

Details

For additional help see selfTraining examples.

Value

Vector with the labels assigned.

---

predict.setred *Predictions of the SETRED method*

Description

Predicts the label of instances according to the setred model.

Usage

```
## S3 method for class 'setred'
predict(object, x, ...)
```

Arguments

object	SETRED model built with the setred function.
x	A object that can be coerced as matrix. Depending on how was the model built, x is interpreted as a matrix with the distances between the unseen instances and the selected training instances, or a matrix of instances.
...	This parameter is included for compatibility reasons.

Details

For additional help see setred examples.

Value

Vector with the labels assigned.

predict.snnrce *Predictions of the SNNRCE method*

Description

Predicts the label of instances according to the snnrce model.

Usage

```
## S3 method for class 'snnrce'
predict(object, x, ...)
```

Arguments

object	SNNRCE model built with the snnrce function.
x	A object that can be coerced as matrix. Depending on how was the model built, x is interpreted as a matrix with the distances between the unseen instances and the selected training instances, or a matrix of instances.
...	This parameter is included for compatibility reasons.

Details

For additional help see snnrce examples.

Value

Vector with the labels assigned.

predict.triTraining *Predictions of the Tri-training method*

Description

Predicts the label of instances according to the triTraining model.

Usage

```
## S3 method for class 'triTraining'
predict(object, x, ...)
```

Arguments

object	Tri-training model built with the triTraining function.
x	A object that can be coerced as matrix. Depending on how was the model built, x is interpreted as a matrix with the distances between the unseen instances and the selected training instances, or a matrix of instances.
...	This parameter is included for compatibility reasons.

Details

For additional help see triTraining examples.

Value

Vector with the labels assigned.

`selfTraining` *Self-training method*

Description

Self-training is a simple and effective semi-supervised learning classification method. The self-training classifier is initially trained with a reduced set of labeled examples. Then it is iteratively retrained with its own most confident predictions over the unlabeled examples. Self-training follows a wrapper methodology using a base supervised classifier to establish the possible class of unlabeled instances.

Usage

```
selfTraining(x, y, x.inst = TRUE, learner, learner.pars = NULL,
  pred = "predict", pred.pars = NULL, max.iter = 50,
  perc.full = 0.7, thr.conf = 0.5)
```

Arguments

x	A object that can be coerced as matrix. This object has two possible interpretations according to the value set in the x.inst argument: a matrix with the training instances where each row represents a single instance or a precomputed (distance or kernel) matrix between the training examples.
y	A vector with the labels of the training instances. In this vector the unlabeled instances are specified with the value NA.
x.inst	A boolean value that indicates if x is or not an instance matrix. Default is TRUE.
learner	either a function or a string naming the function for training a supervised base classifier, using a set of instances (or optionally a distance matrix) and it's corresponding classes.
learner.pars	A list with additional parameters for the learner function if necessary. Default is NULL.
pred	either a function or a string naming the function for predicting the probabilities per classes, using the base classifier trained with the learner function. Default is "predict".
pred.pars	A list with additional parameters for the pred function if necessary. Default is NULL.
max.iter	maximum number of iterations to execute the self-labeling process. Default is 50.

`perc.full` A number between 0 and 1. If the percentage of new labeled examples reaches this value the self-training process is stopped. Default is 0.7.

`thr.conf` A number between 0 and 1 that indicates the confidence threshold. At each iteration, only the newly labelled examples with a confidence greater than this value (`thr.conf`) are added to the training set.

Details

For predicting the most accurate instances per iteration, `selfTraining` uses the predictions obtained with the learner specified. To train a model using the `learner` function, it is required a set of instances (or a precomputed matrix between the instances if `x.inst` parameter is `FALSE`) in conjunction with the corresponding classes. Additionals parameters are provided to the `learner` function via the `learner.pars` argument. The model obtained is a supervised classifier ready to predict new instances through the `pred` function. Using a similar idea, the additional parameters to the `pred` function are provided using the `pred.pars` argument. The `pred` function returns the probabilities per class for each new instance. The value of the `thr.conf` argument controls the confidence of instances selected to enlarge the labeled set for the next iteration.

The stopping criterion is defined through the fulfillment of one of the following criteria: the algorithm reaches the number of iterations defined in the `max.iter` parameter or the portion of the unlabeled set, defined in the `perc.full` parameter, is moved to the labeled set. In some cases, the process stops and no instances are added to the original labeled set. In this case, the user must assign a more flexible value to the `thr.conf` parameter.

Value

A list object of class "selfTraining" containing:

model The final base classifier trained using the enlarged labeled set.

instances.index The indexes of the training instances used to train the `model`. These indexes include the initial labeled instances and the newly labeled instances. Those indexes are relative to `x` argument.

classes The levels of `y` factor.

pred The function provided in the `pred` argument.

pred.pars The list provided in the `pred.pars` argument.

References

David Yarowsky.
Unsupervised word sense disambiguation rivaling supervised methods.
In Proceedings of the 33rd annual meeting on Association for Computational Linguistics, pages 189-196. Association for Computational Linguistics, 1995.

Examples

```
library(ssc)

## Load Wine data set
data(wine)
```

```
cls <- which(colnames(wine) == "Wine")
x <- wine[, -cls] # instances without classes
y <- wine[, cls] # the classes
x <- scale(x) # scale the attributes

## Prepare data
set.seed(20)
# Use 50% of instances for training
tra.idx <- sample(x = length(y), size = ceiling(length(y) * 0.5))
xtrain <- x[tra.idx,] # training instances
ytrain <- y[tra.idx]  # classes of training instances
# Use 70% of train instances as unlabeled set
tra.na.idx <- sample(x = length(tra.idx), size = ceiling(length(tra.idx) * 0.7))
ytrain[tra.na.idx] <- NA # remove class information of unlabeled instances

# Use the other 50% of instances for inductive testing
tst.idx <- setdiff(1:length(y), tra.idx)
xitest <- x[tst.idx,] # testing instances
yitest <- y[tst.idx] # classes of testing instances

## Example: Training from a set of instances with 1-NN as base classifier.
m1 <- selfTraining(x = xtrain, y = ytrain,
                   learner = caret::knn3,
                   learner.pars = list(k = 1),
                   pred = "predict")
pred1 <- predict(m1, xitest)
table(pred1, yitest)

## Example: Training from a distance matrix with 1-NN as base classifier.
dtrain <- as.matrix(proxy::dist(x = xtrain, method = "euclidean", by_rows = TRUE))
m2 <- selfTraining(x = dtrain, y = ytrain, x.inst = FALSE,
                   learner = ssc::oneNN,
                   pred = "predict",
                   pred.pars = list(distance.weighting = "none"))
ditest <- proxy::dist(x = xitest, y = xtrain[m2$instances.index,],
                      method = "euclidean", by_rows = TRUE)
pred2 <- predict(m2, ditest)
table(pred2, yitest)

## Example: Training from a set of instances with SVM as base classifier.
learner <- e1071::svm
learner.pars <- list(type = "C-classification", kernel="radial",
                     probability = TRUE, scale = TRUE)
pred <- function(m, x){
  r <- predict(m, x, probability = TRUE)
  prob <- attr(r, "probabilities")
  prob
}
m3 <- selfTraining(x = xtrain, y = ytrain,
                   learner = learner,
                   learner.pars = learner.pars,
                   pred = pred)
```

```
pred3 <- predict(m3, xitest)
table(pred3, yitest)

## Example: Training from a set of instances with Naive-Bayes as base classifier.
m4 <- selfTraining(x = xtrain, y = ytrain,
                   learner = function(x, y) e1071::naiveBayes(x, y),
                   pred = "predict",
                   pred.pars = list(type = "raw"))
pred4 <- predict(m4, xitest)
table(pred4, yitest)

## Example: Training from a set of instances with C5.0 as base classifier.
m5 <- selfTraining(x = xtrain, y = ytrain,
                   learner = C50::C5.0,
                   pred = "predict",
                   pred.pars = list(type = "prob"))
pred5 <- predict(m5, xitest)
table(pred5, yitest)
```

`selfTrainingG` *Self-training generic method*

Description

Self-training is a simple and effective semi-supervised learning classification method. The self-training classifier is initially trained with a reduced set of labeled examples. Then it is iteratively retrained with its own most confident predictions over the unlabeled examples. Self-training follows a wrapper methodology using one base supervised classifier to establish the possible class of unlabeled instances.

Usage

```
selfTrainingG(y, gen.learner, gen.pred, max.iter = 50, perc.full = 0.7,
  thr.conf = 0.5)
```

Arguments

`y`	A vector with the labels of training instances. In this vector the unlabeled instances are specified with the value `NA`.
`gen.learner`	A function for training a supervised base classifier. This function needs two parameters, indexes and cls, where indexes indicates the instances to use and cls specifies the classes of those instances.
`gen.pred`	A function for predicting the probabilities per classes. This function must be two parameters, model and indexes, where the model is a classifier trained with `gen.learner` function and indexes indicates the instances to predict.

`max.iter` Maximum number of iterations to execute the self-labeling process. Default is 50.

`perc.full` A number between 0 and 1. If the percentage of new labeled examples reaches this value the self-training process is stopped. Default is 0.7.

`thr.conf` A number between 0 and 1 that indicates the confidence theshold. At each iteration, only the newly labelled examples with a confidence greater than this value (`thr.conf`) are added to the training set.

Details

SelfTrainingG can be helpful in those cases where the method selected as base classifier needs `learner` and `pred` functions with other specifications. For more information about the general self-training method, please see the `selfTraining` function. Essentially, the `selfTraining` function is a wrapper of the `selfTrainingG` function.

Value

A list object of class "selfTrainingG" containing:

model The final base classifier trained using the enlarged labeled set.

instances.index The indexes of the training instances used to train the `model`. These indexes include the initial labeled instances and the newly labeled instances. Those indexes are relative to the `y` argument.

Examples

```
library(ssc)

## Load Wine data set
data(wine)

cls <- which(colnames(wine) == "Wine")
x <- wine[, -cls] # instances without classes
y <- wine[, cls] # the classes
x <- scale(x) # scale the attributes

## Prepare data
set.seed(20)
# Use 50% of instances for training
tra.idx <- sample(x = length(y), size = ceiling(length(y) * 0.5))
xtrain <- x[tra.idx,] # training instances
ytrain <- y[tra.idx]  # classes of training instances
# Use 70% of train instances as unlabeled set
tra.na.idx <- sample(x = length(tra.idx), size = ceiling(length(tra.idx) * 0.7))
ytrain[tra.na.idx] <- NA # remove class information of unlabeled instances

# Use the other 50% of instances for inductive testing
tst.idx <- setdiff(1:length(y), tra.idx)
xitest <- x[tst.idx,] # testing instances
yitest <- y[tst.idx] # classes of testing instances
```

```
## Example: Training from a set of instances with 1-NN (knn3) as base classifier.
gen.learner <- function(indexes, cls)
  caret::knn3(x = xtrain[indexes, ], y = cls, k = 1)
gen.pred <- function(model, indexes)
  predict(model, xtrain[indexes, ])

md1 <- selfTrainingG(y = ytrain, gen.learner, gen.pred)

cls1 <- predict(md1$model, xitest, type = "class")
table(cls1, yitest)

## Example: Training from a distance matrix with 1-NN (oneNN) as base classifier.
dtrain <- as.matrix(proxy::dist(x = xtrain, method = "euclidean", by_rows = TRUE))
gen.learner <- function(indexes, cls) {
  m <- ssc::oneNN(y = cls)
  attr(m, "tra.idxs") <- indexes
  m
}

gen.pred <- function(model, indexes)  {
  tra.idxs <- attr(model, "tra.idxs")
  d <- dtrain[indexes, tra.idxs]
  prob <- predict(model, d, distance.weighting = "none")
  prob
}

md2 <- selfTrainingG(y = ytrain, gen.learner, gen.pred)
ditest <- proxy::dist(x = xitest, y = xtrain[md2$instances.index,],
                      method = "euclidean", by_rows = TRUE)
cls2 <- predict(md2$model, ditest, type = "class")
table(cls2, yitest)
```

`setred` *SETRED method*

Description

SETRED (SElf-TRaining with EDiting) is a variant of the self-training classification method (as implemented in the function `selfTraining`) with a different addition mechanism. The SETRED classifier is initially trained with a reduced set of labeled examples. Then, it is iteratively retrained with its own most confident predictions over the unlabeled examples. SETRED uses an amending scheme to avoid the introduction of noisy examples into the enlarged labeled set. For each iteration, the mislabeled examples are identified using the local information provided by the neighborhood graph.

Usage

```
setred(x, y, x.inst = TRUE, dist = "Euclidean", learner,
  learner.pars = NULL, pred = "predict", pred.pars = NULL,
  theta = 0.1, max.iter = 50, perc.full = 0.7)
```

Arguments

x	A object that can be coerced as matrix. This object has two possible interpretations according to the value set in the x.inst argument: a matrix with the training instances where each row represents a single instance or a precomputed (distance or kernel) matrix between the training examples.
y	A vector with the labels of the training instances. In this vector the unlabeled instances are specified with the value NA.
x.inst	A boolean value that indicates if x is or not an instance matrix. Default is TRUE.
dist	A distance function or the name of a distance available in the proxy package to compute the distance matrix in the case that x.inst is TRUE.
learner	either a function or a string naming the function for training a supervised base classifier, using a set of instances (or optionally a distance matrix) and it's corresponding classes.
learner.pars	A list with additional parameters for the learner function if necessary. Default is NULL.
pred	either a function or a string naming the function for predicting the probabilities per classes, using the base classifier trained with the learner function. Default is "predict".
pred.pars	A list with additional parameters for the pred function if necessary. Default is NULL.
theta	Rejection threshold to test the critical region. Default is 0.1.
max.iter	maximum number of iterations to execute the self-labeling process. Default is 50.
perc.full	A number between 0 and 1. If the percentage of new labeled examples reaches this value the self-training process is stopped. Default is 0.7.

Details

SETRED initiates the self-labeling process by training a model from the original labeled set. In each iteration, the learner function detects unlabeled examples for which it makes the most confident prediction and labels those examples according to the pred function. The identification of mislabeled examples is performed using a neighborhood graph created from the distance matrix. When x.inst is TRUE this distance matrix is computed using the dist function. On the other hand, when x.inst is FALSE the matrix provided with x is used both to train a classifier and to create the neighborhood graph. Most examples possess the same label in a neighborhood. So if an example locates in a neighborhood with too many neighbors from different classes, this example should be considered problematic. The value of the theta argument controls the confidence of the candidates selected to enlarge the labeled set. The lower this value is, the more restrictive is the selection of the examples that are considered good. For more information about the self-labeled process and the rest of the parameters, please see selfTraining.

Value

A list object of class "setred" containing:

model The final base classifier trained using the enlarged labeled set.

instances.index The indexes of the training instances used to train the `model`. These indexes include the initial labeled instances and the newly labeled instances. Those indexes are relative to `x` argument.

classes The levels of `y` factor.

pred The function provided in the `pred` argument.

pred.pars The list provided in the `pred.pars` argument.

References

Ming Li and ZhiHua Zhou.
Setred: Self-training with editing.
In Advances in Knowledge Discovery and Data Mining, volume 3518 of Lecture Notes in Computer Science, pages 611-621. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-26076-9. doi: 10.1007/11430919 71.

Examples

```
library(ssc)

## Load Wine data set
data(wine)

cls <- which(colnames(wine) == "Wine")
x <- wine[, -cls] # instances without classes
y <- wine[, cls] # the classes
x <- scale(x) # scale the attributes

## Prepare data
set.seed(20)
# Use 50% of instances for training
tra.idx <- sample(x = length(y), size = ceiling(length(y) * 0.5))
xtrain <- x[tra.idx,] # training instances
ytrain <- y[tra.idx]  # classes of training instances
# Use 70% of train instances as unlabeled set
tra.na.idx <- sample(x = length(tra.idx), size = ceiling(length(tra.idx) * 0.7))
ytrain[tra.na.idx] <- NA # remove class information of unlabeled instances

# Use the other 50% of instances for inductive testing
tst.idx <- setdiff(1:length(y), tra.idx)
xitest <- x[tst.idx,] # testing instances
yitest <- y[tst.idx] # classes of testing instances

## Example: Training from a set of instances with 1-NN as base classifier.
m1 <- setred(x = xtrain, y = ytrain, dist = "euclidean",
             learner = caret::knn3,
             learner.pars = list(k = 1),
             pred = "predict")
pred1 <- predict(m1, xitest)
table(pred1, yitest)

## Example: Training from a distance matrix with 1-NN as base classifier.
```

```
# Compute distances between training instances
library(proxy)
D <- dist(x = xtrain, method = "euclidean", by_rows = TRUE)

m2 <- setred(x = D, y = ytrain, x.inst = FALSE,
            learner = ssc::oneNN,
            pred = "predict",
            pred.pars = list(distance.weighting = "none"))
ditest <- proxy::dist(x = xitest, y = xtrain[m2$instances.index,],
                      method = "euclidean", by_rows = TRUE)
pred2 <- predict(m2, ditest)
table(pred2, yitest)

## Example: Training from a set of instances with SVM as base classifier.
learner <- e1071::svm
learner.pars <- list(type = "C-classification", kernel="radial",
                     probability = TRUE, scale = TRUE)
pred <- function(m, x){
  r <- predict(m, x, probability = TRUE)
  prob <- attr(r, "probabilities")
  prob
}
m3 <- setred(x = xtrain, y = ytrain, dist = "euclidean",
             learner = learner,
             learner.pars = learner.pars,
             pred = pred)
pred3 <- predict(m3, xitest)
table(pred3, yitest)

## Example: Training from a set of instances with Naive-Bayes as base classifier.
m4 <- setred(x = xtrain, y = ytrain, dist = "euclidean",
             learner = function(x, y) e1071::naiveBayes(x, y),
             pred = "predict",
             pred.pars = list(type = "raw"))
pred4 <- predict(m4, xitest)
table(pred4, yitest)

## Example: Training from a set of instances with C5.0 as base classifier.
m5 <- setred(x = xtrain, y = ytrain, dist = "euclidean",
             learner = C50::C5.0,
             pred = "predict",
             pred.pars = list(type = "prob"))
pred5 <- predict(m5, xitest)
table(pred5, yitest)
```

---

setredG *SETRED generic method*

Description

SETRED is a variant of the self-training classification method (selfTraining) with a different addition mechanism. The SETRED classifier is initially trained with a reduced set of labeled examples. Then it is iteratively retrained with its own most confident predictions over the unlabeled examples. SETRED uses an amending scheme to avoid the introduction of noisy examples into the enlarged labeled set. For each iteration, the mislabeled examples are identified using the local information provided by the neighborhood graph.

Usage

```
setredG(y, D, gen.learner, gen.pred, theta = 0.1, max.iter = 50,
  perc.full = 0.7)
```

Arguments

y	A vector with the labels of training instances. In this vector the unlabeled instances are specified with the value NA.
D	A distance matrix between all the training instances. This matrix is used to construct the neighborhood graph.
gen.learner	A function for training a supervised base classifier. This function needs two parameters, indexes and cls, where indexes indicates the instances to use and cls specifies the classes of those instances.
gen.pred	A function for predicting the probabilities per classes. This function must be two parameters, model and indexes, where the model is a classifier trained with gen.learner function and indexes indicates the instances to predict.
theta	Rejection threshold to test the critical region. Default is 0.1.
max.iter	Maximum number of iterations to execute the self-labeling process. Default is 50.
perc.full	A number between 0 and 1. If the percentage of new labeled examples reaches this value the self-training process is stopped. Default is 0.7.

Details

SetredG can be helpful in those cases where the method selected as base classifier needs a learner and pred functions with other specifications. For more information about the general setred method, please see setred function. Essentially, setred function is a wrapper of setredG function.

Value

A list object of class "setredG" containing:

model The final base classifier trained using the enlarged labeled set.

instances.index The indexes of the training instances used to train the model. These indexes include the initial labeled instances and the newly labeled instances. Those indexes are relative to the y argument.

Examples

```
library(ssc)

## Load Wine data set
data(wine)

cls <- which(colnames(wine) == "Wine")
x <- wine[, -cls] # instances without classes
y <- wine[, cls] # the classes
x <- scale(x) # scale the attributes

## Prepare data
set.seed(20)
# Use 50% of instances for training
tra.idx <- sample(x = length(y), size = ceiling(length(y) * 0.5))
xtrain <- x[tra.idx,] # training instances
ytrain <- y[tra.idx]  # classes of training instances
# Use 70% of train instances as unlabeled set
tra.na.idx <- sample(x = length(tra.idx), size = ceiling(length(tra.idx) * 0.7))
ytrain[tra.na.idx] <- NA # remove class information of unlabeled instances

# Use the other 50% of instances for inductive testing
tst.idx <- setdiff(1:length(y), tra.idx)
xitest <- x[tst.idx,] # testing instances
yitest <- y[tst.idx] # classes of testing instances

# Compute distances between training instances
D <- as.matrix(proxy::dist(x = xtrain, method = "euclidean", by_rows = TRUE))

## Example: Training from a set of instances with 1-NN (knn3) as base classifier.
gen.learner <- function(indexes, cls)
  caret::knn3(x = xtrain[indexes, ], y = cls, k = 1)
gen.pred <- function(model, indexes)
  predict(model, xtrain[indexes, ])

md1 <- setredG(y = ytrain, D, gen.learner, gen.pred)

cls1 <- predict(md1$model, xitest, type = "class")
table(cls1, yitest)

## Example: Training from a distance matrix with 1-NN (oneNN) as base classifier
gen.learner <- function(indexes, cls) {
  m <- ssc::oneNN(y = cls)
  attr(m, "tra.idxs") <- indexes
  m
}

gen.pred <- function(model, indexes)  {
  tra.idxs <- attr(model, "tra.idxs")
  d <- D[indexes, tra.idxs]
  prob <- predict(model, d, distance.weighting = "none")
  prob
```

```
}

md2 <- setredG(y = ytrain, D, gen.learner, gen.pred)
ditest <- proxy::dist(x = xitest, y = xtrain[md2$instances.index,],
                      method = "euclidean", by_rows = TRUE)
cls2 <- predict(md2$model, ditest, type = "class")
table(cls2, yitest)
```

---

snnrce *SNNRCE method*

Description

SNNRCE (Self-training Nearest Neighbor Rule using Cut Edges) is a variant of the self-training classification method (selfTraining) with a different addition mechanism and a fixed learning scheme (1-NN). SNNRCE uses an amending scheme to avoid the introduction of noisy examples into the enlarged labeled set. The mislabeled examples are identified using the local information provided by the neighborhood graph. A statistical test using cut edge weight is used to modify the labels of the missclassified examples.

Usage

```
snnrce(x, y, x.inst = TRUE, dist = "Euclidean", alpha = 0.1)
```

Arguments

x	A object that can be coerced as matrix. This object has two possible interpretations according to the value set in the x.inst argument: a matrix with the training instances where each row represents a single instance or a precomputed distance matrix between the training examples.
y	A vector with the labels of the training instances. In this vector the unlabeled instances are specified with the value NA.
x.inst	A boolean value that indicates if x is or not an instance matrix. Default is TRUE.
dist	A distance function available in the proxy package to compute the distance matrix in the case that x.inst is TRUE.
alpha	Rejection threshold to test the critical region. Default is 0.1.

Details

SNNRCE initiates the self-labeling process by training a 1-NN from the original labeled set. This method attempts to reduce the noise in examples by labeling those instances with no cut edges in the initial stages of self-labeling learning. These highly confident examples are added into the training set. The remaining examples follow the standard self-training process until a minimum number of examples will be labeled for each class. A statistical test using cut edge weight is used to modify the labels of the missclassified examples The value of the alpha argument defines the critical region where the candidates examples are tested. The higher this value is, the more relaxed it is the selection of the examples that are considered mislabeled.

Value

A list object of class "snnrce" containing:

model The final base classifier trained using the enlarged labeled set.

instances.index The indexes of the training instances used to train the `model`. These indexes include the initial labeled instances and the newly labeled instances. Those indexes are relative to `x` argument.

classes The levels of `y` factor.

x.inst The value provided in the `x.inst` argument.

dist The value provided in the `dist` argument when x.inst is `TRUE`.

xtrain A matrix with the subset of training instances referenced by the indexes `instances.index` when x.inst is `TRUE`.

References

Yu Wang, Xiaoyan Xu, Haifeng Zhao, and Zhongsheng Hua.
Semisupervised learning based on nearest neighbor rule and cut edges.
Knowledge-Based Systems, 23(6):547-554, 2010. ISSN 0950-7051. doi: http://dx.doi.org/10.1016/j.knosys.2010.03.012.

Examples

```
library(ssc)

## Load Wine data set
data(wine)

cls <- which(colnames(wine) == "Wine")
x <- wine[, -cls] # instances without classes
y <- wine[, cls] # the classes
x <- scale(x) # scale the attributes

## Prepare data
set.seed(20)
# Use 50% of instances for training
tra.idx <- sample(x = length(y), size = ceiling(length(y) * 0.5))
xtrain <- x[tra.idx,] # training instances
ytrain <- y[tra.idx]  # classes of training instances
# Use 70% of train instances as unlabeled set
tra.na.idx <- sample(x = length(tra.idx), size = ceiling(length(tra.idx) * 0.7))
ytrain[tra.na.idx] <- NA # remove class information of unlabeled instances

# Use the other 50% of instances for inductive testing
tst.idx <- setdiff(1:length(y), tra.idx)
xitest <- x[tst.idx,] # testing instances
yitest <- y[tst.idx] # classes of testing instances

## Example: Training from a set of instances with 1-NN as base classifier.
m1 <- snnrce(x = xtrain, y = ytrain,  dist = "Euclidean")
pred1 <- predict(m1, xitest)
table(pred1, yitest)
```

```
## Example: Training from a distance matrix with 1-NN as base classifier.
dtrain <- proxy::dist(x = xtrain, method = "euclidean", by_rows = TRUE)
m2 <- snnrce(x = dtrain, y = ytrain, x.inst = FALSE)
ditest <- proxy::dist(x = xitest, y = xtrain[m2$instances.index,],
                      method = "euclidean", by_rows = TRUE)
pred2 <- predict(m2, ditest)
table(pred2, yitest)
```

`triTraining` *Tri-training method*

Description

Tri-training is a semi-supervised learning algorithm with a co-training style. This algorithm trains three classifiers with the same learning scheme from a reduced set of labeled examples. For each iteration, an unlabeled example is labeled for a classifier if the other two classifiers agree on the labeling proposed.

Usage

```
triTraining(x, y, x.inst = TRUE, learner, learner.pars = NULL,
  pred = "predict", pred.pars = NULL)
```

Arguments

`x`	A object that can be coerced as matrix. This object has two possible interpretations according to the value set in the `x.inst` argument: a matrix with the training instances where each row represents a single instance or a precomputed (distance or kernel) matrix between the training examples.
`y`	A vector with the labels of the training instances. In this vector the unlabeled instances are specified with the value `NA`.
`x.inst`	A boolean value that indicates if `x` is or not an instance matrix. Default is `TRUE`.
`learner`	either a function or a string naming the function for training a supervised base classifier, using a set of instances (or optionally a distance matrix) and it's corresponding classes.
`learner.pars`	A list with additional parameters for the `learner` function if necessary. Default is `NULL`.
`pred`	either a function or a string naming the function for predicting the probabilities per classes, using the base classifiers trained with the `learner` function. Default is `"predict"`.
`pred.pars`	A list with additional parameters for the `pred` function if necessary. Default is `NULL`.

Details

Tri-training initiates the self-labeling process by training three models from the original labeled set, using the `learner` function specified. In each iteration, the algorithm detects unlabeled examples on which two classifiers agree with the classification and includes these instances in the enlarged set of the third classifier under certain conditions. The generation of the final hypothesis is produced via the majority voting. The iteration process ends when no changes occur in any model during a complete iteration.

Value

A list object of class "triTraining" containing:

model The final three base classifiers trained using the enlarged labeled set.

model.index List of three vectors of indexes related to the training instances used per each classifier. These indexes are relative to the `y` argument.

instances.index The indexes of all training instances used to train the three models. These indexes include the initial labeled instances and the newly labeled instances. These indexes are relative to the `y` argument.

model.index.map List of three vectors with the same information in `model.index` but the indexes are relative to `instances.index` vector.

classes The levels of `y` factor.

pred The function provided in the `pred` argument.

pred.pars The list provided in the `pred.pars` argument.

x.inst The value provided in the `x.inst` argument.

References

ZhiHua Zhou and Ming Li.
Tri-training: exploiting unlabeled data using three classifiers.
IEEE Transactions on Knowledge and Data Engineering, 17(11):1529-1541, Nov 2005. ISSN 1041-4347. doi: 10.1109/TKDE.2005. 186.

Examples

```
library(ssc)

## Load Wine data set
data(wine)

cls <- which(colnames(wine) == "Wine")
x <- wine[, -cls] # instances without classes
y <- wine[, cls] # the classes
x <- scale(x) # scale the attributes

## Prepare data
set.seed(20)
# Use 50% of instances for training
tra.idx <- sample(x = length(y), size = ceiling(length(y) * 0.5))
```

```
xtrain <- x[tra.idx,] # training instances
ytrain <- y[tra.idx]  # classes of training instances
# Use 70% of train instances as unlabeled set
tra.na.idx <- sample(x = length(tra.idx), size = ceiling(length(tra.idx) * 0.7))
ytrain[tra.na.idx] <- NA # remove class information of unlabeled instances

# Use the other 50% of instances for inductive testing
tst.idx <- setdiff(1:length(y), tra.idx)
xitest <- x[tst.idx,] # testing instances
yitest <- y[tst.idx] # classes of testing instances

## Example: Training from a set of instances with 1-NN as base classifier.
set.seed(1)
m1 <- triTraining(x = xtrain, y = ytrain,
                  learner = caret::knn3,
                  learner.pars = list(k = 1),
                  pred = "predict")
pred1 <- predict(m1, xitest)
table(pred1, yitest)

## Example: Training from a distance matrix with 1-NN as base classifier.
dtrain <- proxy::dist(x = xtrain, method = "euclidean", by_rows = TRUE)
set.seed(1)
m2 <- triTraining(x = dtrain, y = ytrain, x.inst = FALSE,
                  learner = ssc::oneNN,
                  pred = "predict",
                  pred.pars = list(distance.weighting = "none"))
ditest <- proxy::dist(x = xitest, y = xtrain[m2$instances.index,],
                      method = "euclidean", by_rows = TRUE)
pred2 <- predict(m2, ditest)
table(pred2, yitest)

## Example: Training from a set of instances with SVM as base classifier.
learner <- e1071::svm
learner.pars <- list(type = "C-classification", kernel="radial",
                     probability = TRUE, scale = TRUE)
pred <- function(m, x){
  r <- predict(m, x, probability = TRUE)
  prob <- attr(r, "probabilities")
  prob
}
set.seed(1)
m3 <- triTraining(x = xtrain, y = ytrain,
                  learner = learner,
                  learner.pars = learner.pars,
                  pred = pred)
pred3 <- predict(m3, xitest)
table(pred3, yitest)

## Example: Training from a set of instances with Naive-Bayes as base classifier.
set.seed(1)
m4 <- triTraining(x = xtrain, y = ytrain,
                  learner = function(x, y) e1071::naiveBayes(x, y),
```

```
                  pred.pars = list(type = "raw"))
pred4 <- predict(m4, xitest)
table(pred4, yitest)

## Example: Training from a set of instances with C5.0 as base classifier.
set.seed(1)
m5 <- triTraining(x = xtrain, y = ytrain,
                  learner = C50::C5.0,
                  pred.pars = list(type = "prob"))
pred5 <- predict(m5, xitest)
table(pred5, yitest)
```

`triTrainingCombine` *Combining the hypothesis*

Description

This function combines the predictions obtained by the set of classifiers.

Usage

```
triTrainingCombine(pred)
```

Arguments

`pred`	A list with the predictions of each classifiers

Value

A vector of classes

`triTrainingG` *Tri-training generic method*

Description

Tri-training is a semi-supervised learning algorithm with a co-training style. This algorithm trains three classifiers with the same learning scheme from a reduced set of labeled examples. For each iteration, an unlabeled example is labeled for a classifier if the other two classifiers agree on the labeling proposed.

Usage

```
triTrainingG(y, gen.learner, gen.pred)
```

Arguments

y	A vector with the labels of training instances. In this vector the unlabeled instances are specified with the value NA.
gen.learner	A function for training three supervised base classifiers. This function needs two parameters, indexes and cls, where indexes indicates the instances to use and cls specifies the classes of those instances.
gen.pred	A function for predicting the probabilities per classes. This function must be two parameters, model and indexes, where the model is a classifier trained with gen.learner function and indexes indicates the instances to predict.

Details

TriTrainingG can be helpful in those cases where the method selected as base classifier needs a learner and pred functions with other specifications. For more information about the general triTraining method, please see the triTraining function. Essentially, the triTraining function is a wrapper of the triTrainingG function.

Value

A list object of class "triTrainingG" containing:

model The final three base classifiers trained using the enlarged labeled set.

model.index List of three vectors of indexes related to the training instances used per each classifier. These indexes are relative to the y argument.

instances.index The indexes of all training instances used to train the three models. These indexes include the initial labeled instances and the newly labeled instances. These indexes are relative to the y argument.

model.index.map List of three vectors with the same information in model.index but the indexes are relative to instances.index vector.

Examples

```
library(ssc)

## Load Wine data set
data(wine)

cls <- which(colnames(wine) == "Wine")
x <- wine[, -cls] # instances without classes
y <- wine[, cls] # the classes
x <- scale(x) # scale the attributes

## Prepare data
set.seed(20)
# Use 50% of instances for training
tra.idx <- sample(x = length(y), size = ceiling(length(y) * 0.5))
xtrain <- x[tra.idx,] # training instances
ytrain <- y[tra.idx]  # classes of training instances
# Use 70% of train instances as unlabeled set
```

```
tra.na.idx <- sample(x = length(tra.idx), size = ceiling(length(tra.idx) * 0.7))
ytrain[tra.na.idx] <- NA # remove class information of unlabeled instances

# Use the other 50% of instances for inductive testing
tst.idx <- setdiff(1:length(y), tra.idx)
xitest <- x[tst.idx,] # testing instances
yitest <- y[tst.idx] # classes of testing instances

## Example: Training from a set of instances with 1-NN (knn3) as base classifier.
gen.learner <- function(indexes, cls)
  caret::knn3(x = xtrain[indexes, ], y = cls, k = 1)
gen.pred <- function(model, indexes)
  predict(model, xtrain[indexes, ])

# Train
set.seed(1)
md1 <- triTrainingG(y = ytrain, gen.learner, gen.pred)

# Predict testing instances using the three classifiers
pred <- lapply(
  X = md1$model,
  FUN = function(m) predict(m, xitest, type = "class")
)
# Combine the predictions
cls1 <- triTrainingCombine(pred)
table(cls1, yitest)

## Example: Training from a distance matrix with 1-NN (oneNN) as base classifier.
dtrain <- as.matrix(proxy::dist(x = xtrain, method = "euclidean", by_rows = TRUE))
gen.learner <- function(indexes, cls) {
  m <- ssc::oneNN(y = cls)
  attr(m, "tra.idxs") <- indexes
  m
}

gen.pred <- function(model, indexes)  {
  tra.idxs <- attr(model, "tra.idxs")
  d <- dtrain[indexes, tra.idxs]
  prob <- predict(model, d, distance.weighting = "none")
  prob
}

# Train
set.seed(1)
md2 <- triTrainingG(y = ytrain, gen.learner, gen.pred)

# Predict
ditest <- proxy::dist(x = xitest, y = xtrain[md2$instances.index,],
                      method = "euclidean", by_rows = TRUE)

# Predict testing instances using the three classifiers
pred <- mapply(
  FUN = function(m, indexes){
```

```
    D <- ditest[, indexes]
    predict(m, D, type = "class")
  },
  m = md2$model,
  indexes = md2$model.index.map,
  SIMPLIFY = FALSE
)
# Combine the predictions
cls2 <- triTrainingCombine(pred)
table(cls2, yitest)
```

`wine` *Wine recognition data*

Description

This dataset is the result of a chemical analysis of wine grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines.

Usage

```
data(wine)
```

Format

A data frame with 178 rows and 14 variables including the class.

Details

The dataset is taken from the UCI data repository, to which it was donated by Riccardo Leardi, University of Genova. The attributes are as follows:

- Alcohol
- Malic acid
- Ash
- Alcalinity of ash
- Magnesium
- Total phenols
- Flavanoids
- Nonflavanoid phenols
- Proanthocyanins
- Color intensity
- Hue
- OD280/OD315 of diluted wines
- Proline
- Wine (class)

Source

https://archive.ics.uci.edu/ml/datasets/Wine

Index