Package ‘IRIC’

September 19, 2019

Type (Package)

Title Integrated R Library for Imbalanced Classification

Version 1.1.0

Data 2019-9-19

Author Bing Zhu, Zihan Gao, Junkai Zhao

Maintainer ``Bing Zhu” <zhubing@scu.edu.cn>

Description Imbalanced classification is a challenging issue in data mining and machine learning. To address this issue, a large number of solutions have been proposed. We introduce an R library called IRIC, which integrates a wide set of solutions for imbalanced classification. IRIC not only provides a new implementation of some state-of-art techniques for binary imbalanced classification, but also improves the efficiency of model building using parallel techniques. The library and its source code are made freely available.

License GPL (>= 3)

URL https://github.com/shuzhiquan/IRIC

Depends R(>= 3.0.0)

Imports RWeka, RANN, caret, foreach, doParallel, rpart, parallel

NeedsCompilation No

R topics documented:

ADASYN *Implementation of ADASYN Sampling*

Description

This function implements ADASYN sampling.

Usage

ADASYN (x, y, beta = 0.65, k = 5)

Arguments

x	A data frame of the predictors from training data
y	A vector of response variable from training data
beta	Balance level (0, 1], when beta=1, the dataset is fully balanced
k	Number of nearest neighbors

Value

newData	A data frame generated by ADASYN algorithm

References

H. B. He, E. Garcia, and S. Li. ADASYN: Adaptive synthetic sampling approach for imbalanced learning. IEEE International Joint Conference on Neural Networks, 2008, pp.1322-1328.

Examples

```
data(Korean)
sub <- createDataPartition(Korean$Churn,p=0.75,list=FALSE)
trainset <- Korean[sub,]
testset <- Korean[-sub,]
x <- trainset[, -11]
y <- trainset[, 11]
newData<- ADASYN(x, y, beta=0.8, k=5)
```

BalanceCascade *Implementation of BalanceCascade Algorithm*

Description

This function implements BalanceCascade algorithm for binary class imbalance classification.

Usage

BalanceCascade (x, y, iter = 4)

Arguments

x	A data frame of the predictors from training data
y	A vector of response variable from training data
iter	Number of iterations for base classifiers training
allowParallel	A logical number to control the parallel computing. If allowParallel = TRUE, the function is run using parallel techniques

Value

An object of class BalanceCascade, which is a list with the following components:

call	Function call
iter	Number of iterations for base classifiers training
classLabels	Names of class labels
base	Types of base learner
alphas	Weights of base learners
fits	Fitted ensembled model
thresh	Threshold for classification

References

X. Y. Liu, J. Wu and Z. H. Zhou. April 2009. Exploratory Undersampling for Class-Imbalance Learning. IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics,2009, 39(2), pp. 539-550.

Examples

```
data(Korean)
sub <- createDataPartition(Korean$Churn,p=0.75,list=FALSE)
trainset <- Korean[sub,]
testset <- Korean[-sub,]
x <- trainset[, -11]
y <- trainset[, 11]
model <- BalanceCascade(x, y, allowParallel = TRUE)
output<- predict (model, x)
```

---

bbaging *Implementation of Bagging-based Algorithm*

Description

This function implements bagging-based algorithm for imbalance classification. Four algorithms can be found in the current version: SMOTEBagging, RUSBagging, RBBagging and ROSBagging.

Usage

```
bbaging (x, y, numBag = 40, base = treeBag, type = "SMOTEBagging", allowParallel = FALSE)
```

Arguments

x	A data frame of the predictors from training data
y	A vector of response variable from training data
numBag	Number of bags
base	Types of base learner
type	Type of bagging-based algorithm, including “SMOTEBagging”, “RUSBagging”,“RBBagging” and “ROSBagging”
allowParallel	A logical number to control the parallel computing. If allowParallel = TRUE, the function is run using parallel techniques

Value

An object of class bbag, which is a list with the following components:

call	Function call
base	Types of base learner
type	Type of bagging-based algorithm
numBag	Number of bags
fits	Fitted bagging-based model

References

S. Hido, H. Kashima, Y. Takahashi. Roughly balanced bagging for imbalanced data. Statistical Analysis & Data Mining, 2009, 2(5-6), pp.412-426.

S. Wang, X. Yao. 2009. Diversity analysis on imbalanced data sets by using ensemble models. IEEE Symposium on Computational Intelligence, 2009, pp. 324–331

Examples

```
data(Korean)
sub <- createDataPartition(Korean$Churn, p=0.75,list=FALSE)
trainset <- Korean[sub,]
testset <- Korean[-sub,]
x <- trainset[, -11]
y <- trainset[, 11]
model <- bbagging(x, y, type = "SMOTEBagging", allowParallel=TRUE)
output <- predict (model, x)
```

---

bboost *Implementation of Boost-based Algorithm*

Description

This function implements boost-based algorithm for imbalance classification. Four algorithms can be found in the current version: Adaboost, SMOTEboost, RUSBoost, AdaC2

Usage

```
bboost (x, y, iter = 40, base = treeBoost, type = "AdaBoost")
```

Arguments

x	A data frame of the predictors from training data
y	A vector of response variable from training data
iter	Number of iterations for base classifiers training
base	Types of base learner
type	Type of boosting-based algorithm, including “Adaboost”, “SMOTEboost”, “RUSBoost”, “AdaC2”

Value

An object of class bboost, which is a list with the following components:

call	Function call
type	Type of boosting-based algorithm

base	Types of base learner
classLabels	Names of class labels
fits	Fitted boosting-based model
alpha	Weights of base learners

References

N. Chawla, A. Lazarevic, L. Hall, K.W. Bowyer. SMOTEBoost: improving prediction of the minority class in boosting. Proceeding of PKDD, 2003, pp. 107–119.
Y. Sun, M.S. Kamel, A.K. Wong, Y. Wang. 2007. Cost-sensitive boosting for classification of imbalanced data, Pattern Recognit, 2007, 40 (12), pp. 3358–3378.

Examples

```
data(Korean)
sub <- createDataPartition (Korean$Churn, p=0.75, list=FALSE)
trainset <- Korean[sub,]
testset <- Korean[-sub,]
x <- trainset[, -11]
y <- trainset[, 11]
model <- bboost(x, y, base = treeBoost, type = "AdaBoost")
output <- predict (model, x)
```

CLUS *Implementation of CLUS Sampling Algorithm*

Description

This function implements CLUS sampling (clustering-based undersampling), which selects the representative data for training data to improve the classification accuracy for minority class

Usage

```
CLUS (x, y, k = 3, m = 1.5)
```

Arguments

x	A data frame of the predictors from training data
y	A vector of response variable from training data
k	Number of clusters
m	Imbalanced ratio in output dataset

Value

newData	A data frame of the undersampled data using CLUS method

References

S. J. Yen and Y.S. Lee. Cluster-based under-sampling approaches for imbalanced data distributions. Expert Systems with Applications, 36(3), 2009, pp. 5718 – 5727.

Examples

```
data(Korean)
sub <- createDataPartition(Korean$Churn,p=0.75,list=FALSE)
trainset <- Korean[sub,]
testset <- Korean[-sub,]
x <- trainset[, -11]
y <- trainset[, 11]
newData<- CLUS(x, y, m=2)
```

CSC45 *Implementation of Cost-sensitive C4.5 Decision Tree Algorithm*

Description

This function implements cost-sensitive C4.5 decision tree using an instance-weighting method

Usage

```
CSC45(x, y, pruning = TRUE, minIns = 2, costRatio = 11/56)
```

Arguments

x	A data frame of the predictors from training data
y	A vector of response variable from training data
pruning	A logical number to determine whether to prune the tree. If pruning = TRUE, do the pruning process
minIns	Minimum number of instances for split
costRatio	CostRatio between Majority class and Minority class

Value

pruning	A logical number to indicate whether to prune the tree
tree	Fitted cost-sensitive C4.5 decision tree
classLabels	Names of class labels
costRatio	Ratio of misclassification cost between the majority and minority class

References

M. T. Kai. An instance-weighting method to induce cost-sensitive trees. IEEE Transactions on Knowledge & Data Engineering, 14(3), 2002, pp. 659–665

Examples

```
data(Korean)
sub <- createDataPartition(Korean$Churn,p=0.75,list=FALSE)
trainset <- Korean[sub,]
testset <- Korean[-sub,]
x <- trainset[, -11]
y <- trainset[, 11]
model <- CSC45(x, y, pruning = TRUE)
output <- predict (model, x)
```

EasyEnsemble *Implementation of EasyEnsemble Algorithm*

Description

This function implements EasyEnsemble algorithm for binary imbalance classification

Usage

EasyEnsemble(x, y, iter = 4, allowParallel = FALSE)

Arguments

x	A data frame of the predictors from training data
y	A vector of response variable from training data
iter	Number of iterations for base classifiers training
allowParallel	A logical number to control the parallel computing. If allowParallel = TRUE, the function is run using parallel techniques

Value

An object of class EasyEnsemble, which is a list with the following components:

call	Function call
iter	Number of iterations for base classifiers training
fits	Fitted ensembled model
base	Types of base learner
alphas	Weights of based learners
classLabels	names of class labels

References

X. Y. Liu, J. Wu and Z. H. Zhou. Exploratory Undersampling for Class-Imbalance Learning. IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 2009, 39(2), pp. 539-550.

Examples

```
data(Korean)
sub <- createDataPartition(Korean$Churn,p=0.75,list=FALSE)
trainset <- Korean[sub,]
testset <- Korean[-sub,]
x <- trainset[, -11]
y <- trainset[, 11]
model <- EasyEnsemble(x, y, allowParallel=TRUE)
output <- predict (model, x)
```

MWMOTE *Implementation of MWMOTE Sampling Algorithm*

Description

This function implements MWMOTE sampling (*Majority Weighted Minority Oversampling Technique)*

Usage

MWMOTE (x, y, percOver = 1400, k1 = 5, k2 = 5, CThresh = 3)

Arguments

x	A data frame of the predictors from training data
y	A vector of response variable from training data
percOver	Percent of new instance generated for each minority instance
k1	Number of neighbours for filtering
k2	Number of neighbours for selecting majority instances
CThresh	Threshold to determine the number of clusters

Value

newData	A data frame of the oversampled data using MWMOTE

References

S. Barua, M. M. Islam, X. Yao, K. Murase. MWMOTE–majority weighted minority oversampling technique for imbalanced data set learning. IEEE Transactions on Knowledge & Data Engineering, 2013, 26 (2), pp.405–425

Examples

```
data(Korean)
sub <- createDataPartition(Korean$Churn,p=0.75,list=FALSE)
trainset <- Korean[sub,]
testset <- Korean[-sub,]
x <- trainset[, -11]
y <- trainset[, 11]
newData<- MWMOTE(x, y)
```

---

predict.BalanceCascade *Predict Method for BalanceCascade Object*

Description

Predicting instances in test set using BalanceCascade object.

Usage

predict (object, x, type=”probability”)

Arguments

object	An object of BalanceCascde class.
x	A data frame of the predictors from testing data
type	Types of output, which can be **probability** and **class** (predicted label). Default is **probability**.

Value

Two types of output can be selected:

probability Estimated probability of being a minority instance. The probability is averaged by using an equal-weight majority vote by all weak learners.

class Predicted class of the instance. Instances of probability larger than 0.5 are predicted as 1, otherwise 0.

Examples

```
data(Korean)
sub <- createDataPartition(Korean$Churn,p=0.75,list=FALSE)
trainset <- Korean[sub,]
testset <- Korean[-sub,]
x<- trainset[, -11]
y<- trainset[, 11]
model <- BalanceCascade(x, y, allowParallel=TRUE)
output <- predict(model, x)
```

---

predict.bbaging *Predict Method for bbaging object*

Description

Predicting instances in test set using bbaging object.

Usage

```
predict (object, x, type=" probability")
```

Arguments

object	An object of bbaging class.
x	A data frame of the predictors from testing data
type	Types of output, which can be **probability** and **class** (predicted label). Default is **probability**.

Value

Two types of output can be selected:

probability Estimated probability of being a minority instance. The probability is averaged by using an equal-weight majority vote by all weak learners.

class Predicted class of the instance. Instances of probability larger than 0.5 are predicted as 1, otherwise 0.

Examples

```
data(Korean)
sub <- createDataPartition(Korean$Churn,p=0.75,list=FALSE)
trainset <- Korean[sub,]
testset <- Korean[-sub,]
x<- trainset[, -11]
y<- trainset[, 11]
model <- bbaging(x, y, type = "SMOTEBaging", allowParallel=TRUE)
output <- predict (model, x, type = "probability")    # return probability estimation
```

output <- predict (model, x, type = “class”)   # return predicted class

predict.bboost   *Predict Method for bboost Object*

Description

Predicting instances in test set using bboost object.

Usage

predict (object, x, type=” probability”)

Arguments

object	An object of bboost class.
x	A data frame of the predictors from testing data
type	Types of output, which can be **probability** and **class** (predicted label). Default is **probability**.

Value

Two types of output can be selected:

probability	Estimated probability of being a minority instance. The probability is averaged by using an equal-weight majority vote by all weak learners.
class	Predicted class of the instance. Instances of probability larger than 0.5 are predicted as 1, otherwise 0.

Examples

```
data(Korean)
sub <- createDataPartition(Korean$Churn,p=0.75,list=FALSE)
trainset <- Korean[sub,]
testset <- Korean[-sub,]
x<- trainset[, -11]
y<- trainset[, 11]
model <- bboost(x, y, base = treeBoost, type = "AdaC2")
output <- predict (model, x, type = “probability”)    # return probability estimation
output <- predict (model, x, type = “class”)    # return predicted class
```

predict.CSC45   *Predict Method for CSC4.5 Object*

Description

Predicting instances in test set using CSC4.5 object.

Usage

predict (object, x, y, type=”prob”)

Arguments

object An object of CSC4.5 class.

x A data frame of the predictors from testing data

type Types of output, which can be **prob** (probability) and **class** (predicted label). Default is **prob**.

Value

Two types of output can be selected:

prob Estimated probability of being a minority instance. The probability is averaged by using an equal-weight majority vote by all weak learners.

class Predicted class of the instance. Instances of probability larger than 0.5 are predicted as 1, otherwise 0.

Examples

```
data(Korean)
sub <- createDataPartition(Korean$Churn,p=0.75,list=FALSE)
trainset <- Korean[sub,]
testset <- Korean[-sub,]
x <- trainset[, -11]
y <- trainset[, 11]
model <- CSC45(x, y, pruning = TRUE)
output <- predict (model, x, type = "probability")    # return probability estimation
output <- predict (model, x, type = "class")    # return predicted class
```

---

predict.EasyEnsemble *Predict Method for EasyEnsemble Object*

Description

Predicting instances in test set using EasyEnsemble object.

Usage

```
predict (object, x, type = "probability")
```

Arguments

object An object of EasyEnsemble class.

x A data frame of the predictors from testing data

type Types of output, which can be **probability** and **class** (predicted label). Default is **probability**.

Value

Two types of output can be selected:

probability Estimated probability of being a minority instance. The probability is averaged by using an equal-weight majority vote by all weak learners.

class Predicted class of the instance. Instances of probability larger than 0.5 are predicted as 1, otherwise 0.

Examples

```
data(Korean)
```

```
sub <- createDataPartition(Korean$Churn,p=0.75,list=FALSE)
trainset <- Korean[sub,]
testset <- Korean[-sub,]
x <- trainset[, -11]
y <- trainset[, 11]
model <- EasyEnsemble(x, y, allowParallel=TRUE)
output <- predict (model, x, type = "probability")    # return probability estimation
output <- predict (model, x, type = "class")    # return predicted class
```

RandomSampling *Implementation of Random Sampling Algorithm*

Description

This function implements random undersampling and oversampling algorithm

Usage

```
RandomSampling (x, y, percOver = 0, percUnder = 6.8)
```

Arguments

x	A data frame of the predictors from training data
y	A vector of response variable from training data
percOver	Oversampling percentage
percUnder	Undersampling percentage

Value

newData	A data frame of the random oversampled/undersampled data

Examples

```
data(Korean)
sub <- createDataPartition(Korean$Churn,p=0.75,list=FALSE)
trainset <- Korean[sub,]
testset <- Korean[-sub,]
x <- trainset[, -11]
y <- trainset[, 11]
newData<- RandomSampling(x, y)
```

SMOTE *Implementation of SMOTE Algorithm*

Description

This function implements SMOTE sampling (Synthetic Minority Oversampling Technique)

Usage

```
SMOTE (x, y, percOver = 1400, k = 5)
```

Arguments

x	A data frame of the predictors from training data
y	A vector of response variable from training data
percOver	Percent of new instance generated for each minority instance
k	Number of nearest neighbors

Value

newData	A data frame of the oversampled data using SMOTE

References

Chawla, N., Bowyer, K., Hall, L. and Kegelmeyer, W. SMOTE: Synthetic minority oversampling technique. Journal of Artificial Intelligence Research, 2002, 16(3), pp. 321-357.

Examples

```
data(Korean)
sub <- createDataPartition(Korean$Churn,p=0.75,list=FALSE)
trainset <- Korean[sub,]
testset <- Korean[-sub,]
x <- trainset[, -11]
y <- trainset[, 11]
newData<- SMOTE(x, y)
```

SmoteENN *Implementation of SmoteENN Algorithm*

Description

This function implements SmoteENN *algorithm*, which combined SMOTE and data cleaning techniques ENN(Edited Nearest Neighbor)

Usage

```
SmoteENN (x, y, percOver, k1 = 5, k2 = 3, allowParallel= TRUE)
```

Arguments

x	A data frame of the predictors from training data
y	A vector of response variable from training data
percOver	Percent of new instance generated for each minority instance
k1	Number of the nearest neighbors in SMOTE
k2	Number of nearest neighbors in ENN
allowParallel	A logical number to control the parallel computing. If allowParallel = TRUE, the function is run using parallel techniques

Value

newData	A data frame after the application of SmoteENN

References

G. E. Batista, R. C. Prati, M. C. Monard. A study of the behavior of several methods for balancing machine learning training data. ACM SIGKDD Explorations Newsletter, 2004, 6 (1), pp. 20–29.

Examples

```
data(Korean)
sub <- createDataPartition(Korean$Churn,p=0.75,list=FALSE)
trainset <- Korean[sub,]
testset <- Korean[-sub,]
x <- trainset[, -11]
y <- trainset[, 11]
newData<- SmoteENN(x, y, percOver =1400 , allowParallel= TRUE)
```

SmoteTL *Implementation of SmoteTL Algorithm*

Description

This function implements SmoteTL, which performs over-sampling with SMOTE and clean data with Tomek Links.

Usage

```
SmoteTL (x, y, percOver, k)
```

Arguments

x	A data frame of the predictors from training data
y	A vector of response variable from training data
percOver	Percent of new instance generated for each minority instance
k	Number of nearest neighbors used in Smote

Value

newData	A data frame after the application of SmoteTL

References

G. E. Batista, R. C. Prati, M. C. Monard. A study of the behavior of several methods for balancing machine learning training data. ACM SIGKDD Explorations Newsletter, 2004, 6 (1), pp. 20–29.

Examples

```
data(Korean)
sub <- createDataPartition(Korean$Churn,p=0.75,list=FALSE)
trainset <- Korean[sub,]
testset <- Korean[-sub,]
x <- trainset[, -11]
```

```
y <- trainset[, 11]
newData<- SmoteTL(x, y, percOver = 1400)
```

SPIDER *Implementation of SPIDER Algorithm*

Description

This function implements Perform SPIDER (Selective Preprocessing of Imbalanced Data with ENN Rule) on imbalanced dataset, which filters difficult instances from the majority class after local over-sampling of the minority class

Usage

```
SPIDER (x, y, method = "weak", allowParallel = TRUE)
```

Arguments

x	A data frame of the predictors from training data
y	A vector of response variable from training data
method	Type of modification of the minority class in the second phase, including “weak”, “relabel”, “strong”
allowParallel	A logical number to control the parallel computing. If allowParallel = TRUE, the function is run using parallel techniques

Value

newData	A data frame after the application of SPIDER

References

J. Stefanowski, S. Wilk. Selective pre-processing of imbalanced data for improving classification performance. International Conference on Data Warehousing and Knowledge Discovery, 2008, pp. 283–292

Examples

```
data(Korean)
sub <- createDataPartition(Korean$Churn,p=0.75,list=FALSE)
trainset <- Korean[sub,]
testset <- Korean[-sub,]
x <- trainset[, -11]
y <- trainset[, 11]
newData<- SPIDER(x, y, method = "weak", allowParallel= TRUE)
```