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

April 28, 2016

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 0.0.4.

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 presents a simple example to show how `UBL` package can be used and its impact on the models performance. Sections 4 and 5 describe with detail each method currently implemented in

UBL for classification and regression tasks. Section 6 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 7 concludes this document.

2 Package Installation Guidelines

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

```r
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 0.0.4.

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:

```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:

```r
library(UBL)
```

3 A Simple Illustrative Example

Let us consider a classification task with 3 classes with different frequency. For illustration purposes, we will use the well-known iris data set with some examples removed to simulate a distribution with a rare class. Let us also suppose that the most important class for the user is this rare class (in our example, the virginica class).

```r
library(UBL)  # Loading our infra-structure
library(e1071) # packge containing the svm we will use
data(iris)          # The data set we are going to use
# transforming into an imbalanced problem
dat <- iris[-c(91:125), c(1, 2, 5)]
table(dat$Species)

##
##     setosa versicolor  virginica
##         50         40         25
```

If we now train a svm in a sample of 70% of our data, we obtain the following:

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

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

##
## preds        setosa versicolor virginica
##   setosa         14          0         0
##   versicolor      0         14         5
##   virginica       0          2         0
```

Clearly, the model presents a poor performance on least represented class, the virginica class. However, for the most common class, the setosa the learner always predicts correctly.

Now, we can try to apply a strategy for dealing with utility-based problems, and check again the models performance.

```
# change the train data by applying the smote strategy
newtrain <- SmoteClassif(Species~., train, C.perc="balance")

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

##
## preds        setosa versicolor virginica
##   setosa         14          0         0
##   versicolor      0         13         2
##   virginica       0          3         3
```

We can observe that the least represented class, virginica, now presents an improved result. If the previous model was unable to correctly classify any examples with class label of virginica, now there are three virginica cases which have a correct prediction.

We can also try a simple random under-sampling method:

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

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

##
## preds        setosa versicolor virginica
##   setosa         14          0         0
##   versicolor      0         13         2
##   virginica       0          3         3
```

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

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

- 4.1: Random Under-sampling
- 4.2: Random Over-sampling
- 4.3: Importance Sampling
- 4.4: Tomek Links
- 4.5: Condensed Nearest Neighbors
- 4.6: One-sided Selection
- 4.7: Edited Nearest Neighbors
- 4.8: Neighborhood Cleaning Rule
- 4.9: Gaussian Noise Introduction
- 4.10: Smote Algorithm

4.1 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) # packge containing the svm we will use
data(iris)                       # The data set we are going to use
# transforming into a multiclass imbalanced problem
dat <- iris[-c(91:125), c(1, 2, 5)]
```

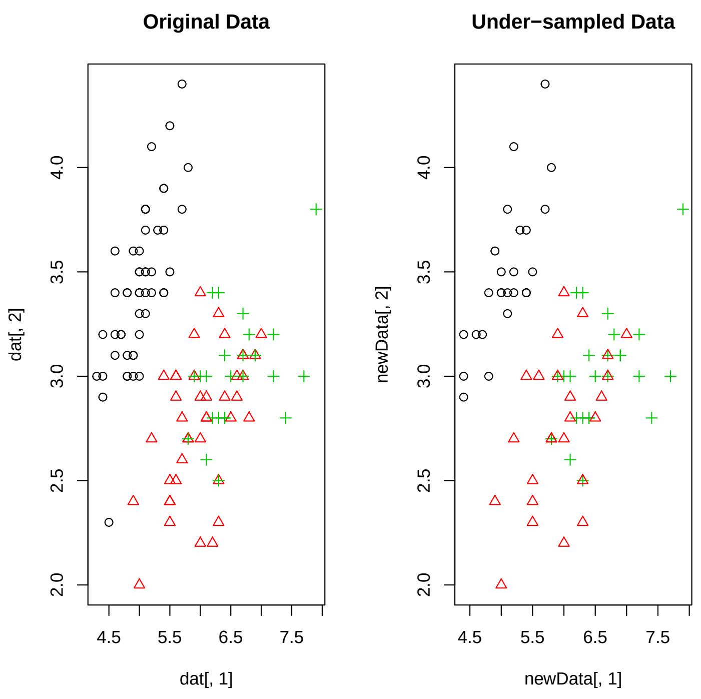


Figure 1: The impact of random under-sampling strategy.

```
# check the unbalanced data
table(dat$Species)

##
##     setosa versicolor  virginica
##         50         40         25

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

newData <- RandUnderClassif(Species ~ ., dat)
table(newData$Species)

##
##     setosa versicolor  virginica
##         25         25         25
```

Figure 1 shows the impact of this strategy in the examples distribution. Another example with the iris data set:

```
  RUmy.ir <- RandUnderClassif(Species~., dat, list(setosa=0.3, versicolor=0.7))
  RUB.ir <- RandUnderClassif(Species~., dat, "balance")
  RUE.ir <- RandUnderClassif(Species~., dat, "extreme")
```

The impact of the strategies on the number of examples in each class of the data set are in Figure2.

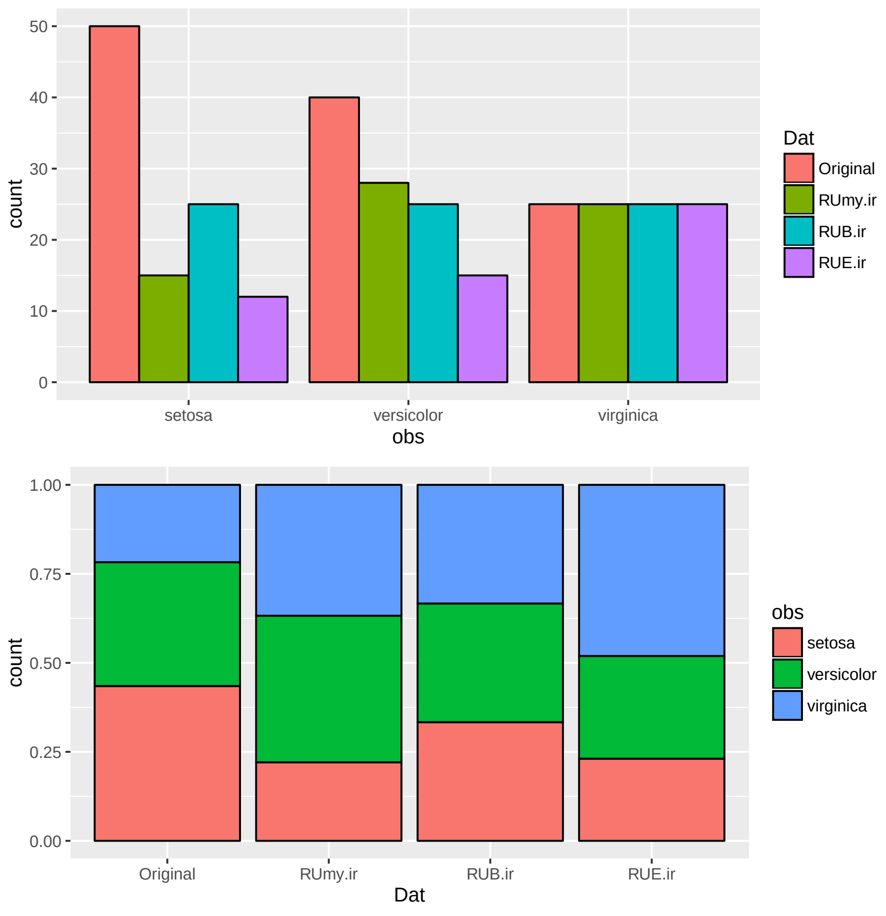


Figure 2: Random Under-sampling strategy for different parameters values.

	setosa	versicolor	virginica
Original	50	40	25
RUmy.ir	15	28	25
RUB.ir	25	25	25
RUE.ir	12	15	25

Table 1: Number of examples in each class for different parameters of random under-sampling strategy.

4.2 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 600% of examples in the
## virginica class
RO.U1<- RandOverClassif(Species ~ ., dat,
                        C.perc=list(virginica=5))
RO.U2<- RandOverClassif(Species ~ ., dat,
                        C.perc=list(versicolor=4, virginica=2.5))
RO.B <- RandOverClassif(Species ~ ., dat, C.perc="balance")
RO.E <- RandOverClassif(Species ~ ., dat, C.perc="extreme")
```

	setosa	versicolor	virginica
Original	50	40	25
RO.U1	50	40	125
RO.U2	50	160	62
RO.B	50	50	50
RO.E	50	62	100

Table 2: Number of examples in each class for different Random over-sampling parameters.

Figure 3 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.

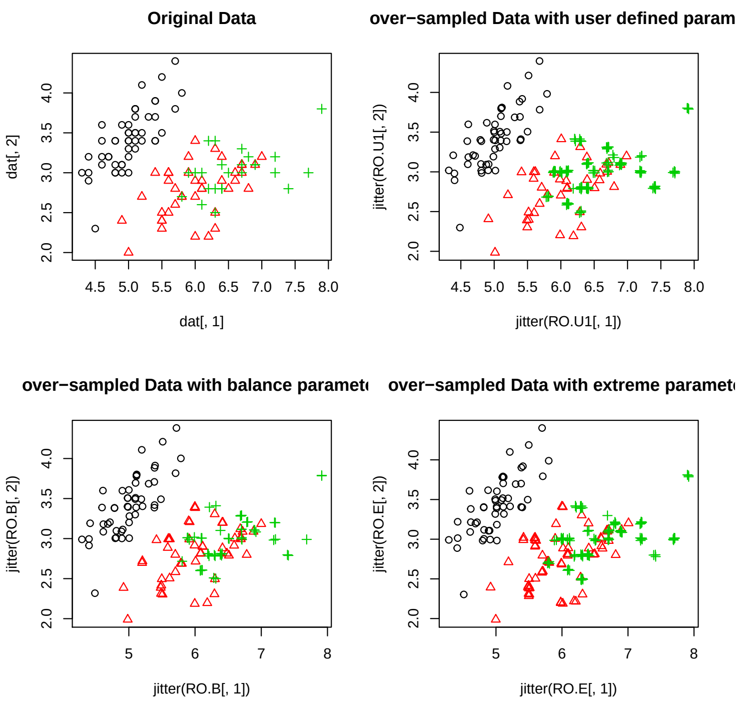


Figure 3: The impact of random over-sampling Strategy.

Figure 4 shows the impact of this strategy on the number of examples in the data set.

4.3 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.

```
# reuse the same unbalanced data obtained from iris data set
table(dat$Species)

##
##     setosa versicolor  virginica
##         50         40         25
```

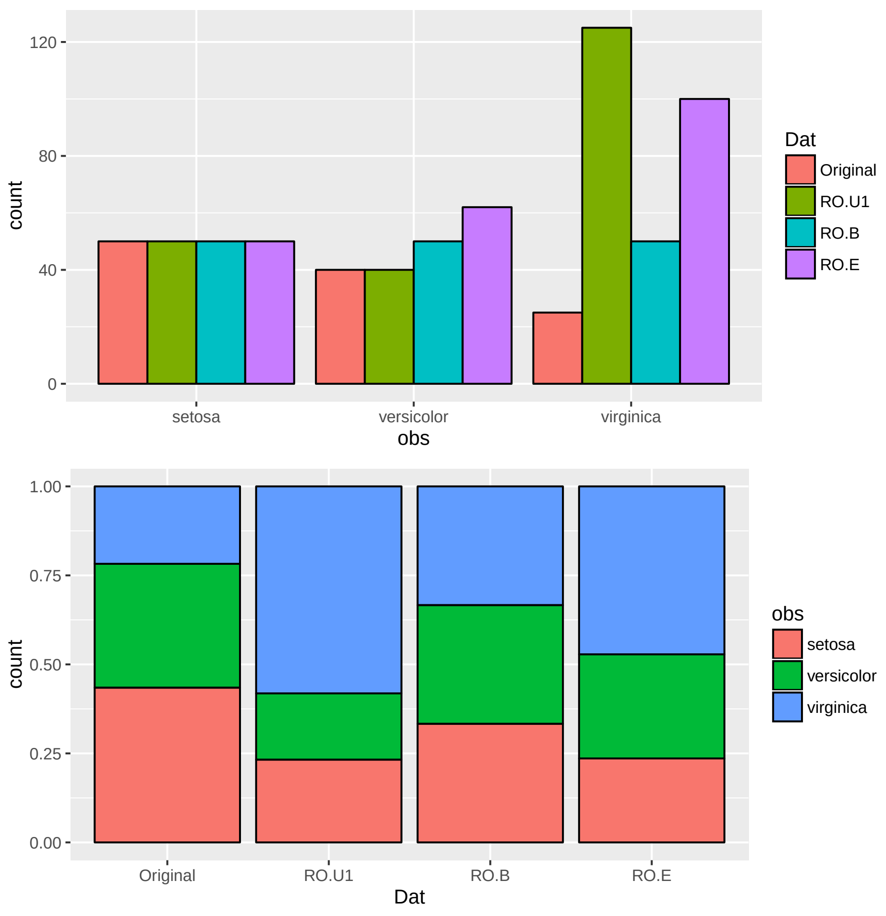


Figure 4: Impact of Random over-sampling strategy for different parameters values.

```
nds <- ImpSampClassif(Species~.,dat, C.perc=list(setosa=0.5, virginica=6))
# notice that when a certain class is not specified it remains unaltered
table(nds$Species)

##
##     setosa versicolor  virginica
##         25         40        150

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

##
##     setosa versicolor  virginica
##         38         38         38

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

##
##     setosa versicolor  virginica
##         27         34         54
```

Figure 5 shows the impact on the unbalanced iris data set of the changes made in the domain with Importance Sampling.

Figure 6 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.

4.4 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 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.

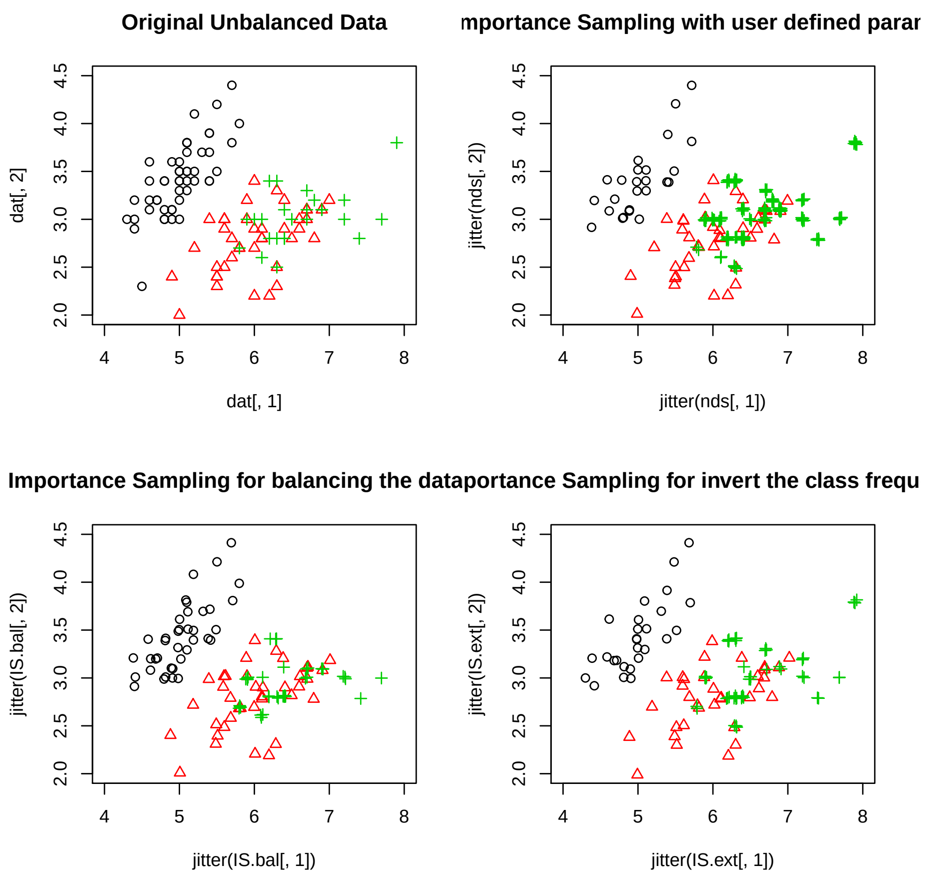


Figure 5: Impact of Importance Sampling strategy in Iris unbalanced data set.

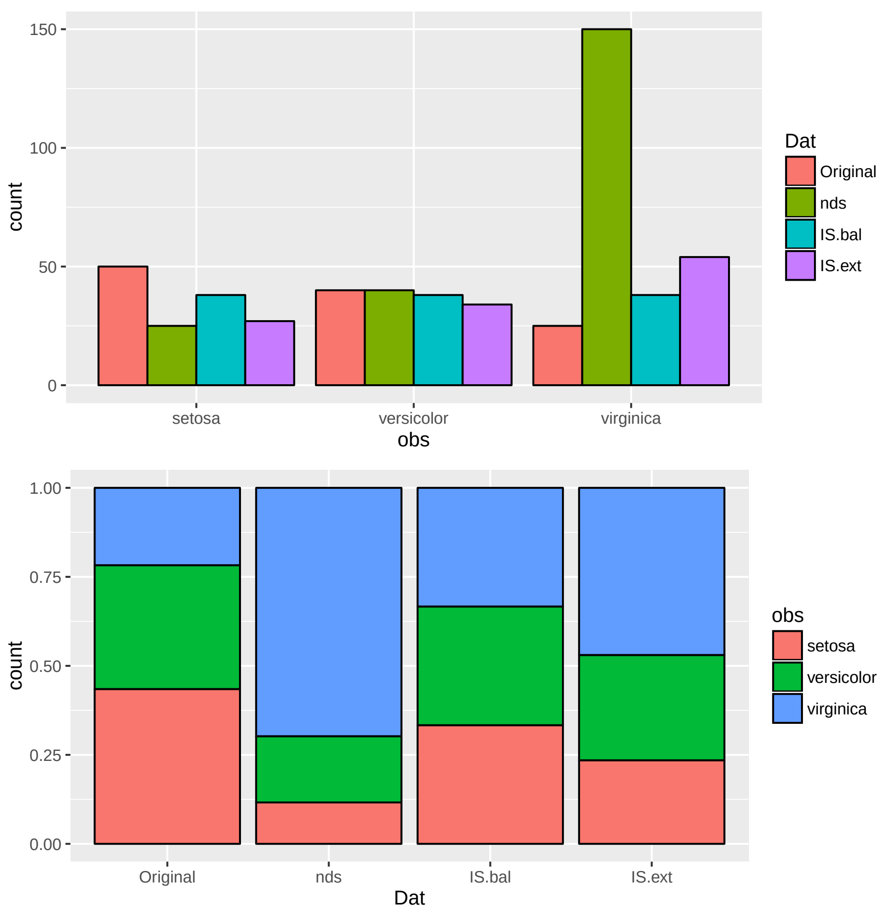


Figure 6: Impact of Importance Sampling strategy.

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 6. 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 iris data set shows how Tomek links can be applied.

```
# 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")

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

##   Sepal.Length    Sepal.Width          Species
##  Min.   :4.300   Min.   :2.000   setosa    :50
##  1st Qu.:5.000   1st Qu.:2.800   versicolor:40
##  Median :5.500   Median :3.100   virginica :19
##  Mean   :5.661   Mean   :3.123
##  3rd Qu.:6.200   3rd Qu.:3.400
##  Max.   :7.900   Max.   :4.400

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

## [1] 103 105 115 112 113 111
```

	setosa	versicolor	virginica
Original	50	40	25
ir	50	34	19
irCheb	50	40	19
irManM	50	34	25
irManB	50	34	19

Table 3: Number of examples in each class for different Tomek Links parameters.

Figure 7 shows the impact on the virginica and versicolor classes of the last experiences.

4.5 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.

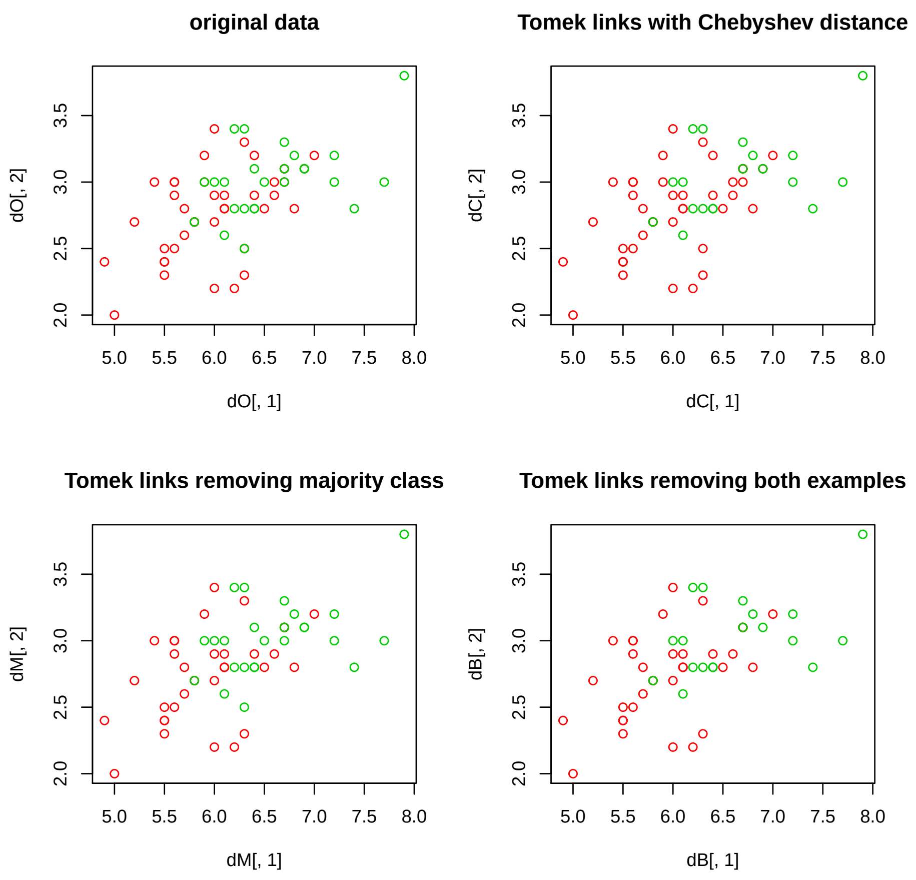


Figure 7: Impact of Tomek links strategy in versicolor and virginica classes.

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 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 to the modified iris data.

```
set.seed(123)
  myCNN <- CNNClassif(Species~., dat, Cl=c("setosa", "virginica"))
  CNN1 <- CNNClassif(Species~., dat, Cl="smaller")
  CNN2 <- CNNClassif(Species~., dat, Cl="versicolor")
  CNN3 <- CNNClassif(Species~., dat, dist="Chebyshev", Cl="virginica")

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

##     setosa versicolor  virginica
##          4         35         25

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

## [1] "virginica"

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

## [1] "setosa"     "versicolor"
```

	setosa	versicolor	virginica
Original	50	40	25
myCNN	50	40	25
CNN1	4	35	25
CNN2	23	40	24
CNN3	9	38	25

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

It is clear from these examples that this method entails a significant reduc-

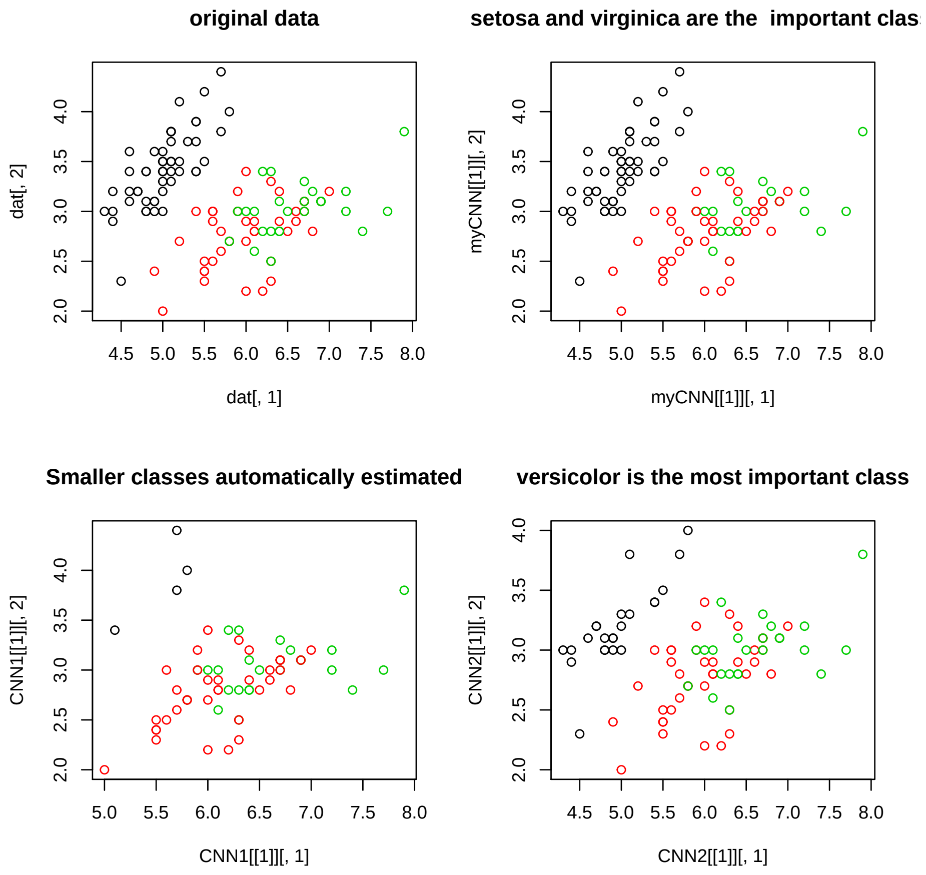


Figure 8: Impact of CNN method for different values of parameter Cl.

tion 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 8 provides a visual illustration of the impact of this method.

4.6 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.

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 $\frac{numberofexamples}{numberofclasses}$. 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 6. 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.

```
set.seed(1234)

ir2 <- OSSClassif(Species~., dat, dist="p-norm", p=3, Cl="virginica")
ir3 <- OSSClassif(Species~., dat, Cl=c("versicolor", "virginica"), start="Tomek")

## Warning in TomekClassif(form, dat, dist = dist, p = p, Cl = otherCl, rem = "both"):
There are no examples to remove!

ir4 <- OSSClassif(Species~., dat)

summary(ir2$Species)

##     setosa versicolor  virginica
##          4         33         25

summary(ir3$Species)

##     setosa versicolor  virginica
##         16         40         25

summary(ir4$Species)

##     setosa versicolor  virginica
##         12         27         25
```

	setosa	versicolor	virginica
Original	50	40	25
ir2	4	33	25
ir3	16	40	25
ir4	12	27	25

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

The impact of these methods on the number of examples in each class are

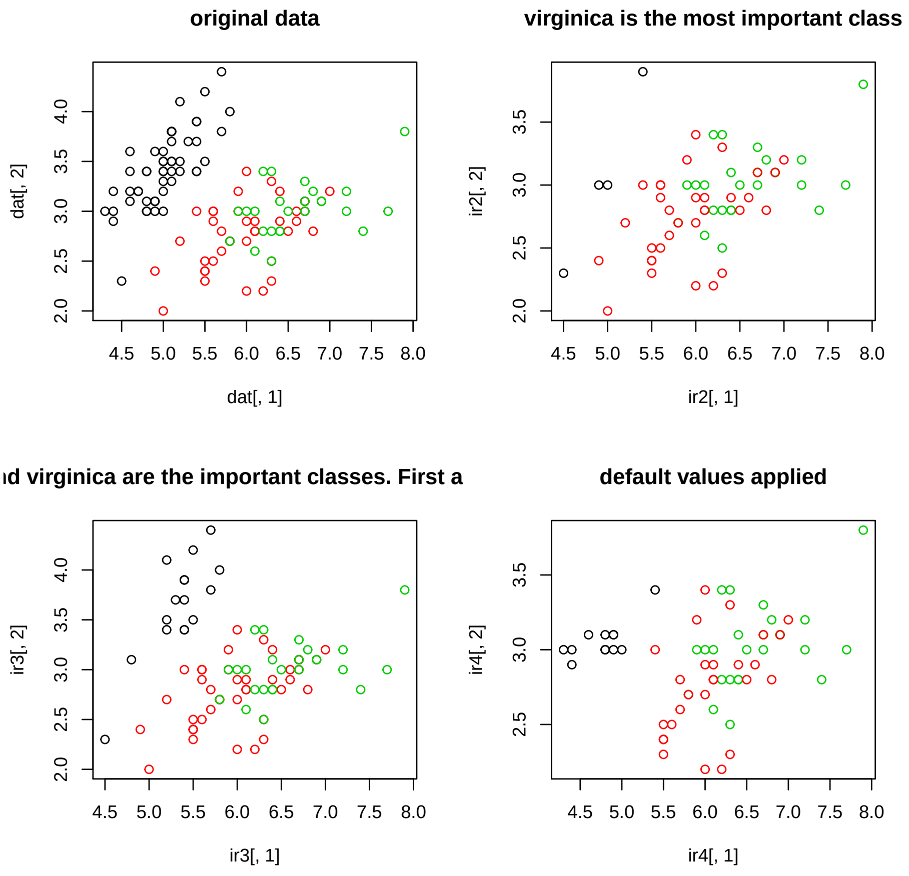


Figure 9: OSS techniques applied to a multiclass imbalanced problem.

in Table 5. The distribution of the results obtained with these methods can be visualized in Figure 9.

4.7 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.

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

In Table 6 we can observe the examples distributions for some parameters settings in ENN strategy and in Figure 10 we can visualize that distribution.

	setosa	versicolor	virginica
Original	50	40	25
Man5	50	29	9
Default	49	31	6
ChebSub7	50	40	7
ChebAll7	50	32	7
HVDM3	49	30	9

Table 6: 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.

Another example with a different data set is shown next using the data set cats from package MASS.

```
library(MASS)
data(cats)
# check the data set
summary(cats$Sex)

##  F  M
## 47 97

# Change the data set using ENN strategy
```

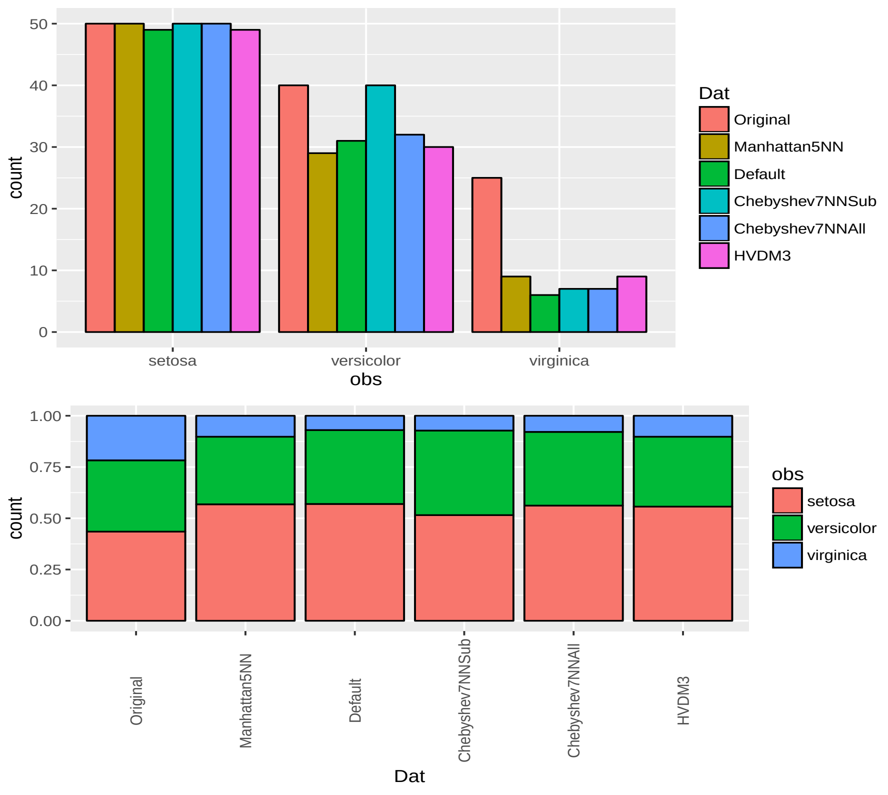
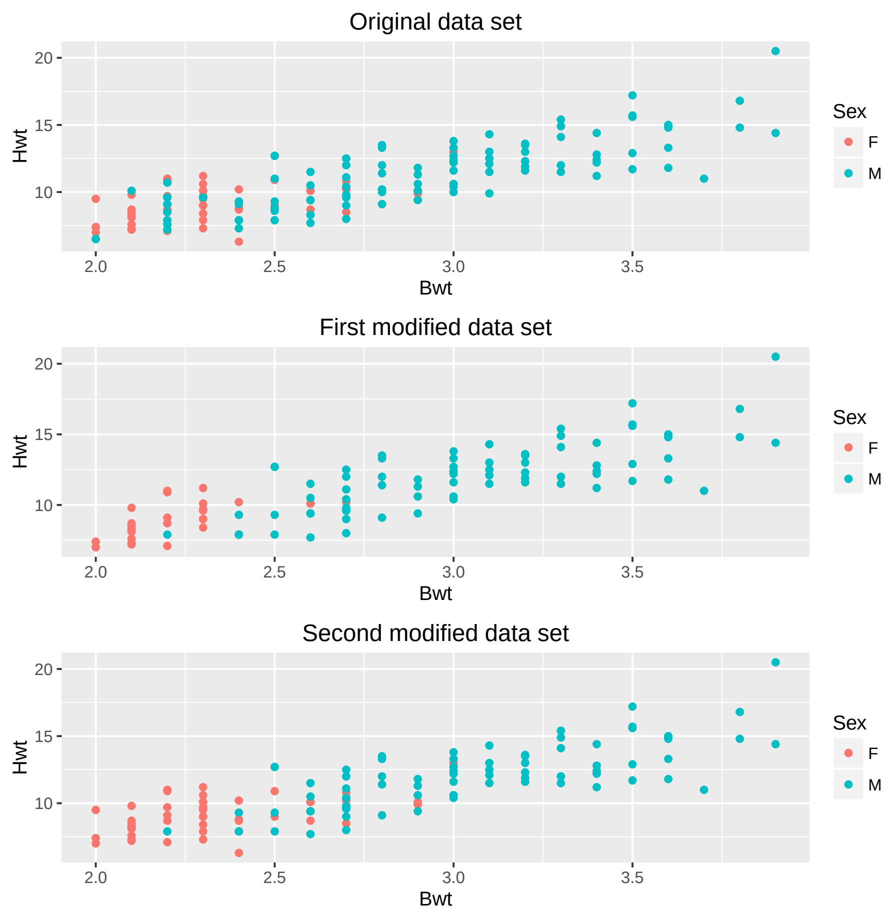


Figure 10: Impact in the Original data set of several parameters for ENN strategy

```
newdata1 <- ENNClassif(Sex~., cats)
newdata2 <- ENNClassif(Sex~., cats, Cl="M")
# check the number of examples in each class
summary(newdata1[[1]]$Sex)

##  F  M
## 28 74

summary(newdata2[[1]]$Sex)

##  F  M
## 47 74

# check visually the examples distribution
g <-ggplot(cats, aes(Bwt, Hwt, col=Sex))+geom_point()+ggtitle("Original data set")

# check visually the impact of the strategies
g1 <-ggplot(newdata1[[1]], aes(Bwt, Hwt, col=Sex))+
     geom_point()+ggtitle("First modified data set")
g2 <-ggplot(newdata2[[1]], aes(Bwt, Hwt, col=Sex))+
     geom_point()+ggtitle("Second modified data set")

do.call(grid.arrange, list(g,g1,g2))
```



Sometimes, this method is not capable of removing any example. When this happens, the original data set remains unchanged and an warning is issued. 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.

4.8 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 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. Table 7 provides the number of examples in each class for different parameters of NCL method and in Figure 11 the changes produced by the use of this method may be visualized.

```
set.seed(1234)
ir.M1 <- NCLClassif(Species~., dat, k=3, dist="p-norm", p=1, Cl="smaller")

## Warning in ENNClassif(form, dat[which(dat[, tgt] %in% otherCl), ], k, dist, :  There
are no examples to remove!

ir.M2<- NCLClassif(Species~., dat, k=1, dist="p-norm", p=1, Cl="smaller")
ir.Def <- NCLClassif(Species~., dat)
ir.Ch <- NCLClassif(Species~., dat, k=7, dist="Chebyshev", Cl="virginica")

## Warning in ENNClassif(form, dat[which(dat[, tgt] %in% otherCl), ], k, dist, :  There
are no examples to remove!

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

	setosa	versicolor	virginica
Original	50	40	25
ir.M1	50	28	25
ir.M2	49	29	25
ir.Def	49	28	25
ir.Ch	50	33	25
ir.Eu	50	28	25

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

4.9 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 4.1.

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).

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 present an example of the impact of applying this technique for different values of the parameters.

```r
set.seed(1234)
irB<- GaussNoiseClassif(Species~., dat, C.perc="balance")
```

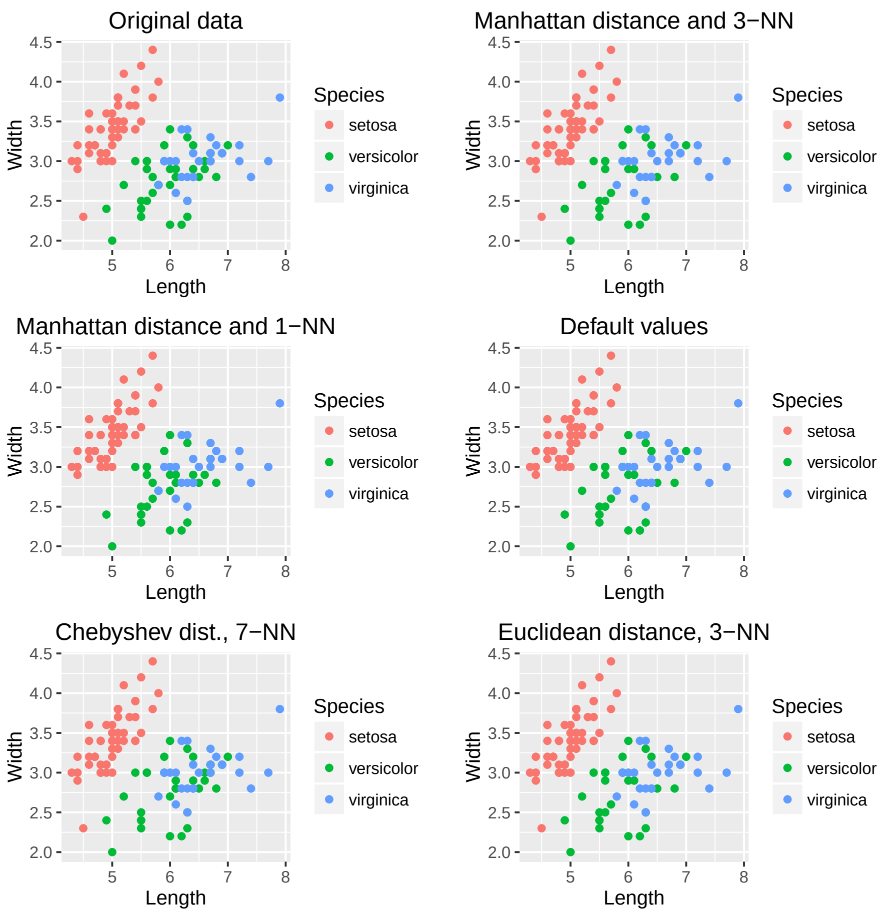


Figure 11: NCL techniques applied to a multiclass imbalanced problem.

```r
irE <- GaussNoiseClassif(Species~., dat,C.perc="extreme")
irU1 <- GaussNoiseClassif(Species~., dat,
                          C.perc=list(setosa=0.3, versicolor=1.5, virginica=4),
                          pert=0.5, repl=TRUE)
irU2 <- GaussNoiseClassif(Species~., dat,
                          C.perc=list(versicolor=3, virginica=2),
                          pert=0.05)
```

Table 8 presents the impact on the number of examples for the considered parameters of this strategy. In Figure 12 we can observe the number of examples on the changed data sets for the parameters considered and Figure 13 presents the distribution of examples for those parameters.

	setosa	versicolor	virginica
Original	50	40	25
irB	38	38	38
irE	27	34	54
irU1	15	60	100
irU2	50	120	50

Table 8: Number of examples in each class for different parameters of Gaussian Noise strategy.

4.10 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 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 14.

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

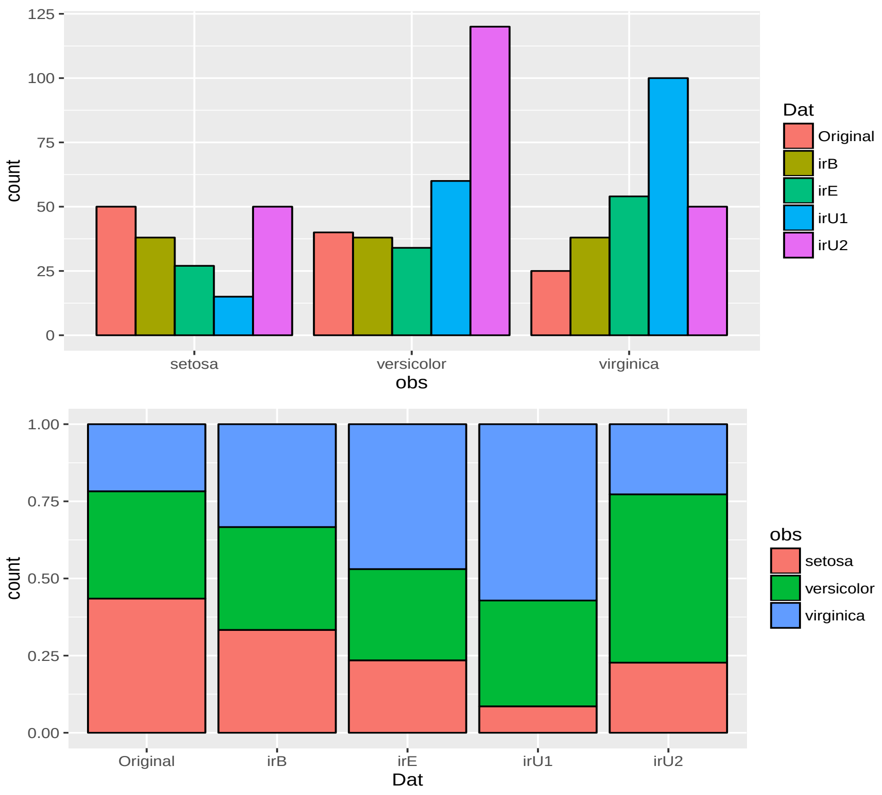


Figure 12: Impact in the Original data set of several parameters in Gaussian noise strategy.

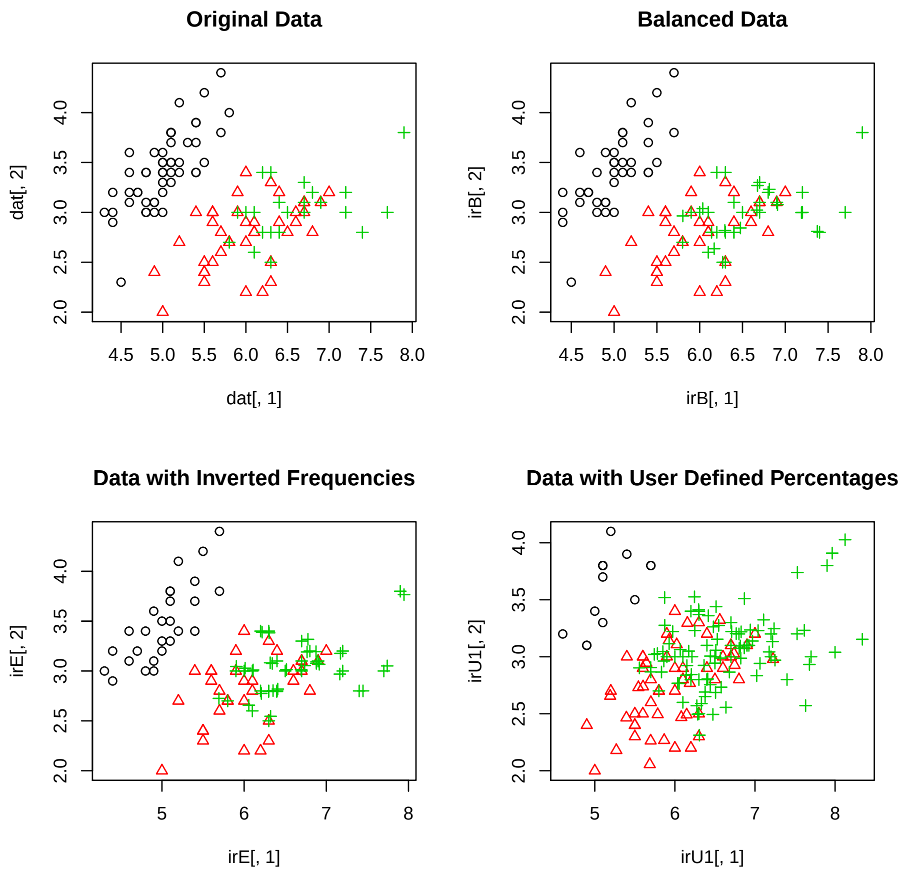


Figure 13: The examples distribution for different parameters in Gaussian Noise strategy.

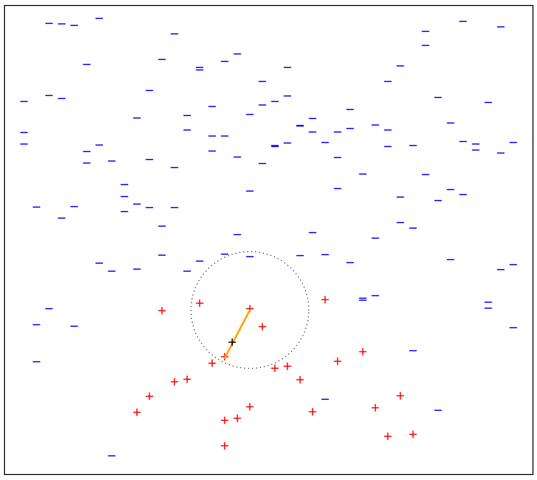


Figure 14: Generation of synthetic examples through Smote algorithm.

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`).

```
mysmote1 <- SmoteClassif(Species~., dat,
                         C.perc=list(setosa=0.6, virginica=1.5))
mysmote2 <- SmoteClassif(Species~., dat,
                         C.perc=list(setosa=0.2, versicolor=4), repl=TRUE)
mysmote3 <- SmoteClassif(Species~., dat,
                         C.perc=list(virginica=6, versicolor=2))
smoteB <- SmoteClassif(Species~., dat,
                         C.perc="balance")
smoteE <- SmoteClassif(Species~., dat,
                         C.perc="extreme")
```

Table 9 show the impact on the number of examples in each class for several parameters of smote technique.

Figures 15 and 16 present the impact of applying smote strategy on an imbalanced data set.

5 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.

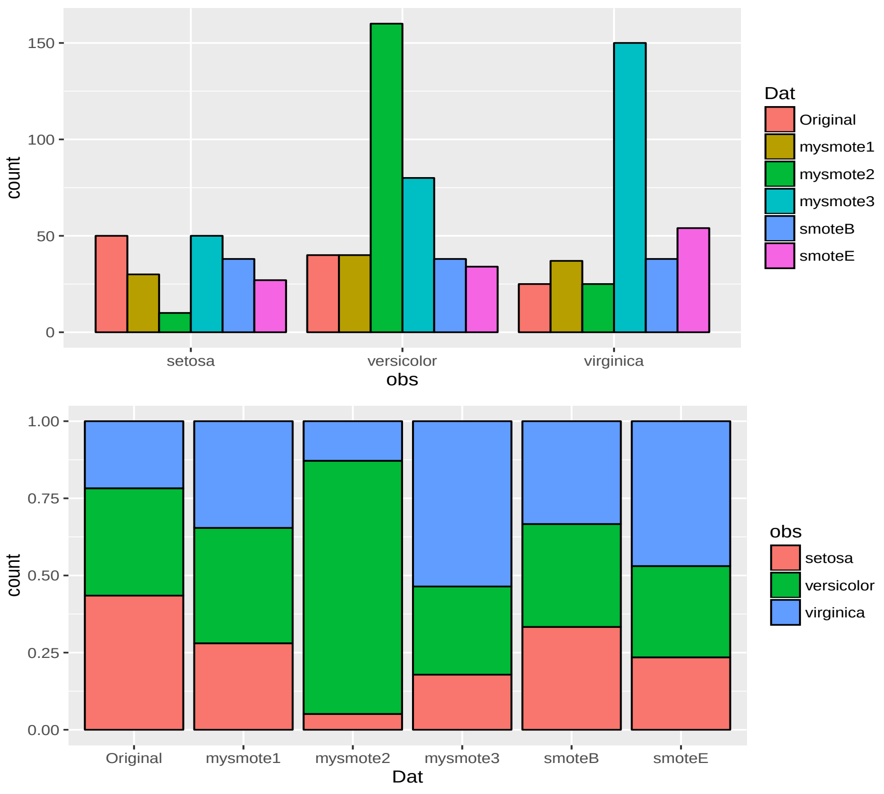


Figure 15: Impact in the Original data set of several parameters in smote strategy.

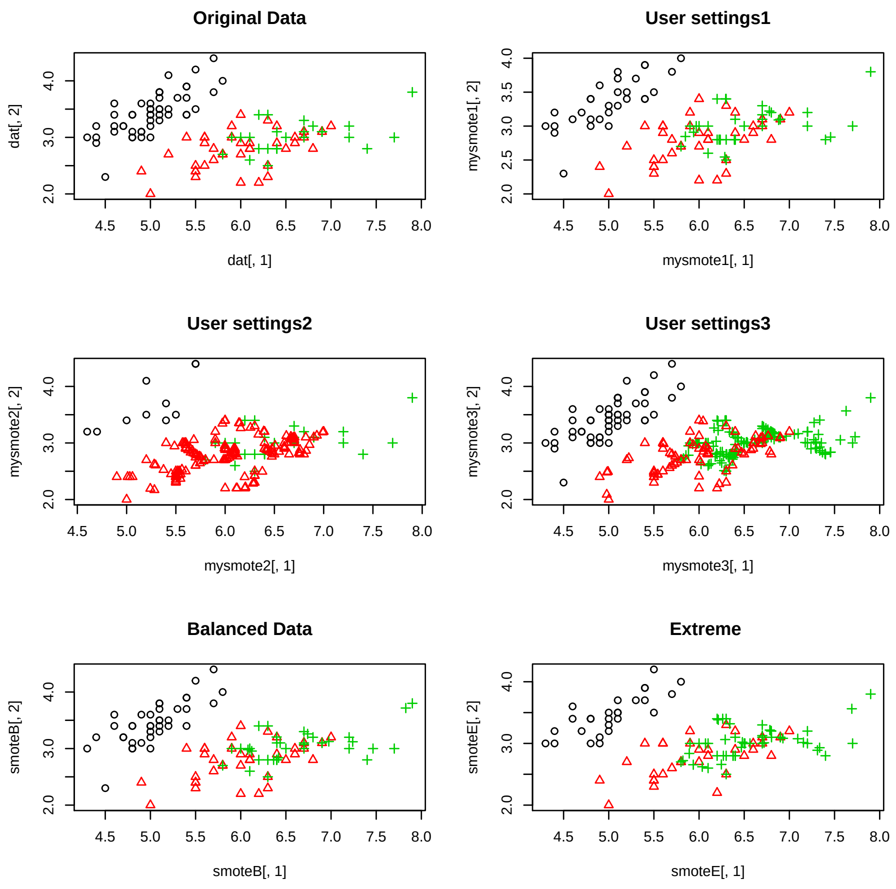


Figure 16: Smote strategy applied with different parameters

	setosa	versicolor	virginica
Original	50	40	25
mysmote1	30	40	37
mysmote2	10	160	25
mysmote3	50	80	150
smoteB	38	38	38
smoteE	27	34	54

Table 9: Number of examples in each class for different parameters of smote strategy.

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 17.

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 17). On the other hand, the target values falling in the intervals $]1.5, 4.5[$ and $]7, 10]$ (represented in red in Figure 17) 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 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. Alternatively, the relevance function may be manually defined with a 3-column matrix as follows:

```
# relevance function represented in the previous example

## method: range
```

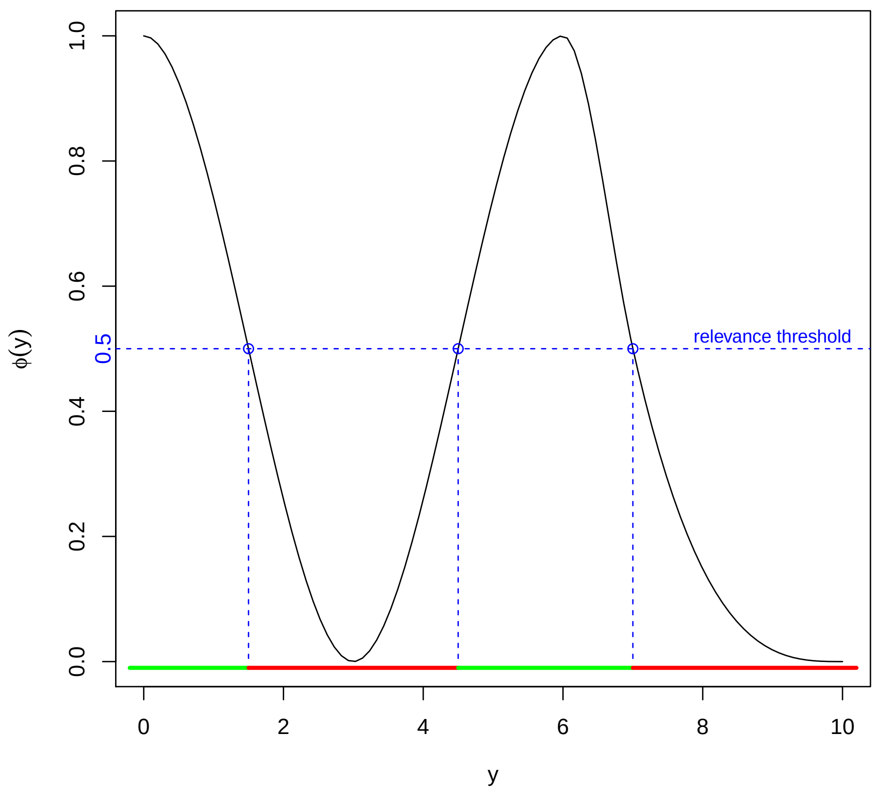


Figure 17: Example of a relevance function

```r
# 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 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. We now provide an example of how to use this automatic method.

```r
## 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 relevance threshold established by the user.

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

- 5.1 Random Under-sampling
- 5.2 Random Over-sampling
- 5.3 Gaussian Noise Introduction
- 5.4 SmoteR Algorithm
- 5.5 Importance Sampling

5.1 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 were considered normal and uninteresting and thus were regarded as candidates to be under-sampled. The user was also asked another parameter that established the proportion between normal (unimportant) and rare (important) cases that the 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 want to apply the range method, then he should provide a 3-column matrix. 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 shows how these parameters can be set.

```r
# use algae data set with NA's removed
library(DMwR)
data(algae)
clean.algae <- algae[complete.cases(algae),]

# We start by using the automatic method for the relevance function
# Since this is the default behaviour, we can simply not mention the
# "rel" parameter

algB <- RandUnderRegress(a7~., clean.algae, C.perc="balance")
algE <- RandUnderRegress(a7~., clean.algae, C.perc="extreme")

# the automatic method for the relevance function provides only one bump
# with values to be under-sampled, thus we only need to indicate one percentage
algMy <- RandUnderRegress(a7~., clean.algae, C.perc=list(0.5))
```

The impact of the previous strategies can be visualized in Figure 18.

Suppose, for the same data set, that we want to consider as relevant the target values close to 5 and above 25. We can define a relevance function that expresses this:

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

# add zero relevance for the target values before five
rel <- rbind(rel, c(0,0,0))
rel <- rbind(rel,c(4,0,0))
```

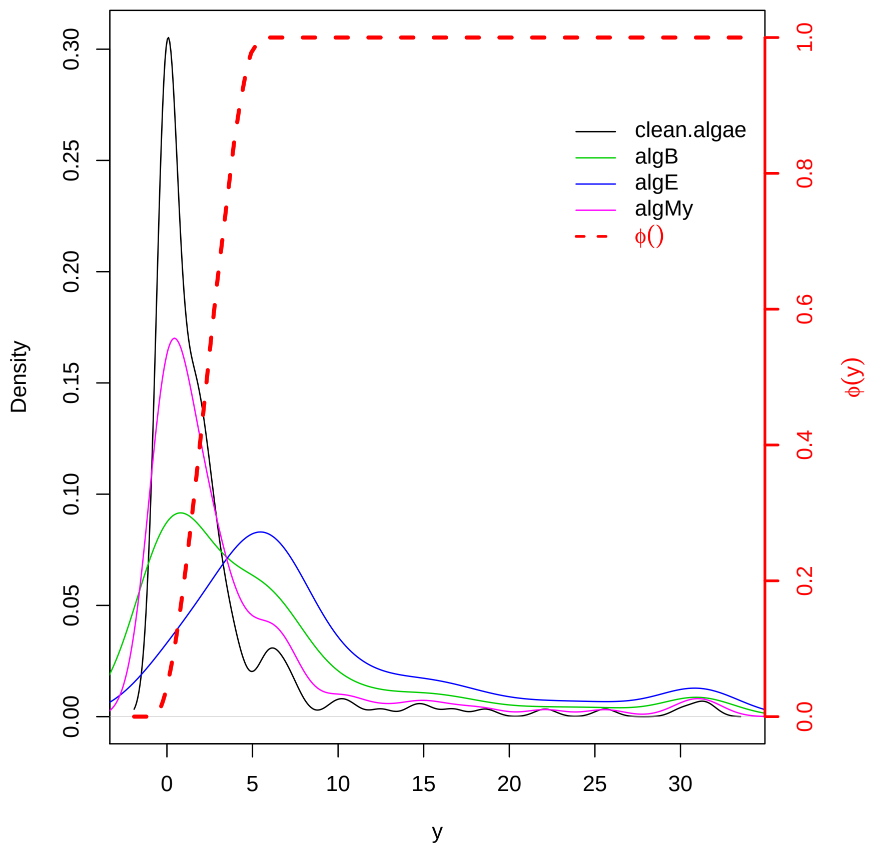


Figure 18: Relevance function and density of the target variable in the original and new data sets using Random Under-sampling strategy

```
# add maximum relevance for the target values close to 5
rel <- rbind(rel, c(5,1,0))

# add some unimportant target values

rel <- rbind(rel, c(7,0,0))
rel <- rbind(rel, c(18,0,0))

# add maximum relevance to points close to and above 20

rel <- rbind(rel, c(20,1,0))
rel <- rbind(rel, c(30,1,0))
```

Now, having defined the relevance threshold as 0.7, we will apply the random under-sampling strategy in the two ranges with uninteresting values of the target variable:

```
RUnew <- RandUnderRegress(a7~., clean.algae, rel=rel, thr.rel=0.7,
                          C.perc=list(0.2,0.8))
RUB2 <- RandUnderRegress(a7~., clean.algae, rel=rel, thr.rel=0.7,
                         C.perc="balance")
RUE2 <- RandUnderRegress(a7~., clean.algae, rel=rel, thr.rel=0.7,
                         C.perc="extreme")
```

The impact of the new strategies and the relevance function defined can be visualized in Figure 19.

We must also highlight that this strategy entails costs which can not be disregarded namely 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, 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 10 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).

	clean.algae	algB	algE	algMy	RUnew	RUB2	RUE2
nr. examples	184	65	40	108	54	13	7

Table 10: Total number of examples in each data set for different parameters of random under-sampling strategy.

5.2 Random Over-sampling

The Random over-sampling method here presented 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 introduction of random copies 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

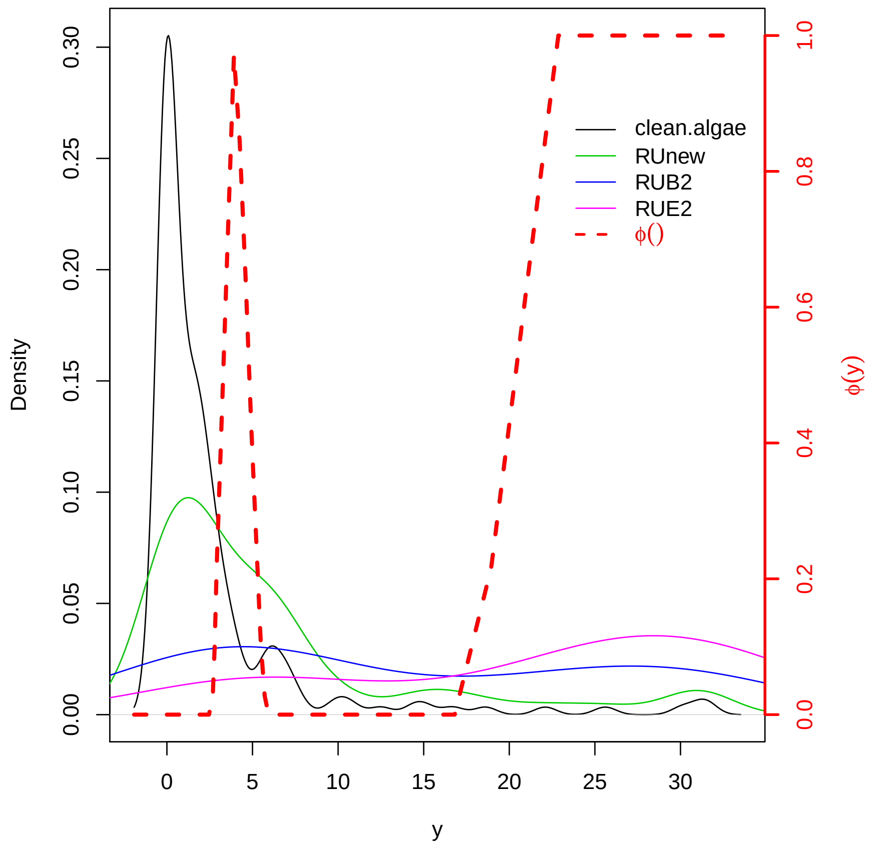


Figure 19: Relevance function and density of the target variable in the original and new data sets with Random Under-sampling strategy.

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
Alg.my <- RandOverRegress(a7~., clean.algae, C.perc=list(2.5))
Alg.Bal <- RandOverRegress(a7~., clean.algae, C.perc="balance")
Alg.Ext0.5 <- RandOverRegress(a7~., clean.algae, C.perc="extreme")

# change the relevance threshold to 0.9
Alg.Ext0.9 <- RandOverRegress(a7~., clean.algae, thr.rel=0.9, C.perc="extreme")
```

Figure 20 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 11 shows the impact of the previous examples on the total number of examples of used the data set.

	clean.algae	Alg.my	Alg.Bal	Alg.Ext0.5	Alg.Ext0.9
nr. examples	184	266	335	874	1195

Table 11: Total number of examples in each data set for different parameters of random over-sampling strategy.

As expected, all the data sets have an increased size. However, for the `Alg.Ext0.9` data set, the size was increased approximately 649%. 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.

5.3 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.

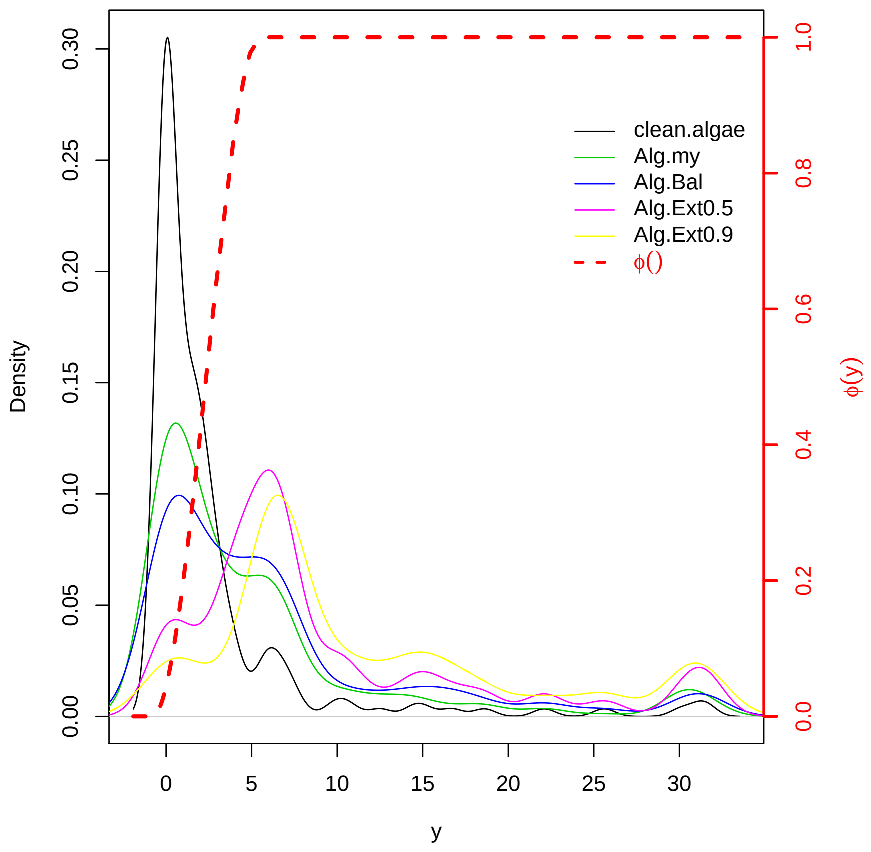


Figure 20: Relevance function and density of the target variable in the original and new data sets using Random over-sampling strategy.

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 5.1. For the over-sampling strategy we use the same procedure which was described for classification tasks in Section 4.9. 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), 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(a7~., clean.algae, thr.rel=thr.rel, C.perc=C.perc)
gnB <- GaussNoiseRegress(a7~., clean.algae, thr.rel=thr.rel, C.perc="balance")
gnE <- GaussNoiseRegress(a7~., clean.algae, thr.rel=thr.rel, C.perc="extreme")
```

Figures 21 and 22 show the impact of this strategy, for the parameters considered, on the examples distribution. In Figure 22 we selected two numeric features (“Cl” and “a5”) to visualize the impact on the 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(a7~., clean.algae, thr.rel=thr.rel, C.perc="balance")

# try two different values for "pert" parameter
gnB2 <- GaussNoiseRegress(a7~., clean.algae, thr.rel=thr.rel, C.perc="balance",
                          pert=0.5)
gnB3 <- GaussNoiseRegress(a7~., clean.algae, thr.rel=thr.rel, C.perc="balance",
                          pert=0.01)
```

The impact of changing the parameter `pert` is represented in Figure 23.

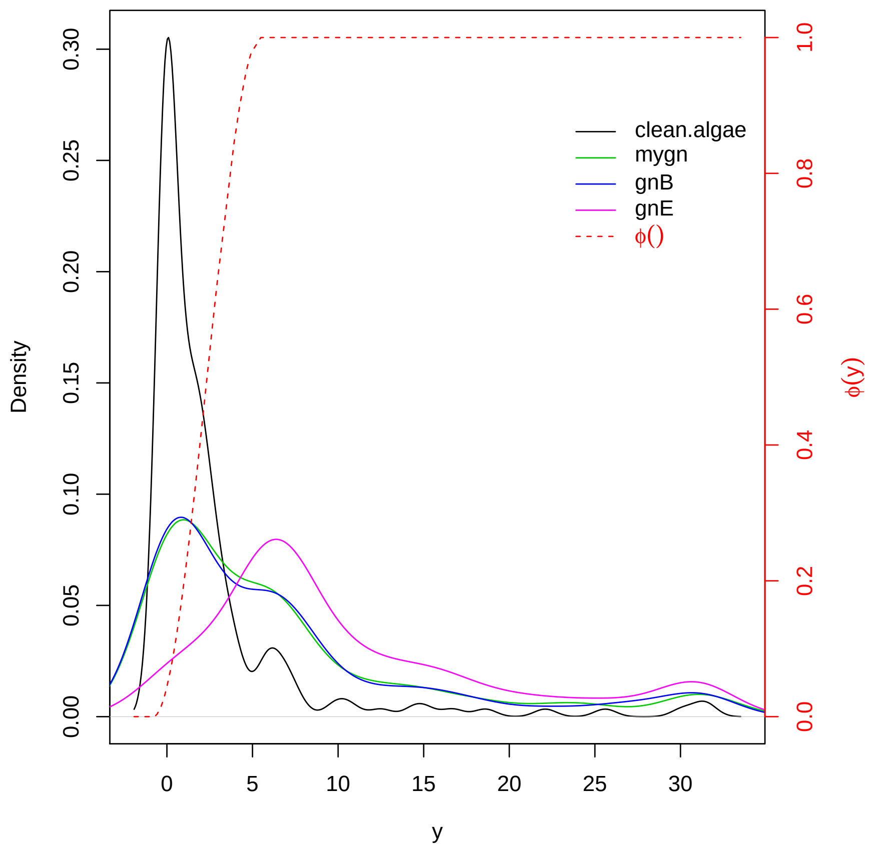


Figure 21: Relevance function and density of the target variable in the original and new data sets using Gaussian noise strategy.

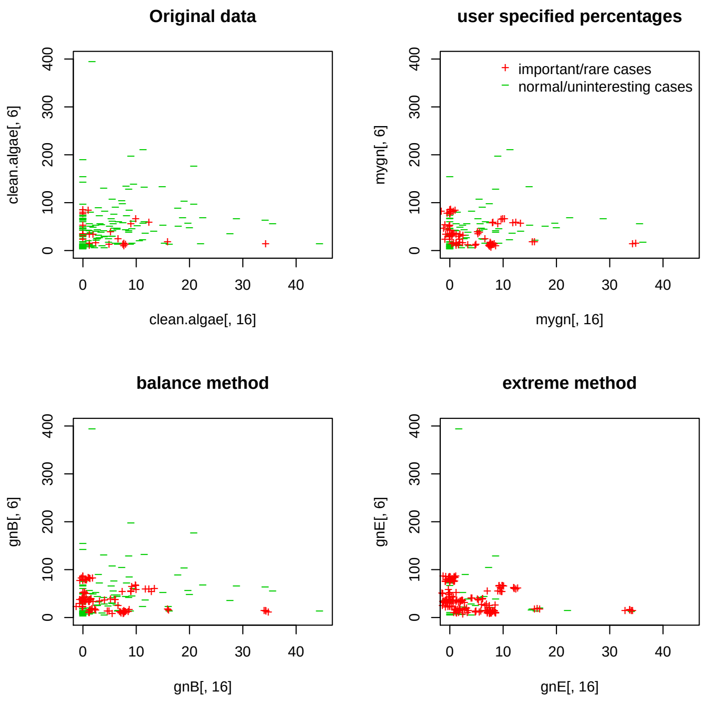


Figure 22: The impact of Gaussian Noise strategy.

5.4 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 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

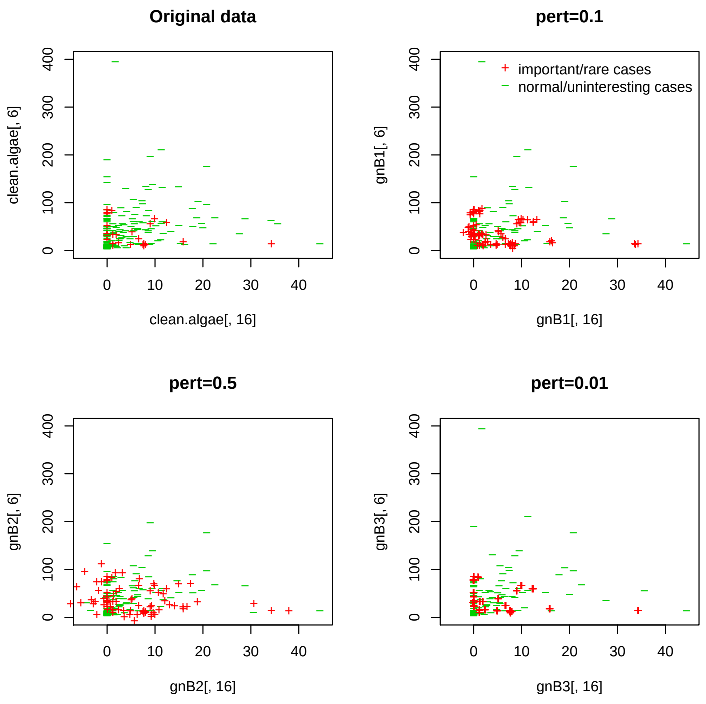


Figure 23: Impact of changing the pert parameter in Gaussian Noise strategy.

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(clean.algae$a7)
phiF.args <- phi.control(y,method="extremes",extr.type="both")
```

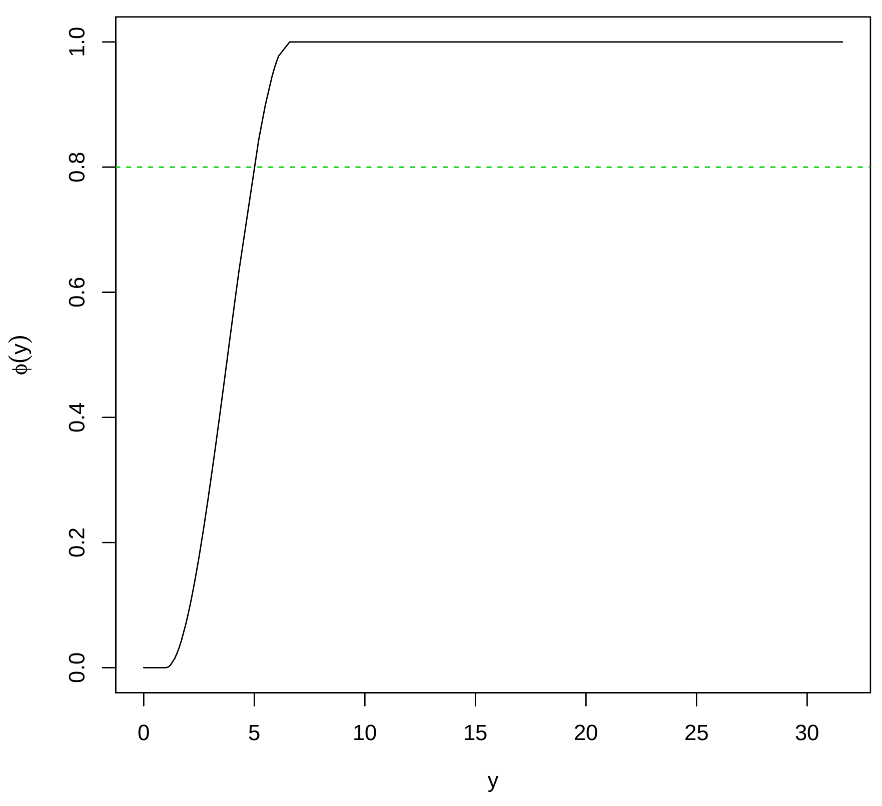


Figure 24: Relevance function obtained automatically for the clean.algae data set

```r
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 24 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:

```r
# 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
```

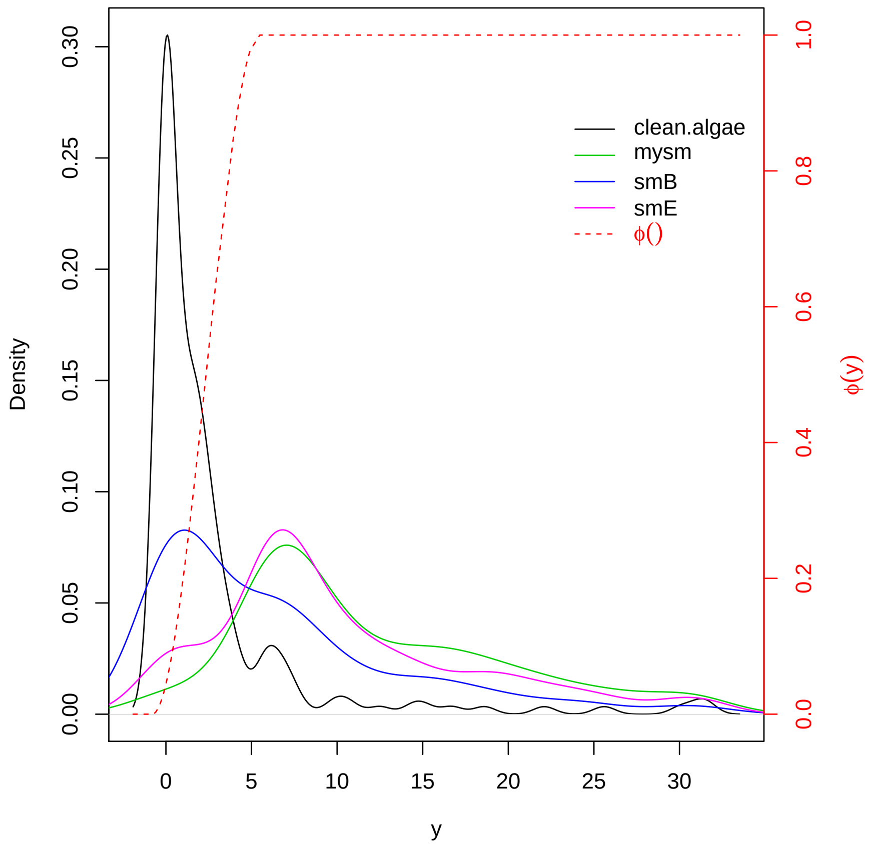


Figure 25: Relevance function and density of the target variable in the original and new data sets using smoteR strategy.

```
# it is necessary to set the distance function to "HEOM" because the
# data set contains nominal and numeric features
mysm <- SmoteRegress(a7~., clean.algae, thr.rel=thr.rel, dist="HEOM", C.perc=C.perc)

# use the automatic method for obtaining a balanced data set
smB <- SmoteRegress(a7~., clean.algae, thr.rel=thr.rel, dist="HEOM", C.perc="balance")

# use the automatic method for invert the frequencies of the bumps
smE <- SmoteRegress(a7~., clean.algae, thr.rel=thr.rel, dist="HEOM", C.perc="extreme")
```

This strategy changes the examples distribution as shown in Figure 25.

We can also obtain the number of examples that each bump contains. Table 12 show that distribution for the considered strategies.

In Figure 26 we can visualize the impact of these approaches on the examples distribution for each bump of relevance.

In Figure 27 we can visualize the impact of the several strategy variants considered for two numeric features selected (the “mnO2” and “Cl” features).

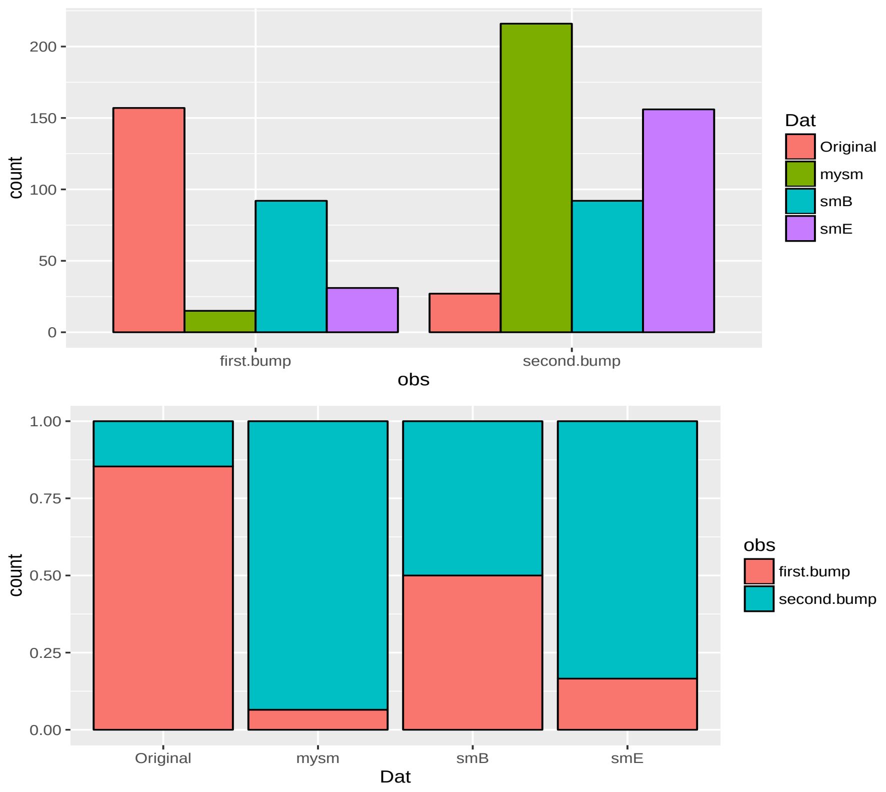


Figure 26: Impact in the distribution of examples for several parameters in smoteR strategy.

	first bump	second bump
clean.algae	157	27
mysm	15	216
smB	92	92
smE	31	156

Table 12: Number of examples in each bump of relevance for different parameters of smoteR strategy.

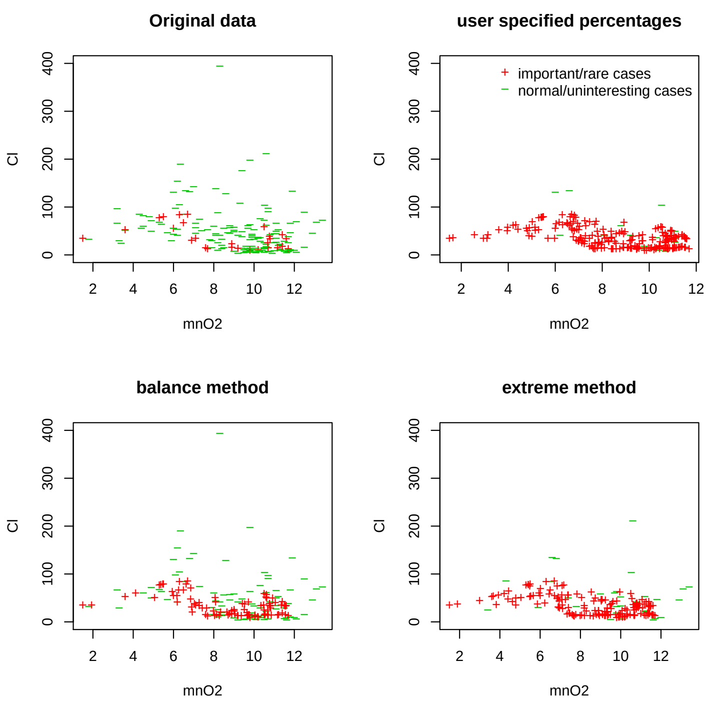


Figure 27: The impact of smoteR strategy.

5.5 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 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.

```r
# relevance function estimated automatically has two bumps
# using the strategy with threshold definition
C.perc=list(0.2,6)
myIS <- ImpSampRegress(a7~., clean.algae, thr.rel=0.8,C.perc=C.perc)
ISB <- ImpSampRegress(a7~., clean.algae, thr.rel=0.8, C.perc="balance")
ISE <- ImpSampRegress(a7~., clean.algae, thr.rel=0.8, C.perc="extreme")
```

Figures 28 and 29 show the impact on the density and distribution of the examples for the new data sets obtained with Importance Sampling strategy.

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

```r
# 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(a7~., clean.algae)
IS1 <- ImpSampRegress(a7~., clean.algae, U=0.9, O=0.2)
IS2 <- ImpSampRegress(a7~., clean.algae, U=0.5, O=0.8)
```

Figures 30 and 31 show the impact on the density and distribution of the examples for the new data sets obtained with Importance Sampling strategy.

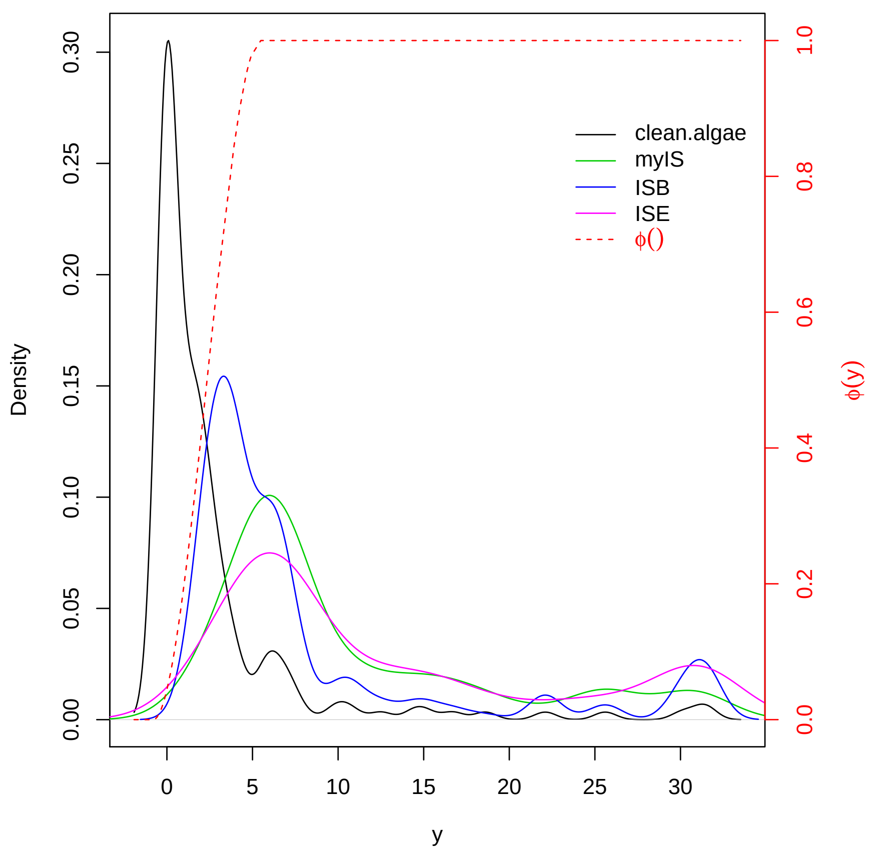


Figure 28: Relevance function and density of the target variable in the original and new data sets using Importance Sampling strategy.

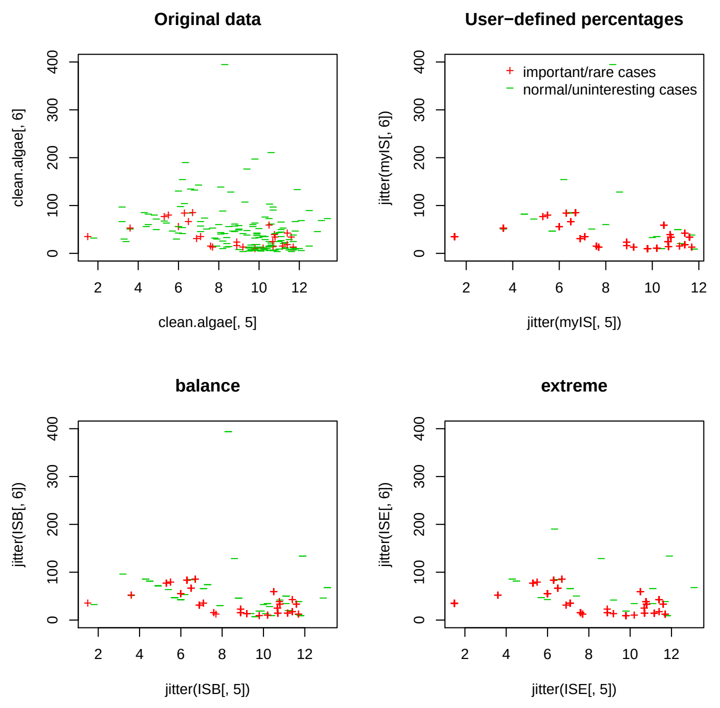


Figure 29: Impact of Importance Sampling strategy.

6 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 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.

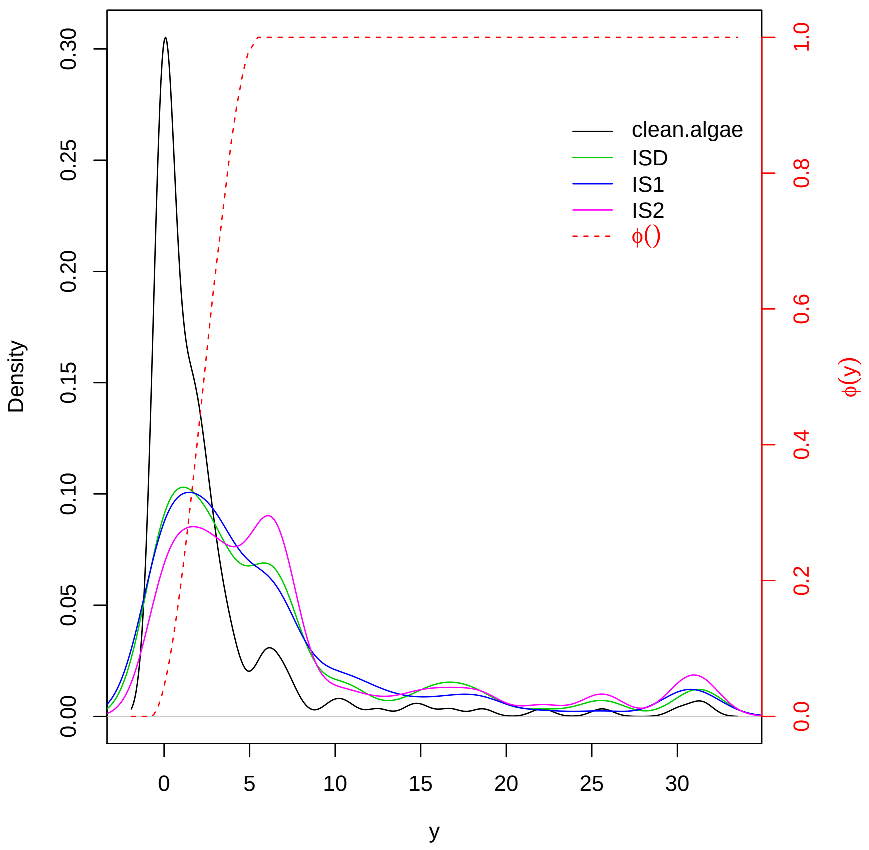


Figure 30: Relevance function and density of the target variable in the original and new data sets using Importance Sampling strategy.

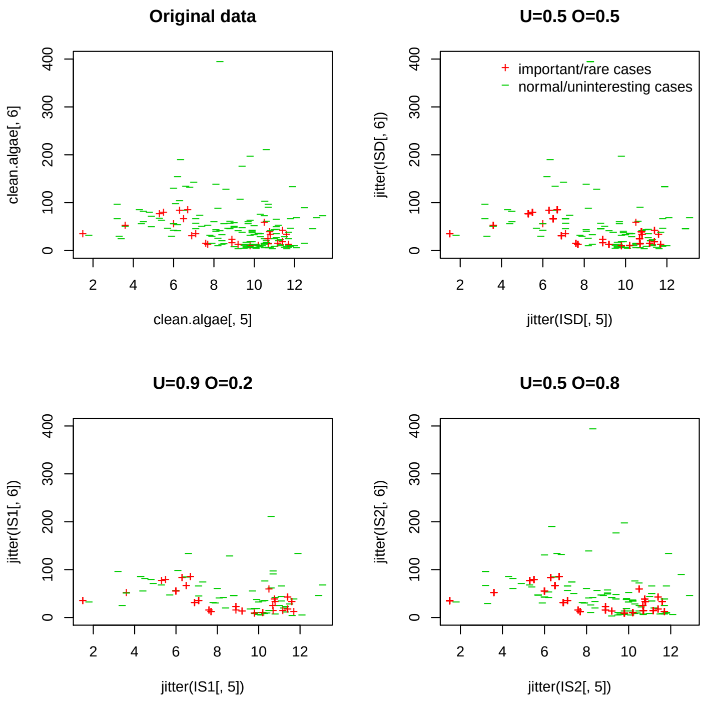


Figure 31: Impact of Importance Sampling strategy.

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 32.

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)
```

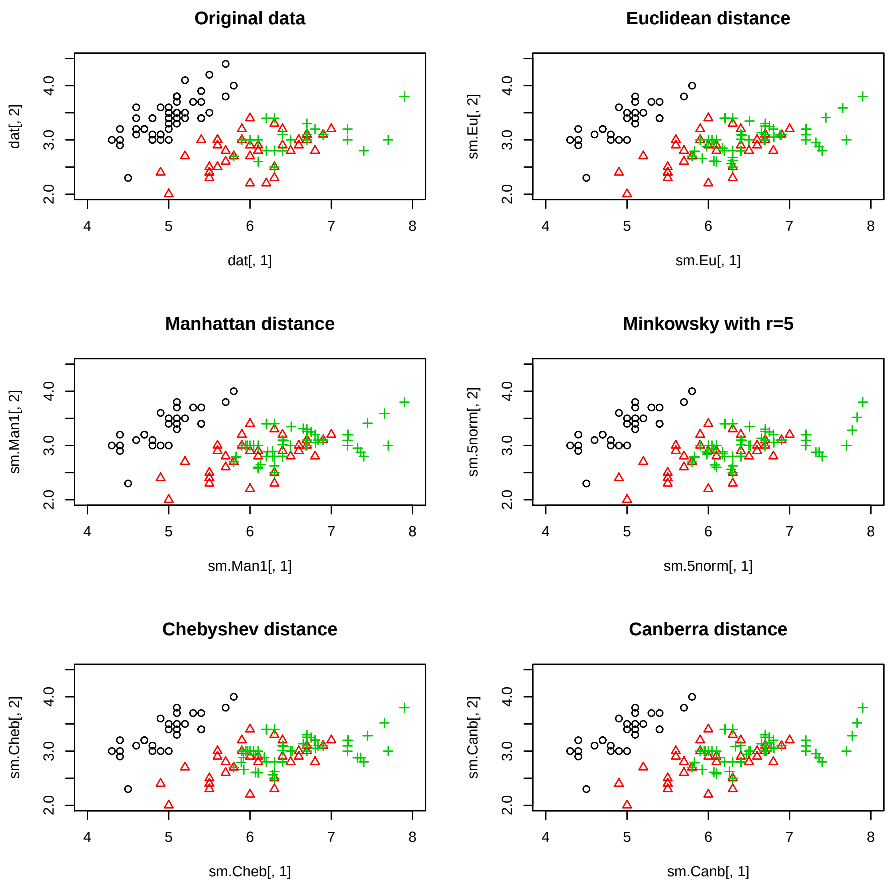


Figure 32: Impact of using different distance functions with smote strategy.

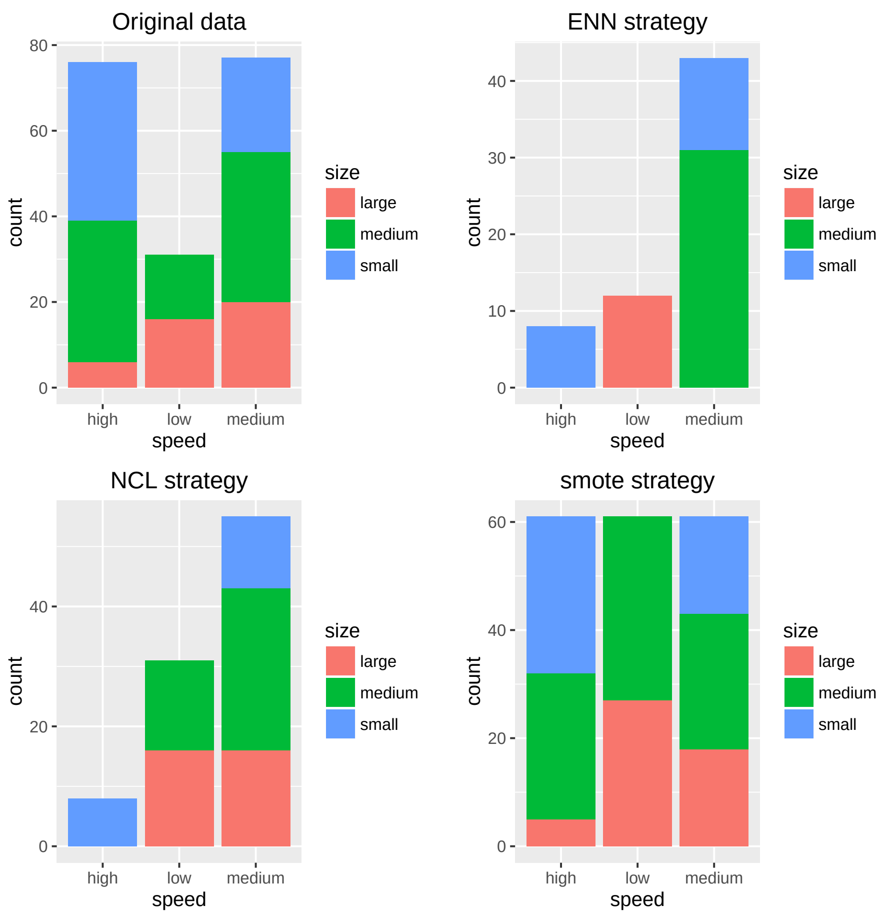


Figure 33: Using Overlap distance function with different strategies on a data set with only nominal features.

```
##     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")
# 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 33 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 34 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.

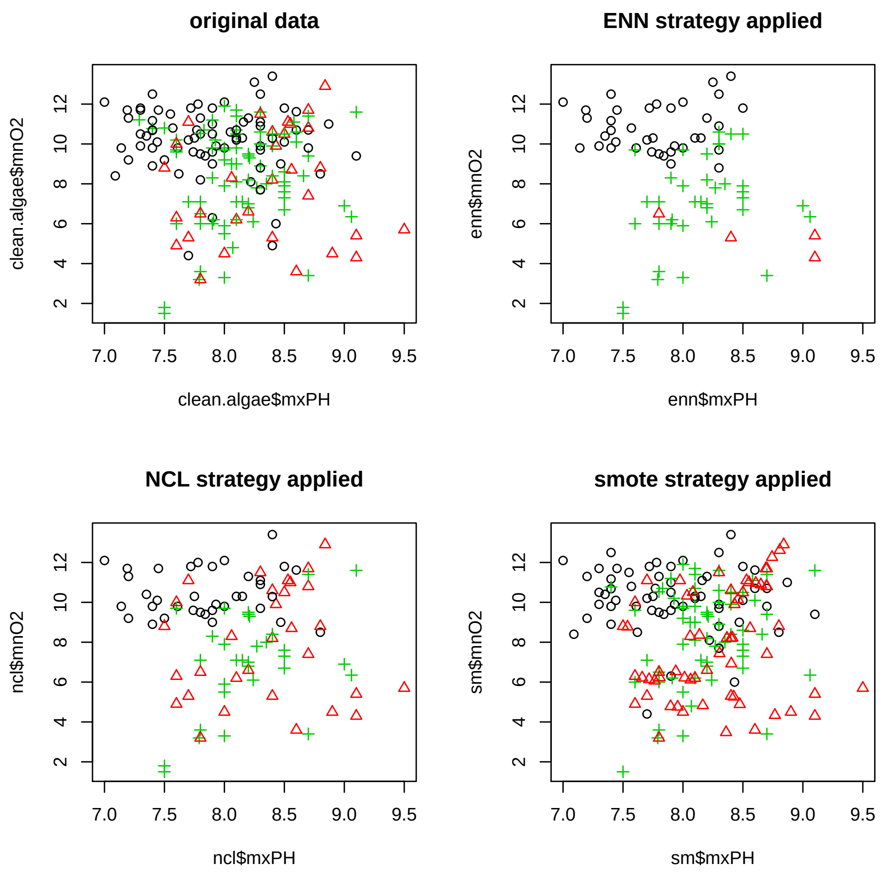


Figure 34: Using HEOM distance function with different strategies on a data set with both nominal and numeric features.

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

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

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

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")
```

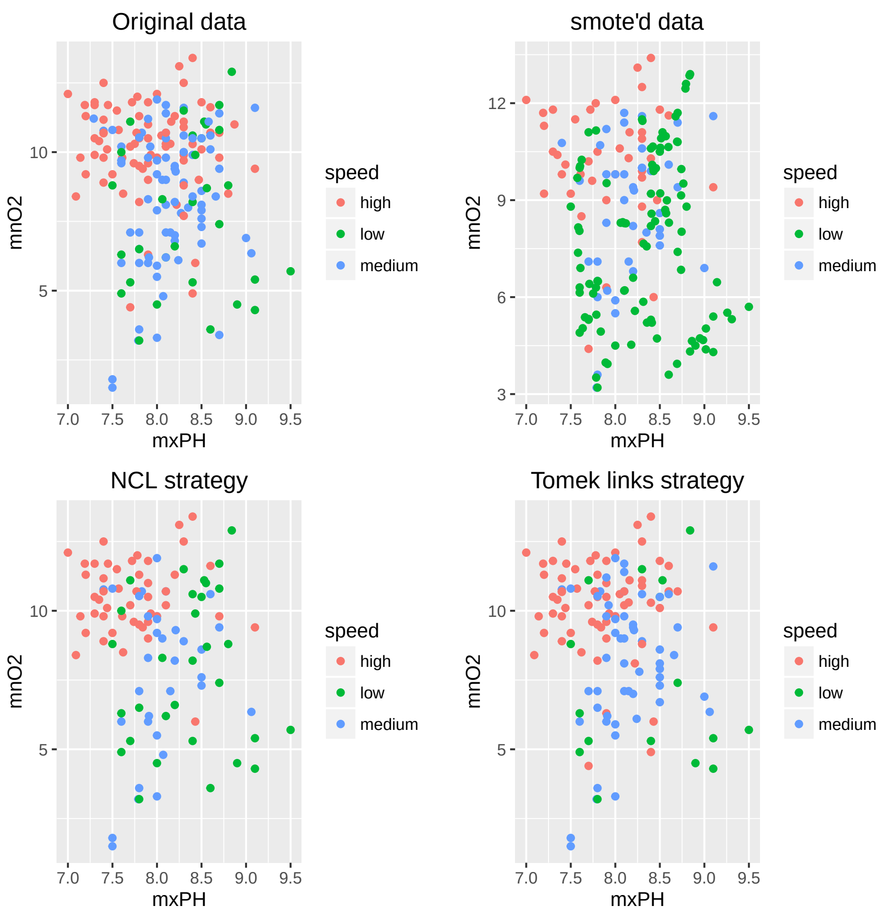


Figure 35: Using HVDM distance function with different strategies.

Figure 35 shows the result of applying HVDM distance function for several different strategies, on a data set consisting of numeric and nominal features.

In Figure 36 the impact of smote strategy applied with different distance functions on a data set can be observed.

7 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 essentially 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

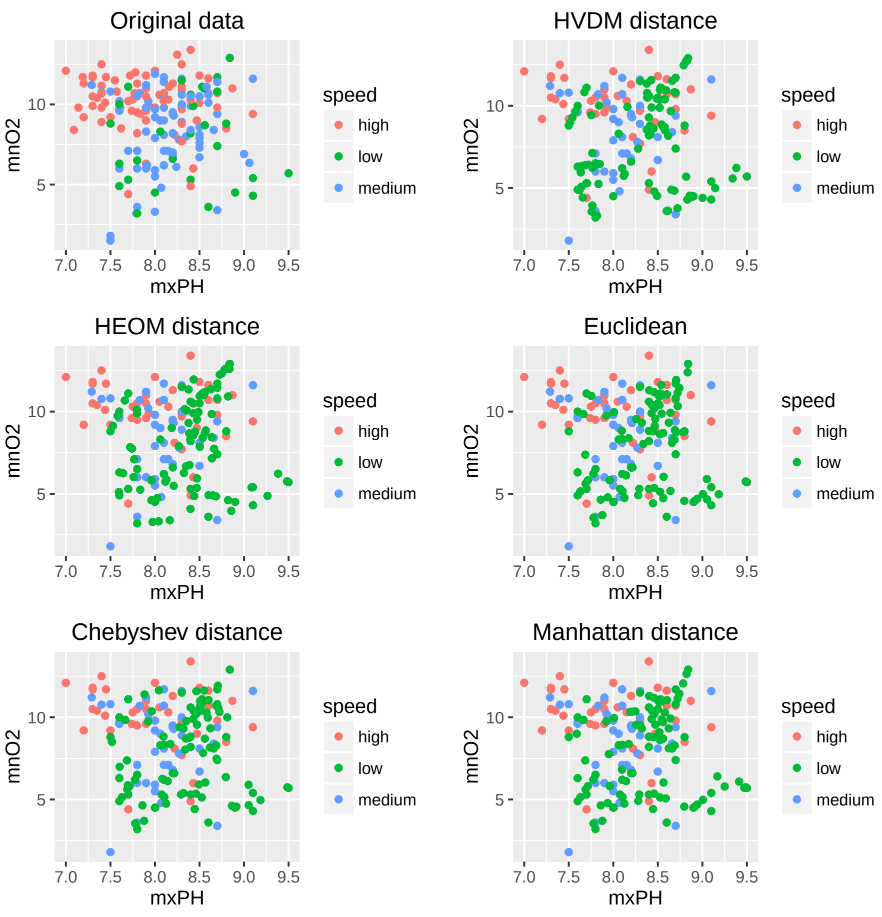


Figure 36: Using different distance functions with smote strategy.

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 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.

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.

[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.

[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 0.7.5.

[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.