

# Package ‘hyperSMURF’

August 18, 2016

**Type** Package

**Title** Hyper-Ensemble Smote Undersampled Random Forests

**Version** 1.1.2

**Date** 2016-08-18

**Author** Giorgio Valentini [aut, cre] -

AnacletoLab, Dipartimento di Informatica, Universita' degli Studi di Milano;

Max Schubach [ctb] - Charite, Universitätsmedizin Berlin;

Matteo Re [ctb] - AnacletoLab, Dipartimento di Informatica, Universita' degli Studi di Milano;

Peter N Robinson [ctb] - The Jackson Laboratory for Genomic Medicine, Farmington CT, USA.

**Maintainer** Giorgio Valentini <valentini@di.unimi.it>

**Description** Machine learning supervised method to learn rare genomic features in imbalanced genetic data sets. This method can be also applied to classify or rank examples characterized by a high imbalance between the minority and majority class. hyperSMURF adopts a hyper-ensemble (ensemble of ensembles) approach, undersampling of the majority class and oversampling of the minority class to learn highly imbalanced data. Both single-core and parallel multi-core version of hyperSMURF are implemented.

**License** GPL (>= 2)

**LazyLoad** yes

**Imports** unbalanced, randomForest, foreach, iterators, doParallel,  
parallel

**NeedsCompilation** no

## R topics documented:

hyperSMURF-package	2
do.random.partition	3
do.stratified.cv.data	4
do.stratified.cv.data.from.folds	5
hyperSMURF.corr.cv.parallel	6
hyperSMURF.cv	7
hyperSMURF.cv.parallel	9
hyperSMURF.test	10

hyperSMURF.test.parallel . . . . .	11
hyperSMURF.test.thresh . . . . .	12
hyperSMURF.train . . . . .	13
hyperSMURF.train.parallel . . . . .	14
imbalanced.data.generator . . . . .	16
smote . . . . .	17
smote_and_undersample . . . . .	18
<b>Index</b>	<b>19</b>

---

hyperSMURF-package	<i>Hyper-Ensemble Smote Undersampled Random Forests</i>
--------------------	---

---

**Description**

Machine learning supervised method to learn rare genomic features in imbalanced genetic data sets. This method can be also applied to classify or rank examples characterized by a high imbalance between the minority and majority class. hyperSMURF adopts a hyper-ensemble (ensemble of ensembles) approach, undersampling of the majority class and oversampling of the minority class to learn highly imbalanced data. Both single-core and parallel multi-core version of hyperSMURF are implemented.

**Details**

The DESCRIPTION file:

Package: hyperSMURF  
Type: Package  
Title: Hyper-Ensemble Smote Undersampled Random Forests  
Version: 1.1.2  
Date: 2016-08-18  
Author: Giorgio Valentini [aut, cre] - AnacletoLab, Dipartimento di Informatica, Universita' degli Studi di Milano; MA  
Maintainer: Giorgio Valentini <valentini@di.unimi.it>  
Description: Machine learning supervised method to learn rare genomic features in imbalanced genetic data sets. This meth  
License: GPL (>= 2)  
LazyLoad: yes  
Imports: unbalanced, randomForest, foreach, iterators, doParallel, parallel

Index of help topics:

do.random.partition	Random partition of the data
do.stratified.cv.data	Construction of random folds for cross-validation
do.stratified.cv.data.from.folds	Construction of folds for cross-validation from predefined folds
hyperSMURF-package	Hyper-Ensemble Smote Undersampled Random

	Forests
hyperSMURF.corr.cv.parallel	hyperSMURF cross-validation with embedded correlation-based feature selection
hyperSMURF.cv	hyperSMURF cross-validation
hyperSMURF.cv.parallel	hyperSMURF cross-validation - parallel implementation
hyperSMURF.test	Test of a hyperSMURF model
hyperSMURF.test.parallel	Test of a hyperSMURF model - parallel version
hyperSMURF.test.thresh	Test of a thresholded hyperSMURF model
hyperSMURF.train	hyperSMURF training
hyperSMURF.train.parallel	hyperSMURF training - parallel version
imbalanced.data.generator	Synthetic imbalanced data generator
smote	SMOTE oversampling
smote_and_undersample	SMOTE oversampling and undersampling

**Author(s)**

Giorgio Valentini [aut, cre] - AnacletoLab, Dipartimento di Informatica, Università degli Studi di Milano; Max Schubach [ctb] - Charite, Universitätsmedizin Berlin; Matteo Re [ctb] - AnacletoLab, Dipartimento di Informatica, Università degli Studi di Milano; Peter N Robinson [ctb] - The Jackson Laboratory for Genomic Medicine, Farmington CT, USA.

Maintainer: Giorgio Valentini <valentini@di.unimi.it>

---

do.random.partition     *Random partition of the data*

---

**Description**

Performs a random partition of the indices that refer to a given data set (data frame or matrix)

**Usage**

```
do.random.partition(n.ex, n.partitions, seed = 0)
```

**Arguments**

n.ex	number of indices to be partitioned
n.partitions	number of partitions
seed	seed for the random generator

**Details**

The partition of the data is performed using only the indices of the data not the data itself

**Value**

a list with n.partitions elements. Each element stores the indices of the partition.

**Examples**

```
do.random.partition(100, 10)
```

---

do.stratified.cv.data *Construction of random folds for cross-validation*

---

**Description**

The function randomly generates fold data for cross-validation

**Usage**

```
do.stratified.cv.data(examples, positives, k = 10, seed = 0)
```

**Arguments**

examples	vector of integer: indices of the examples
positives	vector of integer: Indices of the positive examples. The indices refer to the indices of examples
k	number of folds (def = 10)
seed	seed of the random generator (def=0). If is set to 0 no initialization is performed

**Details**

he folds are separated for positive and negative examples. The elements included in each fold are obtained by random sampling the data.

**Value**

a list with two components;

fold.non.positives

a list with k components. Each component is a vector with the indices of the non positive elements of the fold

old.positives a list with k components. Each component is a vector with the indices of the positive elements of the fold

**See Also**

[do.stratified.cv.data.from.folds](#)

**Examples**

```
do.stratified.cv.data(1:100, 1:20, k = 10)
```

---

```
do.stratified.cv.data.from.folds
```

*Construction of folds for cross-validation from predefined folds*

---

**Description**

The function generates data for cross-validation from pre-computed folds

**Usage**

```
do.stratified.cv.data.from.folds(examples, positives, folds, k = 10)
```

**Arguments**

examples	vector of integer: indices of the examples
positives	vector of integer: Indices of the positive examples. The indices refer to the indices of examples
folds	vector of indices : its length is equal to examples, with values in the interval $[0, k)$ . The value indicates the partition in the cross validation step of the class
k	number of folds (def = 10)

**Details**

The folds are separated for positive and negative examples. The elements included in each fold are obtained from the vector of fold indices folds.

**Value**

a list with two components;

fold.non.positives

a list with k components. Each component is a vector with the indices of the non positive elements of the fold

old.positives

a list with k components. Each component is a vector with the indices of the positive elements of the fold

**See Also**

[do.stratified.cv.data](#)

**Examples**

```
do.stratified.cv.data.from.folds(1:100, 1:10, folds=sample(rep((0:4),20)), k = 5)
```

---

hyperSMURF.corr.cv.parallel

*hyperSMURF cross-validation with embedded correlation-based feature selection*

---

## Description

This function implements the automated cross-validation procedure with hyperSMURF (hyper-ensemble SMote Undersampled Random Forests), using at the same time a correlation-based feature selection to select the best features to train the hyper-ensemble.

## Usage

```
hyperSMURF.corr.cv.parallel(data, y, kk = 5, n.part = 10, fp = 1,
  ratio = 1, k = 5, ntree = 10, mtry = 5, n.feature = 0, seed = 0,
  fold.partition = NULL, ncores = 0, file = "")
```

## Arguments

data	a data frame or matrix with the data
y	a factor with the labels. 0:majority class, 1: minority class.
kk	number of folds (def: 5)
n.part	number of partitions (def. 10)
fp	multiplicative factor for the SMOTE oversampling of the minority class If fp<1 no oversampling is performed.
ratio	ratio of the #majority/#minority
k	number of the nearest neighbours for SMOTE oversampling (def. 5)
ntree	number of trees of the base learner random forest
mtry	number of the features to randomly selected by the decision tree of each base random forest
n.feature	number of the features to be selected in the training set according to the absolute value of the correlation coefficient. If 0 (def), the top 5% are selected.
seed	initialization seed for the random generator (if set to 0(def.) no initialization is performed)
fold.partition	vector of size nrow(data) with values in interval $[0, kk)$ . The values indicate the fold of the cross validation of each example. If NULL (default) the folds are randomly generated.
ncores	number of cores. If 0, the max number of cores - 1 is selected
file	name of the file where the cross-validated hyperSMURF models will be saved. If file=="" (def.) no model is saved.

## Details

The cross-validation is performed by randomly constructing the folds (parameter `fold.partition = NULL`) or using a set of predefined folds listed in the parameter vector `fold.partition`. The cross validation is performed by training and testing in parallel the base random forests. To this end the parameter `ncores` allows to choose the number of cores to be used. Note that by selecting a large number of cores a larger primary memory is needed, and this can be an issue if the data to be analyzed are relatively large with respect to the available RAM memory. At each step of the cross validation a subset of features is selected on the training set by choosing the features most correlated (according to the Pearson correlation) with the response variable and then the selected features are used to train and test the hyper-ensemble.

## Value

a vector with the cross-validated hyperSMURF probabilities (hyperSMURF scores).

## See Also

[hyperSMURF.cv](#), [hyperSMURF.cv.parallel](#)

## Examples

```
d <- imbalanced.data.generator(n.pos=10, n.neg=160, n.features=7,
                              n.inf.features=1, sd=0.3, seed=1);
if (Sys.info()['sysname']!="Windows")
  res<-hyperSMURF.corr.cv.parallel (d$data, d$labels, kk=2, n.part=2, fp=1, ratio=2, k=5,
                                   ntree=5, mtry=2, n.feature=3, seed = 1, fold.partition=NULL, ncores=2, file="");
```

---

hyperSMURF.cv

*hyperSMURF cross-validation*

---

## Description

Automated cross validation of hyperSMURF (hyper-ensemble SMote Undersampled Random Forests)

## Usage

```
hyperSMURF.cv(data, y, kk = 5, n.part = 10, fp = 1, ratio = 1,
               k = 5, ntree = 10, mtry = 5, cutoff = c(0.5, 0.5), thresh = FALSE,
               seed = 0, fold.partition = NULL, file = "")
```

## Arguments

<code>data</code>	a data frame or matrix with the data
<code>y</code>	a factor with the labels. 0: majority class, 1: minority class.
<code>kk</code>	number of folds (def: 5)
<code>n.part</code>	number of partitions (def. 10)

<code>fp</code>	multiplicative factor for the SMOTE oversampling of the minority class. If <code>fp &lt; 1</code> no oversampling is performed.
<code>ratio</code>	ratio of the <code>#majority/#minority</code>
<code>k</code>	number of the nearest neighbours for SMOTE oversampling (def. 5)
<code>ntree</code>	number of trees of the base learner random forest (def. 10)
<code>mtry</code>	number of the features to randomly selected by the decision tree of each base random forest (def. 5)
<code>cutoff</code>	a numeric vector of length 2. Cutoff for respectively the majority and minority class. This parameter is meaningful when used with the thresholded version of hyperSMURF parameter ( <code>thresh=TRUE</code> )
<code>thresh</code>	logical. If <code>TRUE</code> the thresholded version of hyperSMURF is executed (def: <code>FALSE</code> )
<code>seed</code>	initialization seed for the random generator. If set to 0(def.) no initialization is performed
<code>fold.partition</code>	vector of size <code>nrow(data)</code> with values in interval <code>[0,kk)</code> . The values indicate the fold of the cross validation of each example. If <code>NULL</code> (default) the folds are randomly generated.
<code>file</code>	name of the file where the cross-validated hyperSMURF models will be saved. If <code>file==""</code> (def.) no model is saved.

## Details

The cross-validation is performed by randomly constructing the folds (parameter `fold.partition = NULL`) or using a set of predefined folds listed in the parameter `fold.partition`. The cross validation is performed by training and testing in sequence the base random forests. More precisely for each training set constructed at each step of the cross validation a separated random forest is trained sequentially for each of the `n.part` partitions of the data, by oversampling the minority class (parameter `fp`) and undersampling the majority class (parameter `ratio`). The random forest parameters `ntree` and `mtry` are the same for all the random forest of the hyper-ensemble.

## Value

a vector with the cross-validated hyperSMURF probabilities (hyperSMURF scores).

## See Also

[hyperSMURF.corr.cv.parallel](#), [hyperSMURF.corr.cv.parallel](#)

## Examples

```
d <- imbalanced.data.generator(n.pos=10, n.neg=300, sd=0.3);
res<-hyperSMURF.cv(d$data, d$labels, kk=2, n.part=3, fp=1, ratio=1, k=3, ntree=7,
  mtry=2, seed = 1, fold.partition=NULL);
```



---

hyperSMURF.cv.parallel

*hyperSMURF cross-validation – parallel implementation*


---

## Description

Automated cross validation of hyperSMURF (hyper-ensemble SMote Undersampled Random Forests) with both training and testing phase parallelized.

## Usage

```
hyperSMURF.cv.parallel(data, y, kk = 5, n.part = 10, fp = 1, ratio = 1, k = 5,
  ntree = 10, mtry = 5, seed = 0, fold.partition = NULL, ncores = 0, file = "")
```

## Arguments

data	a data frame or matrix with the data
y	a factor with the labels. 0:majority class, 1: minority class.
kk	number of folds (def: 5)
n.part	number of partitions (def. 10)
fp	multiplicative factor for the SMOTE oversampling of the minority class If fp<1 no oversampling is performed.
ratio	ratio of the #majority/#minority
k	number of the nearest neighbours for SMOTE oversampling (def. 5)
ntree	number of trees of the base learner random forest
mtry	number of the features to randomly selected by the decision tree of each base random forest
seed	initialization seed for the random generator (if set to 0(def.) no initialization is performed)
fold.partition	vector of size nrow(data) with values in interval $[0, kk)$ . The values indicate the fold of the cross validation of each example. If NULL (default) the folds are randomly generated.
ncores	number of cores. If 0, the max number of cores - 1 is selected
file	name of the file where the cross-validated hyperSMURF models will be saved. If file=="" (def.) no model is saved.

## Details

The cross-validation is performed by randomly constructing the folds (parameter fold.partition = NULL) or using a set of predefined folds listed in the parameter fold.partition. The cross validation is performed by training and testing in parallel the base random forests. More precisely for each training set constructed at each step of the cross validation a separated random forest is trained in each of the n.part partitions of the data, by oversampling the minority class (parameter

fp) and undersampling the majority class (parameter ratio). The random forest parameters ntree and mtry are the same for all the random forest of the hyper-ensemble. The parameter ncores allows to choose the number of cores to be used. Note that the selection of a large number of cores when data to be analyzed are large can be an issue if the available RAM memory is relatively small.

### Value

a vector with the cross-validated hyperSMURF probabilities (hyperSMURF scores).

### See Also

[hyperSMURF.cv](#), [hyperSMURF.corr.cv.parallel](#)

### Examples

```
# construction of a synthetic unbalanced data set
d <- imbalanced.data.generator(n.pos=10, n.neg=150, n.features=7,
                              n.inf.features=2, sd=0.1);
if (Sys.info()['sysname']!="Windows")
  res<-hyperSMURF.cv.parallel(d$data, d$labels, kk=2, n.part=2, fp=1, ratio=1,
                              k=1, ntree=5, mtry=2, seed = 1, fold.partition=NULL, ncores=2, file="");
```

---

hyperSMURF.test

*Test of a hyperSMURF model*

---

### Description

A hyperSMURF model is tested on a given data set. Predictions of each RF of the hyperensemble are performed sequentially and the scores of each ensemble are finally averaged.

### Usage

```
hyperSMURF.test(data, HSmodel)
```

### Arguments

data	a data frame or matrix with the test data. Rows: examples; columns: features
HSmodel	a list including the trained random forest models. The models have been trained with <a href="#">hyperSMURF.train.parallel</a> or with <a href="#">hyperSMURF.train</a>

### Value

a named vector with the computed probabilities for each example (hyperSMURF score)

### See Also

[hyperSMURF.test.parallel](#), [hyperSMURF.train.parallel](#), [hyperSMURF.train](#)

**Examples**

```

train <- imbalanced.data.generator(n.pos=20, n.neg=1000,
  n.features=10, n.inf.features=2, sd=0.1, seed=1);
HModel <- hyperSMURF.train(train$data, train$label,
  n.part = 5, fp = 1, ratio = 2, k = 5);
test <- imbalanced.data.generator(n.pos=20, n.neg=1000,
  n.features=10, n.inf.features=2, sd=0.1, seed=2);
res <- hyperSMURF.test(test$data, HModel);
y <- ifelse(test$labels==1,1,0);
pred <- ifelse(res>0.5,1,0);
table(pred,y);

```

---

hyperSMURF.test.parallel

*Test of a hyperSMURF model – parallel version*


---

**Description**

A hyperSMURF model is tested on a given data set. Predictions are performed in parallel: more precisely each RF of the hyperensemble is executed independently and in parallel and the scores are finally averaged.

**Usage**

```
hyperSMURF.test.parallel(data, HModel, ncores = 0)
```

**Arguments**

data	a data frame or matrix with the test data. Rows: examples; columns: features
HModel	a list including the trained random forest models. The models have been trained with <a href="#">hyperSMURF.train.parallel</a> or with <a href="#">hyperSMURF.train</a>
ncores	number of cores used for the parallel execution. If 0, the max number of cores - 1 is selected

**Value**

a named vector with the computed probabilities for each example (hyperSMURF score)

**See Also**

[hyperSMURF.test](#), [hyperSMURF.train.parallel](#), [hyperSMURF.train](#)

**Examples**

```

train <- imbalanced.data.generator(n.pos=10, n.neg=200,
                                   n.features=10, n.inf.features=2, sd=0.2, seed=1);
if (Sys.info()['sysname']!="Windows")
  HSmmodel <- hyperSMURF.train.parallel(train$data, train$label,
                                       n.part = 4, fp = 1, ratio = 2, k = 3, ncores=2);
test <- imbalanced.data.generator(n.pos=10, n.neg=200,
                                   n.features=10, n.inf.features=2, sd=0.2, seed=2);
if (Sys.info()['sysname']!="Windows") {
  res <- hyperSMURF.test.parallel(test$data, HSmmodel, ncores=2);
  y <- ifelse(test$labels==1,1,0);
  pred <- ifelse(res>0.5,1,0);
  table(pred,y);
}

```

---

hyperSMURF.test.thresh

*Test of a thresholded hyperSMURF model*


---

**Description**

The predictions of each random forest are discrete, i.e. 1 or 0: the probabilities are thresholded according to the cutoff value set in the training phase. The threshold is embedded in the HSmmodel according to the cutoff parameter set in the training phase. The score computed by the hyperensemble is the average of the discrete predictions generated by each base random forest.

**Usage**

```
hyperSMURF.test.thresh(data, HSmmodel)
```

**Arguments**

data	a data frame or matrix with the test data. Rows: examples; columns: features
HSmmodel	a list including the trained random forest models. The models have been trained with <a href="#">hyperSMURF.train.parallel</a> or with <a href="#">hyperSMURF.train</a> . The threshold is embedded in the model according to the cutoff value set in the training phase.

**Value**

a named vector with the computed probabilities for each example (HyperSMURF thresholded score)

**See Also**

[hyperSMURF.test](#), [hyperSMURF.test.parallel](#), [hyperSMURF.train.parallel](#), [hyperSMURF.train](#)

**Examples**

```
train <- imbalanced.data.generator(n.pos=20, n.neg=500,
                                   n.features=10, n.inf.features=2, sd=0.1, seed=1);
HModel <- hyperSMURF.train(train$data, train$label, n.part = 5,
                           fp = 1, ratio = 2, k = 5, cutoff=c(0.3, 0.7));
test <- imbalanced.data.generator(n.pos=20, n.neg=500,
                                   n.features=10, n.inf.features=2, sd=0.1, seed=2);
res <- hyperSMURF.test.thresh(test$data, HModel);
```

---

hyperSMURF.train	<i>hyperSMURF training</i>
------------------	----------------------------

---

**Description**

A hyperSMURF model is trained on a given data set. Training data are partitioned, and each RF is separately trained on each partition by SMOTE oversampling of the positives (minority class examples) and undersampling of the negatives (majority class examples). Each RF is trained sequentially

**Usage**

```
hyperSMURF.train(data, y, n.part = 10, fp = 1, ratio = 1, k = 5, ntree = 10,
                  mtry = 5, cutoff = c(0.5, 0.5), seed = 0, file = "")
```

**Arguments**

data	a data frame or matrix with the train data. Rows: examples; columns: features
y	a factor with the labels. 0:majority class, 1: minority class.
n.part	number of partitions (def. 10)
fp	multiplicative factor for the SMOTE oversampling of the minority class. If fp<1 no oversampling is performed.
ratio	ratio of the #majority/#minority
k	number of the nearest neighbours for SMOTE oversampling (def. 5)
ntree	number of trees of the base learner random forest (def. 10)
mtry	number of the features to randomly selected by the decision tree of each base random forest (def.5)
cutoff	a numeric vector of length 2. Cutoff for respectively the majority and minority class. This parameter is meaningful when used with the thresholded version of hyperSMURF (parameter thresh=TRUE)
seed	initialization seed for the random generator. If set to 0(def.) no initialization is performed
file	name of the file where the cross-validated hyperSMURF models will be saved. If file=="" (def.) no model is saved.

**Details**

A different random forest is trained on each partition of the training set. If `npos` and `nneg` are the the number of respectively the positive and negative examples, for each partition of the training data `fp*npos` new synthetic positives constructed by the SMOTE algorithm are added to the training set. The number of negatives is set to `ratio*(fp*npos + npos)`. If no enough negatives are available in the partition, then all the negatives in the partition are used to train the base RF associated to the partition.

**Value**

A list of trained RF models. Each element of the list is a `randomForest` objects of the homonymous package.

**See Also**

[hyperSMURF.test](#), [hyperSMURF.test.parallel](#), [hyperSMURF.train.parallel](#)

**Examples**

```
train <- imbalanced.data.generator(n.pos=20, n.neg=1000,
                                   n.features=10, n.inf.features=2, sd=1, seed=1);
HModel <- hyperSMURF.train(train$data, train$label, n.part = 5, fp = 1, ratio = 2);
```

---

`hyperSMURF.train.parallel`

*hyperSMURF training – parallel version*

---

**Description**

A hyperSMURF model is trained on a given data set. Training data are partitioned, and each RF is separately trained on each partition by SMOTE oversampling of the positives (minority class examples) and undersampling of the negatives (majority class examples). Each RF is trained independently and using parallel computation.

**Usage**

```
hyperSMURF.train.parallel(data, y, n.part = 10, fp = 1, ratio = 1, k = 5,
                           ntree = 10, mtry = 5, cutoff = c(0.5, 0.5), seed = 0, ncores = 0, file = "")
```

**Arguments**

<code>data</code>	a data frame or matrix with the train data. Rows: examples; columns: features
<code>y</code>	a factor with the labels. 0:majority class, 1: minority class.
<code>n.part</code>	number of partitions (def. 10)
<code>fp</code>	multiplicative factor for the SMOTE oversampling of the minority class. If <code>fp&lt;1</code> no oversampling is performed.

ratio	ratio of the #majority/#minority
k	number of the nearest neighbours for SMOTE oversampling (def. 5)
ntree	number of trees of the base learner random forest (def. 10)
mtry	number of the features to randomly selected by the decision tree of each base random forest (def.5)
cutoff	a numeric vector of length 2. Cutoff for respectively the majority and minority class. This parameter is meaningful when used with the thresholded version of hyperSMURF (parameter thresh=TRUE)
seed	initialization seed for the random generator. If set to 0(def.) no initialization is performed
ncores	number of cores used for the parallel execution. If 0, the max number of cores - 1 is selected
file	name of the file where the cross-validated hyperSMURF models will be saved. If file=="" (def.) no model is saved.

## Details

A different random forest is trained on each partition of the training set. If npos and nneg are the the number of respectively the positive and negative examples, for each partition of the training data  $fp \times npos$  new synthetic positives constructed by the SMOTE algorithm are added to the training set. The number of negatives is set to  $ratio \times (fp \times npos + npos)$ . If no enough negatives are available in the partition, then all the negatives in the partition are used to train the base RF associated to the partition. Each random forests are trained in parallel by exploiting the multi-core architecture of the processors.

## Value

A list of trained RF models. Each element of the list is a randomForest objects of the homonymous package.

## See Also

[hyperSMURF.test](#), [hyperSMURF.test.parallel](#), [hyperSMURF.train](#)

## Examples

```
train <- imbalanced.data.generator(n.pos=20, n.neg=500,
                                   n.features=10, n.inf.features=2, sd=1, seed=1);
if (Sys.info()['sysname']!="Windows")
  HSmodel <- hyperSMURF.train.parallel(train$data, train$label,
                                       n.part = 6, fp = 1, ratio = 2, k = 3, ncores=2);
```

---

imbalanced.data.generator

*Synthetic imbalanced data generator*


---

## Description

A variable number of minority and majority class examples are generated. All the features of the majority class are distributed according to a Gaussian distribution with mean=0 and sd=1. Of the overall n.features, n.inf. features of the minority class are distributed according to a gaussian centered in 1 with standard deviation sd.

## Usage

```
imbalanced.data.generator(n.pos=100, n.neg=2000,
                          n.features=10, n.inf.features=2, sd=1, seed=0)
```

## Arguments

n.pos	number of positive (minority class) examples (def. 100)
n.neg	number of negative (majority class) examples (def. 2000)
n.features	total number of features (def. 10)
n.inf.features	number of informative features (def. 2)
sd	standard deviation of the informative features (def.1)
seed	initialization seed for the random number generator. If 0 (def) current clock time is used.

## Value

A list with two elements:

data	the matrix of the synthetic data having pos+n.neg rows and n.features columns
labels	a factor with the labels of he examples: 1 for minority and 0 for majority class

## Examples

```
imbalanced.data.generator(n.pos=10, n.neg=200, n.features=6, n.inf.features=2, sd=1)
```



---

smote	<i>SMOTE oversampling</i>
-------	---------------------------

---

## Description

Function to oversample by SMOTE the minority class

## Usage

```
smote(data, fp = 1, k = 5)
```

## Arguments

data	data frame or matrix of data including only the minority class. Rows: examples; columns: features
fp	multiplicative factor for the SMOTE oversampling of the minority class (def=1). If fp<1 no oversampling is performed.
k	number of the nearest neighbours for SMOTE oversampling (def. 5)

## Details

If n is the number of examples of the minority class, then fp\*n new synthetic examples are generated according to the SMOTE algorithm and returned in addition to the original set of positives. If fp<1 no new data are generated and the original data set is returned

## Value

a data frame including the original minority class examples plus the SMOTE oversampled data

## See Also

[smote\\_and\\_undersample](#)

## Examples

```
d <- imbalanced.data.generator(n.pos=20, n.neg=1000, n.features=12, n.inf.features=2, sd=1, seed=1);  
res <- smote(d$data[d$label==1,], fp = 2, k = 3);
```

---

smote\_and\_undersample *SMOTE oversampling and undersampling*


---

### Description

Function to both oversample by SMOTE the minority class and undersample the majority class

### Usage

```
smote_and_undersample(data, y, fp = 1, ratio = 1, k = 5)
```

### Arguments

data	a data frame or matrix. Rows: examples; columns: features
y	a factor with the labels. 0: majority class, 1: minority class.
fp	multiplicative factor for the SMOTE oversampling of the minority class. If fp<1 no oversampling is performed.
ratio	ratio of the #majority/#minority
k	number of the nearest neighbours for SMOTE oversampling (def. 5)

### Details

If n is the number of examples of the minority class, then fp\*n new synthetic examples are generated according to the SMOTE algorithm and ratio\*(fp\*n + n) negative examples are undersampled from the majority class.

### Value

A list with two entries:

X	a data frame including the original minority class examples plus the SMOTE oversampled and undersampled data
Y	a factor with the labels of the data frame

### See Also

[smote](#)

### Examples

```
d <- imbalanced.data.generator(n.pos=20, n.neg=1000, n.features=12, n.inf.features=2, sd=1, seed=1);
res <- smote_and_undersample(d$data, d$label, fp = 2, ratio = 3);
```

# Index

`do.random.partition`, [3](#)  
`do.stratified.cv.data`, [4](#), [5](#)  
`do.stratified.cv.data.from.folds`, [4](#), [5](#)  
  
`hyperSMURF` (`hyperSMURF`-package), [2](#)  
`hyperSMURF`-package, [2](#)  
`hyperSMURF.corr.cv.parallel`, [6](#), [8](#), [10](#)  
`hyperSMURF.cv`, [7](#), [7](#), [10](#)  
`hyperSMURF.cv.parallel`, [7](#), [9](#)  
`hyperSMURF.test`, [10](#), [11](#), [12](#), [14](#), [15](#)  
`hyperSMURF.test.parallel`, [10](#), [11](#), [12](#), [14](#),  
[15](#)  
`hyperSMURF.test.thresh`, [12](#)  
`hyperSMURF.train`, [10–12](#), [13](#), [15](#)  
`hyperSMURF.train.parallel`, [10–12](#), [14](#), [14](#)  
  
`imbalanced.data.generator`, [16](#)  
  
`smote`, [17](#), [18](#)  
`smote_and_undersample`, [17](#), [18](#)