Package 'sambia'

June 6, 2018

Type Package

Title A Collection of Techniques Correcting for Sample Selection Bias

Version 0.1.0

Author Norbert Krautenbacher, Kevin Strauss, Maximilian Mandl, Christiane Fuchs

Maintainer Norbert Krautenbacher <norbert.krautenbacher@tum.de>

Description A collection of various techniques correcting statistical models for sample selection bias is provided. In particular, the resampling-based methods ``stochastic inverse-probability oversampling" and ``parametric inverse-probability bagging" are placed at the disposal which generate synthetic observations for correcting classifiers for biased samples resulting from stratified random sampling. For further information, see the article Krautenbacher, Theis, and Fuchs (2017) <doi:10.1155/2017/7847531>. The methods may be used for further purposes where weighting and generation of new observations is needed.

License GPL-3

LazyData TRUE

RoxygenNote 6.0.1.9000

NeedsCompilation no

Imports stats, mvtnorm, dplyr, smotefamily, e1071, ranger, pROC, FNN

Repository CRAN

Date/Publication 2018-06-06 11:27:57 UTC

R topics documented:

costing *Predicting outcomes using Costing.*

Description

This method trains classifiers by correcting them for sample selection bias via Costing, a method proposed in Zadrozny et al. (2003) . This method is similar to sambia's IP bagging function in terms of resampling from the learning data and aggregation of the learned algorithms. It differs in the implementation of resampling. Observations from the original data enters a resampled data set only once at most.

Usage

```
costing(..., learner, list.train.learner, list.predict.learner, n.bs,
  mod = FALSE)
```

Arguments

`...`	see the parameter rejSamp() of package sambia.
`learner`	a character indicating which classifier is used to train. Note: set learner='rangerTree' if random forest should be applied as in Krautenbacher et al. (2017), i.e. the correction step is incorporated in the inherent random forest resampling procedure.
`list.train.learner`	a list of parameters specific to the classifier that will be trained. Note that the parameter 'data' need not to be provided in this list since the training data which the model will learn on is already attained by new sampled data produced by the method rejSamp().
`list.predict.learner`	a list of parameters specifiying how to predict new data given the trained model. (This trained model is uniquely determined by parameters 'learner' and 'list.train.learner'
`n.bs`	number of bootstramp samples.
`mod`	If mod = TRUE: strategy for always having (at least) two outcome classes in subsets.

Author(s)

Norbert Krautenbacher, Kevin Strauss, Maximilian Mandl, Christiane Fuchs

References

Zadrozny, B., Langford, J., & Abe, N. (2003, November). Cost-sensitive learning by cost-proportionate example weighting. In Data Mining, 2003. ICDM 2003. Third IEEE International Conference on (pp. 435-442). IEEE.

Krautenbacher, N., Theis, F. J., & Fuchs, C. (2017). Correcting Classifiers for Sample Selection Bias in Two-Phase Case-Control Studies. Computational and mathematical methods in medicine, 2017.

Examples

```
## simulate data for a population
require(pROC)

set.seed(1342334)
N = 100000
x1 <- rnorm(N, mean=0, sd=1)
x2 <- rt(N, df=25)
x3 <- x1 + rnorm(N, mean=0, sd=.6)
x4 <- x2 + rnorm(N, mean=0, sd=1.3)
x5 <- rbinom(N, 1, prob=.6)
x6 <- rnorm(N, 0, sd = 1) # noise not known as variable
x7 <- x1*x5 # interaction
x <- cbind(x1, x2, x3, x4, x5, x6, x7)

## stratum variable (covariate)
xs <- c(rep(1,0.1*N), rep(0,(1-0.1)*N))

## effects
beta <- c(-1, 0.2, 0.4, 0.4, 0.5, 0.5, 0.6)
beta0 <- -2

## generate binary outcome
linpred.slopes <-  log(0.5)*xs + c(x %*% beta)
eta <-  beta0 + linpred.slopes

p <- 1/(1+exp(-eta)) # this is the probability P(Y=1|X), we want the binary outcome however:
y<-rbinom(n=N, size=1, prob=p) #

population <- data.frame(y,xs,x)

#### draw "given" data set for training
sel.prob <- rep(1,N)
sel.prob[population$xs == 1] <- 9
sel.prob[population$y == 1] <- 8
sel.prob[population$y == 1 & population$xs == 1] <- 150
ind <- sample(1:N, 200, prob = sel.prob)

data = population[ind, ]

## calculate weights from original numbers for xs and y
w.matrix <- table(population$y, population$xs)/table(data$y, data$xs)
w <- rep(NA, nrow(data))
w[data$y==0 & data$xs ==0] <- w.matrix[1,1]
w[data$y==1 & data$xs ==0] <- w.matrix[2,1]
w[data$y==0 & data$xs ==1] <- w.matrix[1,2]
w[data$y==1 & data$xs ==1] <- w.matrix[2,2]

### draw a test data set
newdata = population[sample(1:N, size=200 ), ]

n.bs = 20
```

```
pred_nb <- sambia::costing(data = data, weights = w,
  learner='naiveBayes', list.train.learner = list(formula=formula(y~.)),
  list.predict.learner = list(newdata=newdata, type="raw"),n.bs = n.bs, mod=TRUE)
roc(newdata$y, pred_nb, direction="<")
```

---

genSample *Generate synthetic observations using inverse-probability weights*

Description

This method corrects a given data set for sample selection bias by generating new observations via Stochastic inverse-probability oversampling or parametric inverse-probability sampling using inverse-probability weights and information on covariance structure of the given strata (Krautenbacher et al, 2017).

Usage

```
genSample(data, strata.variables = NULL, stratum = NULL, weights = rep(1,
  nrow(data)), distr = "mvnorm", type = c("parIP", "stochIP"))
```

Arguments

data	a data frame containing the observations rowwise, along with their corresponding categorical strata feature.
strata.variables	a character vector of the names determined by the categorical stratum features.
stratum	a numerical vector of the length of the number of rows of the data specifying the stratum ID. Either 'strata.variables' or 'stratum' has to be provided. This vector will not be included as a column in the resulting data set.
weights	a numerical vector whose length must coincide with the number of the rows of data. The i-th value contains the inverse-probability e.g. determines how often the i-th observation of data shall be replicated.
distr	character object that describes the distribution
type	character which decides which method is used to correct a given data set for sample selection bias. Stochastic Inverse-Probabiltiy oversampling is applied if type = 'stochIP' or Parametric Inverse-Probability Bagging if type = 'parIP'.

Value

$data data frame containing synthetic data which is corrected for sample selection bias by generating new observations via Stochastic inverse-probability oversampling or parametric inverse-probability oversampling.

$orig.data original data frame which shall to corrected

$stratum vector containing the stratum of each observation

$method a character indicating which method was used. If method = 'stochIP' then Stochastic Inverse-Probabiltiy oversampling was used, if method = 'parIP' the Parametric Inverse-Probability sampling was used.

$strata.tbl a data frame containing all variables and their feature occurences

$N number of rows in data

$n number of rows in original data

Author(s)

Norbert Krautenbacher, Kevin Strauss, Maximilian Mandl, Christiane Fuchs

References

Krautenbacher, N., Theis, F. J., & Fuchs, C. (2017). Correcting Classifiers for Sample Selection Bias in Two-Phase Case-Control Studies. Computational and mathematical methods in medicine, 2017.

Examples

```
## simulate data for a population
require(pROC)

set.seed(1342334)
N = 100000
x1 <- rnorm(N, mean=0, sd=1)
x2 <- rt(N, df=25)
x3 <- x1 + rnorm(N, mean=0, sd=.6)
x4 <- x2 + rnorm(N, mean=0, sd=1.3)
x5 <- rbinom(N, 1, prob=.6)
x6 <- rnorm(N, 0, sd = 1) # noise not known as variable
x7 <- x1*x5 # interaction
x <- cbind(x1, x2, x3, x4, x5, x6, x7)

## stratum variable (covariate)
xs <- c(rep(1,0.1*N), rep(0,(1-0.1)*N))

## effects
beta <- c(-1, 0.2, 0.4, 0.4, 0.5, 0.5, 0.6)
beta0 <- -2

## generate binary outcome
linpred.slopes <-  log(0.5)*xs + c(x %*% beta)
eta <-  beta0 + linpred.slopes

p <- 1/(1+exp(-eta)) # this is the probability P(Y=1|X), we want the binary outcome however:
y<-rbinom(n=N, size=1, prob=p) #

population <- data.frame(y,xs,x)

#### draw "given" data set
sel.prob <- rep(1,N)
```

```
sel.prob[population$xs == 1] <- 9
sel.prob[population$y == 1] <- 8
sel.prob[population$y == 1 & population$xs == 1] <- 150
ind <- sample(1:N, 200, prob = sel.prob)

data = population[ind, ]

## calculate weights from original numbers for xs and y
w.matrix <- table(population$y, population$xs)/table(data$y, data$xs)
w <- rep(NA, nrow(data))
w[data$y==0 & data$xs ==0] <- w.matrix[1,1]
w[data$y==1 & data$xs ==0] <- w.matrix[2,1]
w[data$y==0 & data$xs ==1] <- w.matrix[1,2]
w[data$y==1 & data$xs ==1] <- w.matrix[2,2]
## parametric IP bootstrap sample
sample1 <- sambia::genSample(data=data, strata.variables = c('y', 'xs'),
                          weights = w, type='parIP')
## stochastic IP oversampling; treating 'y' and 'xs' as usual input variable
## but using strata info unambiguously defined by the weights w
sample2 <- sambia::genSample(data=data,
                            weights = w, type='stochIP', stratum= round(w))
```

IPbag *Predicting outcomes using non-parametric Inverse-Probability bagging*

Description

This method trains classifiers by correcting them for sample selection bias via non-parametric inverse-probability bagging. This method fits classifiers from different resampled data whose observations are increased per stratum to correct for the bias in the original sample. The so attained ensemble of predictors is aggregated by averaging.

Usage

```
IPbag(..., learner, list.train.learner, list.predict.learner, n.bs)
```

Arguments

`...`	see the parameter ipOversampling() of package sambia.
`learner`	a character indicating which classifier is used to train. Note: set learner='rangerTree' if random forest should be applied as in Krautenbacher et al. (2017), i.e. the correction step is incorporated in the inherent random forest resampling procedure.
`list.train.learner`	a list of parameters specific to the classifier that will be trained. Note that the parameter 'data' need not to be provided in this list since the training data which the model will learn on is already attained by new sampled data produced by the method genSample().

`list.predict.learner`
: a list of parameters specifiying how to predict new data given the learned model. (This learned model is uniquely determined by parameters 'learner' and 'list.train.learner').

`n.bs`
: number of bootstramp samples.

Author(s)

Norbert Krautenbacher, Kevin Strauss, Maximilian Mandl, Christiane Fuchs

References

Krautenbacher, N., Theis, F. J., & Fuchs, C. (2017). Correcting Classifiers for Sample Selection Bias in Two-Phase Case-Control Studies. Computational and mathematical methods in medicine, 2017.

Examples

```
## simulate data for a population
require(pROC)

set.seed(1342334)
N = 100000
x1 <- rnorm(N, mean=0, sd=1)
x2 <- rt(N, df=25)
x3 <- x1 + rnorm(N, mean=0, sd=.6)
x4 <- x2 + rnorm(N, mean=0, sd=1.3)
x5 <- rbinom(N, 1, prob=.6)
x6 <- rnorm(N, 0, sd = 1) # noise not known as variable
x7 <- x1*x5 # interaction
x <- cbind(x1, x2, x3, x4, x5, x6, x7)

## stratum variable (covariate)
xs <- c(rep(1,0.1*N), rep(0,(1-0.1)*N))

## effects
beta <- c(-1, 0.2, 0.4, 0.4, 0.5, 0.5, 0.6)
beta0 <- -2

## generate binary outcome
linpred.slopes <-  log(0.5)*xs + c(x %*% beta)
eta <-  beta0 + linpred.slopes

p <- 1/(1+exp(-eta)) # this is the probability P(Y=1|X), we want the binary outcome however:
y<-rbinom(n=N, size=1, prob=p) #

population <- data.frame(y,xs,x)

#### draw "given" data set for training
sel.prob <- rep(1,N)
sel.prob[population$xs == 1] <- 9
sel.prob[population$y == 1] <- 8
sel.prob[population$y == 1 & population$xs == 1] <- 150
```

```
ind <- sample(1:N, 200, prob = sel.prob)

data = population[ind, ]

## calculate weights from original numbers for xs and y
w.matrix <- table(population$y, population$xs)/table(data$y, data$xs)
w <- rep(NA, nrow(data))
w[data$y==0 & data$xs ==0] <- w.matrix[1,1]
w[data$y==1 & data$xs ==0] <- w.matrix[2,1]
w[data$y==0 & data$xs ==1] <- w.matrix[1,2]
w[data$y==1 & data$xs ==1] <- w.matrix[2,2]

### draw a test data set
newdata = population[sample(1:N, size=200 ), ]

n.bs = 5
pred_nb <- sambia::IPbag(data = data, weights = w,
           learner='naiveBayes', list.train.learner = list(formula=formula(y~.)),
           list.predict.learner = list(newdata=newdata, type="raw"),
           n.bs = n.bs)
roc(newdata$y, pred_nb, direction="<")
```

`ipOversampling` *Plain replication of each observation by inverse-probability weights*

Description

This method corrects for the sample selection bias by the plain replication of each observation in the sample according to its IP weight, i.e. in a stratified random sample one replicates an observation of stratum h by the factor w_h.

Usage

```
ipOversampling(data, weights, normalize = FALSE)
```

Arguments

`data`	a data frame containing the observations rowwise, along with their corresponding categorical strata feature(s).
`weights`	a numerical vector whose length must coincide with the number of the rows of data. The i-th value contains the inverse-probability e.g. determines how often the i-th observation of data shall be replicated.
`normalize`	If weight vector should be normalized, i.e. the smallest entry of the vector will be set to 1.

Details

If the numeric vector contains numbers which are not natural but real, they will be rounded.

Author(s)

Norbert Krautenbacher, Kevin Strauss, Maximilian Mandl, Christiane Fuchs

Examples

```
library(smotefamily)
library(sambia)
data.example <- sample_generator(100,ratio = 0.80)
result <- gsub('n','0',data.example[,'result'])
result <- gsub('p','1',result)
data.example[,'result'] <- as.numeric(result)
weights <- data.example[,'result']
weights <- ifelse(weights==1,1,4)
sample <- sambia::ipOversampling(data.example,weights)
```

---

rejSamp *Rejection Sampling is a method used in sambia's function 'costing' (Krautenbacher et al, 2017).*

Description

Rejection Sampling is a method used in sambias costing function. It is sampling scheme that allows us to draw examples independently from a distribution X, given examples drawn independently from distribution Y.

Usage

```
rejSamp(data, weights)
```

Arguments

data	a data frame containing the observations rowwise, along with their corresponding categorical strata feature
weights	a numerical vector whose length must coincide with the number of the rows of data. The i-th value contains the inverse-probability e.g. determines how often the i-th observation of data shall be replicated.

Author(s)

Norbert Krautenbacher, Kevin Strauss, Maximilian Mandl, Christiane Fuchs

References

Krautenbacher, N., Theis, F. J., & Fuchs, C. (2017). Correcting Classifiers for Sample Selection Bias in Two-Phase Case-Control Studies. Computational and mathematical methods in medicine, 2017.

Examples

```
library(smotefamily)
library(sambia)
data.example <- sample_generator(100,ratio = 0.80)
result <- gsub('n','0',data.example[,'result'])
result <- gsub('p','1',result)
data.example[,'result'] <- as.numeric(result)
weights <- data.example[,'result']
weights <- ifelse(weights==1,1,4)
rej.sample <- sambia:::rejSamp(data=data.example, weights = weights)
```

`smoteMod` *smoteMod is a modified version of the 'synthetic minority oversampling technique to generate new data.*

Description

This method adapts SMOTE to the context of stratified random samples. Rather than enlarging only the minority class, smoteMod generates synthetic data for all strata with a weight bigger than 1. Note: this function has to apply SMOTE H-1 times: 1. subsample data by smallest stratum and a stratum to oversample 2. oversample with modified SMOTE function according to weight of the stratum 3. do this for the other H-2 to subsamples 4. build new data set with strata where H-1 strata contain synthetic data (stratum with smallest weight remains as is)

Usage

```
smoteMod(data.x, stratum, weights, data.y = NULL, K)
```

Arguments

`data.x`	A data frame or matrix of numeric-attributed dataset
`stratum`	a numerical vector of the same length as the number of the rows of data. Depending on the number of strata variables and their number of exposures each such combination is assigned to a numeric class id. The i-th entry of stratum contains the class id (and therefore class belonging) of the i-th row (=observation) of data.
`weights`	a numerical vector whose length must coincide with the number of the rows of data. The i-th value contains the inverse-probability e.g. determines how often the i-th observation of data shall be replicated.
`data.y`	A vector of a target class attribute corresponding to a dataset data.x.
`K`	The number of nearest neighbors during sampling process

Author(s)

Norbert Krautenbacher, Kevin Strauss, Maximilian Mandl, Christiane Fuchs

Examples

```
## simulate data for a population
require(pROC)

set.seed(1342334)
N = 100000
x1 <- rnorm(N, mean=0, sd=1)
x2 <- rt(N, df=25)
x3 <- x1 + rnorm(N, mean=0, sd=.6)
x4 <- x2 + rnorm(N, mean=0, sd=1.3)
x5 <- rbinom(N, 1, prob=.6)
x6 <- rnorm(N, 0, sd = 1) # noise not known as variable
x7 <- x1*x5 # interaction
x <- cbind(x1, x2, x3, x4, x5, x6, x7)

## stratum variable (covariate)
xs <- c(rep(1,0.1*N), rep(0,(1-0.1)*N))

## effects
beta <- c(-1, 0.2, 0.4, 0.4, 0.5, 0.5, 0.6)
beta0 <- -2

## generate binary outcome
linpred.slopes <-  log(0.5)*xs + c(x %*% beta)
eta <-  beta0 + linpred.slopes

p <- 1/(1+exp(-eta)) # this is the probability P(Y=1|X), we want the binary outcome however:
y<-rbinom(n=N, size=1, prob=p) #

population <- data.frame(y,xs,x)

#### draw "given" data set for training
sel.prob <- rep(1,N)
sel.prob[population$xs == 1] <- 9
sel.prob[population$y == 1] <- 8
sel.prob[population$y == 1 & population$xs == 1] <- 150
ind <- sample(1:N, 200, prob = sel.prob)

data = population[ind, ]

## calculate weights from original numbers for xs and y
w.matrix <- table(population$y, population$xs)/table(data$y, data$xs)
w <- rep(NA, nrow(data))
w[data$y==0 & data$xs ==0] <- w.matrix[1,1]
w[data$y==1 & data$xs ==0] <- w.matrix[2,1]
w[data$y==0 & data$xs ==1] <- w.matrix[1,2]
w[data$y==1 & data$xs ==1] <- w.matrix[2,2]

### draw a test data set
newdata = population[sample(1:N, size=200 ), ]

K = 5
```

```
genData = smoteMod(data.x = data[ , -which(colnames(data) %in% c('y', 'xs'))] ,
stratum = w, data.y = data$y, weights = w, K=K)
```

---

smoteNew *smoteNew is a necessary function that modifies the SMOTE algorithm.*

Description

smoteNewis a necessary function that modifies the SMOTE algorithm in the following ways: (1) correct bug in original smotefamily::SMOTE() function and (2) lets the user specifiy which class to be oversampled.

Usage

```
smoteNew(data.x, data.y, K, dup_size = 0, class.to.oversample)
```

Arguments

data.x	A data frame or matrix of numeric-attributed dataset
data.y	A vector of a target class attribute corresponding to a dataset X
K	The number of nearest neighbors during sampling process
dup_size	The number or vector representing the desired times of synthetic minority instances over the original number of majority instances
class.to.oversample	Class to be oversampled

Author(s)

Norbert Krautenbacher, Kevin Strauss, Maximilian Mandl, Christiane Fuchs

Examples

```
library(smotefamily)
library(sambia)
data.example = sample_generator(10000,ratio = 0.80)
genData = sambia:::smoteNew(data.example[,-3],data.example[,3],K = 5,class.to.oversample = 'p')
```

`synthIPbag` *Train a classifier via synthetic observations using inverse-probability weights*

Description

This method trains classifiers by correcting them for sample selection bias via stochastic inverse-probability oversampling or parametric inverse-probability bagging (Krautenbacher et al 2017). Classifiers are trained from differently resampled data whose observations are weighted by inverse-probability weights per stratum to correct for the bias in the original sample. The so attained ensemble of predictors is aggregated by averaging.

Usage

```
synthIPbag(..., learner, list.train.learner, list.predict.learner, n.bs)
```

Arguments

`...`	see the parameter genSample() of package sambia.
`learner`	a character indicating which classifier is used to train. Note: set learner='rangerTree' if random forest should be applied as in Krautenbacher et al. (2017), i.e. the correction step is incorporated in the inherent random forest resampling procedure.
`list.train.learner`	a list of parameters specific to the classifier that will be trained. Note that the parameter 'data' need not to be provided in this list since the training data which the model will learn on is already attained by new sampled data produced by the method genSample().
`list.predict.learner`	a list of parameters specifiying how to predict new data given the trained model.
`n.bs`	number of bootstramp samples. This trained model is uniquely determined by parameters 'learner' and 'list.train.learner'.

Author(s)

Norbert Krautenbacher, Kevin Strauss, Maximilian Mandl, Christiane Fuchs

References

Krautenbacher, N., Theis, F. J., & Fuchs, C. (2017). Correcting Classifiers for Sample Selection Bias in Two-Phase Case-Control Studies. Computational and mathematical methods in medicine, 2017.

Krautenbacher, N., Theis, F. J., & Fuchs, C. (2017). Correcting Classifiers for Sample Selection Bias in Two-Phase Case-Control Studies. Computational and mathematical methods in medicine, 2017.

Examples

```
## simulate data for a population
require(pROC)

set.seed(1342334)
N = 100000
x1 <- rnorm(N, mean=0, sd=1)
x2 <- rt(N, df=25)
x3 <- x1 + rnorm(N, mean=0, sd=.6)
x4 <- x2 + rnorm(N, mean=0, sd=1.3)
x5 <- rbinom(N, 1, prob=.6)
x6 <- rnorm(N, 0, sd = 1) # noise not known as variable
x7 <- x1*x5 # interaction
x <- cbind(x1, x2, x3, x4, x5, x6, x7)

## stratum variable (covariate)
xs <- c(rep(1,0.1*N), rep(0,(1-0.1)*N))

## effects
beta <- c(-1, 0.2, 0.4, 0.4, 0.5, 0.5, 0.6)
beta0 <- -2

## generate binary outcome
linpred.slopes <-  log(0.5)*xs + c(x %*% beta)
eta <-  beta0 + linpred.slopes

p <- 1/(1+exp(-eta)) # this is the probability P(Y=1|X), we want the binary outcome however:
y<-rbinom(n=N, size=1, prob=p) #

population <- data.frame(y,xs,x)

#### draw "given" data set for training
sel.prob <- rep(1,N)
sel.prob[population$xs == 1] <- 9
sel.prob[population$y == 1] <- 8
sel.prob[population$y == 1 & population$xs == 1] <- 150
ind <- sample(1:N, 200, prob = sel.prob)

data = population[ind, ]

## calculate weights from original numbers for xs and y
w.matrix <- table(population$y, population$xs)/table(data$y, data$xs)
w <- rep(NA, nrow(data))
w[data$y==0 & data$xs ==0] <- w.matrix[1,1]
w[data$y==1 & data$xs ==0] <- w.matrix[2,1]
w[data$y==0 & data$xs ==1] <- w.matrix[1,2]
w[data$y==1 & data$xs ==1] <- w.matrix[2,2]

### draw a test data set
newdata = population[sample(1:N, size=200 ), ]

n.bs = 10
```

```
## glm
pred_glm <- sambia::synthIPbag(data = data, weights = w, type='parIP',
                               strata.variables = c('y', 'xs'), learner='glm',
                          list.train.learner = list(formula=formula(y~.),family="binomial"),
                            list.predict.learner = list(newdata=newdata, type="response"),
                               n.bs = n.bs)
roc(newdata$y, pred_glm, direction = "<")

## random forest
pred_rf <- sambia::synthIPbag(data = data, weights = w, type='parIP',
                              strata.variables = c('y','xs'), learner='rangerTree',
                              list.train.learner = list(formula=formula(as.factor(y)~.)),
                              list.predict.learner = list(data=newdata),
                              n.bs = n.bs)
roc(newdata$y, pred_rf, direction = "<")
```

Index