UBL: an R Package for Utility-Based Learning

Paula Branco, Rita P. Ribeiro and Luis Torgo
FCUP - LIAAD/INESC Tec
University of Porto
{paula.branco,rpribeiro,ltorgo}@dcc.fc.up.pt

September 13, 2018

Abstract

This document describes the R package UBL that allows the use of several methods for handling utility-based learning problems. Classification and regression problems that assume non-uniform costs and/or benefits pose serious challenges to predictive analytic tasks. In the context of meteorology, finance, medicine, ecology, among many other, specific domain information concerning the preference bias of the users must be taken into account to enhance the models predictive performance. To deal with this problem, a large number of techniques was proposed by the research community for both classification and regression tasks. The main goal of UBL package is to facilitate the utility-based predictive analytic task by providing a set of methods to deal with this type of problems in the R environment. It is a versatile tool that provides mechanisms to handle both regression and classification (binary and multiclass) tasks. Moreover, UBL package allows the user to specify his domain preferences, but it also provides some automatic methods that try to infer those preference bias from the domain, considering some common known settings.

1 Introduction

This document describes the methods available in package UBL [1] to deal with utility-based problems. UBL package aims at providing a diverse set of methods to address predictive tasks where the user has a non-uniform preference bias across the domain. The package provides tools suitable for both classification and regression tasks. All the methods available in UBL package were extended for being able to deal with multiclass problems and with regression problems possibly containing several relevant regions across the target variable domain.

Utility-based problems are defined in the context of predictive tasks where the user has a differentiated interest over the domain. This means that, in this type of problems, the user has non-uniform benefits for the correct predictions and/or assumes non-uniform costs for different errors. Many real world applications are utility-based learning problems because they encompass domain specific information which, if disregarded, may strongly penalize the performance of predictive models. This happens in the context of meteorology,

[1] This document was written for UBL package version 0.0.7.

finance, medicine, ecology, among many other, where specific domain information concerning the user preferences must be taken into account to enhance the models predictive performance.

In the utility-based learning framework we can frequently witness the conjugation of two important factors: i) an increased interest in some particular range(s)/class(es) of the target variable values and ii) a scarce representation of the examples belonging to that range(s)/class(es). This situation occurs in both classification and regression tasks and is usually known as the problem of imbalanced domains [BTR15].

Utility-based learning assumes non-uniform costs and/or benefits which are usually expressed through a cost/benefit matrix (in classification) or a cost/benefit surface (in regression). However, frequently this information is just not available, or is hard/expensive to obtain because it often requires the intervention of a domain expert. This means that for many domains, there is only an informal knowledge regarding which are the most costly mistakes and which are the most important classes/ranges of the target variable. In fact, considering the particular problem of imbalanced classes it is frequent to observe the assumption that "the minority class is the most important one". This is an important information regarding the preferences of the user. However, it is stated in a very informal way, an no cost/benefit matrix is available in this situation. The approaches proposed in package `UBL` are able to deal with these situations because they allow the use of both user specified preferences and automatic methods.

Several types of approaches exist for handling utility-based learning problems. These approaches were categorized into: pre-processing, change the learning algorithms, post-processing or hybrid [BTR15]. The **pre-processing** approaches act before the learning stage by manipulating the examples distribution to match the user preferences. The methods that **change the learning algorithms** try to incorporate the user preference bias into the selected learning algorithm. There are also strategies that are applied as a **post-processing** step by changing the predictions made by a standard learner using the original data set. Finally there are **hybrid** approaches that combine some of the previous strategies.

In package `UBL` we have focused on pre-processing strategies to address the problem of utility-based learning. These strategies change the original distribution of examples by removing or/and adding examples, i.e., by performing under-sampling or over-sampling. The under-sampling strategies may be random or focused. By focused under-sampling we mean that the discarded examples satisfy a given requirement, such as: are possibly noisy examples, are too distant from the decision border, are too close to the border, etc. Regarding the over-sampling methods there are two main options: over-sampling is accomplished by the introduction of replicas of examples or by the generation of new synthetic examples. For the strategies which include copies of existing examples, the cases may be selected randomly or in an informed fashion. Approaches that build synthetic cases differ among themselves in the generation process adopted. Several strategies combine under-sampling and over-sampling methods in different ranges/classes of the target variable.

This document is organized as follows. In Section 2 some general installation guidelines for `UBL` package are provided. Section 3 briefly describes the two synthetic data sets provided with `UBL` package . Section 4 presents two simple examples to show how `UBL` package can be used both in classification and re-

gression contexts and its impact on the models performance. Sections 5 and 6 describe with detail each method currently implemented in UBL for classification and regression tasks. Section 7 describes the distance functions available in package UBL which allow to asses the distance between examples in data sets containing nominal and/or numeric features. Finally, Section 8 concludes this document.

2 Package Installation Guidelines

The installation of any R package available on CRAN is performed as follows:

```
install.packages("UBL")
```

This is mandatory, if you want to use the approaches available in package UBL or even if you just want to try out the examples presented in the next sections. This installs the current stable version of UBL package which is version 0.0.7.

You may also install the development version of the package, that is available on the following GitHub Web page: https://github.com/paobranco/UBL. However, we strongly recommend the use of the CRAN stable version. If you still want to install the development version, you should do this with extreme care because this version is still being tested and therefore is more prone to bugs. To install the development version from GitHub you should do the following in R:

```
library(devtools)
install_github("paobranco/UBL",ref="development")
```

Further instructions may be found at the mentioned GitHub page. For reporting issues related with UBL package you can use: https://github.com/paobranco/UBL/issues.

After installation using any of the above procedures, the package can be used as any other R package by doing:

```
library(UBL)
```

3 Synthetic Data for Classification and Regression Tasks

Package UBL includes two artificially generated data sets: one for classification (ImbC) and another for regression (ImbR). Both data sets were generated to depict situations of imbalanced domains. Thus, in both data sets, we assume the usual setting where the under-represented values of the target variable (either nominal or numeric) are the most important to the user. Table 1 summarizes the main characteristics of these data sets.

ImbC data consists of multiclass classification data set with 1000 cases and two features, $X1$ (numeric) and $X2$ (nominal). The target variable, denoted as $Class$, has two minority classes (*rare1* and *rare2*) and a majority class (*normal*).

Dataset	Task	Total Cases	Relevant Cases		Features					Target
					name	type	min	mean	max	
ImbC	Classif.	1000	rare1	rare2	X1	num.	-13.58	-0.11	12.78	minority: rare1; rare2
			10	131	X2	nom.	"cat" 300	"fish" 300	"dog 400"	majority: normal
ImbR	Regress.	1000	50		X1	num.	0.37	9.94	19.06	min: 10.00
					X2	num.	0.20	10.08	19.47	mean: 10.98
										max: 23.17

Table 1: Description of artificial data sets of `UBL` package .

The percentage of cases of classes *rare1* and *rare2* is 1% and 13.1%, respectively, while the *normal* class has 85.9% of the cases. This data set mimics the usual setting where the most relevant classes for the user are under-represented. This data set also simulates the existence of both class overlap and small disjuncts. Both issues are known for increasing the difficulty of dealing with imbalanced domains [LFG+13]. Figure 1 shows this data set with some noise added to the nominal variable $X2$ to make the examples distribution more visible.

ImbC data set was generated as follows:

- $X1 \sim \mathbf{N}(0, 4)$
- $X2$ labels "cat", "fish" and "dog" where randomly distributed with the restriction of having a frequency of 30%, 30% and 40% respectively.
- To obtain the target variable $Class$, we have define the following sets:
  - $S_1 = \{(X1, X2) : X1 > 9 \wedge (X2 \in \{"cat", "dog"\})\}$
  - $S_2 = \{(X1, X2) : X1 > 7 \wedge X2 = "fish"\}$
  - $S_3 = \{(X1, X2) : -1 < X1 < 0.5\}$
  - $S_4 = \{(X1, X2) : X1 < -7 \wedge X2 = "fish"\}$
- The following conditions define the target variable distribution of the ImbC synthetic data set:
  - Assign class label "rare1" to: a random sample of 90% of set S_1 and a random sample of 40% of set S_2
  - Assign class label "rare2" to: a random sample of 80% of set S_3 and a random sample of 70% of set S_4
  - Assign class label "normal" to the remaing examples.

The regression data set ImbR has two numeric features ($X1$ and $X2$) and a continuous target variable Tgt. ImbR was generated in the following way: it includes 50 cases sampled from a circumference with white noise and the remaining 950 cases were sampled from a two dimensional normal distribution. Regarding the values of the continuous target variable (Tgt), they were obtained through a sample of two different Gamma distributions (one with higher values used for generating the target variable values of the examples in the circumference, and another with lower values used for the target variable values of the remaining examples). ImbR data simulates the usual setting in regression where the most relevant values are under-represented. In this case, we consider that the higher values of the target variable, whose predictor variables where sampled from a cirfumference, are the most important ones.

More formally, ImbR data was obtained as follows:

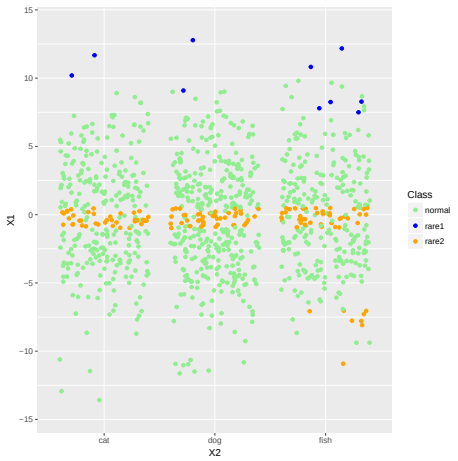

Figure 1: ImbC: artificial data set for classification.

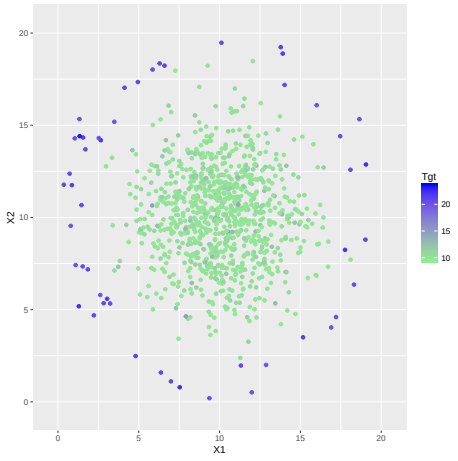

Figure 2: ImbR: artificial data set for regression.

- lower Tgt values:
  - $(X1, X2) \sim \mathbf{N}_2(\mathbf{10}_2, \mathbf{2.5}_2)$
  - $Tgt \sim \mathbf{\Gamma}(0.5, 1) + 10$
- higher Tgt values:
  - $(X1, X2) \sim (\rho * cos(\theta) + 10, \rho * sin(\theta) + 10)$, where $\rho \sim \mathbf{9}_2 + \mathbf{N}_2(\mathbf{0}_2, \mathbf{I}_2)$ and $\theta \sim \mathbf{U}_2(\mathbf{0}_2, 2\pi\mathbf{I}_2)$
  - $Tgt \sim \mathbf{\Gamma}(1, 1) + 20$

Figure 2 shows the examples distribution of ImbR data set.

4 Two Simple Illustrative Examples

In this section we will show two simple examples of how to use the `UBL` package . We will use the two data sets provided with the package (ImbC and ImbR) to illustrate a classification and a regression task.

Consider the ImbC synthetic data set. Let us suppose that the most important classes are the two minority classes, rare1 and rare2, and that we do not consider class normal relevant. Assuming this domain information, we begin by briefly observing the data set characteristics.

```
library(UBL)      # Loading our infra-structure
library(e1071)    # packge containing the svm we will use
data(ImbC)        # The synthetic data set we are going to use
summary(ImbC)     # Summary of the ImbC data

##        X1                X2         Class
##  Min.   :-13.5843   cat :300   normal:859
##  1st Qu.: -2.6930   dog :400   rare1 : 10
##  Median : -0.1592   fish:300   rare2 :131
##  Mean   : -0.1064
##  3rd Qu.:  2.4633
##  Max.   : 12.7836

table(ImbC$Class)

##
## normal  rare1  rare2
##    859     10    131
```

Now we will obtain a random sample of 70% of our data to train a svm. Then, we observe the results on the remaining 30% of data left for testing. We obtain the following:

```
set.seed(123)
samp <- sample(1:nrow(ImbC), nrow(ImbC)*0.7)
train <- ImbC[samp,]
test <- ImbC[-samp,]

model <- svm(Class~., train)
preds <- predict(model,test)
table(preds, test$Class) # confusion matrix
```

```
##
## preds    normal rare1 rare2
##   normal    256     3    41
##   rare1       0     0     0
##   rare2       0     0     0
```

Clearly, the model presents a poor performance on least represented classes. In effect, in this case, the model always predicts class normal.

Now, we can try to apply a pre-processing strategy for dealing with utility-based problems, and check again the models performance. In this case we selected the common strategy of balancing the data set classes and we used the SMOTE algorithm proposed by [CBHK02].

```
# change the train data by applying the smote strategy
# notice that we have to set the dist parameter to for instance "HEOM"
# because the default distance (Euclidean) is not possible to use with nominal features
newtrain <- SmoteClassif(Class~., train, C.perc="balance", dist="HEOM")


# generate a new model with the changed data
newmodel <- svm(Class~., newtrain)
preds <- predict(newmodel,test)
table(preds, test$Class)

##
## preds    normal rare1 rare2
##   normal    104     0     4
##   rare1      12     3     0
##   rare2     140     0    37
```

We can observe that the least represented classes, rare1 and rare2, now present an improved result. If the previous model was unable to correctly classify any examples of these classes, now most of those cases have a correct prediction. However, it is also important to highlight the increase in misclassification of class normal.

We can also observe the results obtained by applying a simple random over-sampling method. Again, we opted to balance the problem classes.

```
# apply random over-sampling strategy
newtrain2 <- RandOverClassif(Class~., train, C.perc="balance")

#generate a new model with the modified data set
newmodel2 <- svm(Class~., newtrain2)
preds <- predict(newmodel2, test)
table(preds, test$Class)

##
## preds    normal rare1 rare2
##   normal    129     0     4
##   rare1       9     3     0
##   rare2     118     0    37
```

Again, the pre-processing method applied allowed to improve the performance of the model on the least represented (and more important) class.

Let us now see how the pre-processing strategies can be applied on a regression task using the ImbR synthetic data set.

We start by loading the data and observe its main characteristics. Let us consider a random sample of 70% of ImbR data for training a model. The remaining 30% will be used as test set.

```
library(UBL)
data(ImbR)
summary(ImbR)

##        X1                X2              Tgt
##  Min.   : 0.3654   Min.   : 0.201   Min.   :10.00
##  1st Qu.: 8.2821   1st Qu.: 8.246   1st Qu.:10.06
##  Median : 9.9811   Median :10.129   Median :10.22
##  Mean   : 9.9418   Mean   :10.078   Mean   :10.98
##  3rd Qu.:11.7202   3rd Qu.:11.903   3rd Qu.:10.72
##  Max.   :19.0565   Max.   :19.474   Max.   :23.17

set.seed(123)
samp <- sample(1:nrow(ImbR), as.integer(0.7*nrow(ImbR)))
trainD <- ImbR[samp,]
testD <- ImbR[-samp,]
```

It is required that the user states which are the cases that he considers more/less relevant. The `UBL` package includes an automatic method for defining a relevance of the examples based on the data distribution. This method, proposed by [Rib11], allows to obtain a relevance function that maps each target variabel value into a $[0, 1]$ scale of relevance, where 1 represents maximum relevance and 0 represent minimum relevance. The key aspect of this automatic method is the assignment of an higher relevance to the scarcely represented cases which is the most common setting. We will use this automatic method in the ollowing example. A more detail explanation of this method is provided in Section 6.

Let us train a model using the original train data. We choose a random forest available through the `randomForest` package and obtained the predictions on the test set.

```
library(randomForest)
model <- randomForest(Tgt~., trainD)
preds <- predict(model, testD)
```

Let us now apply a pre-processing strategy in the original train data and observe the impact on the predictions.

```
# use the Introduction of Gaussian Noise with the default parameters
newTrain <- GaussNoiseRegress(Tgt~., trainD)
newModel <-randomForest(Tgt~., newTrain)
newPreds <- predict(newModel, testD)
```

The results obtained by the two random forest models are displayed in Figure 3. In this case, it is clear that the predictions obtained by the model trained with the changed data set are better in the high rare cases. For the higher range of values of Tgt the model trained with the original data displays mostly under-predictions showing a focus in the normal range of values.

We can also observe the use impact of using the simple random undersampling pre-processing strategy.

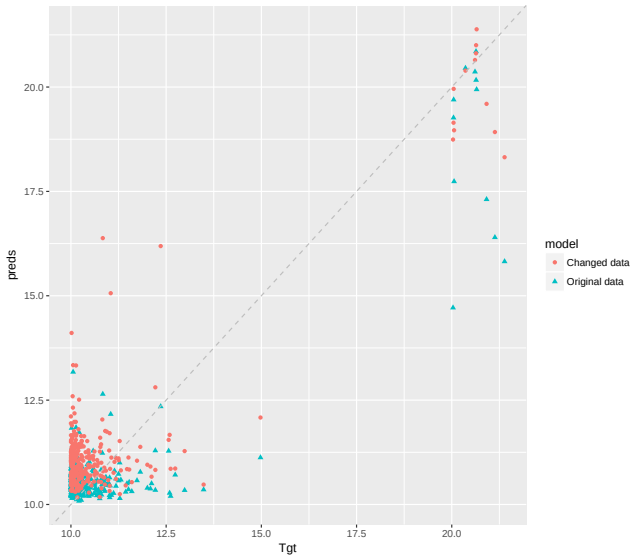

Figure 3: Predictions obtained with the original and the new data modified through the Gaussian Noise strategy.

```
# random under-sampling strategy setting the under-sampling percentage to 0.3
trainRU <-RandUnderRegress(Tgt~., trainD, C.perc=list(0.3))
ModelRU <-randomForest(Tgt~., trainRU)
PredsRU <- predict(ModelRU, testD)
```

Figure 4 shows the predictions obtained with the original training set and the training data modified through random under-sampling strategy.

A more complex pre-processing can also be tried. In this case, we apply smoteR algorithm with low percentage of over-sampling. Then, we add more synthetic examples using the Gaussian Noise strategy using only the cases with a relevance value above the 0.8 threshold. Figure 5 shows the predictions obtained with these changes.

```
train1 <- SmoteRegress(Tgt~., trainD, C.perc=list(0.9, 2))
train2 <- GaussNoiseRegress(Tgt~., train1, thr.rel=0.8, C.perc=list(0.8, 2), pert=0.01)
ModelC <-randomForest(Tgt~., train2)
PredsC <- predict(ModelC, testD)
```

Figure 6 shows each test set example marked by a point with size and color varying according to the error magnitude for all the models previously obtained. This means that larger blue points represent larger errors and small green points represent a lower magnitude of the error in the example.

5 Methods for Addressing Utility-based Classification Tasks

In this section we describe the methods implemented in package `UBL` . We provide detailed examples of each function, and discuss how the several parameters

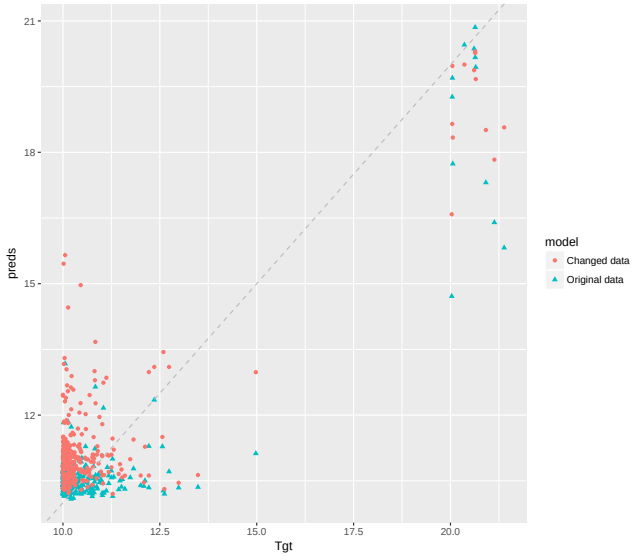

Figure 4: Predictions obtained with the original and the new data modified through the random under-sampling strategy.

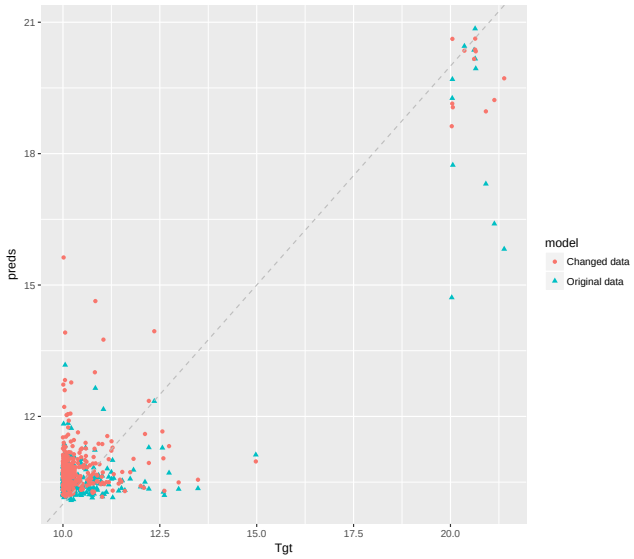

Figure 5: Predictions obtained with the original and the new data modified through the combination of strategies.

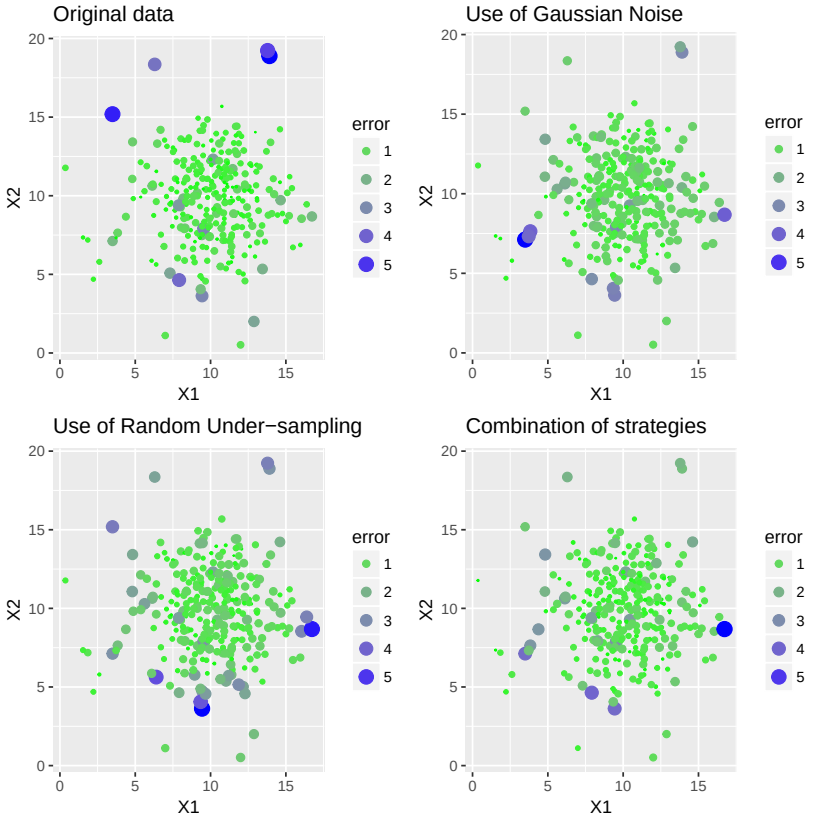

Figure 6: Predictions obtained with the original data and the two training sets modified through random under-sampling and Gaussian Noise strategies.

can be used and their impact. The methods explained in this section are the following:

- 5.1: Random Under-sampling
- 5.2: Random Over-sampling
- 5.3: Importance Sampling
- 5.4: Tomek Links
- 5.5: Condensed Nearest Neighbors
- 5.6: One-sided Selection
- 5.7: Edited Nearest Neighbors
- 5.8: Neighborhood Cleaning Rule
- 5.9: Gaussian Noise Introduction
- 5.10: Smote Algorithm
- 5.11: Adasyn Algorithm

5.1 Random Under-sampling

The random under-sampling strategy is among the simplest strategies for dealing with the class imbalanced problem. To force the learners to focus on the most important and least represented class(es) this technique randomly removes examples from the most represented and less important classes. This process allows to obtain a more balanced data set, although some important data may have been discarded with this technique. Another side effect of this strategy is a big reduction on the number of examples in the data set which facilitates the learners task although some important data may be ignored.

This strategy is implemented in `UBL` taking into consideration the possible existence of several minority classes. The user may define through `C.perc` parameter which are the normal and less important classes and the under-sampling percentages to apply in each one of them. Another possibility is to select "balance" or "extreme" for the parameter `C.perc`. These two options automatically estimate the under-sampling percentages to apply to the classes. The "balance" option obtains a balanced number of examples in all the existing classes, and the "extreme" option inverts the existing frequencies, transforming the most frequent classes into the less frequent and vice-versa. The following examples show how these options can be used and their impact.

```
library(UBL)  # Loading our infra-structure
library(e1071) # package containing the sum we will use
data(ImbC)                      # Our synthetic multiclass data set

table(ImbC$Class)

##
## normal  rare1  rare2
##    859     10    131
```

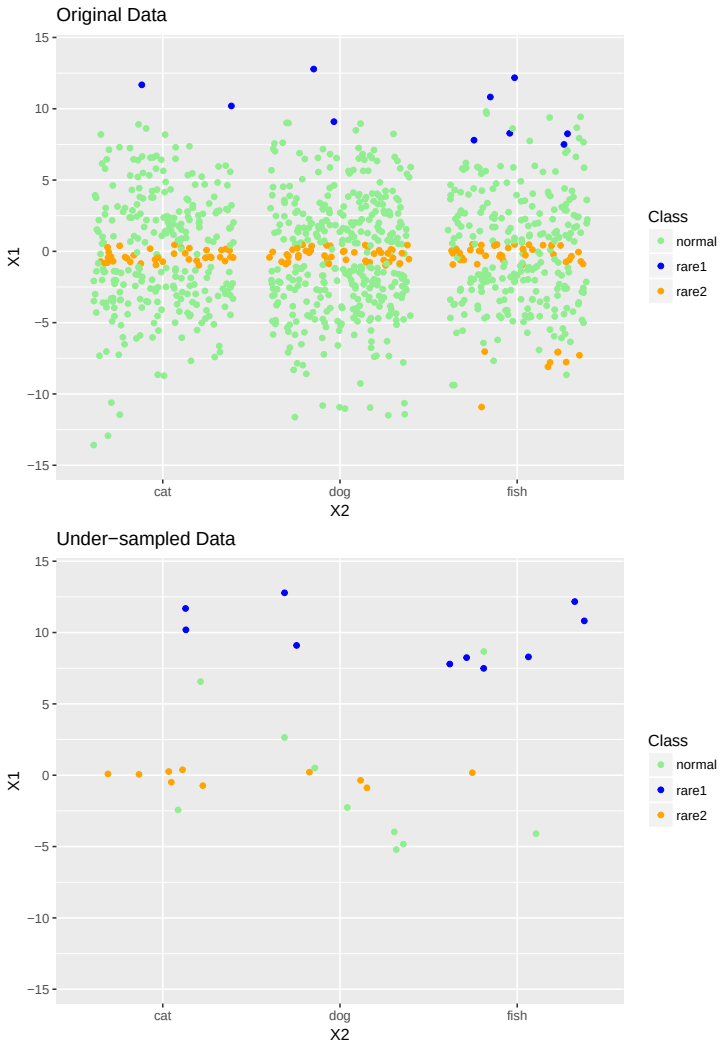

Figure 7: The impact of random under-sampling strategy.

```
## now, using random under-sampling to create a
## "balanced problem" automatically

newData <- RandUnderClassif(Class ~ ., ImbC)
table(newData$Class)

##
## normal  rare1  rare2
##     10     10     10
```

We highlight that, because this method only allows the removal of cases, in order to balance the examples distribution (the function default that was used), it has a strong impact in the total number of examples in the changed data set. This happens because one of the minority classes has only 10 examples. Figure 7 shows the impact of this strategy in the examples distribution.

Another example with ImbC data set:

```
RUmy <- RandUnderClassif(Class~., ImbC, list(normal=0.1, rare2=0.9))
RUB <- RandUnderClassif(Class~., ImbC, "balance")
RUE <- RandUnderClassif(Class~., ImbC, "extreme")
```

	normal	rare1	rare2
Original	859	10	131
RUmy	85	10	117
RUB	10	10	10
RUE	0	10	0

Table 2: Number of examples in each class for different parameters of random under-sampling strategy.

The impact of the strategies on the number of examples in each class of the data set are in Figure8.

5.2 Random Over-sampling

The random over-sampling strategy introduces replicas of already existing examples in the data set. The replicas to include are randomly selected among the least populated and more important classes. This allows to obtain a better balanced data set without discarding any examples. However, this method has a strong impact on the number of examples of the new data set which can represent a difficulty to the used learner.

This strategy is implemented in package `UBL` taking into consideration the possible existence of several minority classes. The user may define through `C.perc` parameter which are the most important classes and their respective over-sampling percentages. The parameter `C.perc` may also be set to "balance" or "extreme". These two options automatically estimate the classes and over-sampling percentages to apply. Similarly to the previous strategy the "balance" option allows to obtain a balanced number of examples in all the existing classes, and the "extreme" option inverts the existing frequencies, transforming the most frequent classes into the less frequent and vice-versa. The following examples show how these options can be used and their impact:

```
## now using random over-sampling to create a
## data with more 500% of examples in the
## rare1 class
RO.U1<- RandOverClassif(Class ~ ., ImbC,
                        C.perc=list(rare1=5))
RO.U2<- RandOverClassif(Class ~ ., ImbC,
                        C.perc=list(rare1=4, rare2=2.5))
RO.B <- RandOverClassif(Class ~ ., ImbC, C.perc="balance")
RO.E <- RandOverClassif(Class ~ ., ImbC, C.perc="extreme")
```

Figure 9 shows the impact of this strategy in the examples distribution. We have introduced a small perturbation on the examples position to be more clear the replicas that were introduced.

Figure 10 shows the impact of this strategy on the number of examples in the data set.

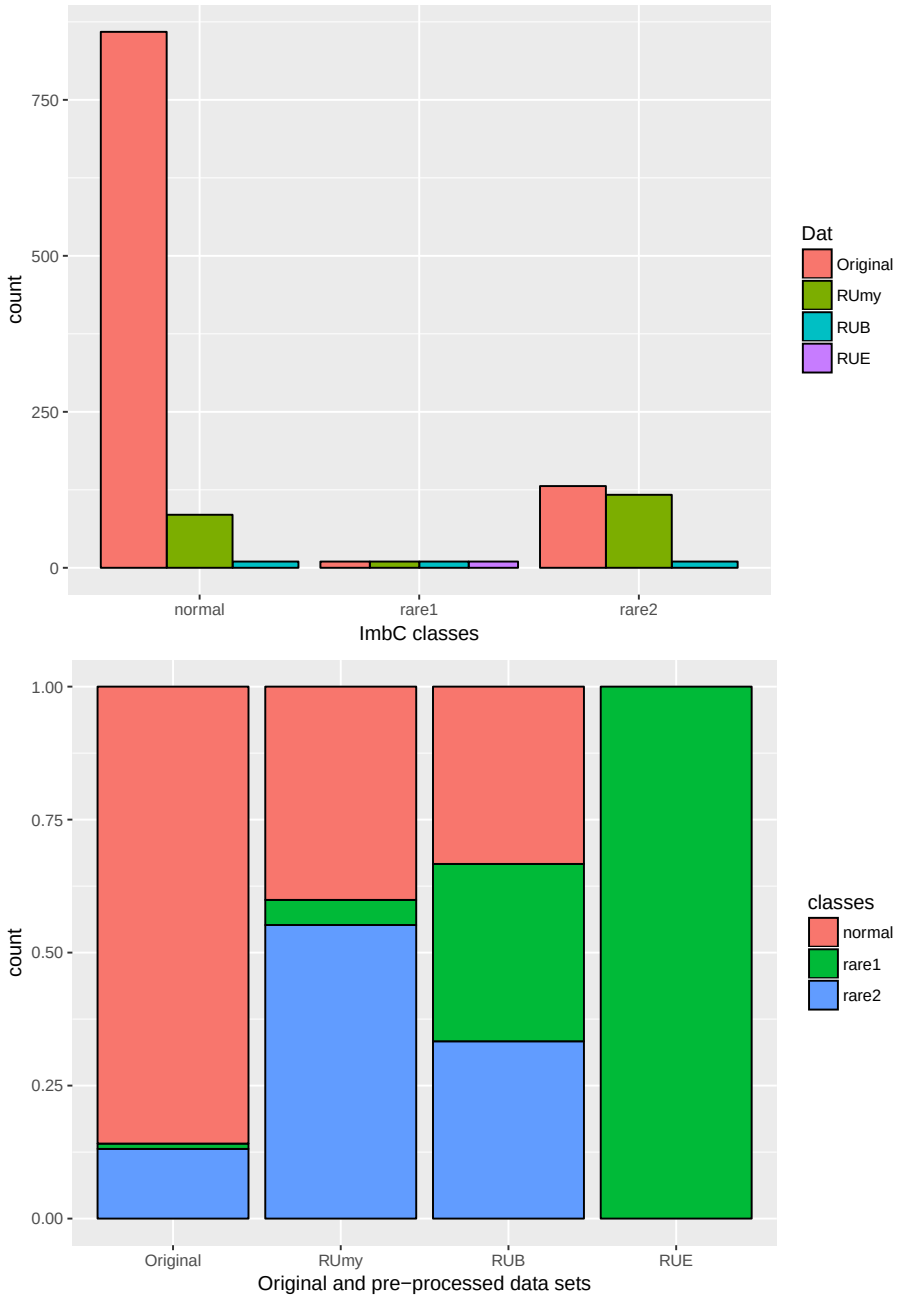

Figure 8: Random Under-sampling strategy for different parameters values.

	normal	rare1	rare2
Original	859	10	131
RO.U1	859	50	131
RO.U2	859	40	327
RO.B	859	859	859
RO.E	859	73788	5633

Table 3: Number of examples in each class for different Random over-sampling parameters.

5.3 Importance Sampling

The main idea of Importance Sampling strategy is to perform random over- or under-sampling in each class according to the importance assigned by the user. This means that for each class the user can specify its relevance. Then, this relevance is used to change each class frequency by selecting randomly examples from each class. Alternatively, the user may consider that each class is equally important or may chose to invert the classes frequencies.

This strategy is available in `UBL` package through the function `ImpSampClassif`. The user may specify using parameter `C.perc` the classes where over-/under-sampling must be applied, by indicating the corresponding percentages. If all classes are equally important and a perfectly balanced data set should be obtained, the `C.perc` parameter must be set to "balance". On the other hand, if the classes frequencies should be inverted, then this parameter should be "extreme". The following example illustrate the use of this function.

```
# use the synthetic imbalanced data set ImbC provided with UBL package
table(ImbC$Class)

##
## normal  rare1  rare2
##    859     10    131

nds <- ImpSampClassif(Class~.,ImbC, C.perc=list(normal=0.4, rare1=6))
# notice that when a certain class is not specified it remains unaltered
table(nds$Class)

##
## normal  rare1  rare2
##    343     60    131

# to obtain a balanced data set
IS.bal <- ImpSampClassif(Class~., ImbC) # or use C.perc="balance"
table(IS.bal$Class)

##
## normal  rare1  rare2
##    333    333    333

# to obtain a data set with inverted frequencies
IS.ext <- ImpSampClassif(Class~., ImbC, C.perc="extreme")
table(IS.ext$Class)

##
## normal  rare1  rare2
##     11    919     70
```

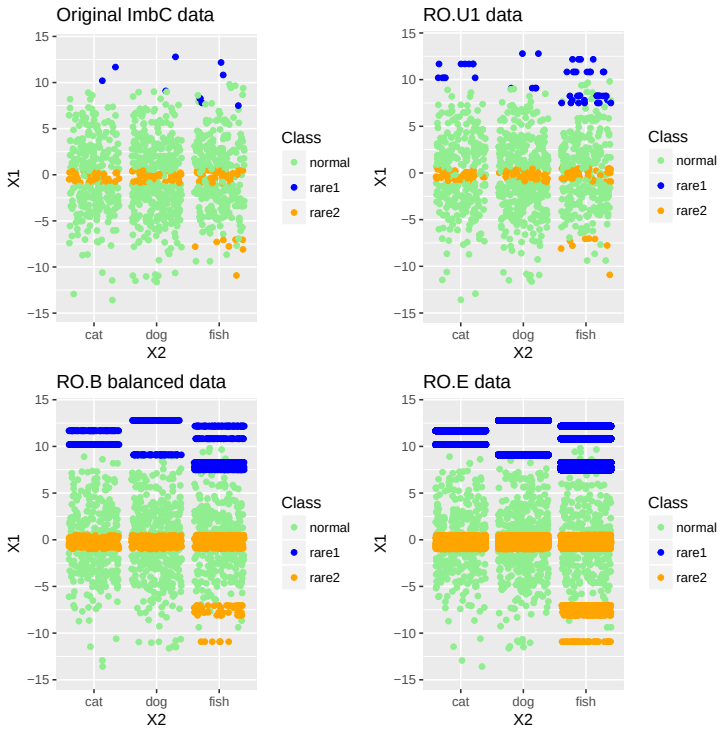

Figure 9: The impact of random over-sampling Strategy.

Figure 11 shows the impact on the imbalanced ImbC data set of the changes made in the domain with Importance Sampling.

Figure 12 shows the impact of this strategy on the number of examples in the data set.

We must highlight that random under- and over-sampling also allow to balance and invert the classes frequencies. Importance Sampling strategy, although also allowing this type of impact, acts differently because it combines both under- and over-sampling strategies. This means that a balanced data set can be obtained through random under-sampling, random over-sampling or importance sampling strategy. However, the resulting data sets will be different. If we use random under-sampling the final size of the data set is reduced, while if we use the random over-sampling approach the changed data set is significantly larger than the original one. If we select the importance sampling, the combination of the strategies allows to roughly maintain the data set size.

5.4 Tomek Links

Tomek Links [Tom76] can be defined as follows: two examples form a Tomek Link if and only if they belong to different classes and are each other nearest neighbors. This is a property existing between a pair of examples (S_i, S_j) having different class labels and for which

$\nexists S_k : dist(S_i, S_k) < dist(S_i, S_j) \vee dist(S_j, S_k) < dist(S_i, S_j)$

Having determined the examples which form Tomek Links, these connections

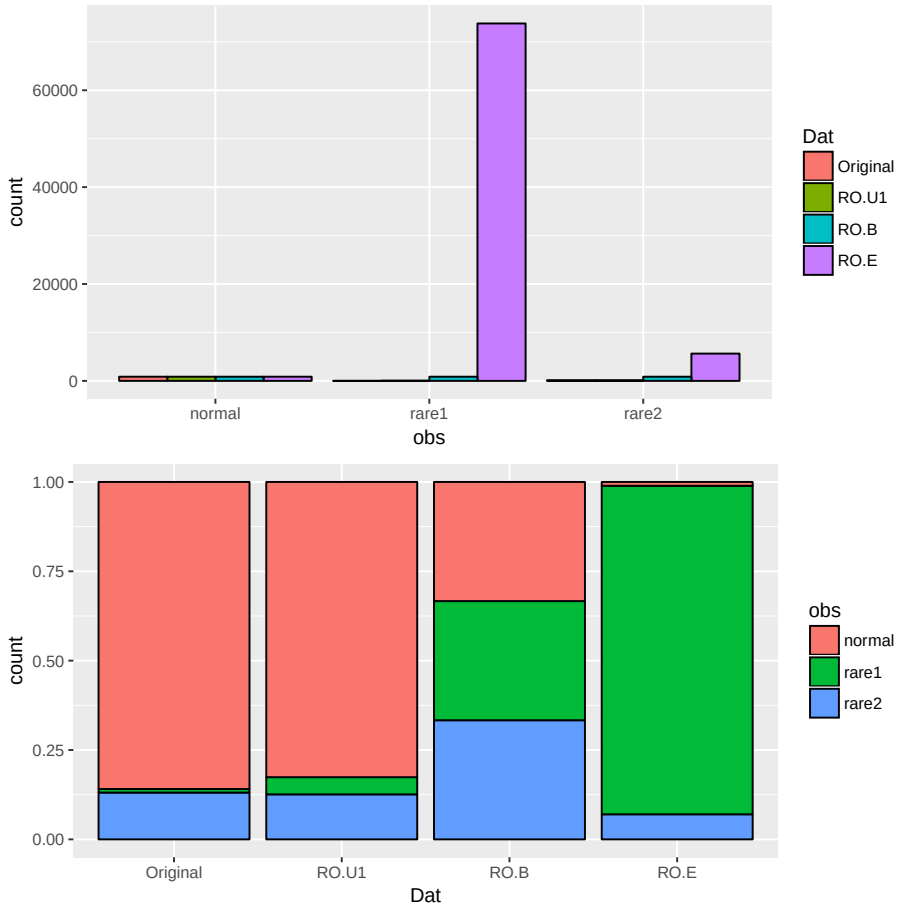

Figure 10: Impact of Random over-sampling strategy for different parameters values.

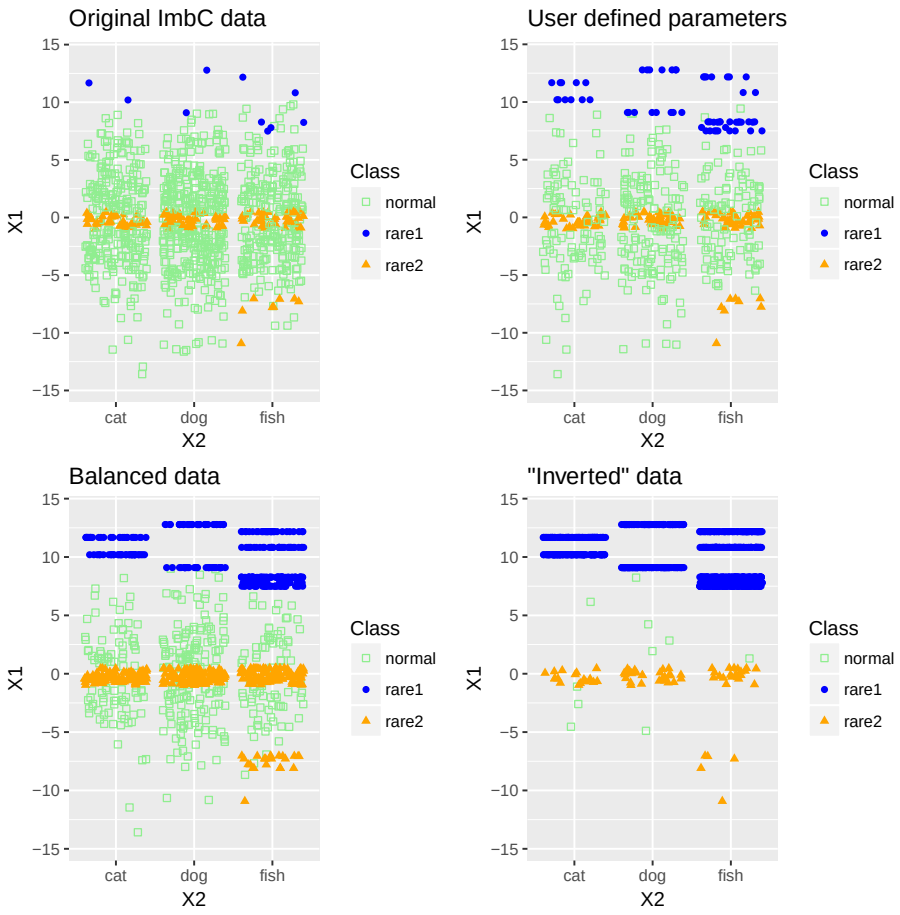

Figure 11: Impact of Importance Sampling strategy in ImbC data set.

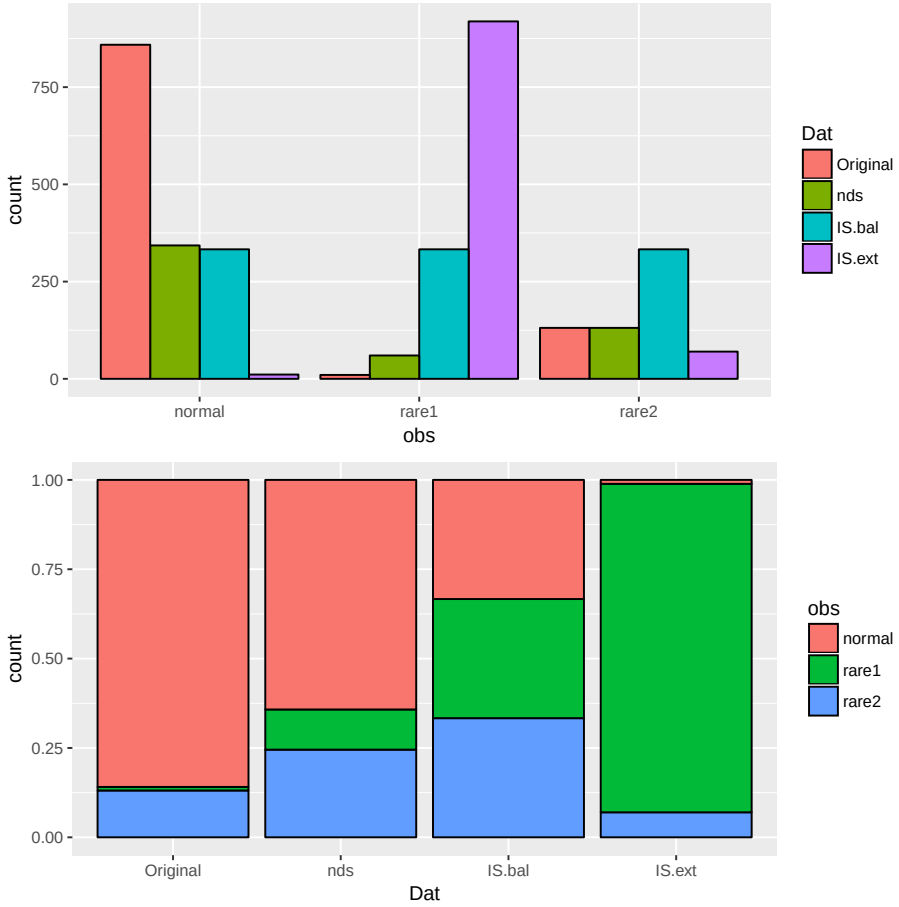

Figure 12: Impact of Importance Sampling strategy.

may be explained because either the examples are both borderline examples or one of the examples may be considered as noise. Therefore, there are two possibilities of using Tomek links to accomplish under-sampling:

- remove the two examples forming a Tomek link, or
- only remove the example from the most populated class which forms a Tomek link.

These two options correspond to using Tomek link as cleaning technique (by removing both borderline examples) or as an under-sampling method for balancing the classes (by removing the majority class example).

In package `UBL` we have adapted this technique for being able to deal with multiclass imbalanced problems. For working with more than two classes some issues were considered:

- allow the user to select which classes should be under-sampled (if not defined, the default is to under-sample all the existing classes);
- if the user selects a given number of classes what to do to break the link, i.e., how to decide which example(s) to remove (if any).

So, in `UBL` the user may chose from which classes he is interested in removing examples through the `Cl` parameter. Moreover, the user can also decide if both examples are removed or if just one is discarded using the `rem` parameter. If this can be easily understood in two class problems, the impact of these parameters may not be so clear for multiclass imbalanced tasks. In fact,the options set for `Cl` and `rem` parameters may "disagree". In those cases, the preference is given to the `Cl` options once the user choose that specific set of classes to under-sample and not the other ones (even if the defined classes are not the larger ones). This means that, when making a decision on how many and which examples will be removed the first criteria used will be the `Cl` definition.

For a better clarification of the behavior stated we now provide some possible scenarios for multiclass problems and the corresponding expected behavior:

- `Cl` is set to one class which is neither the most nor the least frequent, and `rem` is set to "maj". The expected behavior is the following: - if a Tomek link exists connecting the largest class and another class(not included in `Cl`): no example is removed; - if a Tomek link exists connecting the larger class and the class defined in `Cl`: the example from the `Cl` class is removed (because the user expressly indicates that only examples from class `Cl` should be removed);
- `Cl` includes two classes and `rem` is set to "both". This function will do the following: - if a Tomek link exists between an example with class in `Cl` and another example with class not in `Cl`, then, only the example with class in `Cl` is removed; - if the Tomek link exists between two examples with classes in `Cl`, then, both are removed.
- `Cl` includes two classes and `rem` is set to "maj". The behavior of this function is the following: -if a Tomek link exists connecting two classes included in `Cl`, then only the example belonging to the more populated class is removed; -if a Tomek link exists connecting an example from a

class included in `Cl` and another example whose class is not in `Cl` and is the largest class, then, no example is removed.

We must also highlight that this strategy strongly depends on the distance metric considered for the nearest neighbors computation. We provide in package `UBL` several different distance measures which are able to deal with numeric and/or nominal features, such as Manhattan distance, Euclidean Distance, HEOM or HVDM. For more details on the available distance functions check Section 7. The user may set the desired distance metric through the `dist` parameter.

The implementation provided in this package returns a list containing: the new data set modified through the Tomek links strategy and the indexes of the examples removed. Under certain situations, this strategy is not able to remove any example of the data set. In this case, a warning is issued to advert the user that no example was removed.

The following examples with ImbC data set show how Tomek links can be applied.

```
# using the default parameters except for the distance function
# ImbC has nominal and numeric features and this requires the use of
# specific distance functions
  ds <- TomekClassif(Class~., ImbC, dist="HEOM")
# using HEOM distance metric, and selecting only one class to under-sample
  dsHEOM <- TomekClassif(Class~., ImbC, dist="HEOM",
                         Cl="normal")

# check the new dsHEOM data set
  summary(dsHEOM[[1]])

##        X1              X2         Class
##  Min.   :-13.5843   cat :294   normal:843
##  1st Qu.: -2.7206   dog :394   rare1 : 10
##  Median : -0.1603   fish:296   rare2 :131
##  Mean   : -0.1141
##  3rd Qu.:  2.5029
##  Max.   : 12.7836

# check the indexes of the examples removed:
  dsHEOM[[2]]

##  [1] 359 830 103 172 182 748 314 322 341 981 943 544 703 681 920 996

# using HVDM distance, enable the removal of examples from all classes, and
# select to break the link by only removing the example from the majority class
  dsHVDMM <- TomekClassif(Class~., ImbC, dist="HVDM", Cl="all", rem="maj")
# similar but breaking the Tomek link by removing both examples that for it
  dsHVDMB <- TomekClassif(Class~., ImbC, dist="HVDM", Cl="all", rem="both")
```

Figure 13 shows the impact in ImbC examples of the last experiences.

Let us now consider the iris data set, changed with the goal of having an imbalanced distribution of the classes.

```
data(iris)
dat <- iris[-c(61:85, 116:150),c(1,2,5)]
summary(dat)

##   Sepal.Length    Sepal.Width          Species
##  Min.   :4.300   Min.   :2.300   setosa    :50
```

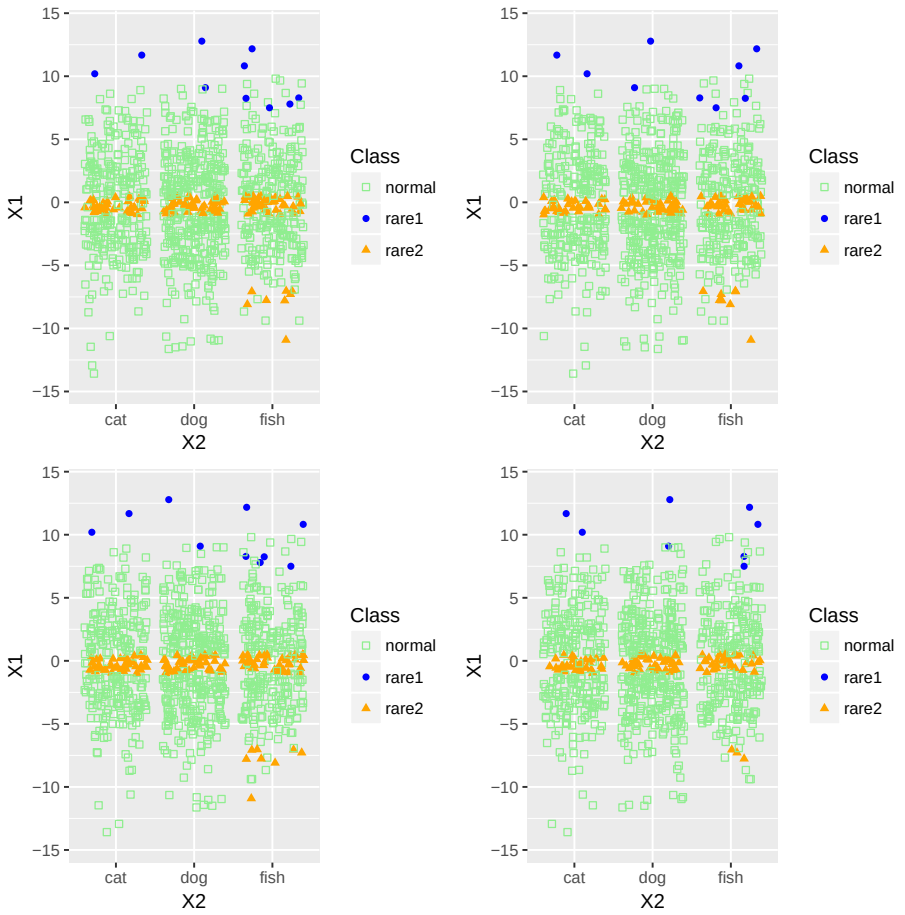

Figure 13: Impact of Tomek links strategy in ImbC synthetic data set. (top left: ImbC data; top right: ds data; bottom left:dsHVDMM data; and bottom right: dsHVDMB data)

	normal	rare1	rare2
Original	859	10	131
ds	843	9	116
dsHEOM	843	10	131
dsHVDMM	833	10	131
dsHVDMB	833	8	107

Table 4: Number of examples in each class for different Tomek Links parameters.

```
##  1st Qu.:5.000   1st Qu.:2.900   versicolor:25
##  Median :5.350   Median :3.150   virginica :15
##  Mean   :5.497   Mean   :3.170
##  3rd Qu.:5.800   3rd Qu.:3.475
##  Max.   :7.600   Max.   :4.400
```

Let us observe the impact of applying Tomek links strategy in this data.

```
# using the default in all parameters
  ir <- TomekClassif(Species~., dat)
# using chebyshev distance metric, and selecting only two classes to under-sample
  irCheb <- TomekClassif(Species~., dat, dist="Chebyshev",
                         Cl=c("virginica", "setosa"))
# using Manhattan distance, enable the removal of examples from all classes, and
# select to break the link by only removing the example from the majority class
  irManM <- TomekClassif(Species~., dat, dist="Manhattan", Cl="all", rem="maj")
  irManB <- TomekClassif(Species~., dat, dist="Manhattan", Cl="all", rem="both")
```

	setosa	versicolor	virginica
Original	50	25	15
ir	50	19	9
irCheb	50	25	10
irManM	50	19	15
irManB	50	19	9

Table 5: Number of examples in each class for different Tomek Links parameters.

Figure 14 shows the impact of the previously described Tomek links strategies in the iris subset considered.

5.5 Condensed Nearest Neighbors

The Condensed nearest neighbors rule (CNN) was presented by [Har68]. The goal of this strategy is to perform under-sampling by building a subset of examples which is consistent with the original data. A subset is consistent with another if the elements in the subset classify correctly all the original examples using a 1-NN.

To build a consistent subset we have adapted the proposal of [KM97] to multiclass problems. The user starts by defining which are the most relevant classes in the data set using the `Cl` parameter. If the user prefers, an automatic option that corresponds to setting `Cl` to "smaller", evaluates the distribution of the classes and determines which classes are candidates for being the smaller

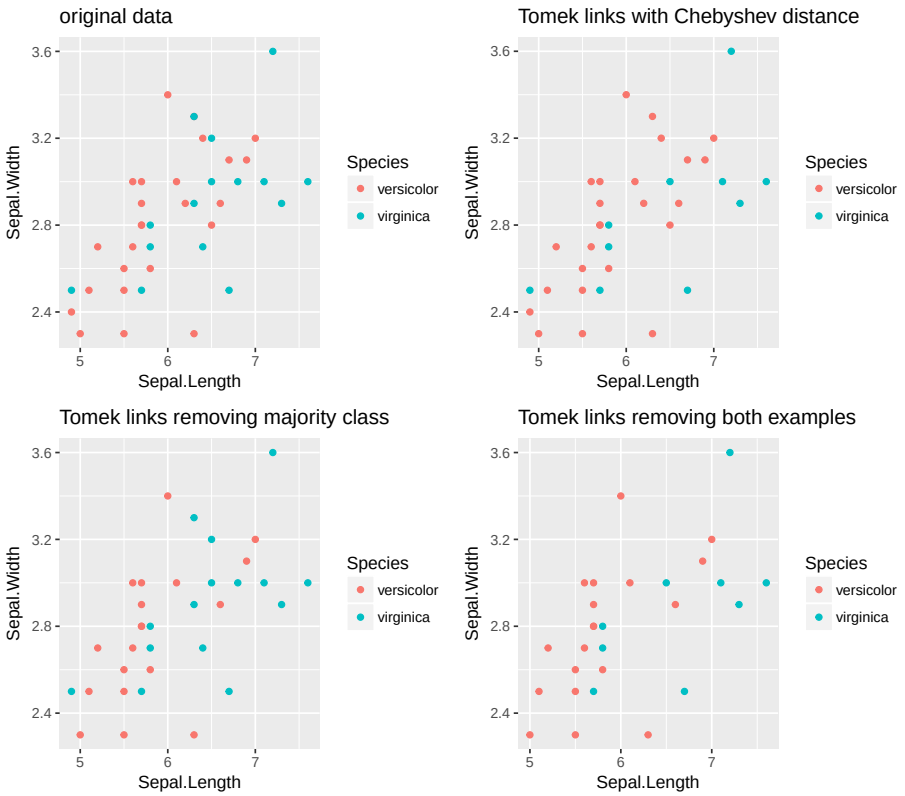

Figure 14: Impact of Tomek links strategy in classes virginica and versicolor of the subset of iris data (top left: original iris subset; top right: irCheb data; bottom left: irManM data and bottom right: irManB data).

and most important. By default, this parameter is set to "smaller" which means that the most relevant classes are automatically estimated from the data and correspond to those classes containing less than $\frac{\text{total number of examples}}{\text{number of classes}}$ examples. For instance, if a data set has 5 classes and a total number of examples of 100, the classes with less than 20 ($\frac{100}{5}$) examples will be considered the most important. The examples of the most relevant classes are then joined with one randomly selected example from each of the other classes. A 1-NN is computed with the distance metric provided by the user through the `dist` parameter. Then, all the examples from the original data set which were mislabeled in this procedure are also added to the reduced data set. This allows to obtain a smaller data set by removing examples from the larger and less important classes which are farther from the decision border.

This strategy is available through the `CNNClassif` function. This function returns a list containing: the modified data set, the classes that were considered important, and finally the unimportant classes.

We can now see some examples of this approach on the sythetic ImbC data and in the subset of iris data previously defined.

```
# select a distance that is appropriate for dealing with
# both nominal and numeric features
# the default considers the two minority classes as the most important ones
IHEOM <- CNNClassif(Class~., ImbC, dist="HEOM")

# considering only rare1 class is important
IHEOM1 <- CNNClassif(Class~., ImbC, dist="HEOM", Cl="rare1")

# considering only rare2 class as important
IHEOM2 <- CNNClassif(Class~., ImbC, dist="HEOM", Cl="rare2")

# use HVDM distance and the default of conisdering
# both minority classes as the most important
IHVDM <- CNNClassif(Class~., ImbC, dist="HVDM")

# now we select rare1 as the important class
IHVDM1 <- CNNClassif(Class~., ImbC, dist="HVDM", Cl="rare1")

# this selects the class rare2 as the most important one
IHVDM2 <- CNNClassif(Class~., ImbC, dist="HVDM", Cl="rare2")
```

	normal	rare1	rare2
Original	859	10	131
IHEOM	777	10	131
IHEOM1	640	10	87
IHEOM2	698	5	131
IHVDM	637	10	131
IHVDM1	535	10	1
IHVDM2	685	1	131

Table 6: Number of examples in each class of IMbC pre-processed data sets for different CNN parameters.

Table 6 shows the number of examples that remain in each class of the pre-processed data sets. In this case, it is evident that both the distance function

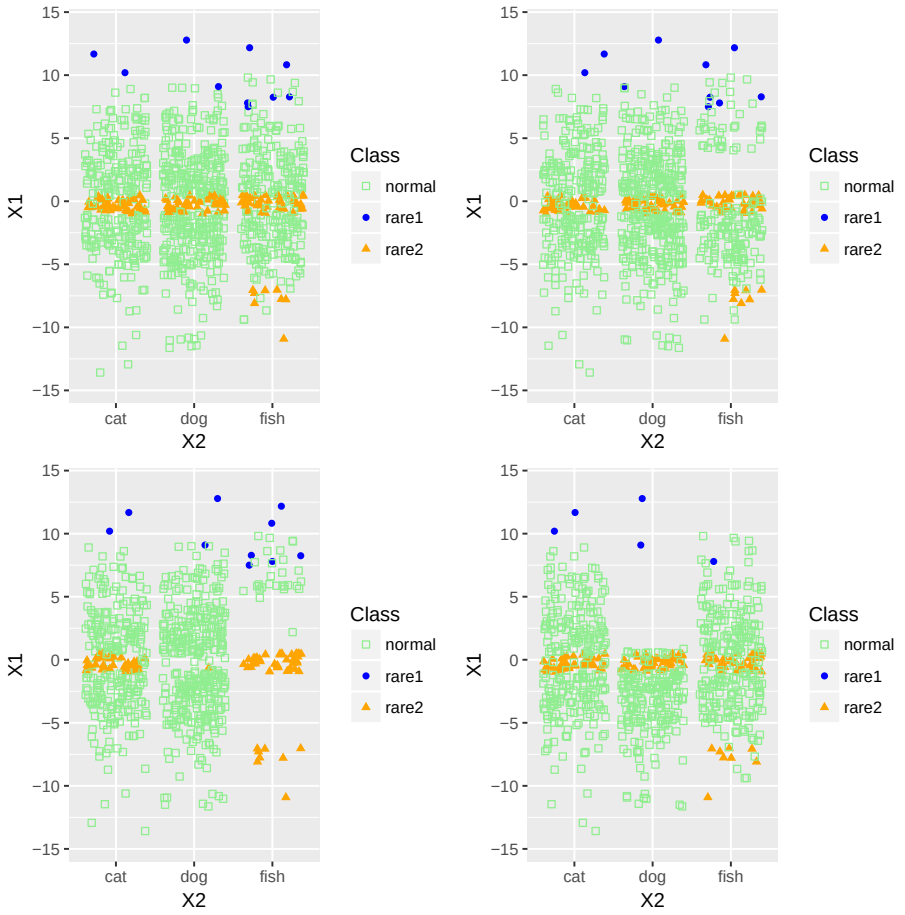

Figure 15: Impact on ImbC data of CNN method with HEOM distance for different values of parameter Cl.

selected and the classes provided to be considered important have a significant impact on this strategy. Figures 15 and 16 show the impact of the previously described strategies on ImbC data using HEOM and HVDM distances.

Let us now consider the subset of iris data. In this case, the two features are numeric which allows the use of different distance functions.

```
# just to remember the considered subset of iris data set
  summary(dat)

##   Sepal.Length    Sepal.Width          Species
##  Min.   :4.300   Min.   :2.300   setosa    :50
##  1st Qu.:5.000   1st Qu.:2.900   versicolor:25
##  Median :5.350   Median :3.150   virginica :15
##  Mean   :5.497   Mean   :3.170
##  3rd Qu.:5.800   3rd Qu.:3.475
##  Max.   :7.600   Max.   :4.400

# use of the default distance: Euclidean
  myCNN <- CNNClassif(Species~., dat, Cl=c("setosa", "virginica"))
```

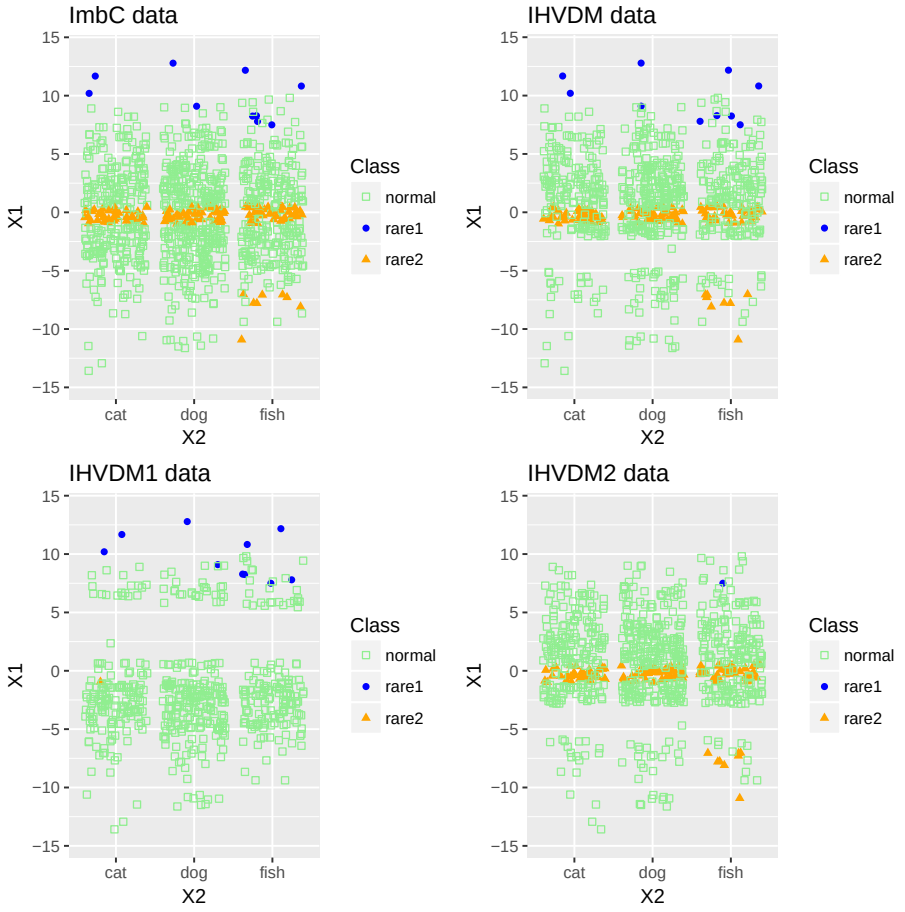

Figure 16: Impact on ImbC data of CNN method with HVDM distance for different values of parameter Cl.

```
  CNN1 <- CNNClassif(Species~., dat, Cl="smaller")
# try other distance functions
  CNN2 <- CNNClassif(Species~., dat, dist="Chebyshev", Cl="versicolor")
  CNN3 <- CNNClassif(Species~., dat, dist="HVDM", Cl="virginica")
  CNN4 <- CNNClassif(Species~., dat, dist="p-norm", p=3, Cl="setosa")

# check the new data set obtained in CNN1
summary(CNN1[[1]]$Species)

##     setosa versicolor  virginica
##         17         25         15

# check the classes which were considered important
CNN1[[2]]

## [1] "versicolor" "virginica"

# check the classes which were considered unimportant
CNN1[[3]]

## [1] "setosa"
```

	setosa	versicolor	virginica
Original	50	25	15
myCNN	50	24	15
CNN1	17	25	15
CNN2	36	25	15
CNN3	23	20	15

Table 7: Number of examples in each class for different CNN parameters.

It is clear from these examples that this method entails a significant reduction on the number of examples left in the modified data set. Moreover, since there is a random selection of points belonging to the less important class(es) the obtained data set may differ for different runs. Figure 17 provides a visual illustration of the impact of this method in the previously considered subset of iris data.

5.6 One-sided Selection

[KM97] proposed a new method for modifying a given data set by applying the Tomek links under-sampling strategy and afterwards the CNN technique. [BPM04] also tested the reverse order for applying the techniques: first apply CNN method and then Tomek links. The main motivation for this was to apply Tomek links to an already reduced data set because Tomek links technique is a more computationally demanding task.

In `UBL` we have gathered under the same function, `OSSClassif`, both techniques. To distinguish between the two methods, we included a parameter `start` which defaults to "CNN". The user may therefore select the order in which we want to apply the two techniques: CNN and Tomek links. In this implementation, when Tomek links are applied, they always imply the removal of both examples forming the Tomek link.

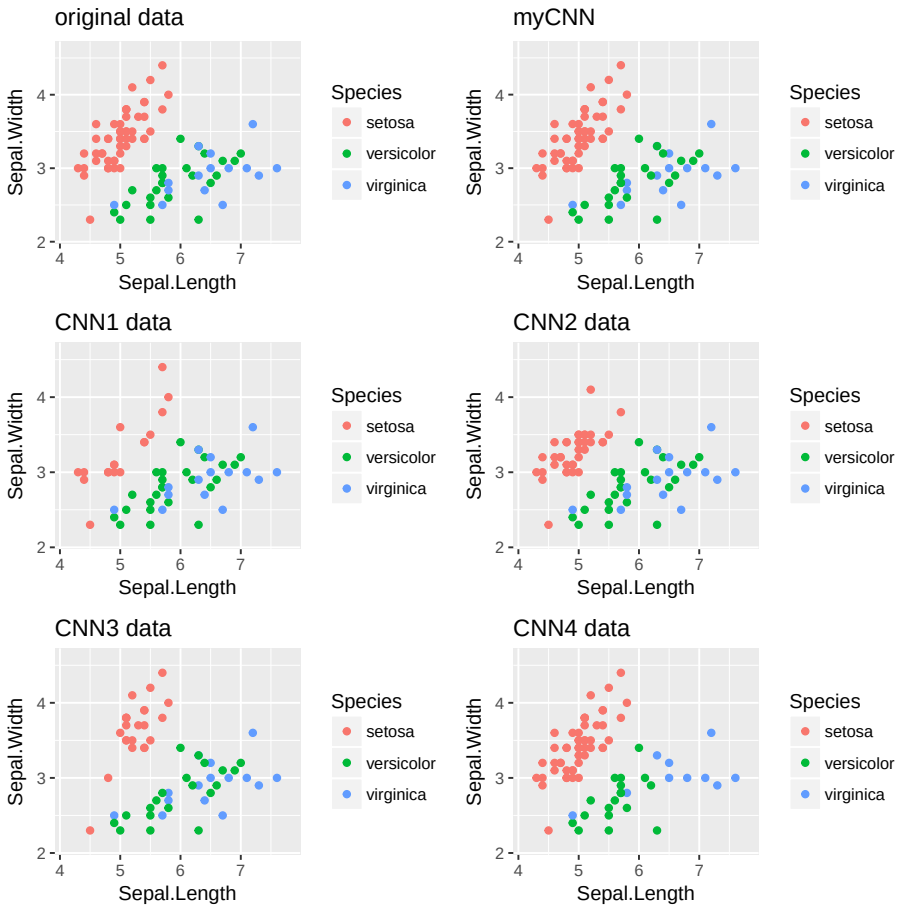

Figure 17: Impact of CNN method for different values of parameter Cl and different distance functions on the subset of iris data (top left: original iris subset; top right:myCNN data; middle left: CNN1 data; middle right: CNN2 data; bottom left: CNN3 data; and bottom right: CNN4 data).

We have adapted both methods for dealing with multiclass imbalanced problems. To do so, we have included the parameter `Cl` which allows the user to specify the most important classes. Similarly to the behavior of CNN strategy, the user may define for the `Cl` parameter the value "smaller". In this case, the most important classes are automatically determined using the same method presented in CNN strategy. When the relevant classes are chosen with this automatic method, the less frequent classes (which are considered the most relevant ones) are those which have a frequency below $\frac{number of examples}{number of classes}$. This means that all the classes with a frequency below the mean frequency of the data set classes is considered a minority class. The `OSSClassif` function also allows to specify which distance metric should be used in the neighbors computation. For more details on the available distance functions see Section 7. We must also mention that this strategy may potentially produce warnings due to the use of Tomek links strategy. As previously mentioned when Tomek links approach was presented, this method may not change the provided data set. In this case a warning is issued to advert the user. This warning may also occur when using OSS strategy if the Tomek links method produce it.

Let us observe how this method can be used with ImbC data.

```
# OSS method with HEOM distance
HEOM1 <- OSSClassif(Class~., ImbC, dist="HEOM")
HEOM2 <- OSSClassif(Class~., ImbC, dist="HEOM", start="Tomek", Cl="rare1")

# OSS method with HVDM distance
HVDM1 <- OSSClassif(Class~., ImbC, dist="HVDM", Cl="rare1")
HVDM2 <- OSSClassif(Class~., ImbC, dist="HVDM", start="Tomek", Cl="rare2")
```

Table 8 shows the impact of OSS strategy with different parameters on the number of examples in each class of the pre-processed data sets and Figure 18 shows the examples distribution on the pre-processed data sets.

	normal	rare1	rare2
Original	859	10	131
HEOM1	777	10	131
HEOM2	635	10	73
HVDM1	328	10	123
HVDM2	659	3	131

Table 8: Number of examples in each class for different OSS parameters.

The use of this method with data sets with all numeric features allows the use of several other distance functions. Let us briefly observe the impact of this method in the previously defined iris subset.

```
set.seed(1234)

# using all the defaults
ir1 <- OSSClassif(Species~., dat)

## Warning:  TomekClassif found no examples to remove!

# using distance functions only suitable for numeric features
ir2 <- OSSClassif(Species~., dat, dist="p-norm", p=3,
                  Cl="virginica")
```

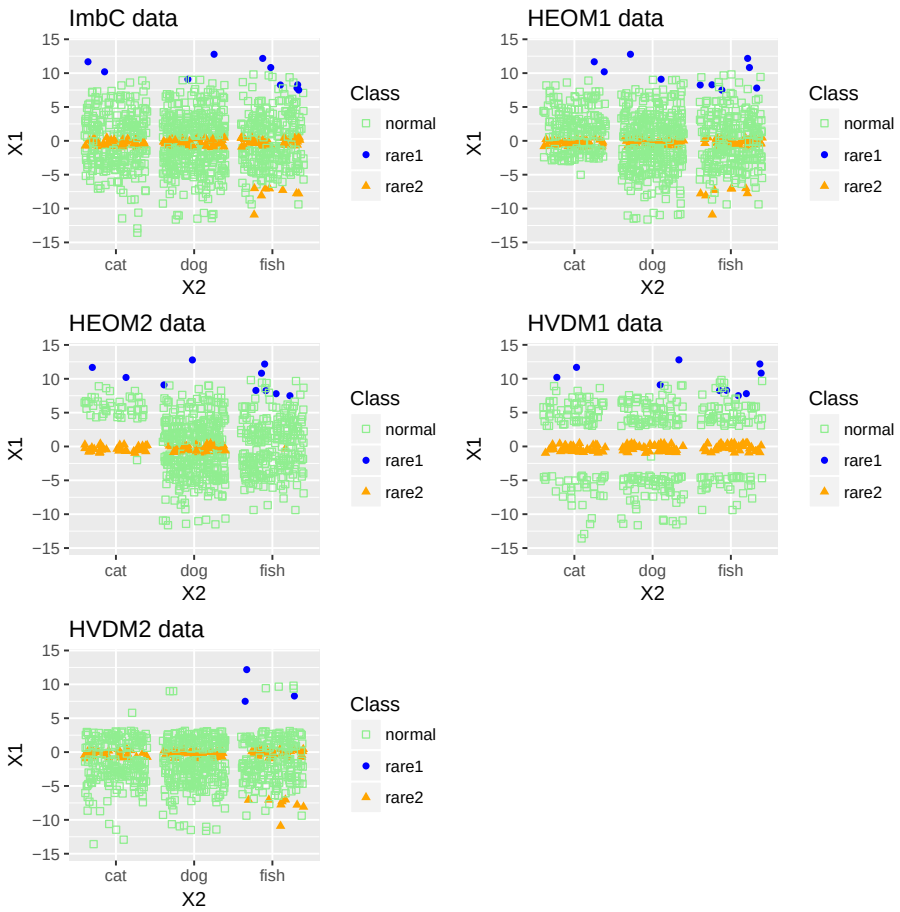

Figure 18: Impact on ImbC data of OSS method with different parameters.

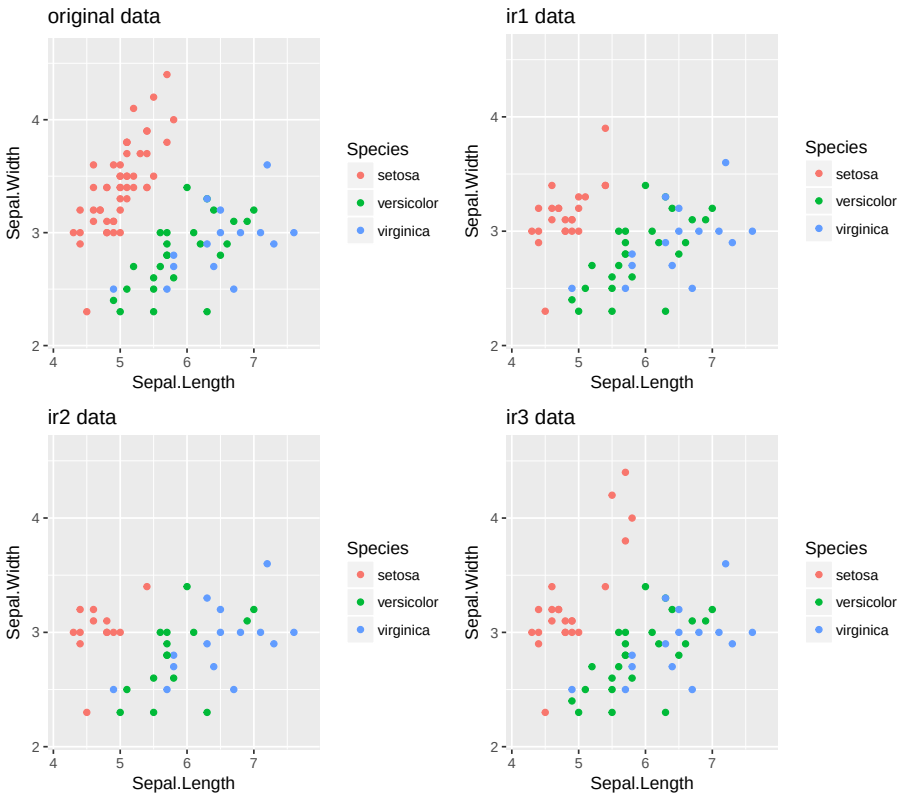

Figure 19: OSS technique with different parameters applied to the imbalanced iris subset.

```
ir3 <- OSSClassif(Species~., dat, dist="Chebyshev",
                  Cl=c("versicolor", "virginica"), start="Tomek")

## Warning:  TomekClassif found no examples to remove!

summary(ir1$Species)

##     setosa versicolor  virginica
##         23         25         15

summary(ir2$Species)

##     setosa versicolor  virginica
##         13         15         15

summary(ir3$Species)

##     setosa versicolor  virginica
##         22         25         15
```

The results obtained with the variants of OSS method on iris subset can be visualized in Figure 19.

5.7 Edited Nearest Neighbors

The Edited Nearest Neighbor (ENN) algorithm was proposed by [Wil72]. This method falls within the under-sampling approaches and has been used to address imbalanced classification problems. The original ENN algorithm uses a 3-NN classifier to remove the examples whose class is different from the class of at least two of its neighbors.

We have implemented this approach for being able to tackle multiclass problems, allowing the user to specify through the `Cl` parameter a subset of classes which should be under-sampled. Moreover, in our implementation, the user may also define the number of nearest neighbors that should be considered by the algorithm. This means that an example is removed if its class label is different from the class label of at least half of its k-nearest neighbors and if it belongs to the subset of classes candidates for removal. The ENN algorithm is available in `UBL` through the function `ENNClassif`. The number of neighbors to consider is set through the parameter `k` and the subset of classes that are candidates for being under-sampled are defined through the `Cl` parameter. The default of `Cl` is "all", meaning that all classes are candidates for having examples removed. The user can also specify which distance metric he wants to use in the nearest neighbors computation. The function `ENNClassif` returns a list containing the new under-sampled data set and the indexes of the examples removed.

It is possible that ENN finds no examples to remove, which means that, for the parameters selected, there are no examples satisfying the necessary conditions to be removed. In this case, a warning is issued with the goal of adverting the user that the strategy is not modifying the data set provided.

We can use this strategy in ImbC data as follows:

```
# use of default parameters except for the distance function
ENN1 <- ENNClassif(Class~., ImbC, dist="HVDM")

ENN2 <- ENNClassif(Class~., ImbC, dist="HVDM", Cl="rare1")

ENN3 <- ENNClassif(Class~., ImbC, dist="HVDM", Cl="rare2")

# now using the HEOM distance
ENN4 <- ENNClassif(Class~., ImbC, dist="HEOM")

# vary the number of neighbors considered by this method
ENN5 <- ENNClassif(Class~., ImbC, k=5, dist="HEOM")

ENN6 <- ENNClassif(Class~., ImbC, k=1, dist="HEOM")
```

Table 9 shows the impact of ENN strategy on ImbC synthetic data set with different parameters and Figure 20 illustrates the examples distribution in the original and changed data sets.

We can also use this strategy on the imbalanced iris subset previously defined. In this case, given that the data contains only numeric features it is possible to use different distance functions.

```
set.seed(123)
Man5 <- ENNClassif(Species~., dat, k=5, dist="Manhattan", Cl="all")
Default <- ENNClassif(Species~., dat)
ChebSub7 <- ENNClassif(Species~., dat, k=7, dist="Chebyshev",
                       Cl=c("virginica", "setosa"))
ChebAll7 <- ENNClassif(Species~., dat, k=7, dist="Chebyshev")
```

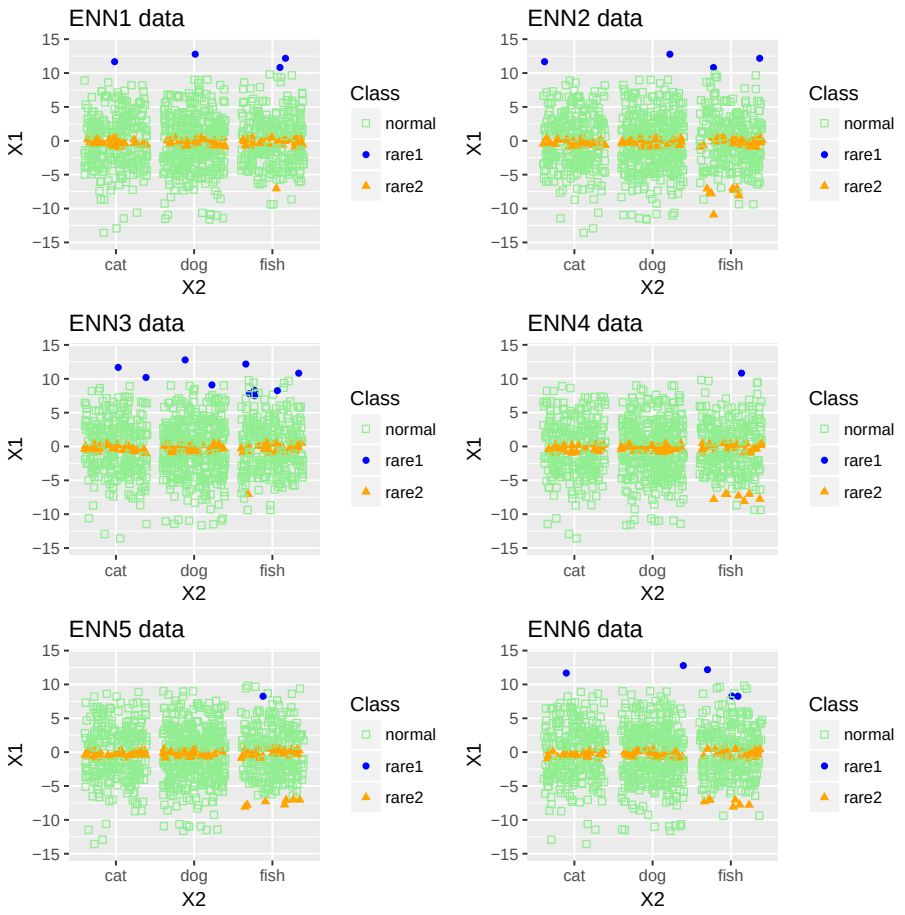

Figure 20: Impact on ImbC data of ENN method with different parameters.

	normal	rare1	rare2
Original	859	10	131
ENN1	826	4	112
ENN2	859	4	131
ENN3	859	10	112
ENN4	829	1	114
ENN5	823	1	119
ENN6	834	5	106

Table 9: Number of examples in each class for different parameters of ENN strategyapplied in ImbC data.

```
HVDM3 <- ENNClassif(Species~., dat, k=3, dist="HVDM")
```

In Table 10 we can observe the examples distributions for some parameters settings in ENN strategy and in Figure 21 we can visualize that distribution.

	setosa	versicolor	virginica
Original	50	25	15
Man5	49	14	3
Default	49	16	2
ChebSub7	49	25	2
ChebAll7	49	19	2
HVDM3	49	18	3

Table 10: Number of examples in each class for different parameters of ENN strategy.

This strategy has an unexpected behavior at first sight. In fact, the ENN method has further reduced the already minority classes. This can be explained by the goal of the ENN method which, being a cleaning technique, discards examples which may introduce errors no mater to which class they belong. As we know, in the iris data set the classes versicolor and virginica are the ones which are more difficult to classify. Therefore, the applied ENN strategy will try to remove examples exactly from those two classes.

Sometimes, ENN method is not capable of removing any example. When this happens, the original data set remains unchanged and an warning is issued. This warning is made only to advert the user that no examples were removed. On the other hand, with some data sets, this algorithm may completely remove one or more classes. This behavior may jeopardize the use of standard learning algorithms because they are provided with data set with only one class in the target variable. To overcome this issue, when a class is completely removed with the ENN strategy we randomly chose one example of that class to add to the under-sampled data set.

5.8 Neighborhood Cleaning Rule

The Neighborhood Cleaning Rule (NCL) algorithm was proposed in [Lau01]. This approach starts by splitting the data set D in two: a subset C with the

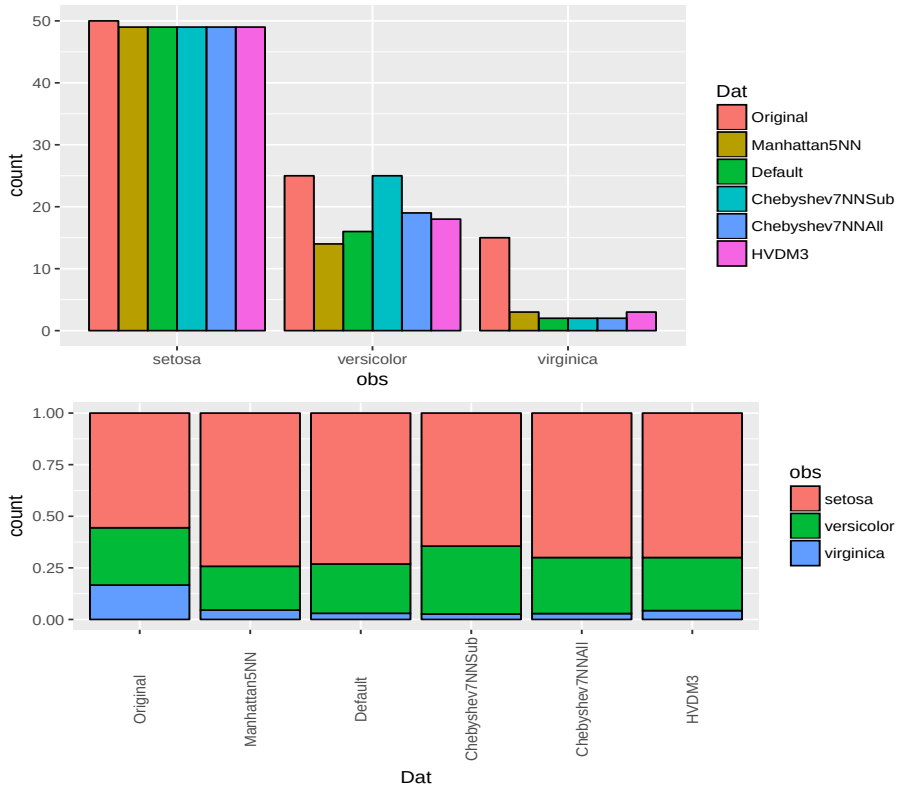

Figure 21: Impact in the subset of iris data of several parameters for ENN strategy.

examples belonging to the most important (an usually less frequent) class(es) and another subset O containing the examples from the less important class(es). A new set A_1 of examples is formed with the noisy examples belonging to the subset O which are identified using the ENN method. Then, another set A_2 of examples is built as follows. For each class C_i in O, the k nearest neighbors of each example in C_i are scanned. The example is included in A_2 if all the scanned k nearest neighbors have a class label not contained in C and if the example belongs to a class which has a cardinal of at least $\frac{1}{2}$ of the cardinal of smaller class in C. This last constraint forces the algorithm to keep the examples of classes with to few examples. Finally, the examples in A_1 and A_2 are removed from the original data set.

Since this strategy internally uses the ENN approach we highlight that it is possible that warnings are issued. As mentioned before, the user is always adverted if ENN does not alter the data set. This can also happen with NCL if internally the ENN does not remove any example.

The NCL approach is available in `UBL` through the `NCLClassif` function. In addition to providing a formula describing the prediction problem (`form`) and a data set (`dat`) the user may set the parameters corresponding to the number of neighbors considered (`k`), the distance function used (`dist`) and the classes that should be under-sampled (`Cl`). This last parameter may be set to `smaller`. In this case, the smaller classes are automatically estimated, and assumed to be the most important ones. All the other least important classes are candidates for the under-sampling of NCL method to be applied. We now provide some examples of application of the NCL method.

We will begin using the ImbC data set. As this data contains numeric and nominal features it is necessary to use suitable distance functions, such as "HEOM" or "HVDM".

```
IHEOM1 <- NCLClassif(Class~., ImbC, k=10, dist="HEOM", Cl="smaller")
IHEOM2 <- NCLClassif(Class~., ImbC, k=1, dist="HEOM")
IHEOM3 <- NCLClassif(Class~., ImbC, k=1, dist="HEOM", Cl="rare1")

IHVDM1<- NCLClassif(Class~., ImbC, k=10, dist="HVDM", Cl="smaller")
IHVDM2<- NCLClassif(Class~., ImbC, k=5, dist="HVDM", Cl="rare1")
IHVDM3<- NCLClassif(Class~., ImbC, k=1, dist="HVDM", Cl="rare2")
```

Table 11 summarizes the impact produced in the number of examples in the classes on the new data sets chenaged through NCL strategy and Figure 22 show the examples distributions in these data sets.

	normal	rare1	rare2
Original	859	10	131
IHEOM1	849	10	131
IHEOM2	835	10	131
IHEOM3	831	10	106
IHVDM1	839	10	131
IHVDM2	818	10	116
IHVDM3	828	5	131

Table 11: Number of examples in each class for different parameters of NCL strategyon ImbC data set.

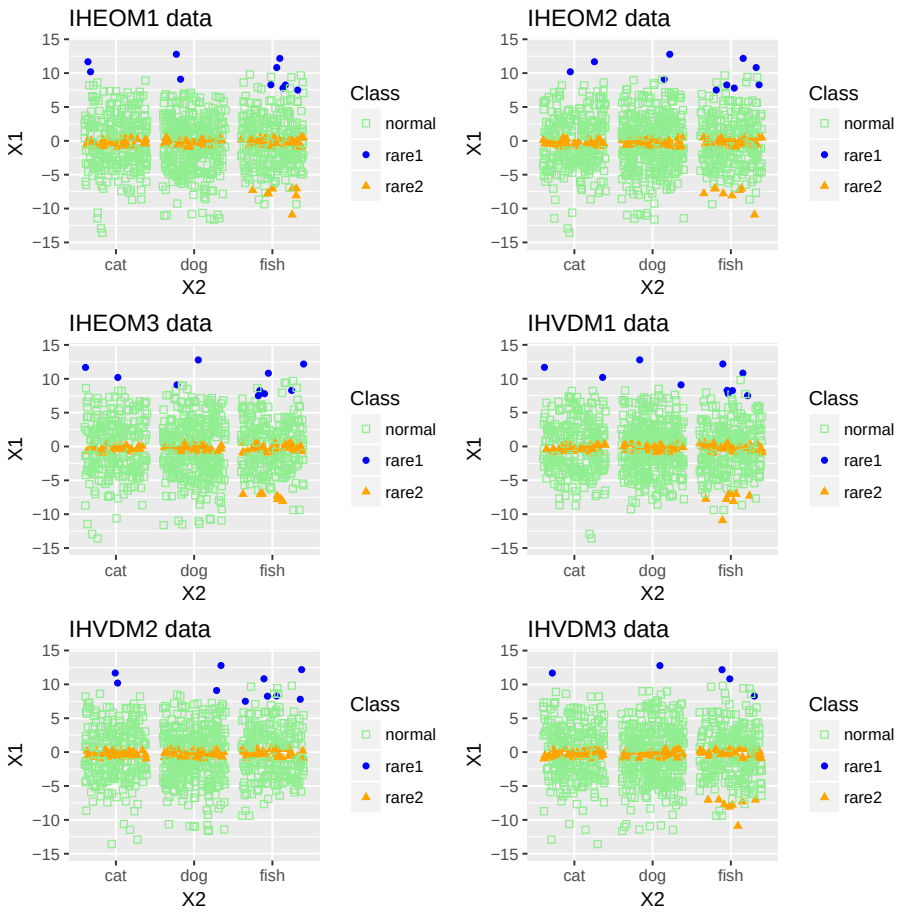

Figure 22: Impact on ImbC data of NCL method with different parameters.

Let us now observe how this technique can be used in the imbalanced iris subset previously defined.

```
set.seed(1234)
ir.M1 <- NCLClassif(Species~., dat, k=3, dist="p-norm", p=1, Cl="smaller")
ir.M2<- NCLClassif(Species~., dat, k=1, dist="p-norm", p=1, Cl="setosa")
ir.Def <- NCLClassif(Species~., dat)
ir.Ch <- NCLClassif(Species~., dat, k=7, dist="Chebyshev", Cl="virginica")

## Warning:  ENNClassif found no examples to remove!

ir.Eu <- NCLClassif(Species~., dat, k=3, dist="Euclidean",
                    Cl=c("setosa", "virginica"))
```

Table 12 provides the number of examples in each class for different parameters of NCL method and in Figure 23 the changes produced by the use of this method may be visualized.

	setosa	versicolor	virginica
Original	50	25	15
ir.M1	50	25	15
ir.M2	50	16	3
ir.Def	50	25	15
ir.Ch	47	21	15
ir.Eu	50	19	15

Table 12: Number of examples in each class for different parameters of NCL strategy.

5.9 Generation of synthetic examples by the introduction of Gaussian Noise

The use of Gaussian Noise to introduce a small perturbation in the data set examples was proposed by [Lee99] and then extended in [Lee00]. The proposed method consisted of producing replicas of the examples of the minority class by introducing normally distributed noise. In this approach, the majority class remained unchanged while the minority class was increased. The noise introduced depends on a fraction of the standard deviation of each numeric feature.

We have adapted this technique to multiclass imbalanced problems. Moreover, we have also included the possibility of combining this over-sampling procedure with the random under-sampling technique described in Section 5.1.

Regarding the over-sampling method, a new example from an important class is obtained by perturbing each numeric feature according to a random value following a normally distributed percentage of its standard deviation (with the standard deviation evaluated on the examples of that class). This means that, for a given value of `pert` defined by the user, each feature value (i) of the new example (new_i) is built as follows: $new_i = ex_i + rnorm(0, sd(i) \times pert)$, where ex_i represents the original example value for feature i, and $sd(i)$ represents the evaluated standard deviation for feature i in the class under consideration. For nominal features, the new example selects a label with a probability directly proportional to the frequency of the existing labels(with the frequency evaluated on the examples of that class).

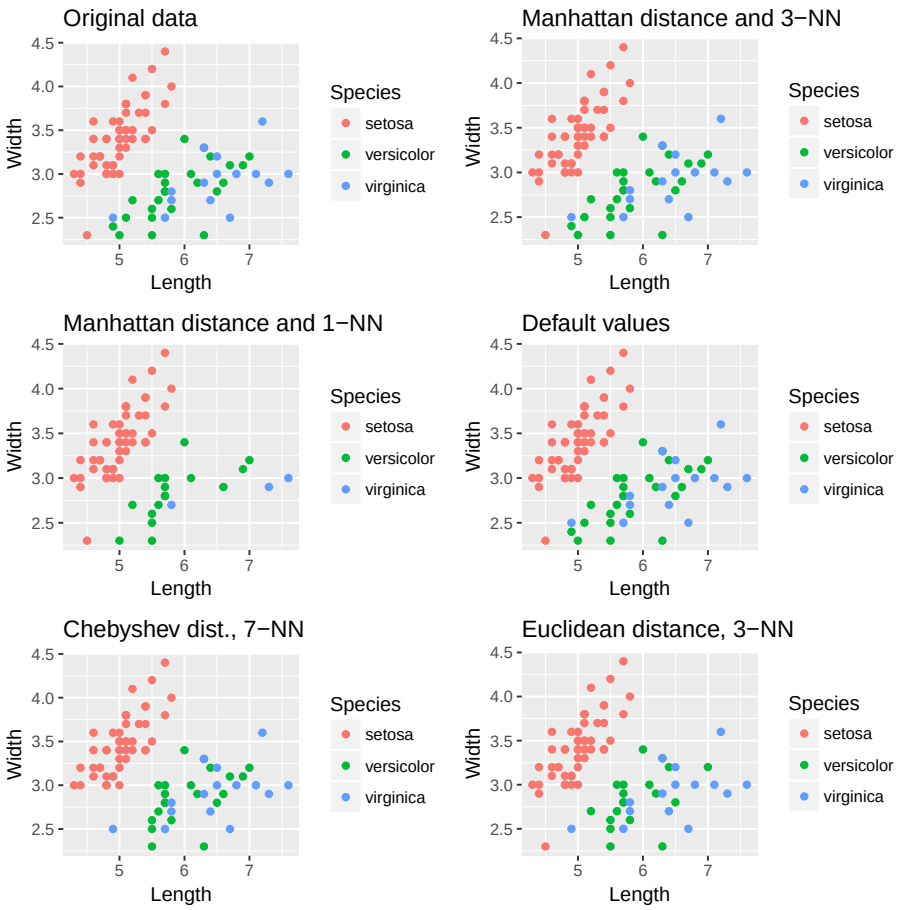

Figure 23: NCL techniques applied to a multiclass imbalanced problem.

The user may express which are the most relevant and the less important classes of the data set through the parameter `C.perc`. With this parameter the user also indicates the percentages of under and over-sampling to apply in each class. If a class is not referred in this parameter it will remain unchanged. Moreover, this parameter can also be set to "balance" or "extreme", cases where the under and over-sampling percentages are automatically estimated to achieve a balanced data set or a data set with the frequencies of the classes inverted. The perturbation applied to the numeric features is set using the `pert` parameter. Finally, the user may also specify if, when performing the random under-sampling strategy, it is allowed to perform sampling with repetition or not.

We now present an example of the impact of applying this technique for different values of the parameters in IMbC data.

```
# using the default parameters that balance the problem classes
GN1 <-GaussNoiseClassif(Class~., ImbC)

# increase the neighborhood radius for generating the new synthetic examples
# the default is pert = 0.1
GN2 <- GaussNoiseClassif(Class~., ImbC, pert = 0.5)
# select the percentages to apply in each class
GN3 <- GaussNoiseClassif(Class~., ImbC,
                         C.perc = list("normal" = 0.5, "rare1" = 10, "rare2" = 3))

#select the re-sampling percentages and the perturbation radius
GN4 <- GaussNoiseClassif(Class~., ImbC,
                         C.perc = list("normal" = 0.3, "rare1" = 5, "rare2" = 2),
                         pert = 0.05)

# use the option the inverts the classes frequencies
GN5 <- GaussNoiseClassif(Class~., ImbC, C.perc="extreme")
```

Table 13 presents the impact on the number of examples for the considered parameters of this strategy. In Figure 24 we can observe the number of examples on the changed data sets for the parameters considered and Figure 25 presents the distribution of examples for those parameters.

	normal	rare1	rare2
Original	859	10	131
GN1	333	332	332
GN2	333	332	332
GN3	429	100	393
GN4	257	50	262
GN5	11	919	70

Table 13: Number of examples in each class for different parameters of Gaussian Noise strategy applied in ImbC data.

5.10 The Smote Algorithm

The well known Smote algorithm was proposed by [CBHK02]. This algorithm presents a new strategy to address the problem of imbalanced domains through the generation of synthetic examples. The new synthetic cases are generated by interpolation of two cases from the minority (positive) class. To obtain a new

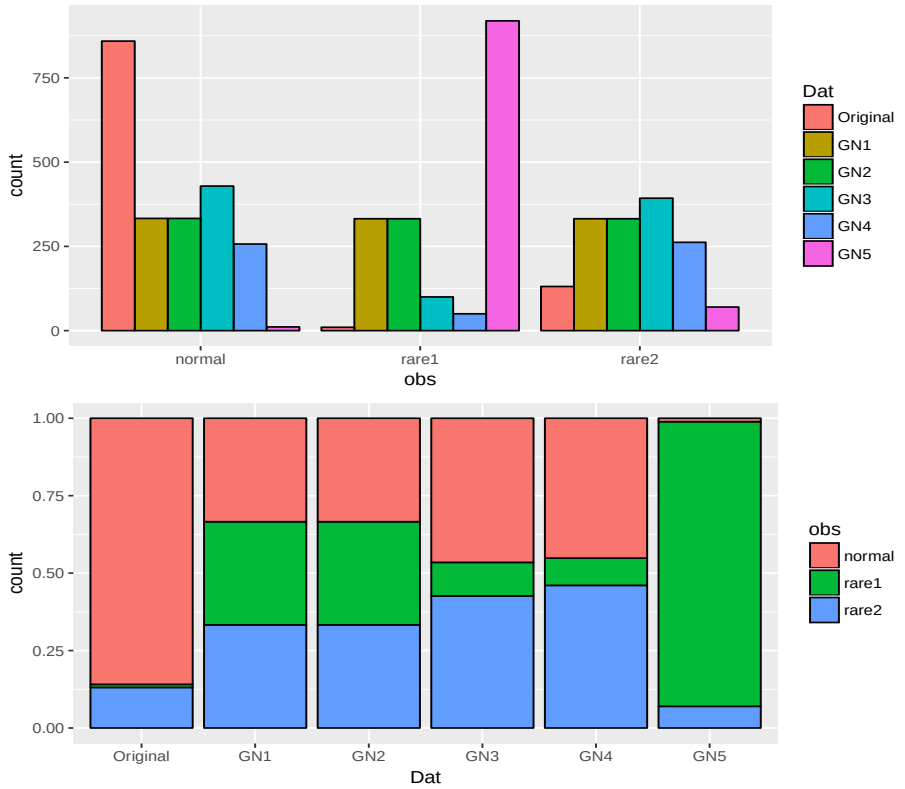

Figure 24: Impact in the Original ImbC data set of several parameters in Gaussian noise strategy.

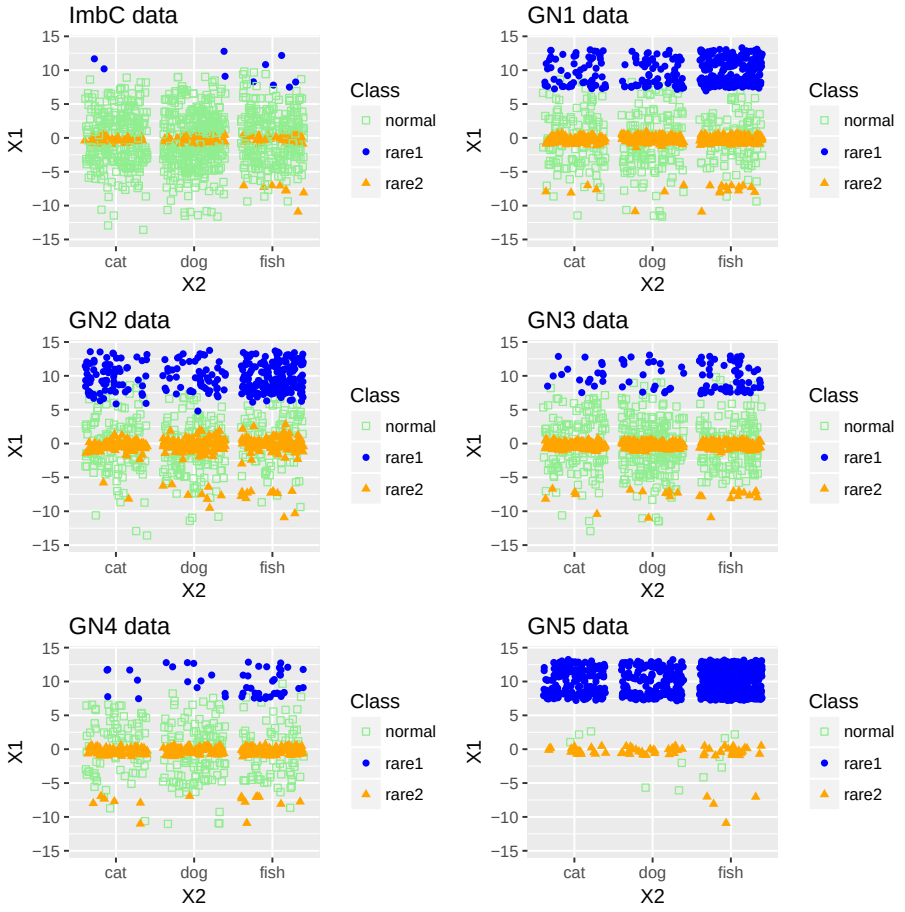

Figure 25: Impact on the examples distribution of ImbC data for different parameters in Gaussian Noise strategy.

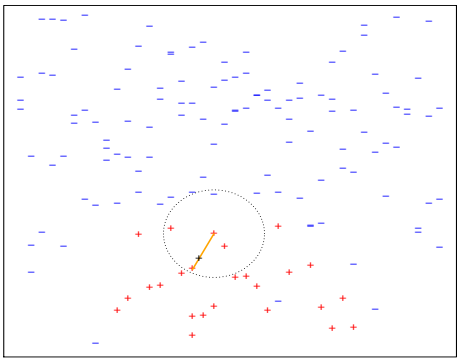

Figure 26: Generation of synthetic examples through Smote algorithm.

example from the minority class, the algorithm uses a seed example from that class, and randomly selects one of its k nearest neighbors. Then, having the two examples, a new synthetic case is obtained by interpolating the examples features. This procedure is illustrated in Figure 26.

This over-sampling strategy was also combined with random under-sampling of the majority class in [CBHK02].

Our implementation of this method is available through the `SmoteClassif` function and is able to deal with multiclass tasks. The user can specify which are the most important and the less relevant classes using the `C.perc` parameter. Using the same parameter the user also expresses the percentages of over and under-sampling that should be applied to each class. When the data set includes nominal features, the interpolation of two examples for these features is solved by randomly selecting among the two values of the seed examples. Two automatic methods are provided for both the estimation of the relevant classes and the percentages of over and under-sampling to apply. These methods are available through the `C.perc` parameter which can be set to "balance" or "extreme". In both cases, it is ensured that the new obtained data set includes roughly the same number of examples as the original data set. When "balance" or "extreme" are chosen, both the minority/majority classes and the percentages of over/under-sampling are automatically estimated. The "balance" option provides a balanced data set and the "extreme" option provides a data set with the classes frequencies inverted, i.e., the most frequent classes in the original data set are the less frequent on the new data set and vice-versa.

Finally, the user may also express if the under-sampling process may include repetition of examples or not (using the `repl` parameter), may choose the number of nearest neighbors to use (parameter `k`) and can select the distance metric to be used in the nearest neighbors evaluation (parameter `dist`).

The following example shows how this strategy can be used to modify the synthetic ImbC data set.

```r
IC1 <- SmoteClassif(Class~., ImbC, dist = "HEOM")

IC2 <- SmoteClassif(Class~., ImbC, k = 1, dist="HEOM")

IC3 <- SmoteClassif(Class~., ImbC,
                    C.perc = list("normal" = 0.4, "rare1" = 8, "rare2" = 6),
                    dist = "HEOM")

IC4 <- SmoteClassif(Class~., ImbC, dist = "HVDM")

IC5 <- SmoteClassif(Class~., ImbC, k = 1, dist = "HVDM")

# class rare2 is not refered in the C.perc parameter. This means that
# this class will remain unchanged
IC6 <- SmoteClassif(Class~., ImbC, dist = "HVDM",
                    C.perc = list("normal"=0.2, "rare1"=10))

IC7 <- SmoteClassif(Class~., ImbC, dist = "HVDM", C.perc="extreme")
```

Table 15 show the impact on the number of examples in each class for several parameters of smote technique.

	normal	rare1	rare2
Original	859	10	131
IC1	333	332	332
IC2	333	332	332
IC3	343	80	786
IC4	333	332	332
IC5	333	332	332
IC6	171	100	131
IC7	11	919	70

Table 14: Number of examples in each class for different parameters of smote strategy.

Figures 27 and 28 present the impact of applying smote strategy on an imbalanced data set.

5.11 ADASYN Algorithm

The ADASYN algorithm was proposed by [HBGL08] for addressing the problem of imbalanced domains. ADASYNs' key idea is to use, for different minority class cases, a different weight that is associated with their level of difficulty in learning. More synthetic minority class examples are generated for the minority examples that are harder to learn while less sytnhetic examples are generated for the easier to learn. To determine the number of new examples to be generated for each minority class case the authors build a density distribution that is obtained as follows:

i) for each case x_i determine the number $d_i \leq k$ of k-nearest neighbours that belong to the majority class;

ii) calculate $r_i = d_i/k$

iii) When all r_i are calculated, obtain $\hat{r}_i$ by normalizing r_i so that $\hat{r}_i$ is a density distribution.

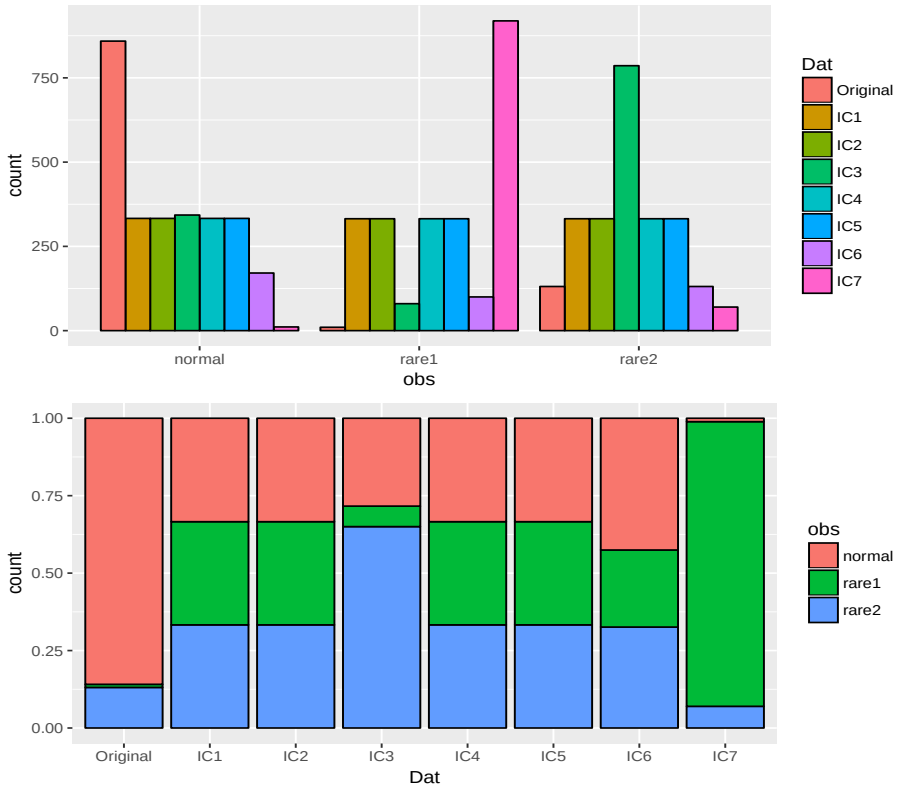

Figure 27: Impact in the Original data set of several parameters in smote strategy.

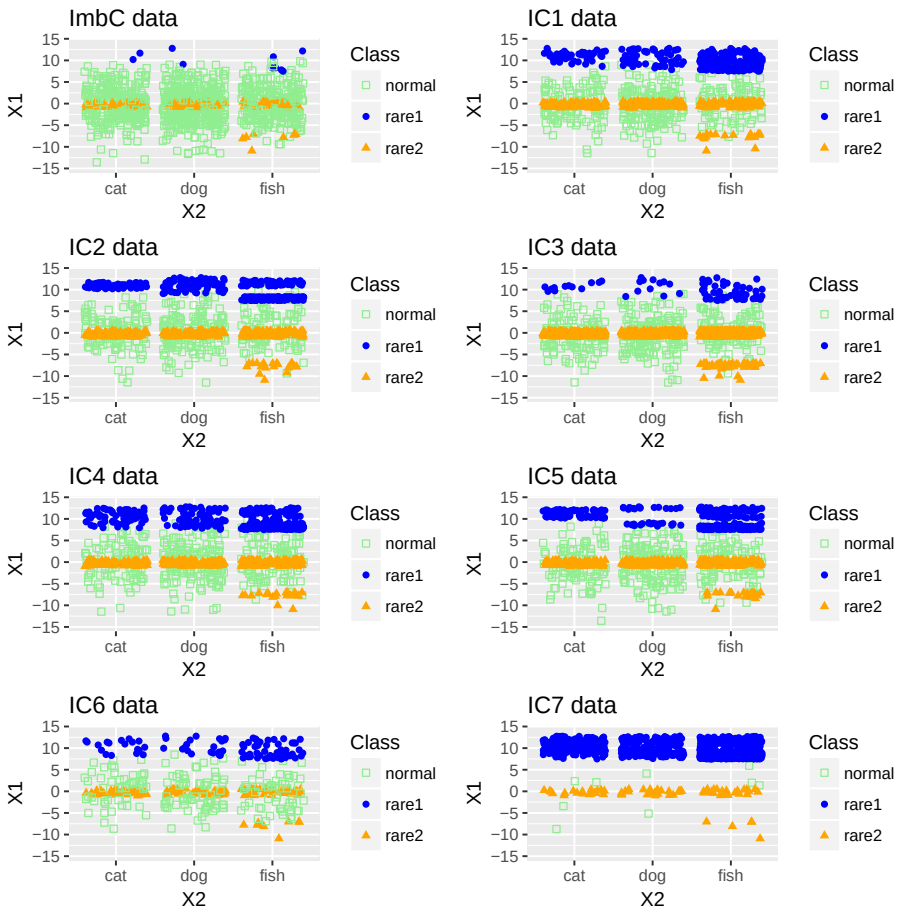

Figure 28: Smote strategy applied to ImbC data with different parameters.

Now, each case has an associated weight ($\hat{r}_i$) that is used as follows:

i) calculate the total number G of new synthetic examples that are necessary to generate, according to the problem imbalance ratio and the user provided parameter beta that sets the balance desired in new distribution.

ii) for example x_i generate $\hat{r}_i \times G$ new examples.

The new synthetic examples are obtained through SMOTE algorithm [CBHK02].

The authors of ADASYN algorithm claim that this algorithm is able: i) to reduce the learning bias introduced by the original imbalanced distribution; and ii) to adaptively shift the decision border towards the most difficult examples.

ADASYN algorithm was proposed for binary classification problems [HBGL08]. In the UBL package we have extended the presented ADASYN strategy and have also implemented it for multiclass problems. To achieve this we introduced parameter `baseClass` that allows the user to specify which is the reference class, i.e., the class against which all other classes will be compared to. If the user does not specifies this parameter, then the reference class is assumed to be most frequent class. The ADASYN algorithm also requires that a parameter beta is specified by the user. This parameter sets the desired balance level to obtain after the synthetic examples generation. In a binary class setting a single number is sufficient for beta. When beta is set to 1, the default, then the new data set to obtain will have the classes roughly balanced. If a single number is also used in a ulticlass setting, it will be used to express the balance level of each class against the `baseClass`. We also allow the user to provide a named list containing the classes names and the corresponding beta value to use.

Let us now observe the impact of aplying the ADADYN algorithm to the synthetic ImbC data set.

```
Adas1 <- AdasynClassif(Class~., ImbC, dist = "HEOM")

Adas2 <- AdasynClassif(Class~., ImbC, k = 1, dist="HEOM")

Adas3 <- AdasynClassif(Class~., ImbC,
                    C.perc = list("normal" = 0.4, "rare1" = 8, "rare2" = 6),
                    dist = "HEOM")

## Error in AdasynClassif(Class ~ ., ImbC, C.perc = list(normal = 0.4, rare1 = 8, :
unused argument (C.perc = list(normal = 0.4, rare1 = 8, rare2 = 6))

Adas4 <- AdasynClassif(Class~., ImbC, dist = "HVDM")

Adas5 <- AdasynClassif(Class~., ImbC, k = 1, dist = "HVDM")

# class rare2 is not refered in the C.perc parameter. This means that
# this class will remain unchanged
Adas6 <- AdasynClassif(Class~., ImbC, dist = "HVDM",
                    C.perc = list("normal"=0.2, "rare1"=10))

## Error in AdasynClassif(Class ~ ., ImbC, dist = "HVDM", C.perc = list(normal = 0.2,
:  unused argument (C.perc = list(normal = 0.2, rare1 = 10))

Ada7 <- AdasynClassif(Class~., ImbC, dist = "HVDM", C.perc="extreme")

## Error in AdasynClassif(Class ~ ., ImbC, dist = "HVDM", C.perc = "extreme"):  unused
argument (C.perc = "extreme")
```

Table **??** show the impact on the number of examples in each class for several parameters of Adasyn technique.

	normal	rare1	rare2
Original	859	10	131
IC1	333	332	332
IC2	333	332	332
IC3	343	80	786
IC4	333	332	332
IC5	333	332	332
IC6	171	100	131
IC7	11	919	70

Table 15: Number of examples in each class for different parameters of smote strategy.

Figures 29 and 30 present the impact of applying smote strategy on an imbalanced data set.

6 Methods for Addressing Utility-based Regression Tasks

Utility-based problems also occur for regression tasks. However, for these problems we have a continuous target variable and therefore are no classes defined. Instead, the user may consider some ranges of the target variable domain more important (which are usually less represented) while other regions of that variable are less important. As proposed by [TR07, Rib11], utility-based regression problems depend on the definition of a continuous relevance function ($\phi()$) which expresses the importance of the target variable values across its domain. This function $\phi()$, varies between 0 and 1, where 0 represents points in the target variable domain which are not relevant and 1 identifies the most important values. Usually, the user is also asked to provide a relevance threshold (a numeric value in $[0, 1]$) which helps to clearly distinguish between the important and unimportant values.

[Rib11] proposed a framework for defining the relevance function of a given continuous target variable. This framework has an automatic method that allows to obtain the relevance function from the target variable distribution. The assumption made to achieve this goal regards the most usual setting, where the extreme rare values are the most important to the user. This framework also allows the user to manually specify which are the relevant and irrelevant values using a matrix. The R package `uba` [RwcfLT14], available in `http://www.dcc.fc.up.pt/~rpribeiro/uba/`, includes several other functionalities for dealing with utility-based regression. We use in `UBL` package the functions regarding the relevance function.

Considering a target variable with domain $[0, 10]$, a possible relevance function could be the one represented in Figure 31. For this particular regression task, the relevance function selected and the chosen relevance threshold of 0.5 characterize the most important ranges of the target variable and the bumps of relevance. In this case, we have established two bumps which include the most important values (also named "rare" cases) of the target variable ($[0, 1.5]$ and $[4.5, 7]$ represented in green in Figure 31). On the other hand, the target values

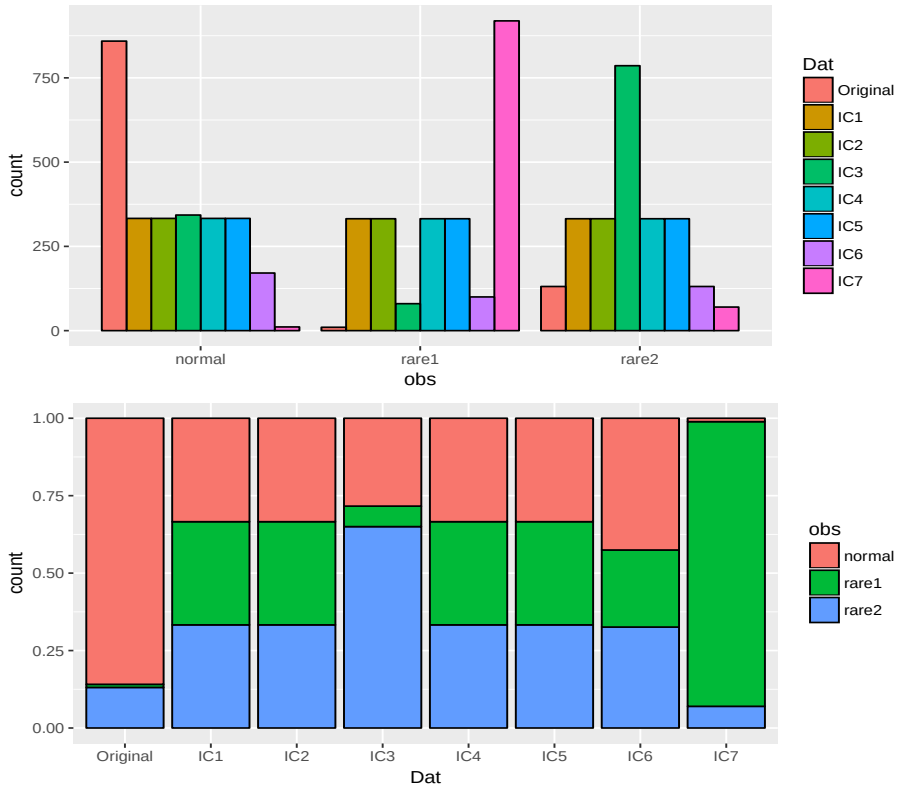

Figure 29: Impact in the Original data set of several parameters in smote strategy.

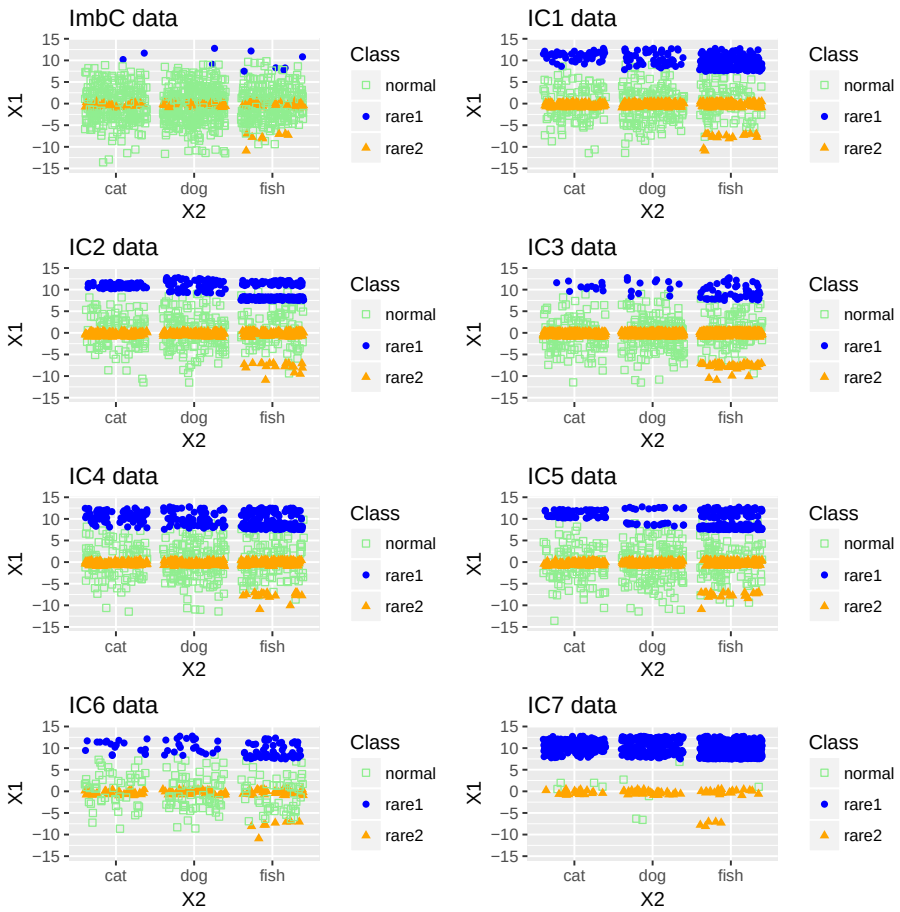

Figure 30: Smote strategy applied to ImbC data with different parameters.

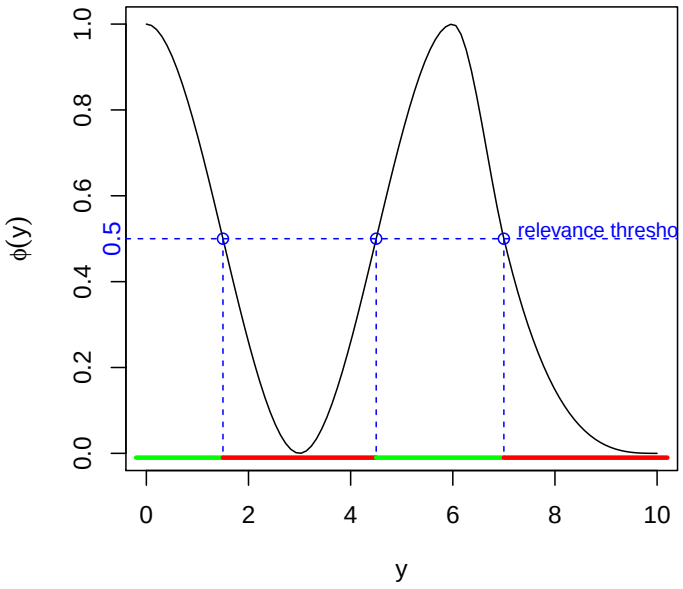

Figure 31: Example of a relevance function.

falling in the intervals $]1.5, 4.5[$ and $]7, 10]$ (represented in red in Figure 31) are the less relevant and "normal" cases.

The user has the responsibility of defining a relevance function suitable for the regression task he is considering. We provide the mechanism proposed by [Rib11] and also implemented in `uba` package to assist the user in this task. This method is called `range`, and depends on the introduction by the user of reference points for the y, and corresponding $\phi()$ and $\phi'()$ values. With the method `range` the relevance function may be manually defined with a 3-column matrix containing the interpolating points as follows:

```
# relevance function represented in the previous example

## method: range
# the user should provide a matrix with y, phi(y), phi'(y)

rel <- matrix(0, ncol = 3, nrow = 0)

# for the target value of zero the relevance function should be one and
# the derivative at that point should be zero
rel <- rbind(rel, c(0,1,0))

# for the value three the relevance assigned is zero and the derivative is zero
rel <- rbind(rel, c(3,0,0))
rel <- rbind(rel, c(6,1,0))
rel <- rbind(rel, c(7,0.5,1))
rel <- rbind(rel, c(10,0,0))
# after defining the relevance function the user may obtain the
```

```
# phi values as follows:

# use method "range" when defining a matrix
phiF.args <- phi.control(y,method = "range", control.pts = rel)

# obtain the relevance values for the target variable y
y.phi <- phi(y,control.parms = phiF.args)
```

In order to facilitate the user task, we also provide an automatic mechanism proposed by [Rib11] and also implemented in **uba** package, for defining the relevance function. This automatic method, called **extremes** is based on the boxplot of the target variable values and assigns a larger importance to the least represented values. In this case, the user does not need to provide interpolating points because this method assumes that the least represented ranges of the target variable are the most important. We now provide an example of how to use this automatic method.

```
## method: extremes

## for considering only the high extremes
phiF.args <- phi.control(y,method = "extremes",extr.type = "high")
y.phi <- phi(y,control.parms = phiF.args)

## for considering only the low extremes
phiF.args <- phi.control(y,method = "extremes",extr.type = "low")
y.phi <- phi(y,control.parms = phiF.args)

## for considering both extreme types (low and high)
phiF.args <- phi.control(y,method = "extremes",extr.type = "both")
y.phi <- phi(y,control.parms = phiF.args)
```

All the implemented methods for utility-based regression tasks depend on the definition of a relevance function, and the majority of them also rely on a user-defined relevance threshold.

Let us now observe the impact of using the automatic method for defining a relevance function with the synthetic ImbR data provided with UBL package .

```
# define that the automatic method will be used and
# specify that we are only interested in the high extreme values
phiF.args <- phi.control(ImbR$Tgt, method = "extremes", extr.type = "high")
y.phi <- phi(sort(ImbR$Tgt),control.parms = phiF.args)
```

However, the user has also the possibility to define its own relevance function as follows:

```
# specify the y, phi(y) and phi'(y) in each row of the matrix
rel <- matrix(c(10, 1, 0, 11, 0, 0, 18, 0.5, 1, 19, 0.8, 0, 21, 1, 0),
              ncol = 3, nrow = 5, byrow = TRUE)
phiF.argsR <- phi.control(ImbR$Tgt, method = "range", control.pts = rel)
y.phiR <- phi(sort(ImbR$Tgt), control.parms = phiF.argsR)
```

Figures 32 and 33 show the two relevance functions previously obtained for ImbC data (the first one is built with the automatic and the second uses the matrix with interpolating points provided by the user). The automatic method "extremes" takes into account the examples distribution while the "range" method uses the information provided by the user regardless of the domain distribution.

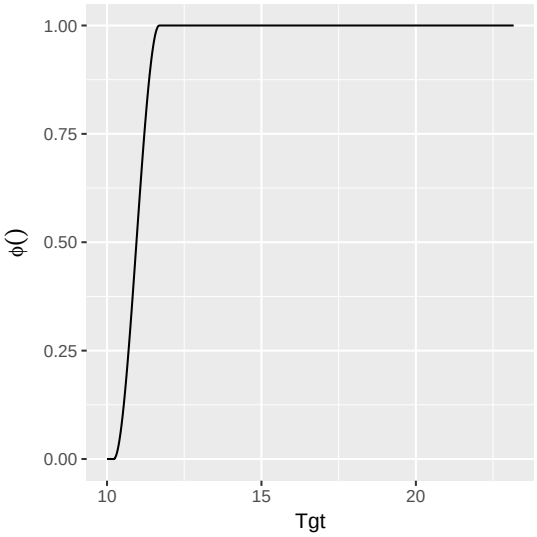

Figure 32: Relevance function automatically obtained.

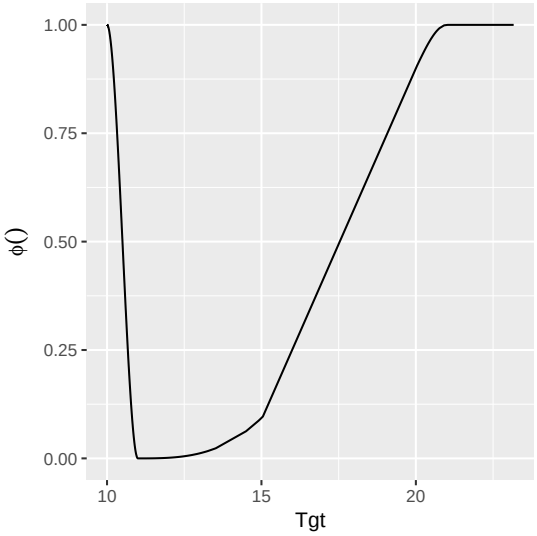

Figure 33: Relevance function obtained through a matrix with interpolating points provided by the user.

In the relevance function specified with "range" method the lower and higher values are both considered extremely relevant.

In the next sections we describe the following methods for tackling utility-based regression tasks:

- 6.1 Random Under-sampling
- 6.2 Random Over-sampling
- 6.3 Gaussian Noise Introduction
- 6.4 SmoteR Algorithm
- 6.5 Importance Sampling

6.1 Random Under-sampling

Random under-sampling strategy for regression problems was first proposed by [TRPB13]. This strategy is similar to the strategy presented for classification. It depends on the definition of both a relevance function and a relevance threshold. In this proposal, all the target values below the relevance threshold are considered normal and uninteresting and thus are regarded as candidates to be under-sampled. The user is also asked to set another parameter that establishes the proportion between normal (unimportant) and rare (important) cases that the new under-sampled data set should contain.

In the implementation of this strategy in package `UBL` , we ask for the user to define the relevance function (manually through the method "range" or using the automatic method, called "extremes", previously described). This means that the user may define as many relevance bumps as wanted. Parameter `rel` is used to indicate the relevance function. For using the automatic method the parameter `rel` should be set to "auto" (the default). If the user wants to apply the range method, then, as previously explained, a 3-column matrix should be provided. It is also necessary for the user to define a relevance threshold through the `thr.rel` parameter. Having this set, all the target variable values with relevance below the relevance threshold are candidates to be under-sampled. Finally, the user can also express using the `C.perc` parameter which under-sampling percentage should be applied in each bump with uninteresting values, or alternatively this parameter may be set to "balance" or "extreme". If "balance" is chosen the under-sampling percentage is automatically estimated in order to balance the normal/important and rare/unimportant cases. On the other hand, the "extreme" option will invert the existing frequencies. The following example uses the regression data set provided with `UBL` package , ImbR, to show how these parameters can be set and their impact on the changed data.

```
data(ImbR) # load the synthetic data set provided with UBL

# Using the automatic method for defining the relevance function
# This is the default behaviour, therefore, we can simply
# not mention the "rel" parameter

# default of C.perc parameter balances the examples in the bumps
IRU1 <- RandUnderRegress(Tgt~., ImbR)

IRU2 <- RandUnderRegress(Tgt~., ImbR, C.perc = "extreme")
```

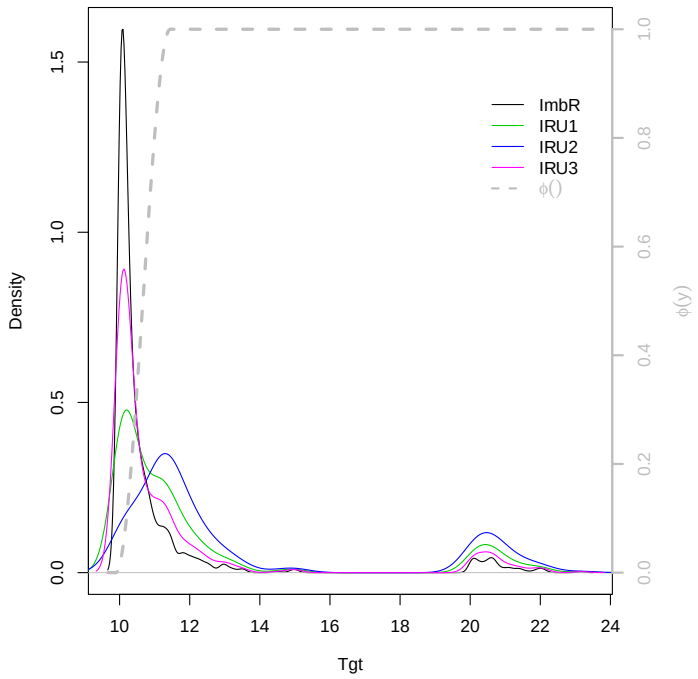

Figure 34: Automatic relevance function and density of the target variable in the original ImbR data and the changed data sets using Random Under-sampling strategy

```
# the automatic method for the relevance function generates only
# one bump with uninteresting values, thus we only need to set
# one under-sampling percentage
IRU3 <- RandUnderRegress(Tgt~., ImbR, C.perc = list(0.5))
```

Figure 34 shows the impact of the applied strategies on the density of target variable of ImbR data.

We can also observe the impact of the previously defined changes on the examples distribution in Figure 35.

Let us now assume that we have some domain knowledge that leads us to considered a different relevance function. Suppose that the most relevant cases are the target varaible values close to 15. In the following example we define a new relvance function suitable for this context and apply the random under-sampling strategy to change the original data set with different parameters.

```
rel <- matrix(c(14, 0, 0, 15, 1, 0, 16, 0, 0, 20, 1, 0, 21, 0, 0),
              ncol=3, nrow=5, byrow=TRUE)
dsU1 <- RandUnderRegress(Tgt~., ImbR, rel=rel)
dsU2 <- RandUnderRegress(Tgt~., ImbR, rel=rel, C.perc=list(0.5))
dsU3 <- RandUnderRegress(Tgt~., ImbR, rel=rel, C.perc=list(0.2))
```

Figures 36 and 37 show the density of the target variable in the original ImbR data and the pre-processed data sets and the examples distribution.

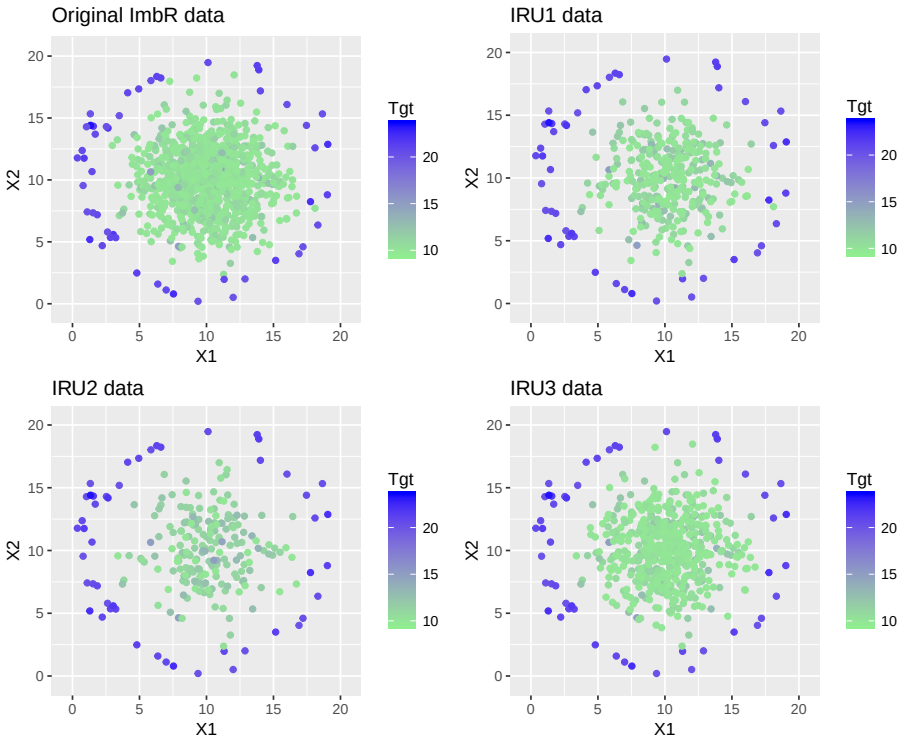

Figure 35: Original and changed data sets using different parameters of the random under-sampling strategy.

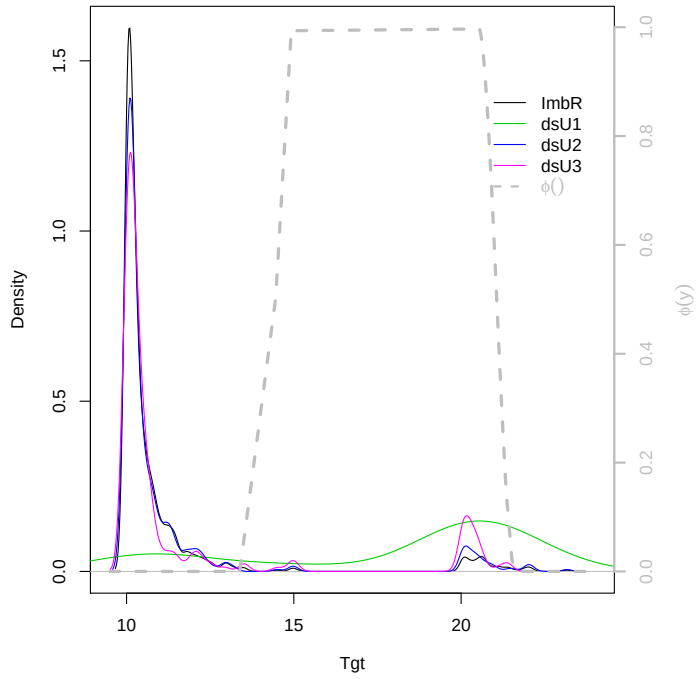

Figure 36: Density of the target variable in the original ImbR data and the changed data sets using Random Under-sampling strategy and a user defined relevance function.

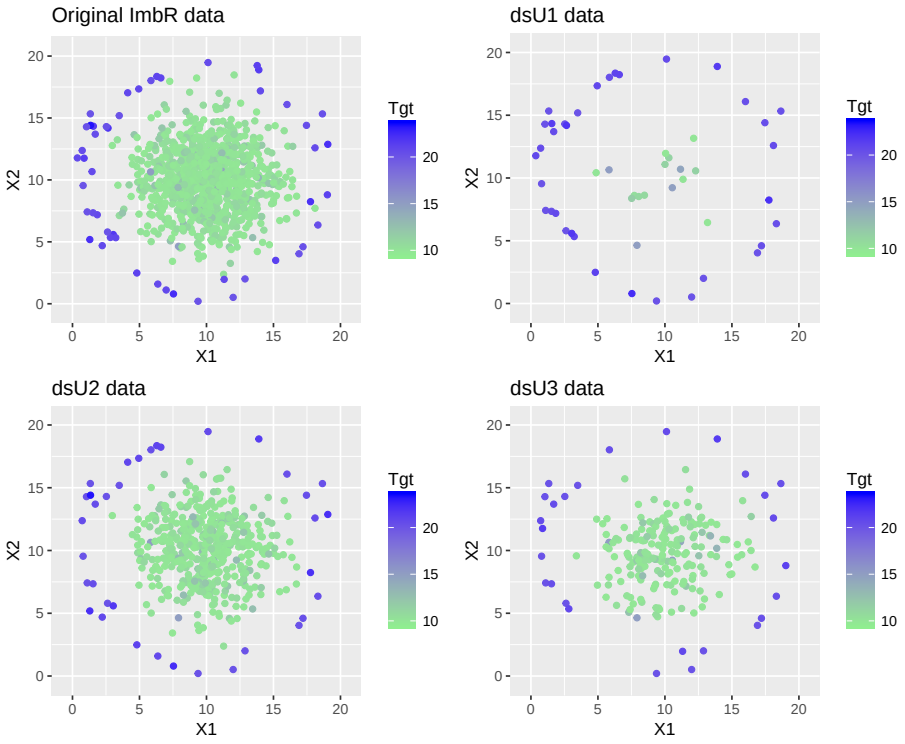

Figure 37: Original and changed data sets using different parameters of the random under-sampling strategy and a user defined relevance function.

We must highlight that this strategy entails some consequences that should not be disregarded. Namely, there can be a sever impact on the total number of examples in the modified data sets. If we are considering a large data set, possibly removing 100 points may have a negligible impact. However, if the data set is already small, then removing 100 examples may have an huge impact.

This can be observed in the previous examples. In fact, the `C.perc` parameter must be thought carefully due to the consequences on the total number of examples. In Table 16 we can check the impact of the several strategies on the data set for the two relevance functions considered (the obtained through the automatic method and the defined with a 3-column matrix).

	ImbR	IRU1	IRU2	IRU3	dsU1	dsU2	dsU3
nr. examples	1000	390	242	597	51	513	220

Table 16: Total number of examples in each data set for different parameters of random under-sampling strategy.

6.2 Random Over-sampling

The Random over-sampling method proposed is an adaptation of the Random over-sampling method proposed for classification tasks using the previously presented relevance function for utility-based regression tasks. This technique is available through `randomOverRegress` function, and is simply based on the random introduction of replicas of examples of the original data set. These replicas are only introduced in the most important ranges of the target variable, i.e., in the ranges where the relevance is above a user-defined threshold. Similarly to what happened in Random under-sampling, the user may define its own relevance function or use the automatic method provided to generate one. It is also the user responsibility to define the relevance threshold (using the `thr.rel` parameter) and the percentages of over-sampling to apply in each bump of relevance (through the `C.perc` parameter). Alternatively, the user may set the `C.perc` parameter as "balance" or "extreme", cases which automatically evaluate the percentages of over-sampling to apply for obtaining a new balanced data set or for inverting the frequencies of examples in the defined bumps. In the following example we can see how to use this function.

```
# using the automatic method for defining the relevance function and
# the default threshold of 0.5
IRO <- RandOverRegress(Tgt~., ImbR, C.perc=list(2.5))
IROBal <- RandOverRegress(Tgt~., ImbR, C.perc="balance")
IROExt <- RandOverRegress(Tgt~., ImbR, C.perc="extreme")

# change the relevance threshold to 0.9
IRO0.9 <- RandOverRegress(Tgt~., ImbR, thr.rel=0.9)
```

Figure 38 shows the impact of this method for several parameters.

This method also carries a strong impact on the total number of examples in the modified data set. While the random Under-sampling method is able to produce a significant reduction of the data set, the random over-sampling technique will increase, sometimes drastically, the data set size. Table 17 shows

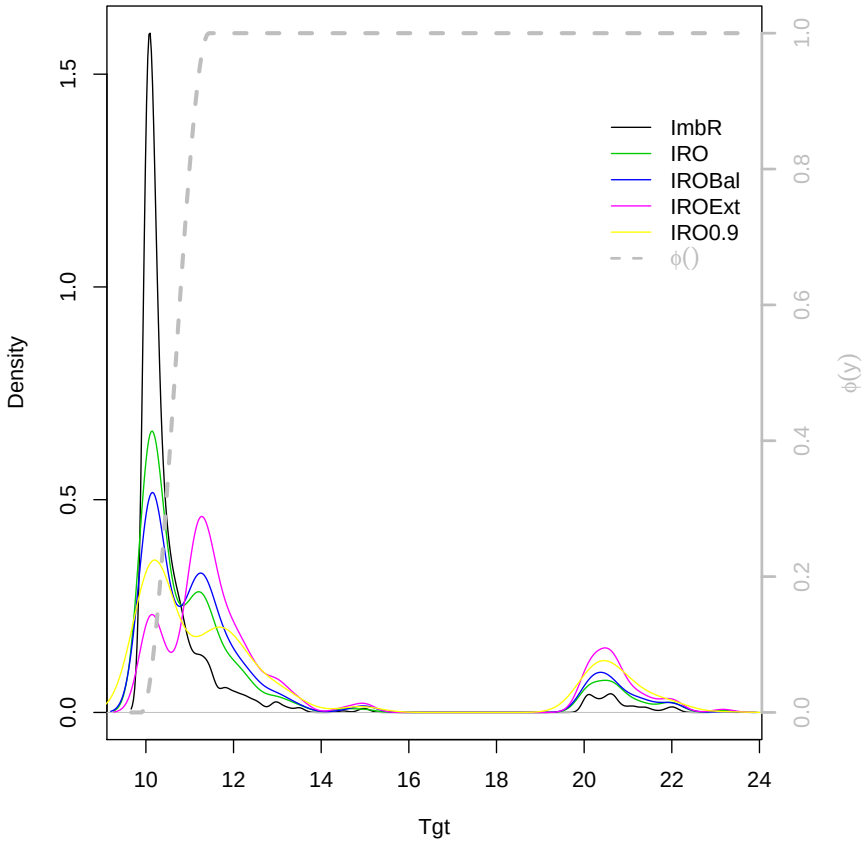

Figure 38: Relevance function and density of the target variable in the original and new data sets using Random over-sampling strategy.

	ImbR	IRO	IROBal	IROExt	IRO0.9
nr. examples	1000	1487	1805	4323	1866

Table 17: Total number of examples in each data set for different parameters of random over-sampling strategy.

the impact of the previous examples on the total number of examples of used the data set.

As expected, all the data sets have an increased size. However, for the `IRO0.9` data set, the size was increased approximately 187%. This "side effect" must be taken into consideration when applying this technique because it may impose constraints on the used learners. We must also highlight that, although the data set size can be strongly increased, we are in fact only introducing replicas of already existing examples, and thus no new information is being inserted.

6.3 Generation of synthetic examples by the introduction of Gaussian Noise

The generation of synthetic examples through the introduction of small perturbations based on Gaussian Noise was a strategy proposed for classification tasks [Lee99, Lee00]. The main idea of this strategy is to generate new synthetic examples with a desired class label, by perturbing the features of examples of that class a certain amount of the respective standard deviation.

We have adapted this over-sampling technique to regression problems and have combined it with the random under-sampling method. To accomplish this it is required that the user defines a relevance function and a relevance threshold. The examples which have a target variable value with relevance higher than the threshold set will be over-sampled, and the remaining will be randomly under-sampled. The under-sampling strategy used is the same described in Section 6.1. For the over-sampling strategy we use the same procedure which was described for classification tasks in Section 5.9. The only difference on the over-sampling method is in the target variable value. For classification tasks, the target variable value was easily assigned: it was the rare class under consideration. For regression tasks we have decided to extend the technique applied for numeric features also to the target variable. This means that the new example target variable value is obtained by a random normal perturbation of the original target value based on the target value standard deviation.

In order to use this method the user must provide a relevance function through the `rel` parameter (or use the automatic method for estimating it by setting rel to "auto"), a threshold on the relevance (parameter `thr.rel`) and the perturbation to be used (parameter `pert`). Moreover, the user may also express using the parameter `C.perc` the percentages of over and under-sampling to apply in each bump defined, or alternatively he may set this parameter to "balance" or "extreme". Similarly to the behavior described in the previous techniques, setting this parameter to "balance" or "extreme" causes the percentages of over and under-sampling to be automatically estimated. The option "balance" will try to distribute the examples evenly across the existing bumps while maintaining the total number of examples in the modified data set. If the choice is "extreme" then the frequencies of the examples in the bumps will be

inverted. The user can also indicate if the under-sampling process can be made with repetition of examples or not using the `repl` parameter.

We now show some examples of usage of the function `GaussNoiseRegress`.

```
# relevance function estimated automatically has two bumps
# defining the desired percentages of under and over-sampling to apply
C.perc=list(0.5, 3)
# define the relevance threshold
thr.rel=0.8
mygn <- GaussNoiseRegress(Tgt~., ImbR, thr.rel=thr.rel, C.perc=C.perc)
gnB <- GaussNoiseRegress(Tgt~., ImbR, thr.rel=thr.rel, C.perc="balance")
gnE <- GaussNoiseRegress(Tgt~., ImbR, thr.rel=thr.rel, C.perc="extreme")
```

Figures 39 and 40 show the impact of this strategy, for the parameters considered, on the examples distribution. In Figure 40 we have binarized the data sets into rare/important (+) and normal/unimportant cases (−). We have considered the threshold of 19 on the target varaible to distinguish between the two "classes". Figure 41 shows the true distribution of the target variable in the original ImbR data and the pre-processed data sets.

In the following example we check the impact of changing the perturbation introduced.

```
# the default uses the value of 0.1 for "pert" parameter
gnB1 <- GaussNoiseRegress(Tgt~., ImbR, thr.rel=thr.rel, C.perc="balance")

# try two different values for "pert" parameter
gnB2 <- GaussNoiseRegress(Tgt~., ImbR, thr.rel=thr.rel, C.perc="balance",
                          pert=0.5)
gnB3 <- GaussNoiseRegress(Tgt~., ImbR, thr.rel=thr.rel, C.perc="balance",
                          pert=0.01)
```

The impact of changing the parameter `pert` is represented in Figures 43 and 42.

6.4 The SmoteR Algorithm

The SmoteR algorithm was presented in [TRPB13]. This proposal is an adaptation for regression problems under imbalanced domains of the existing smote algorithm [CBHK02] for classification tasks. As with other methods addressing regression tasks on imbalanced data distributions it is the user responsability to provide a relevance function and a relevance threshold. This function determines which are the relevant and the unimportant examples. This algorithm combines an over-sampling strategy by interpolation of examples with a random under-sampling approach. For the generation of new examples by interpolation, the same procedure proposed in smote algorithm is used. Regarding the generation of the target variable value of the new generated examples the proposed smoteR algorithm uses an weighted average of the values of target variable of the two examples used. The weights are calculated as an inverse function of the distance of the generated case to each of the two seed examples. This means that, the further away the new example is from the seed case less weight will be given for the generation of the target variable value. The random under-sampling approach is applied in the bumps containing the normal and unimportant cases.

The smoteR algorithm is available through the `SmoteRegress` function. The user may define its own relevance function or use the automatic method, as in

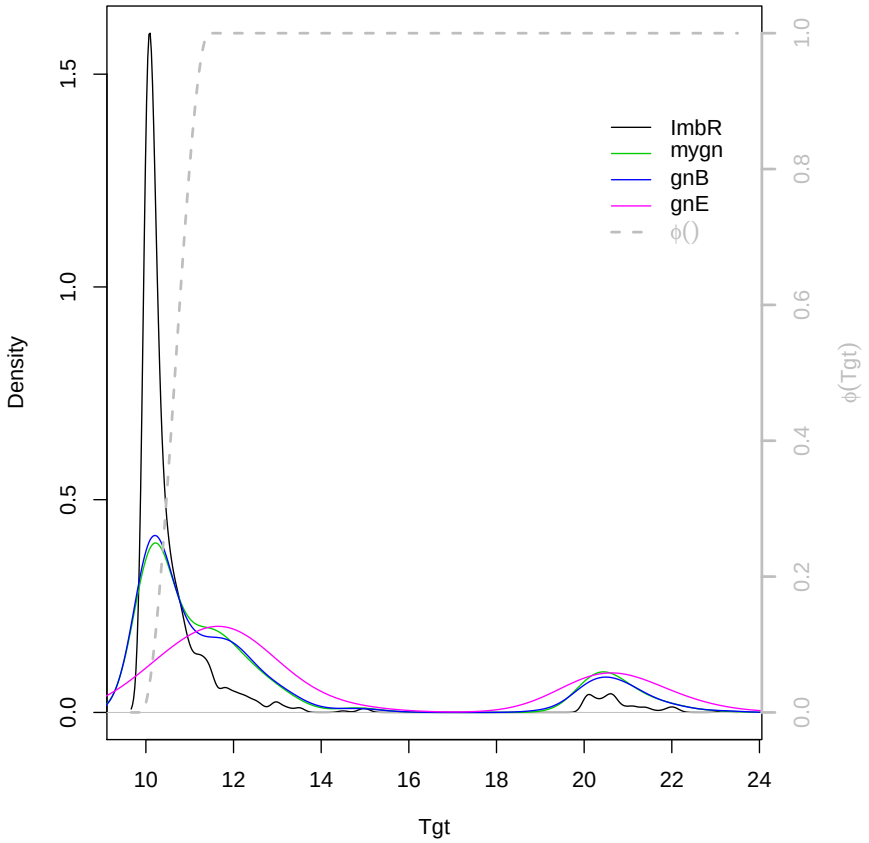

Figure 39: Relevance function and density of the target variable in the original and new data sets using Gaussian noise strategy.

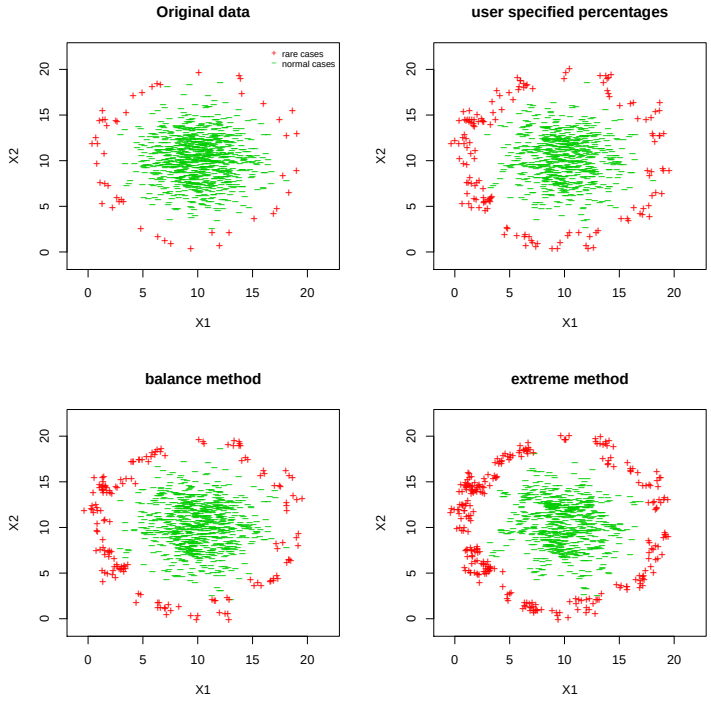

Figure 40: The impact of Gaussian Noise strategy in a binarized version of ImbR data considering Tgt values above 19 as rare and below 19 as normal cases.

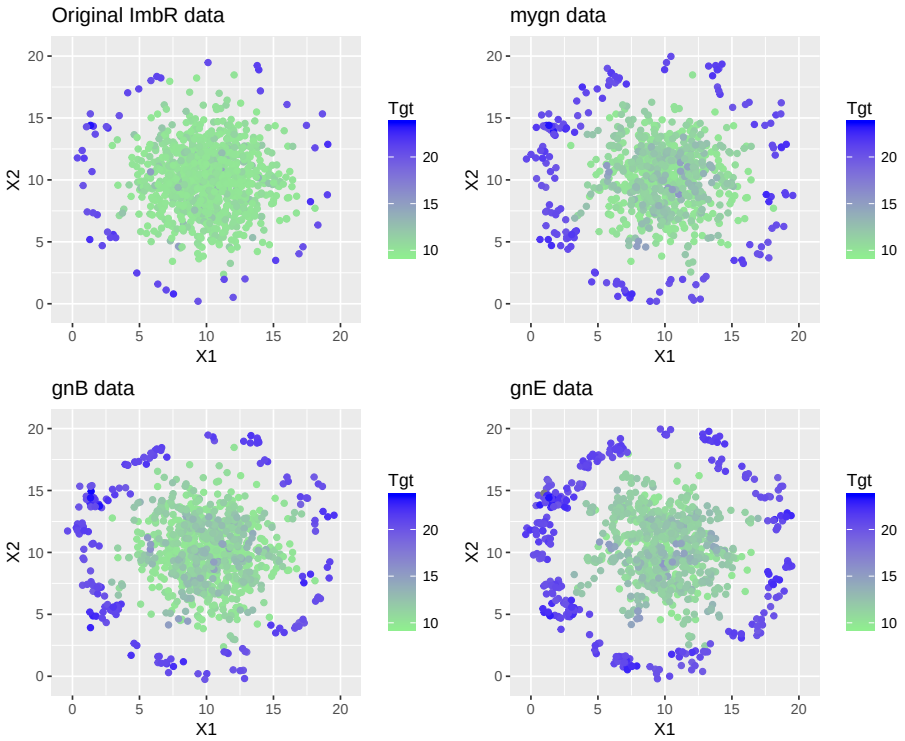

Figure 41: Target variable distribution on ImbR and data sets changed through Gaussian Noise strategy.

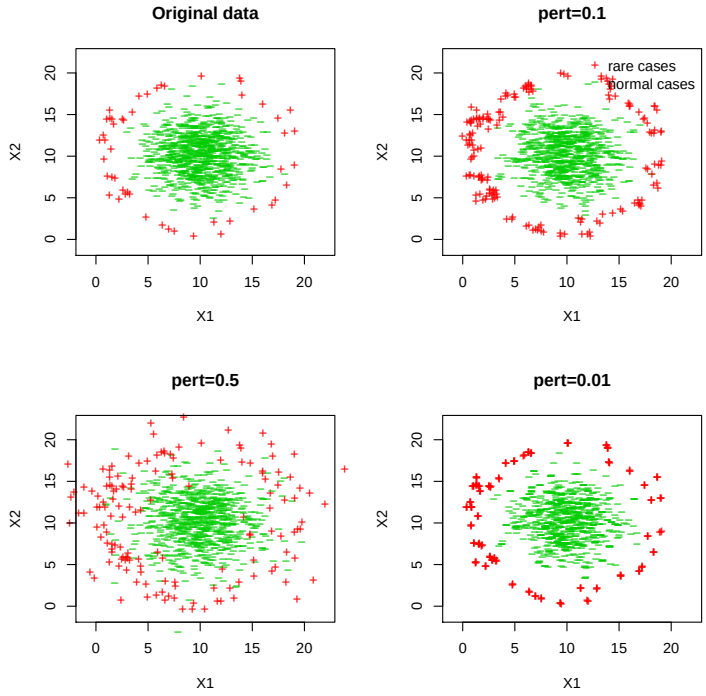

Figure 42: Impact of changing the pert parameter in Gaussian Noise strategy in a binarized version of ImbR data.

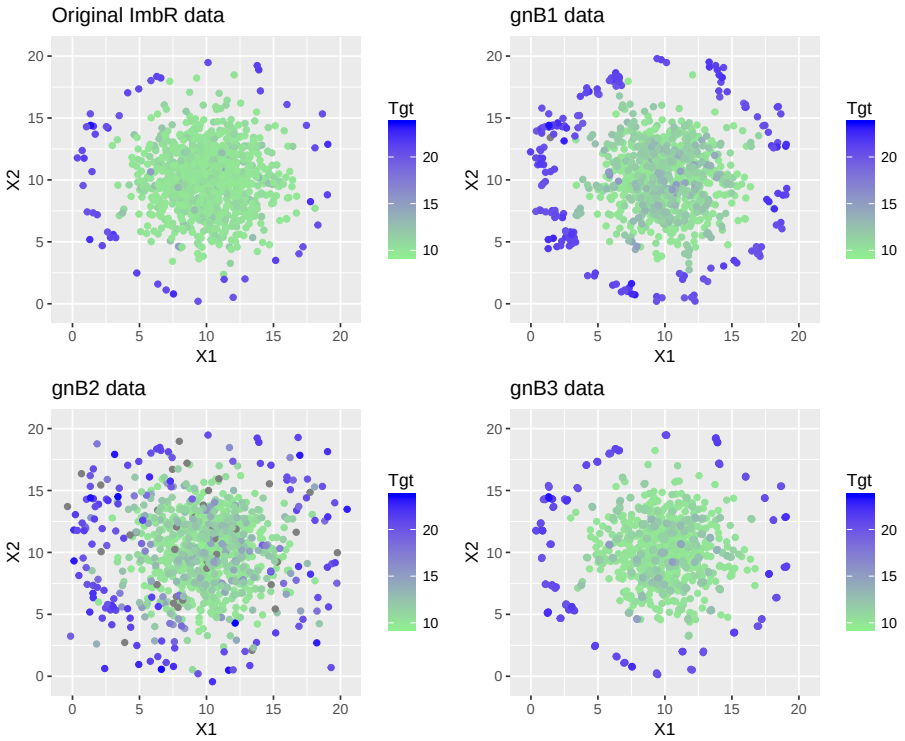

Figure 43: Target variable distribution on ImbR and data sets changed through Gaussian Noise strategy.

the previously described techniques. The user must also define the relevance threshold. Regarding the generation of examples it is required to specify the number of nearest neighbors to consider in smoteR algorithm. This is available through the parameter `k` and the default is set to 5. The user may then use the `C.perc` parameter to either express the percentages of under and over-sampling to use in each bump of relevance or to set which automatic method should be used for determining these percentages. Similarly to the other approaches, the automatic methods available are "balance" and "extreme" which estimate both where to apply the under/over-sampling and the corresponding percentages. The method "balance" changes the examples distribution by assigning roughly the same number of examples to each bump while the "extreme" method inverts the frequencies of each bump. Both methods approximately maintain the total number of examples. The parameter `repl` allows to select if the random under-sampling strategy is applied with repetition of examples or not. The user can also specify which distance function should be used for the nearest neighbors computation using the `dist` parameter.

The following examples illustrate how this method can be used.

```
# we will use the automatic method for defining the relevance function and will
# set the relevance threshold to 0.8
# this method splits the data set in two: a first range of values normal and less
# important and a second range with the interesting cases

# to check this, we can plot the relevance function obtained automatically
# as follows:

y <- sort(ImbR$Tgt)
phiF.args <- phi.control(y, method = "extremes", extr.type = "both")
y.phi <- phi(y, control.parms = phiF.args)

# plot the relevance function
plot(y, y.phi, type="l",
     ylab = expression(phi(y)), xlab = expression(y))

#add the relevance threshold to the plot
abline(h = 0.8, col = 3, lty = 2)
```

Figure 44 shows that we are considering two different bumps: a first bump with the normal and less important cases and a second bump with the rare and interesting cases. Thus, to address this problem we can do the following:

```
# we have two bumps: the first must be under-sampled and the second over-sampled.
# Thus, we can chose the following percentages:
thr.rel = 0.8
C.perc = list(0.1, 8)

# using these percentages and the relevance threshold of 0.8 with all
# the other parameters default values
# we can select any distance function
# because the data set contains only numeric features
mysm <- SmoteRegress(Tgt~., ImbR, thr.rel=thr.rel, dist="Manhattan", C.perc=C.perc)

# use the automatic method for obtaining a balanced data set
smB <- SmoteRegress(Tgt~., ImbR, thr.rel=thr.rel, dist="Manhattan", C.perc="balance")

# use the automatic method for invert the frequencies of the bumps
smE <- SmoteRegress(Tgt~., ImbR, thr.rel=thr.rel, dist="Manhattan", C.perc="extreme")
```

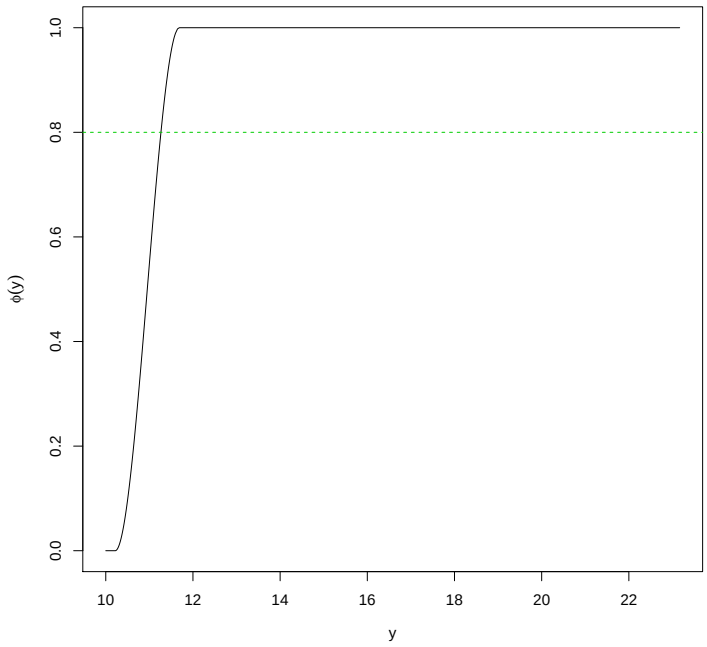

Figure 44: Relevance function obtained automatically for the ImbR data set

This strategy changes the examples distribution as shown in Figure 45.

Figure 46 shows the Original ImbR data and the new data sets pre-processed with SmoteR strategy.

We can also obtain the number of examples that each bump contains. Table 18 shows the examples distribution for the considered strategies.

	first bump	second bump
ImbR	849	151
mysm	84	1208
smB	499	500
smE	169	845

Table 18: Number of examples in each bump of relevance for different parameters of smoteR strategy.

In Figure 47 we can visualize the impact of these approaches on the examples distribution for each bump of relevance.

6.5 Importance Sampling

The Importance Sampling method is a new proposal whose main idea is to use the relevance function ($\phi()$) defined for a regression problem as a probability for resampling the examples combining over with under-sampling. This method

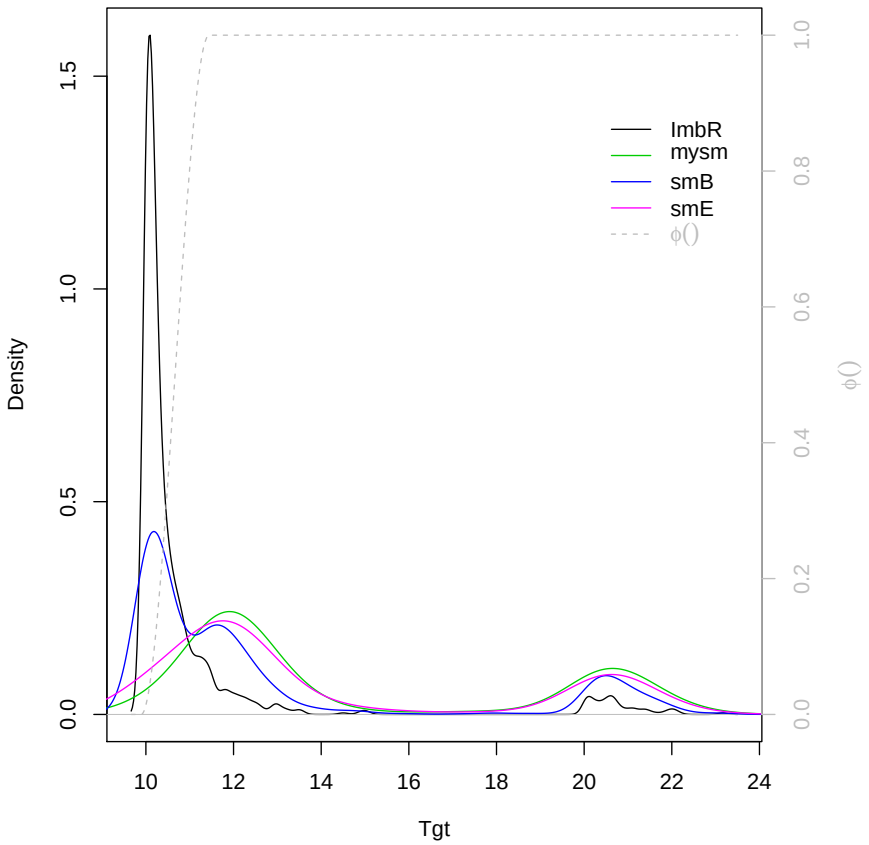

Figure 45: Relevance function and density of the target variable in the original and new data sets using smoteR strategy.

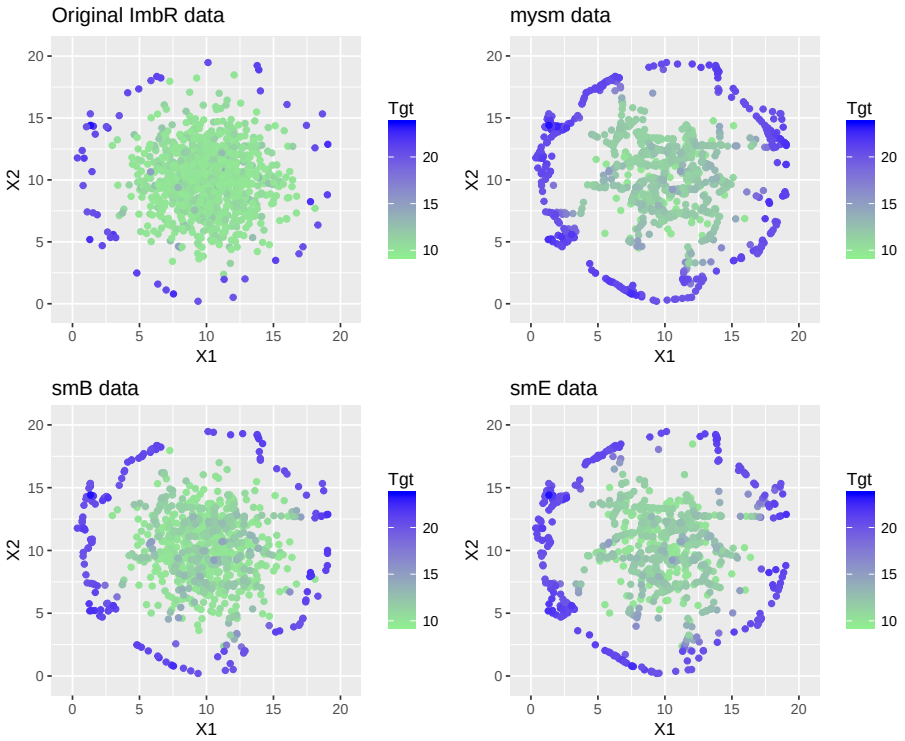

Figure 46: Target variable distribution on ImbR and data sets changed through SmoteR strategy.

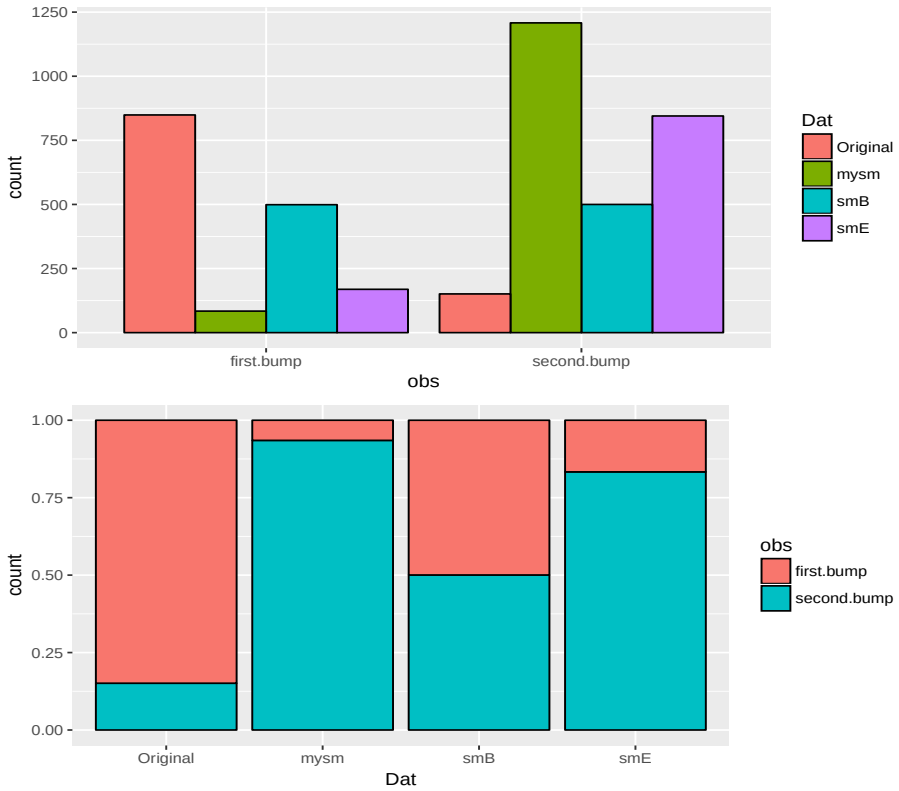

Figure 47: Impact in the distribution of examples for several parameters in smoteR strategy.

simply removes some of the examples and includes in the data set replicas of other existing examples. There is no generation of new synthetic examples. For the over-sampling strategy, replicas of examples are introduced by selecting examples according to the relevance function defined, i.e., the higher the relevance of an example, the higher is the probability of being selected as a new replica to include. The under-sampling strategy selects examples to remove according to the function $1 - \phi(y)$, i.e, the higher the relevance value of an example, the lower will be the probability of removing it.

This method includes two main behaviors which can be distinguished by the definition or not of a threshold on the relevance function. This means that, if the user decides to chose a relevance threshold the strategy will take this value into consideration with under and over-sampling being applied only on the defined bumps. However, if the user decides not to set a threshold on the relevance then over sampling and under-sampling strategies will also be applied but without a strict bound, i.e., there may be regions of the target variable values where under-sampling and over-sampling are performed together.

The strategy that depends on the definition of a relevance threshold, has the relevance bumps well defined. For these bumps, the user has several alternatives available through the `C.perc` parameter: the percentages of over and under-sampling to apply may be explicitly defined, or one of the options "balance" or "extreme" may be chosen. These last two option for the `C.perc` parameter allow to estimate the under and over-sampling percentages automatically. The option "balance" allows to obtain a balanced data set across the different existing bumps. The "extreme" option will produce a new data set with the examples frequencies in the bumps inverted. In this setting, there is no range of the target variable where both under and over-sampling techniques are applied.

As previously mentioned, there is the possibility of not defining a relevance threshold, and simply use the relevance function to decide which examples should be replicated and which should be removed. In this case, the user does not set a threshold on the relevance, but he can define the importance that over and under-sampling should have. In this case, the `C.perc` parameter is ignored and two other parameters(`U` and `O`) are considered instead. The parameters `U` and `O` allow the user to define (in a $[0, 1] scale$) the importance that the under/over-sampling have, i.e, these parameters assign a weight to the two methods. The higher is `O` parameter, the higher is the number of replicas selected. In a similar way, the higher is `U` parameter the higher is the number of examples removed.

The function `ImpSampRegress` allows the use of Importance Sampling strategy. Some examples on how to use this approach are provided next.

```
# relevance function estimated automatically has two bumps
# using the strategy with threshold definition
C.perc=list(0.2,6)
myIS <- ImpSampRegress(Tgt~., ImbR, thr.rel=0.8,C.perc=C.perc)
ISB <- ImpSampRegress(Tgt~., ImbR, thr.rel=0.8, C.perc="balance")
ISE <- ImpSampRegress(Tgt~., ImbR, thr.rel=0.8, C.perc="extreme")
```

Figures 48 and 49 show the impact on the density and distribution of the examples for the new data sets obtained with Importance Sampling strategy. Figure 50 shows a binarized version of the previous data sets considering the value 19 as the threshold between the rare/important cases and the normal/unimportant cases.

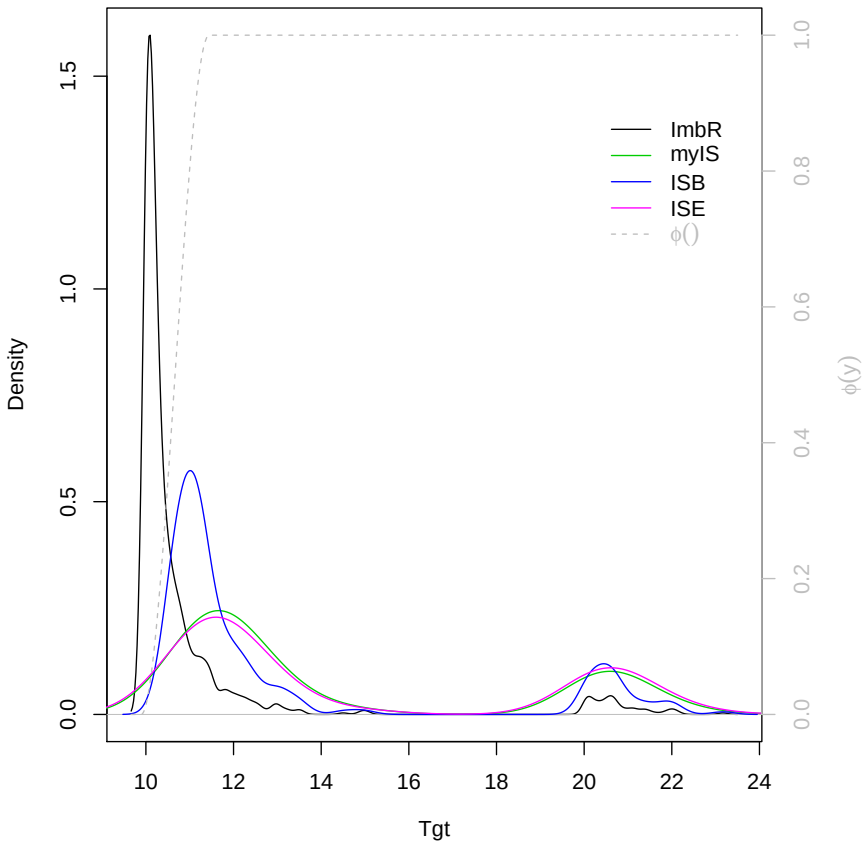

Figure 48: Relevance function and density of the target variable in the original and new data sets using Importance Sampling strategy.

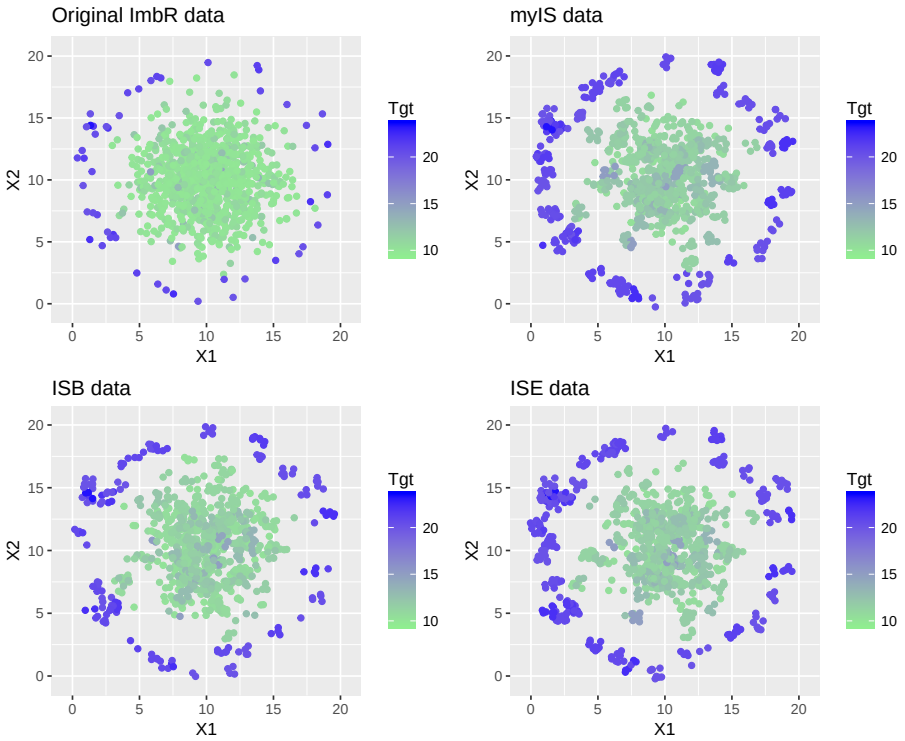

Figure 49: Target variable distribution on ImbR and data sets changed through Importance Sampling strategy.

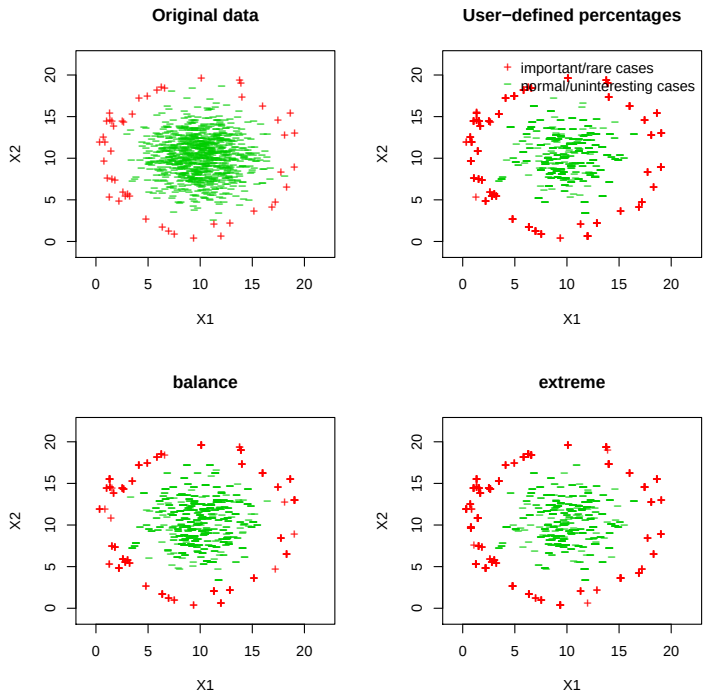

Figure 50: Impact of Importance Sampling strategy in a binarized version of the data sets considering the value 19 as the threshold between rare and normal cases.

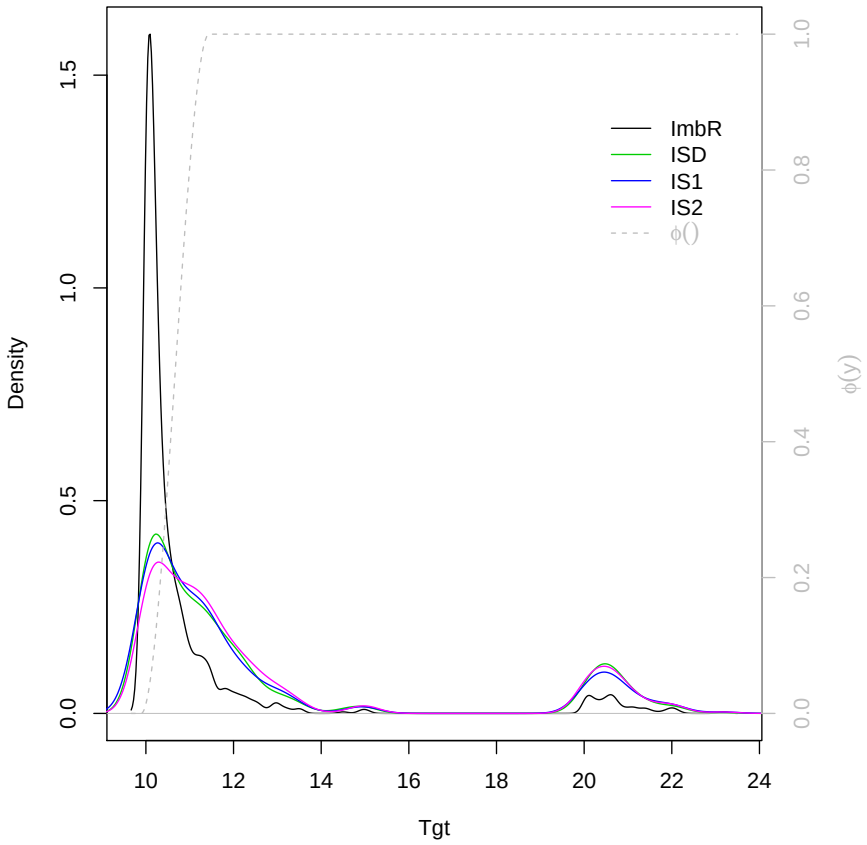

Figure 51: Relevance function and density of the target variable in the original and new data sets using Importance Sampling strategy.

We now provide some examples of the use of this strategy without the definition of a relevance threshold.

```
# relevance function is also estimated automatically
# the default is not to use a relevance threshold and to assign equal
# importance to under and over-sampling, i.e., U=0.5 and O=0.5
ISD <- ImpSampRegress(Tgt~., ImbR)
IS1 <- ImpSampRegress(Tgt~., ImbR, U=0.9, O=0.2)
IS2 <- ImpSampRegress(Tgt~., ImbR, U=0.5, O=0.8)
```

Figures 51 and 52 show the impact on the density and distribution of the examples for the new data sets obtained with Importance Sampling strategy.

7 Distance Functions

In this section we briefly explain the different distance functions implemented, which can be used for calculating the neighbors of the examples along several strategies for classification or regression tasks. The implementation of these

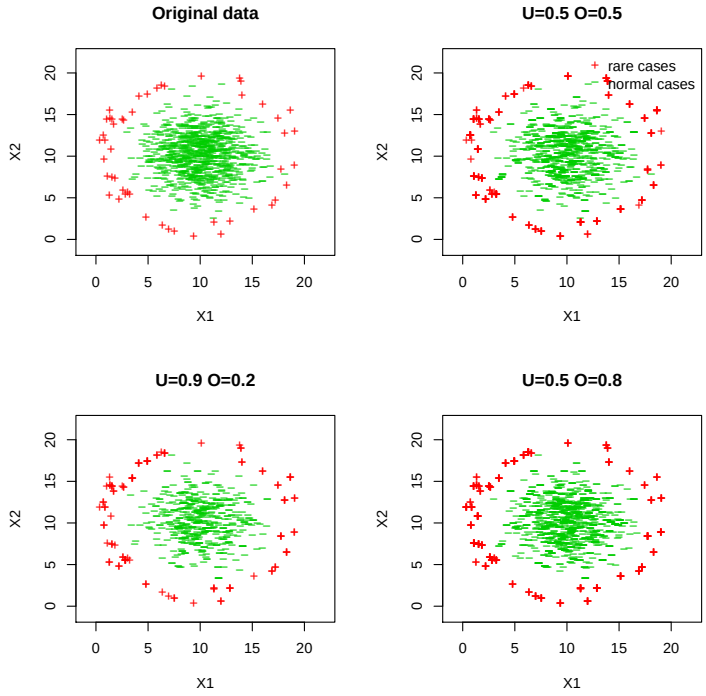

Figure 52: Impact of Importance Sampling strategy in a binarized version of the data sets with the value 19 as the threshold between rare and normal cases.

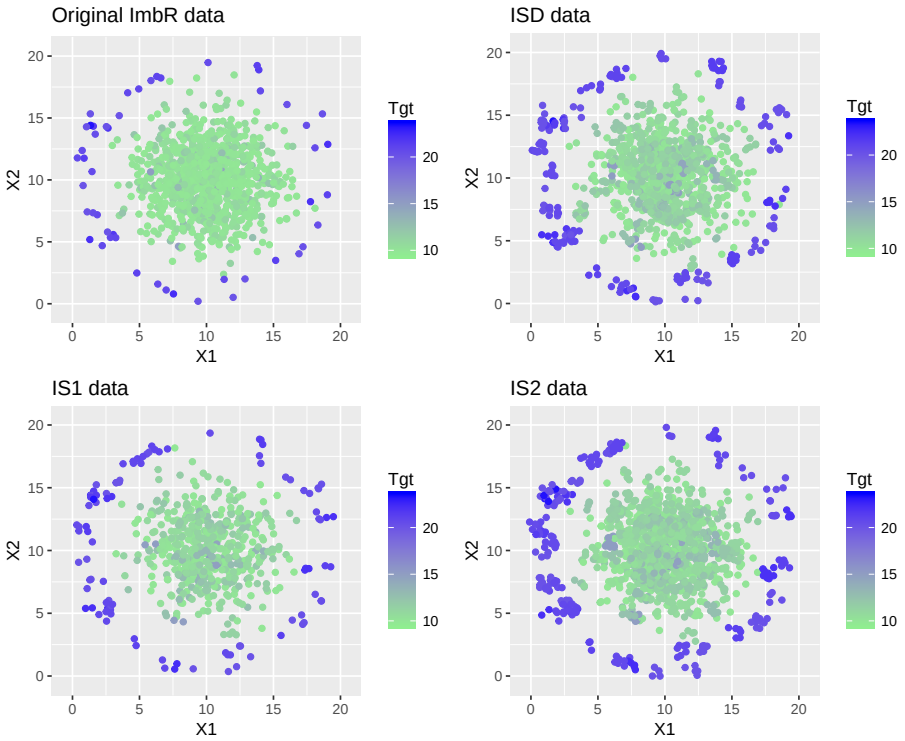

Figure 53: Target variable distribution on ImbR and data sets changed through Importance Sampling strategy.

functions was motivated by the inclusion in UBL of several methods which depend on the nearest neighbors computation. Although several efficient tools exist for evaluating the nearest neighbors, they are mostly limited to the use of the Euclidean distance. In this context, restricting the user to the use of the Euclidean distance can be a limitation, namely because several data sets include nominal features which can and should also be considered in the neighbors computation. In fact, all the features contained in the data set, whether nominal or numeric, should be taken into account when computing the nearest neighbors. Thus, in order to avoid the restriction of computing nearest neighbors based only on the data set numeric features we have implemented several possible measures which can be used for data sets containing only nominal or numeric features or simultaneously both types. By the implementation of several distance functions, we aim at providing an increased flexibility for computing the nearest neighbors while ensuring that no feature information is wasted.

Several distance measures exist which can deal only with numeric or nominal features or can integrate both types in the distance evaluation. Distance functions such as `Canberra`, `Euclidean` or `Chebyshev` are able to deal solely with numeric attributes while the `Overlap` measure handles only nominal features. Other measures such as `HEOM` or `HVDM` try to use both types of features.

We now briefly describe the distance functions implemented in this package. We begin with the distance functions suitable for data sets with only numeric features. Let us suppose x and y are two examples of a data set with m features. The well-known Euclidean distance can be computed as shown in Equation 1. The Manhattan distance, also known as city-block distance or taxicab metric, may be calculated with Equation 2.

$$D(x,y) = \sqrt{\sum_{i=1}^{m}(x_i - y_i)^2} \tag{1}$$

$$D(x,y) = \sum_{i=1}^{m}|x_i - y_i| \tag{2}$$

A generalization of these distance functions is obtained with the Minkowsky distance (cf. Equation 3). In this case, by setting r to 1 or 2 we can obtain respectively the Manhattan and Euclidean distance functions.

$$D(x,y) = \left(\sum_{i=1}^{m}|x_i - y_i|^r\right)^{\frac{1}{r}} \tag{3}$$

The Canberra distance, defined in Equation 4, and the Chebyshev distance (Equation 5) are also functions which can be applied to evaluate the distance between examples described only by numeric features.

$$D(x,y) = \sum_{i=1}^{m}\frac{|x_i - y_i|}{|x_i| + |y_i|} \tag{4}$$

$$D(x,y) = \max_{i=1}^{m}|x_i - y_i| \tag{5}$$

All the previous distance functions can be used in UBL for computing the nearest neighbors. After selecting an appropriate approach to apply on a data

set, it is only necessary to set the parameter `dist` of the approach to the desired distance function and the `p` parameter if it is a Minkowsky distance. We illustrate this in the next example.

```
dat <- iris[-c(91:125),]
# using the default of smote to invert the frequencies of the data set
set.seed(123)
sm.Eu <- SmoteClassif(Species~., dat, dist="Euclidean",
                      C.perc="extreme", k=3)
set.seed(123)
sm.Man1 <- SmoteClassif(Species~., dat, dist="Manhattan",
                        C.perc="extreme", k=3)
set.seed(123)
sm.Man2 <- SmoteClassif(Species~., dat, dist="p-norm", p=1,
                        C.perc="extreme", k=3)
set.seed(123)
sm.5norm <- SmoteClassif(Species~., dat, dist="p-norm", p=5,
                         C.perc="extreme", k=3)
set.seed(123)
sm.Cheb <- SmoteClassif(Species~., dat, dist="Chebyshev",
                        C.perc="extreme", k=3)
set.seed(123)
sm.Canb <- SmoteClassif(Species~., dat, dist="Canberra",
                        C.perc="extreme", k=3)
```

The impact of using these distance functions with smote strategy can be visualized in Figure 54.

All the previously described metrics do not perform any type of normalization. This step, if wanted, should be performed previously by the user.

Regarding nominal attributes, a distance function which is suitable for handling this type of variables is the overlap measure, which is defined in Equation 6.

$$overlap(x, y) = \begin{cases} 1 & \text{if } x \neq y \\ 0 & \text{if } x = y. \end{cases} \tag{6}$$

This distance function can be used in strategies that require the computation of nearest neighbors as follows:

```
# build a data set with all nominal features
library(DMwR)
data(algae)
clean.algae <- algae[complete.cases(algae),1:3]

# speed is considered the target class
summary(clean.algae)

##     season       size       speed
##  autumn:36   large :42   high  :76
##  spring:48   medium:83   low   :31
##  summer:43   small :59   medium:77
##  winter:57

ndat1 <- ENNClassif(speed~., clean.algae, dist="Overlap",  Cl=c("high", "medium"))
ndat2 <- ENNClassif(speed~., clean.algae, dist="Overlap",  Cl="all")

#all the smaller classes are the most important
ndat3 <- NCLClassif(speed~., clean.algae, dist="Overlap",  Cl="smaller")
```

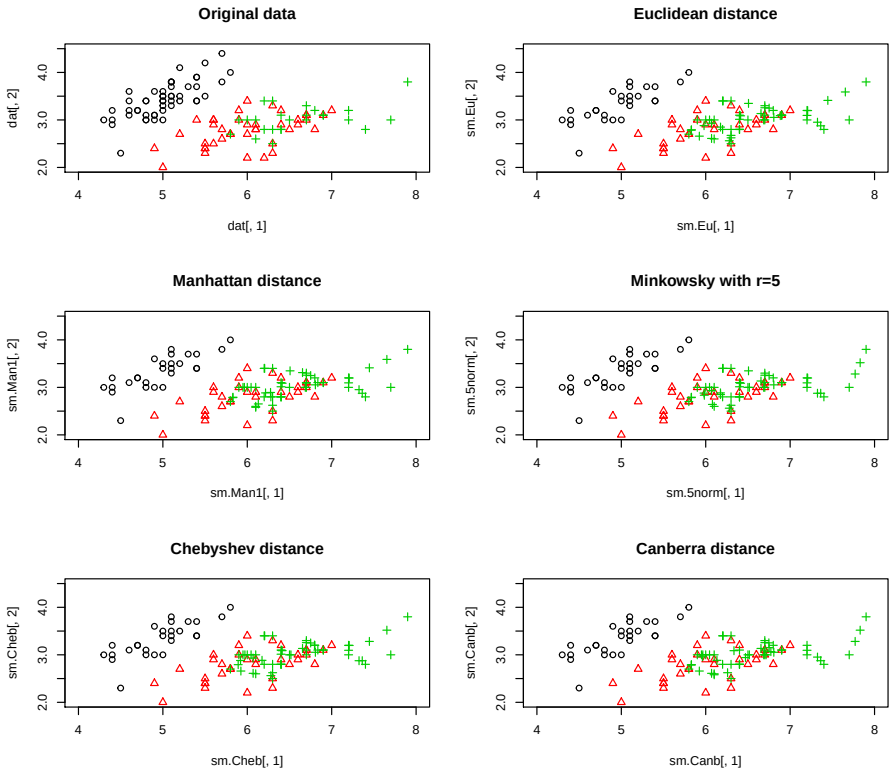

Figure 54: Impact of using different distance functions with smote strategy.

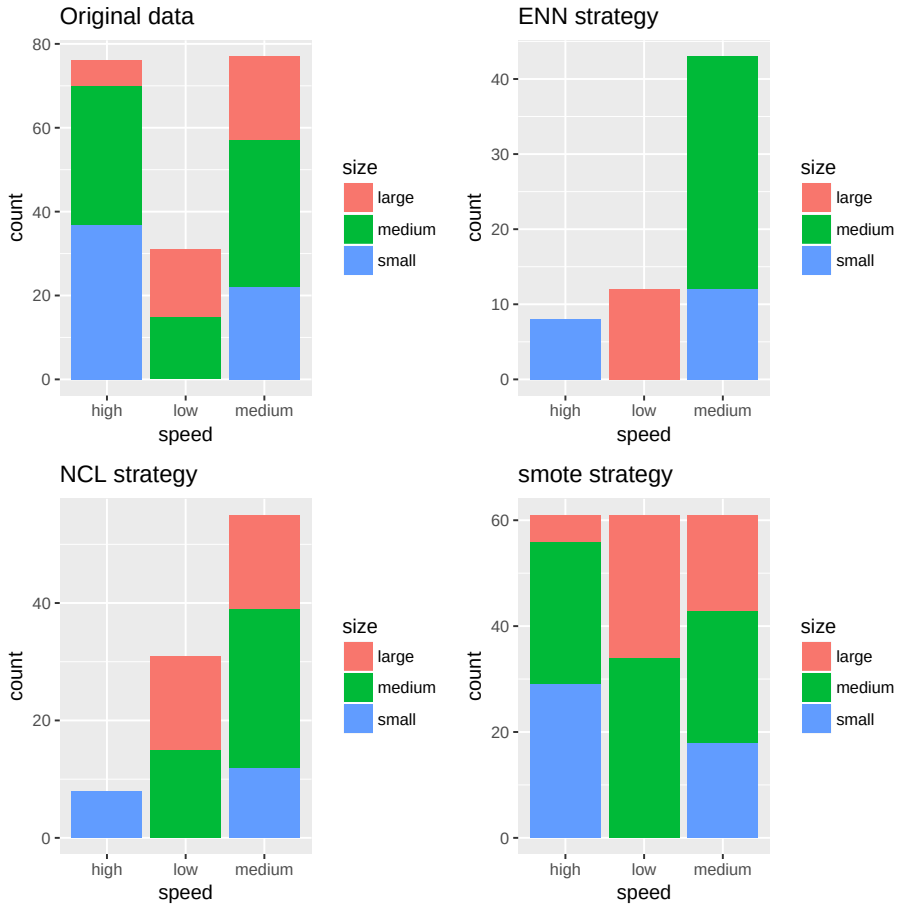

Figure 55: Using Overlap distance function with different strategies on a data set with only nominal features.

```
# the most important classes are "high" and "low"
ndat4 <- NCLClassif(speed~., clean.algae, dist="Overlap",  Cl=c("high", "low"))

ndat5 <- SmoteClassif(speed~., clean.algae, dist="Overlap", C.perc="balance")
```

Figure 55 shows the impact of using the overlap distance function, with several different strategies, on a data set consisting of only nominal variables.

To evaluate the distance between examples described by nominal and numeric variables a simple adaptation of the previous distance functions can be performed. The Heterogeneous Euclidean-Overlap Metric function (HEOM) is a popular solution for these situations. Equations 7 and 8 describe how this distance is computed.

$$HEOM(x,y) = \sqrt{\sum_{a=1}^{m} d_a^2(x_a, y_a)} \tag{7}$$

$$\text{where } d_a(x,y) = \begin{cases} 1 & \text{if } x \vee y \text{ are unknown, else} \\ overlap(x,y) & \text{if } a \text{ is nominal, else} \\ \frac{|x-y|}{range_a} & \end{cases} \tag{8}$$

where $range_a = max_a - min_a$

```
# build a data set with nominal and numeric features
library(DMwR)
data(algae)
clean.algae <- algae[complete.cases(algae),1:5]

# speed is the target class
summary(clean.algae)

##     season       size       speed         mxPH            mnO2
##  autumn:36   large :42   high  :76   Min.   :7.000   Min.   : 1.500
##  spring:48   medium:83   low   :31   1st Qu.:7.777   1st Qu.: 7.675
##  summer:43   small :59   medium:77   Median :8.100   Median : 9.750
##  winter:57                           Mean   :8.078   Mean   : 9.019
##                                      3rd Qu.:8.400   3rd Qu.:10.700
##                                      Max.   :9.500   Max.   :13.400

enn <- ENNClassif(speed~., clean.algae, dist="HEOM",  Cl="all", k=5)[[1]]
#consider all the smaller classes as the most important
ncl <- NCLClassif(speed~., clean.algae, dist="HEOM",  Cl="smaller")
sm <- SmoteClassif(speed~., clean.algae, dist="HEOM", C.perc="balance")
```

In Figure 56 we can observe the impact of using the HEOM distance function with several strategies.

Other proposals, such as the Heterogeneous Value Difference Metric (HVDM), were tested for handling both nominal and numeric features. The HVDM uses the notion of Value Distance Metric (VDM) which was introduced by [SW86] to address the distance computation with nominal variables. The VDM metric is described in Equation 9.

$$VDM_a(x,y) = \sum_{c=1}^{C} \left| \frac{N_{a,x,c}}{N_{a,x}} - \frac{N_{a,y,c}}{N_{a,y}} \right|^q \tag{9}$$

where,
a is the nominal attribute under consideration;
C is the number of classes existing on the data set;
q is a constant;
$N_{a,x,c}$ represents the number of examples which have value x for the feature a and class label c;
$N_{a,x}$ is the number of examples that have value x for the feature a.

The HVDM distance function was proposed by [WM97] and its definition, presented in Equations 10and 11, is similar to the HEOM.

$$HVDM(x,y) = \sqrt{\sum_{a=1}^{m} d_a^2(x_a, y_a)} \tag{10}$$

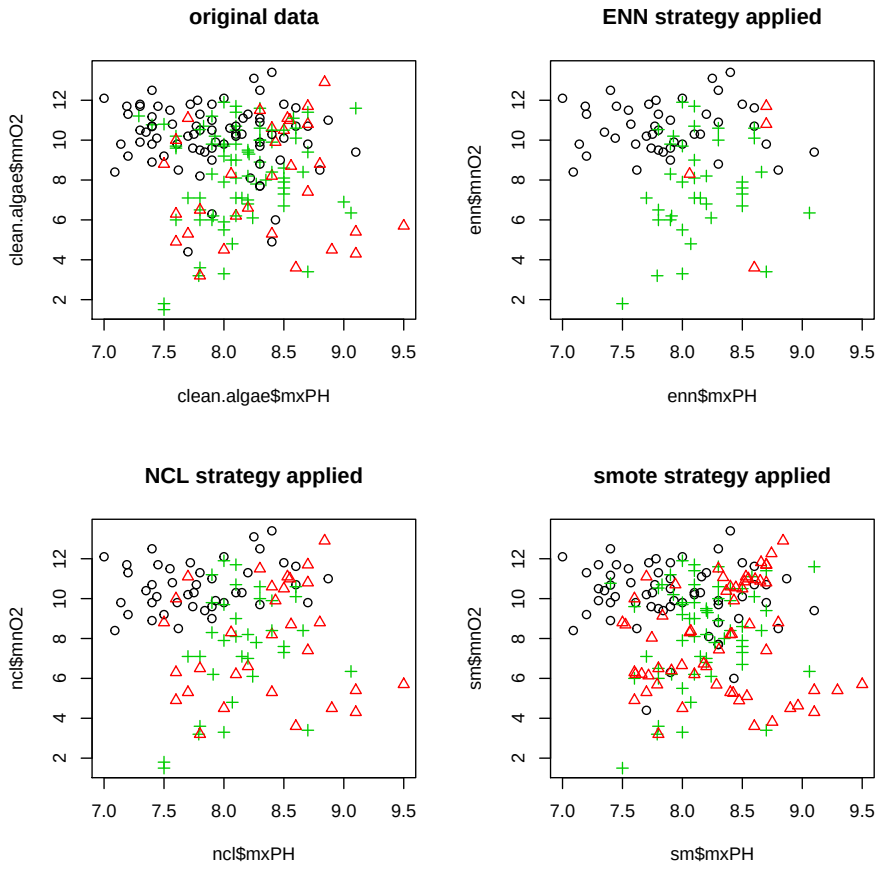

Figure 56: Using HEOM distance function with different strategies on a data set with both nominal and numeric features.

$$\text{where } d_a(x,y) = \begin{cases} 1 & \text{if } x \vee y \text{ are unknown, otherwise} \\ norm - vdm_a(x,y) & \text{if } a \text{ is nominal} \\ norm - diff_a(x,y) & \text{if } a \text{ is numeric} \end{cases} \tag{11}$$

The HVDM distance function uses a normalized version of the absolute value of the difference between two examples for the numeric attributes (Equation 13) and uses for the nominal attributes an also normalized version of the VDM measure for the nominal attributes (Equation 12) .

$$norm - vdm_a(x,y) = \sqrt{VDM_a(x,y)} = \sqrt{\sum_{c=1}^{C} \left| \frac{N_{a,x,c}}{N_{a,x}} - \frac{N_{a,y,c}}{N_{a,y}} \right|^2} \tag{12}$$

$$norm - diff_a(x,y) = \frac{|x-y|}{4\sigma_a} \tag{13}$$

Regarding Equation 12, several normalization of the VDM measure were proposed and tested in [WM97]. The version presented here and implemented in `UBL` was the one that achieved the best performance. We also highlight that the distance function proposed for the numeric attributes uses a different normalization which relies on the standard deviation of of each attribute σ_a.

The HVDM distance can be used simply by setting the `dist` parameter to "HVDM". Although it is a function suitable for both nominal and numeric, if the data set provided contains only one type of attributes only the corresponding distance will be used.

```
# build a data set with both nominal and numeric features
library(DMwR)
data(algae)
clean.algae <- algae[complete.cases(algae),c(1:6)]

# speed is considered the target class
summary(clean.algae)

##     season       size       speed         mxPH            mnO2             Cl
##  autumn:36   large :42   high  :76   Min.   :7.000   Min.   : 1.500   Min.   :  0.80
##  spring:48   medium:83   low   :31   1st Qu.:7.777   1st Qu.: 7.675   1st Qu.: 11.85
##  summer:43   small :59   medium:77   Median :8.100   Median : 9.750   Median : 35.08
##  winter:57                           Mean   :8.078   Mean   : 9.019   Mean   : 44.88
##                                      3rd Qu.:8.400   3rd Qu.:10.700   3rd Qu.: 58.52
##                                      Max.   :9.500   Max.   :13.400   Max.   :391.50

dat1 <- SmoteClassif(speed~., clean.algae, dist="HVDM", C.perc="extreme")

dat2 <- NCLClassif(speed~., clean.algae, k=3, dist="HVDM", Cl="smaller")

dat3 <- TomekClassif(speed~., clean.algae, dist="HVDM", Cl="all", rem="both")
```

Figure 57 shows the result of applying HVDM distance function for several different strategies, on a data set consisting of numeric and nominal features.

In Figure 58 the impact of smote strategy applied with different distance functions on a data set can be observed.

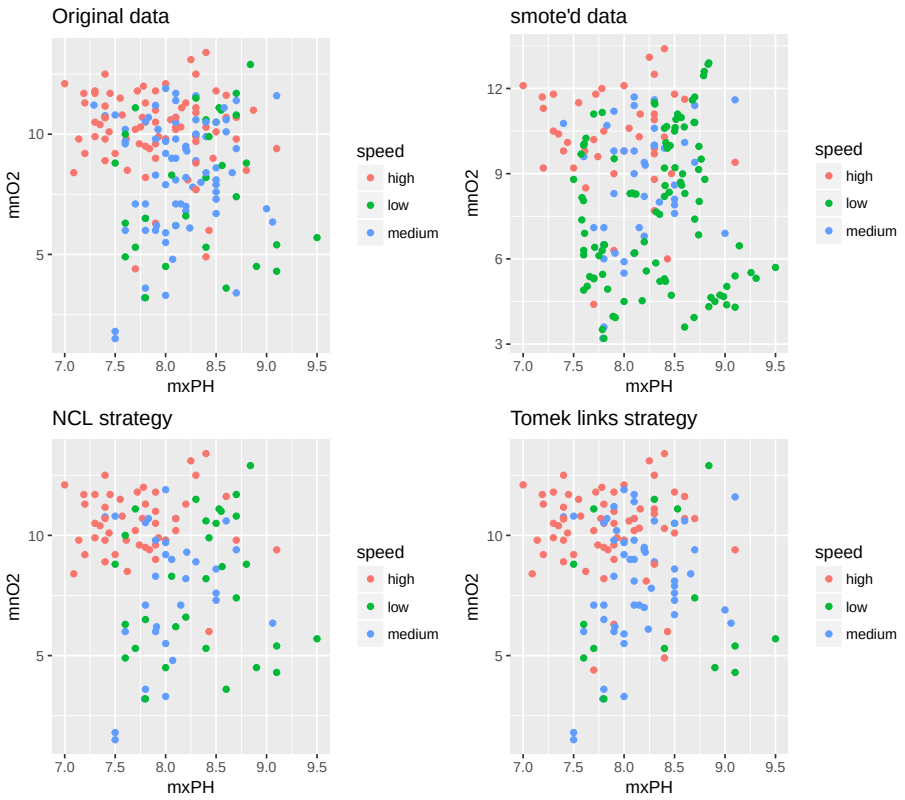

Figure 57: Using HVDM distance function with different strategies.

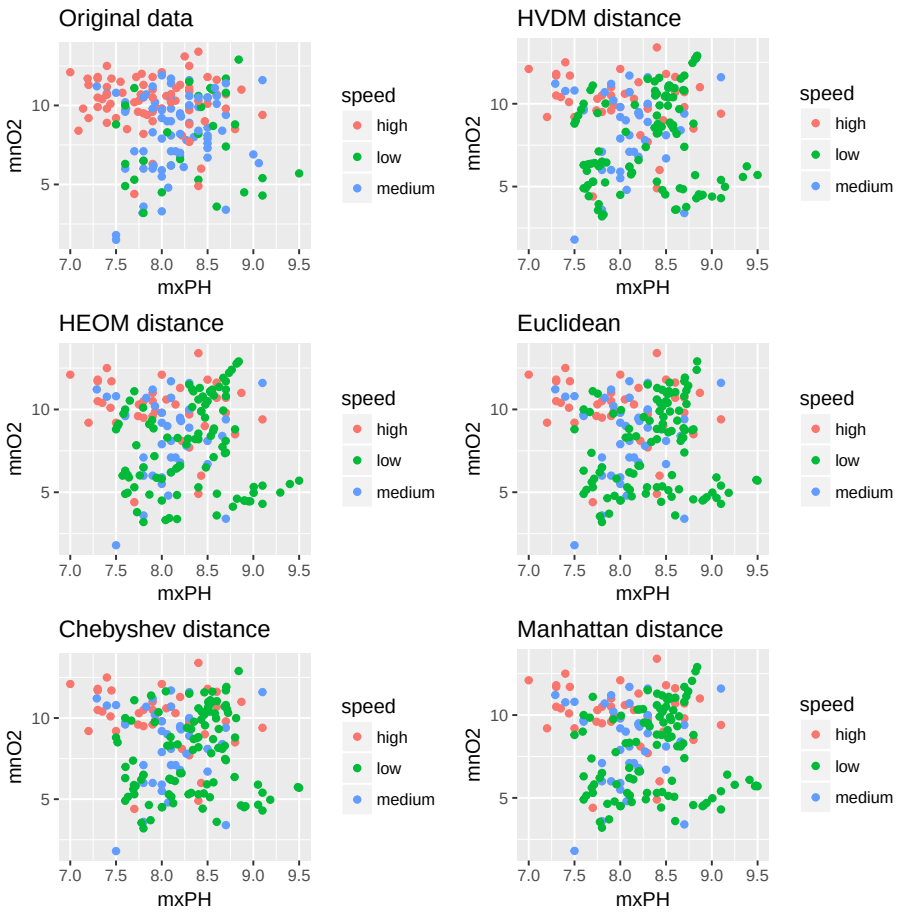

Figure 58: Using different distance functions with smote strategy.

8 Conclusions

We have presented package `UBL` that aims at dealing with utility-based predictive tasks. This package offers several methods for multiclass and regression problems. The approaches implemented are pre-processing methods for changing the target variable distribution. This change in the data set is performed with the goal of incorporating the user preference bias. The use of pre-processing methods that change the original data set force the learning algorithms to focus on the most relevant cases for the user.

The existing strategies for dealing with utility-based problems as a pre-processing step present the advantage of allowing the use of any standard learning algorithm without changing it. Moreover, these methods do not compromise the interpretability of the models used. As possible disadvantages we must point the difficulty of determining the ideal distribution of the domain. In fact, a perfectly balanced distribution of examples is not always the solution that provides the best results.

`UBL` package is a versatile tool for tackling problems at a pre-processing level that have some information regarding the domain. This package extends some methods previously developed for binary classification to a multiclass setting and also allows to deal with regression problems with multiple important regions. It offers to the user the possibility of manually defining how the data set should be changed for a selected pre-processing approach. Moreover, for each implemented approach it also enables the use of automatic methods for estimating the changes to apply. These automatic methods use the original domain disribution for assigning more importance to the least represented examples, a commom setting when learning from imbalanced domains.

References

[BPM04] Gustavo EAPA Batista, Ronaldo C Prati, and Maria Carolina Monard. A study of the behavior of several methods for balancing machine learning training data. *ACM SIGKDD Explorations Newsletter*, 6(1):20–29, 2004.

[BTR15] Paula Branco, Luis Torgo, and Rita Ribeiro. A survey of predictive modelling under imbalanced distributions. *arXiv preprint arXiv:1505.01658*, 2015.

[CBHK02] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: Synthetic minority over-sampling technique. *JAIR*, 16:321–357, 2002.

[Har68] P. E. Hart. The condensed nearest neighbor rule. *IEEE Transactions on Information Theory*, 14:515–516, 1968.

[HBGL08] Haibo He, Yang Bai, Edwardo A Garcia, and Shutao Li. Adasyn: Adaptive synthetic sampling approach for imbalanced learning. In *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, pages 1322–1328. IEEE, 2008.

[KM97] M. Kubat and S. Matwin. Addressing the curse of imbalanced training sets: One-sided selection. In *Proc. of the 14th Int. Conf. on Machine Learning*, pages 179–186. Morgan Kaufmann, 1997.

[Lau01] Jorma Laurikkala. *Improving identification of difficult small classes by balancing class distribution*. Springer, 2001.

[Lee99] Sauchi Stephen Lee. Regularization in skewed binary classification. *Computational Statistics*, 14(2):277, 1999.

[Lee00] Sauchi Stephen Lee. Noisy replication in skewed binary classification. *Computational statistics & data analysis*, 34(2):165–191, 2000.

[LFG+13] Victoria López, Alberto Fernández, Salvador García, Vasile Palade, and Francisco Herrera. An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics. *Information Sciences*, 250:113–141, 2013.

[Rib11] R Ribeiro. *Utility-based regression*. PhD thesis, PhD thesis, Dep. Computer Science, Faculty of Sciences-University of Porto, 2011.

[RwcfLT14] Rita P. Ribeiro and with contributions from Luis Torgo. *uba: Utility-based Algorithms*, 2014. R package version 0.7.6.

[SW86] Craig Stanfill and David Waltz. Toward memory-based reasoning. *Communications of the ACM*, 29(12):1213–1228, 1986.

[Tom76] Ivan Tomek. Two modifications of cnn. *IEEE Trans. Syst. Man Cybern.*, (11):769–772, 1976.

[TR07] Luis Torgo and Rita Ribeiro. Utility-based regression. In *Knowledge Discovery in Databases: PKDD 2007*, pages 597–604. Springer, 2007.

[TRPB13] Luís Torgo, Rita P Ribeiro, Bernhard Pfahringer, and Paula Branco. Smote for regression. In *Progress in Artificial Intelligence*, pages 378–389. Springer, 2013.

[Wil72] Dennis L Wilson. Asymptotic properties of nearest neighbor rules using edited data. *Systems, Man and Cybernetics, IEEE Transactions on*, (3):408–421, 1972.

[WM97] D. Randall Wilson and Tony R. Martinez. Improved heterogeneous distance functions. *JAIR*, 6:1–34, 1997.