Resampling Strategies for Regression

Paula Branco, **Luis Torgo**, Rita Ribeiro and Bernhard Pfahringer

Departamento de Ciência de Computadores
Faculdade de Ciências/Universidade do Porto

January, 2017

Predictive Modeling with Imbalanced Distributions

- Many predictive tasks involve handling a target variable that has an imbalanced distribution in the available training data.
- Problem thoroughly explored in classification tasks.
- Similar problems occur in some regression problems.
- Important applications in real world domains (finance, ecology, meteorology)

You may check our extensive survey on existing methods for classification and regression:
Paula Branco, Luis Torgo and Rita Ribeiro (2016). A Survey of Predictive Modeling on Imbalanced Domains. ACM Comput. Surv. 49, 2.

Imbalanced Distributions in Regression

- Frequently occur when the main goal of the end user are extreme values of the target - e.g. unusually high(low) returns of a stock
- These extreme values are usually rare, i.e. poorly represented in the training data.
- As within imbalanced classification, models will be biased towards the more frequent cases in the training data.
- As a result performance on the cases that matter will be disappointing.

Problem Definition

Predicting Rare Extreme Values

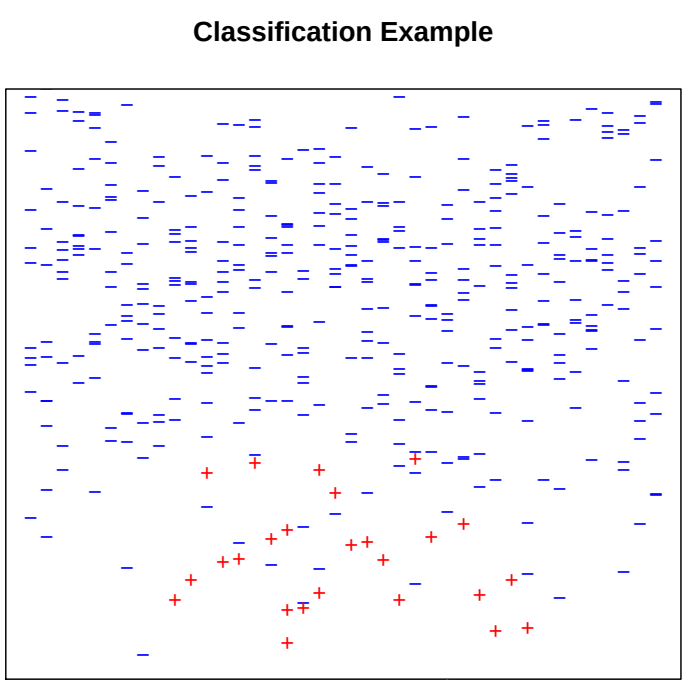
- Goal: obtain a good approximation of the unknown function $Y = f(X_1, X_2, \cdots, X_p)$
- Training set: $D = \{\langle \mathbf{x}_i, y_i \rangle\}_{i=1}^{n}$
- a subset of the range of the target variable values Y has an higher importance to the user
- the most important subset of Y is under-represented in the available training sample

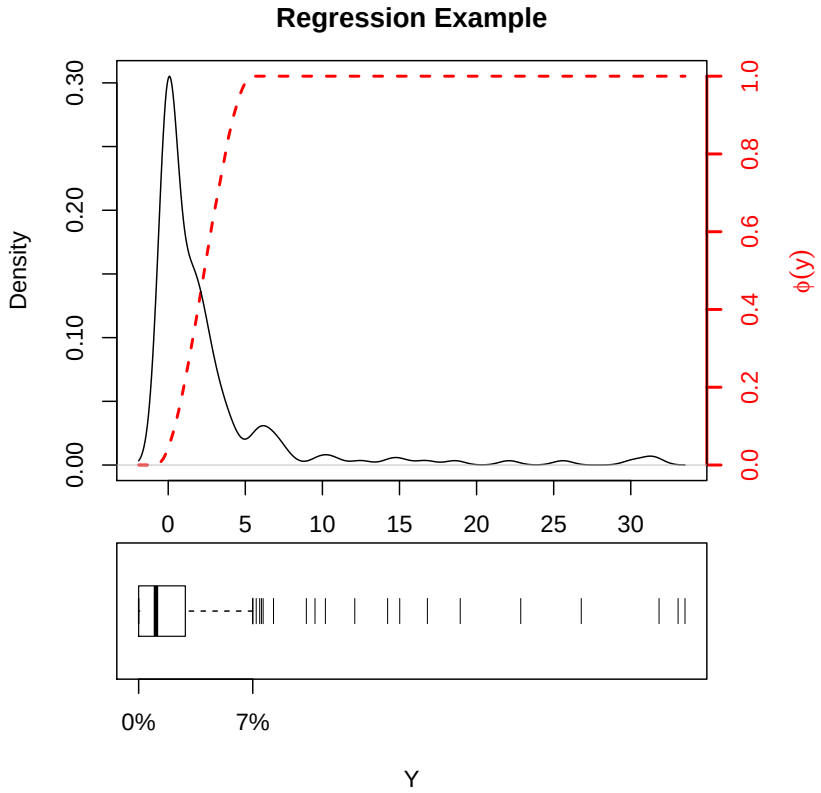
Problem Definition

More formally

- Given a relevance function: $\phi(Y) : \mathcal{Y} \rightarrow [0, 1]$, where 1 is maximal importance and 0 represents minimum relevance
- Ask user for a threshold t_R on relevance
- $\mathcal{Y}_R = \{y \in \mathcal{Y} : \phi(y) > t_R\}$ and $\mathcal{Y}_N = \mathcal{Y} \setminus \mathcal{Y}_R$
- given a training set D partition it in D_R where $y \in \mathcal{Y}_R$ and $D_N = D \setminus D_R$
- an imbalanced prediction tasks satisfies:
  - non-uniform importance of the predictive performance of the models across the domain of Y
  - $|D_R| << |D_N|$

Imbalanced Domains





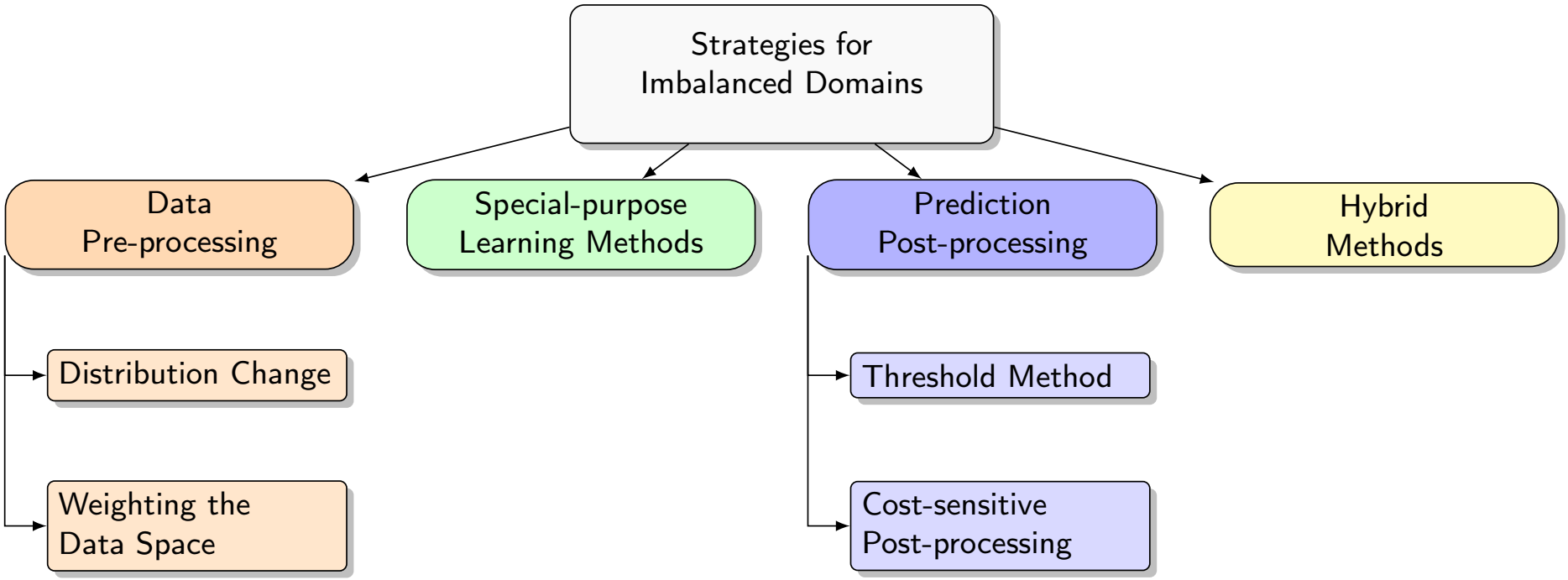
Problems created by imbalanced distributions

The combination of the specific preferences of the user with the poor representation of these situations creates problems at several levels.

Typically we need:

1. special purpose evaluation metrics that are biased towards the performance of the models on these rare cases,
2. making the learning algorithms focus on these rare events.

A Taxonomy of strategies for handling Imbalanced Domains



Strategies for Handling Imbalanced Domains

Data pre-processing

- **Goal**: change the examples distribution before applying any learning algorithm;
- **Advantages**: any standard learning algorithm can then be used;
- **Disadvantages**: difficult to decide the optimal distribution (a perfect balance does not always provide the optimal results); the strategies applied may severely increase/decrease the total number of examples;

Strategies for Handling Imbalanced Domains

Special-purpose learning methods

- **Goal**: change existing algorithms to provide a better fit to the imbalanced distribution;
- **Advantages**: very effective in the contexts for which they were designed; more comprehensible to the user
- **Disadvantages**: difficult task because it requires a deep knowledge of both the learning algorithm and the target domain; often unavailable cost-benefit matrix; difficulty of using an already adapted method in a different learning system;

Strategies for Handling Imbalanced Domains

Prediction post-processing

- **Goal**: change the predictions after applying any learning algorithm;
- **Advantages**: any standard learning algorithm can be used;
- **Disadvantages**: potential loss of models interpretability;

Relevance Function

- Handling imbalanced regression requires defining the important values
- This can be done through the definition of a relevance function, $\phi()$, that maps the domain of the target into a range of importance
- $\phi() \in [0, 1]$ where 0 represents values of the target variable that are not relevant and 1 identifies the most important values.

Relevance Definition

Ribeiro (2011) proposed a framework for defining the relevance function of a given continuous target variable. This framework:

- includes an automatic method that allows to obtain the relevance function from the target variable sample distribution (assumes extreme rare values are the most important to the user).
- allows the user to manually specify which are the relevant and irrelevant values using a matrix.

R. Ribeiro (2011): Utility-based regression. PhD thesis, Dep. Computer Science, Faculty of Sciences-University of Porto.

Resampling Strategies for Regression Tasks

- Random Undersampling
- Random Oversampling
- Introduction of Gaussian Noise
- SmoteR
- WEighted Relevance-based Combination Strategy (WERCS)

L. Torgo, P. Branco, R. Ribeiro and B. Pfahringer (2015). Re-sampling Strategies for Regression. Expert Systems, vol. 32 (3), pp. 465-476.

The R Package UBL

- We have created an R package (UBL) that implements a large set of approaches for both classification and regression
- We will use it for illustration purposes
- More information about the package (and further examples) can be obtained in:
  P. Branco, R. Ribeiro and L. Torgo (2016). A UBL: an R package for Utility-based Learning. CoRR abs/1604.08079.

Random Undersampling

- In Random Undersampling all cases with target value with a relevance lower then the threshold are candidates for undersampling
- User decides this threshold as well as the target proportion between normal (unimportant) and rare (important) cases in the produced data set.
- This resampled dataset is obtained by randomly removing examples from the uninteresting cases.

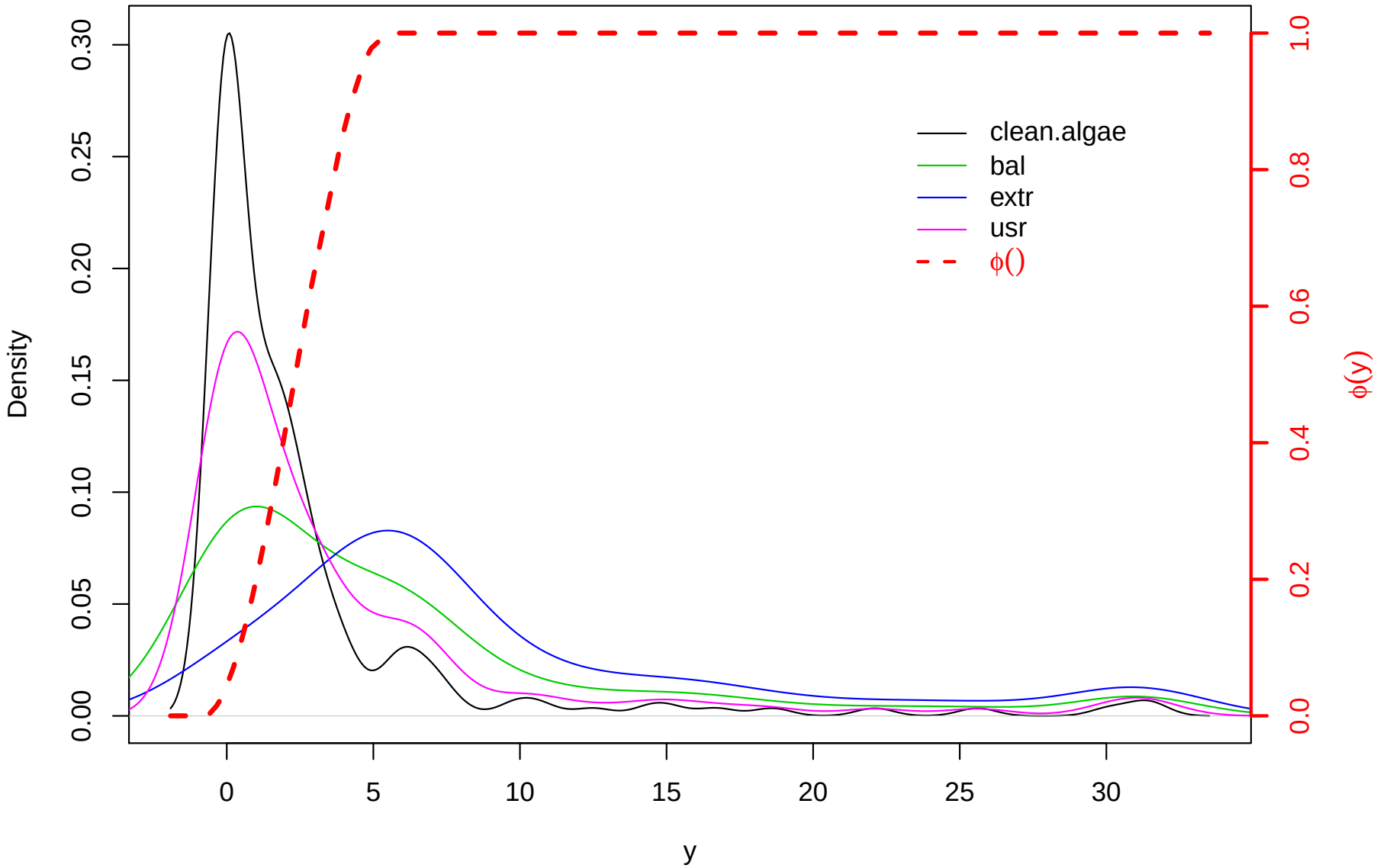
Random Undersampling in UBL

```
# Using the automatic method for defining the relevance function
# This is the default behaviour
  library(UBL)
  # default of C.perc parameter balances the examples
  bal <- RandUnderRegress(a7~., clean.algae)

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

  # the automatic method for the relevance function generates only
  # one "bump" with uninteresting values, thus we only need to set
  # one under-sampling percentage
  usr <- RandUnderRegress(a7~., clean.algae, C.perc = list(0.5))
```

Impact of different settings of Random Undersampling



Random Oversampling

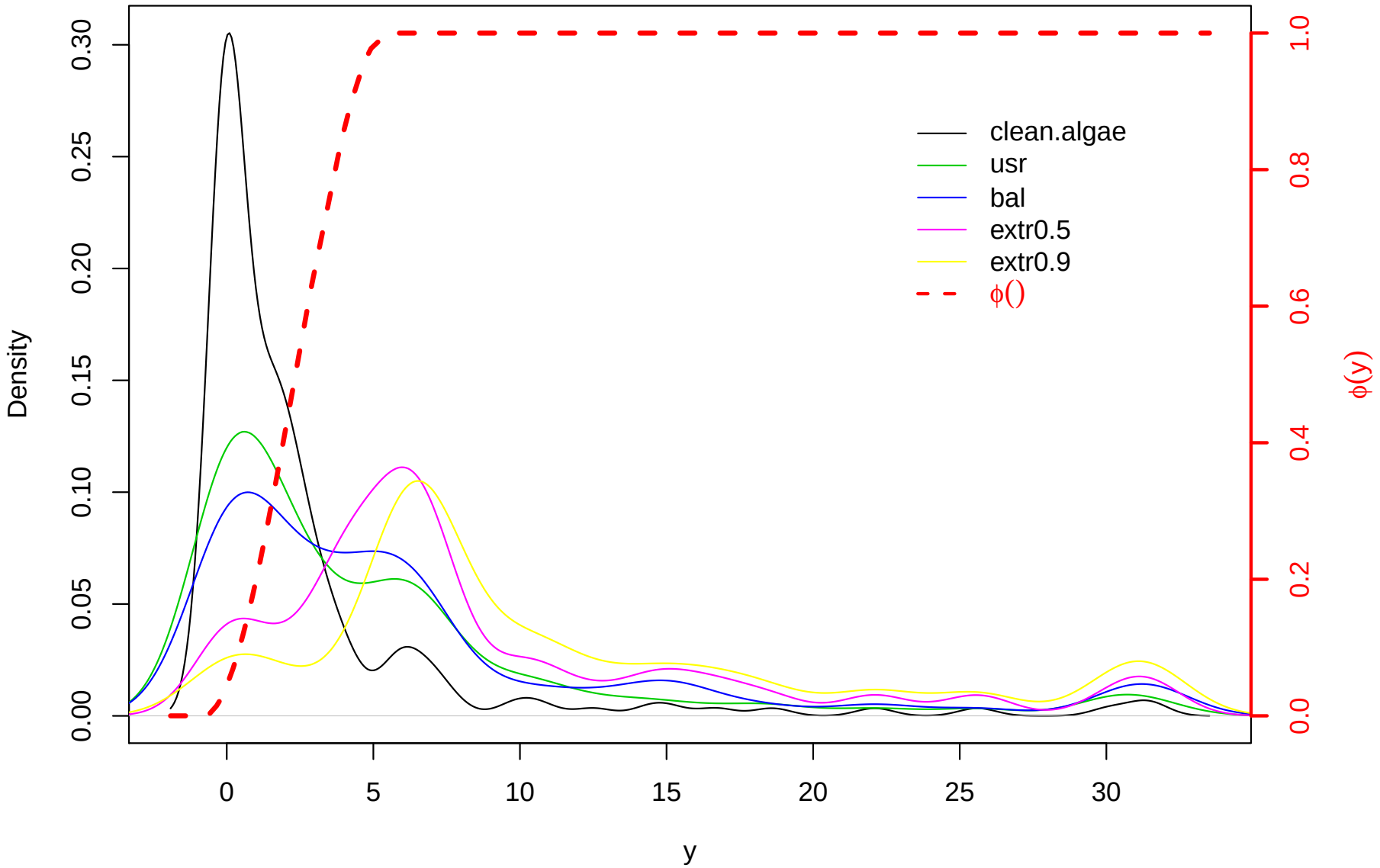
- The Random Oversampling approach is simply based on the introduction of random copies of examples from the original data set
- These replicas are only introduced in the most important ranges of the target variable, i.e., in the ranges where the relevance is above a user-defined threshold.
- User needs to define a relevance function, a relevance threshold and the percentage of oversampling to perform.

Random Oversampling using UBL

```
## using the automatic method for defining the relevance function and
## the default threshold (0.5)
usr <- RandOverRegress(a7~., clean.algae, C.perc=list(2.5))
bal <- RandOverRegress(a7~., clean.algae, C.perc="balance")
extr0.5 <- RandOverRegress(a7~., clean.algae, C.perc="extreme")

# change the relevance threshold to 0.9
extr0.9 <- RandOverRegress(a7~., clean.algae, thr.rel=0.9, C.perc="extreme")
```

Impact of different settings of Random Oversampling



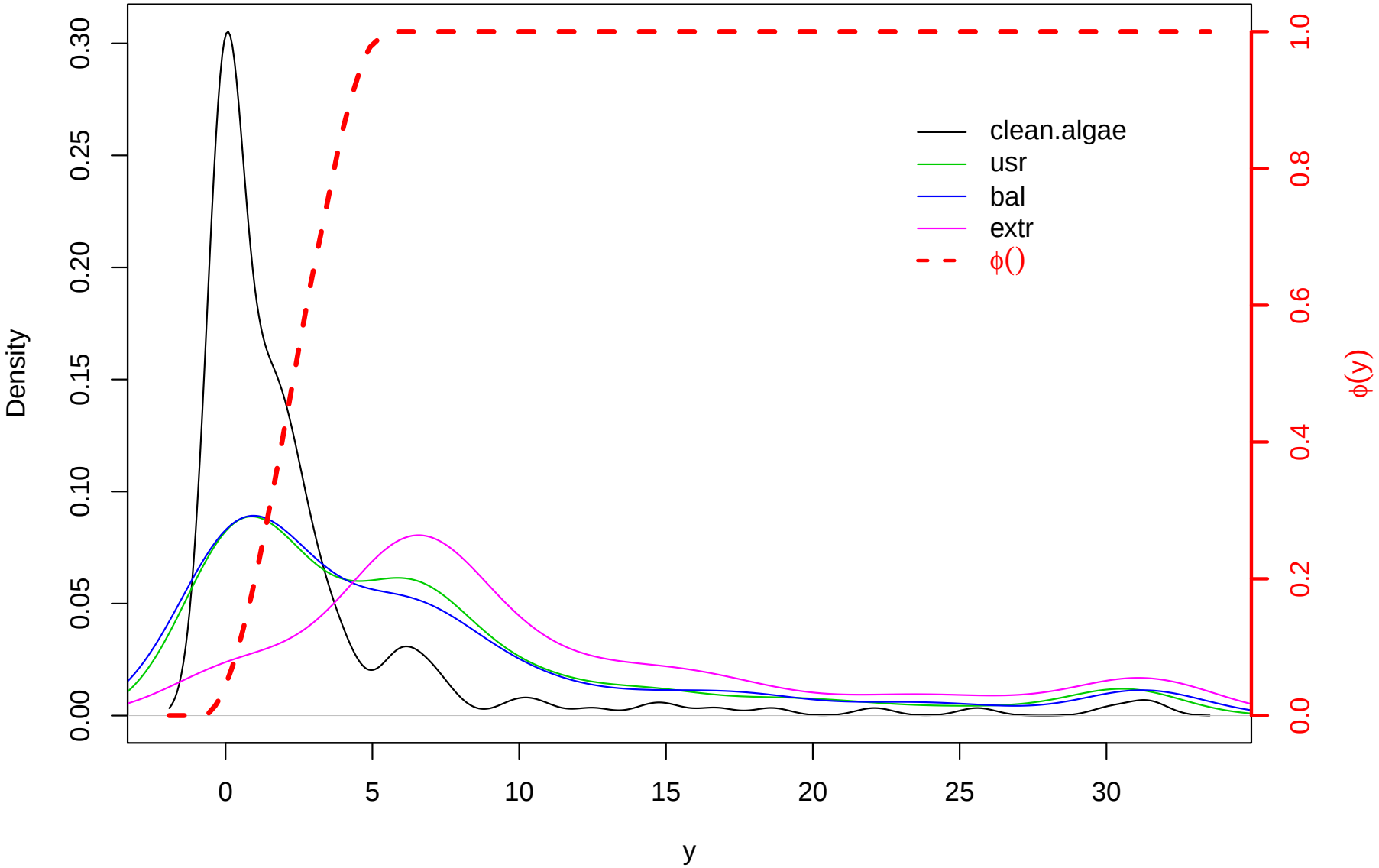
Introduction of Gaussian Noise

- This strategy combines oversampling by generating new synthetic examples with small perturbations with the random undersampling strategy.
- The relevance function and the user-defined threshold distinguish among the ranges where over and undersampling are applied.
- The target variable of the generate cases is obtained by introducing a small perturbation based on the sample standard deviation.

Introduction of Gaussian Noise in UBL

```
# relevance function estimated automatically has two bumps
# defining the desired percentages of under and oversampling to apply
C.perc <- list(0.5, 3)
# define the relevance threshold
thr.rel=0.8
usr <- GaussNoiseRegress(a7~., clean.algae, thr.rel=thr.rel, C.perc=C.perc)
bal <- GaussNoiseRegress(a7~., clean.algae, thr.rel=thr.rel, C.perc="balance")
extr <- GaussNoiseRegress(a7~., clean.algae, thr.rel=thr.rel, C.perc="extreme")
```

Impact of different settings of Gaussian Noise

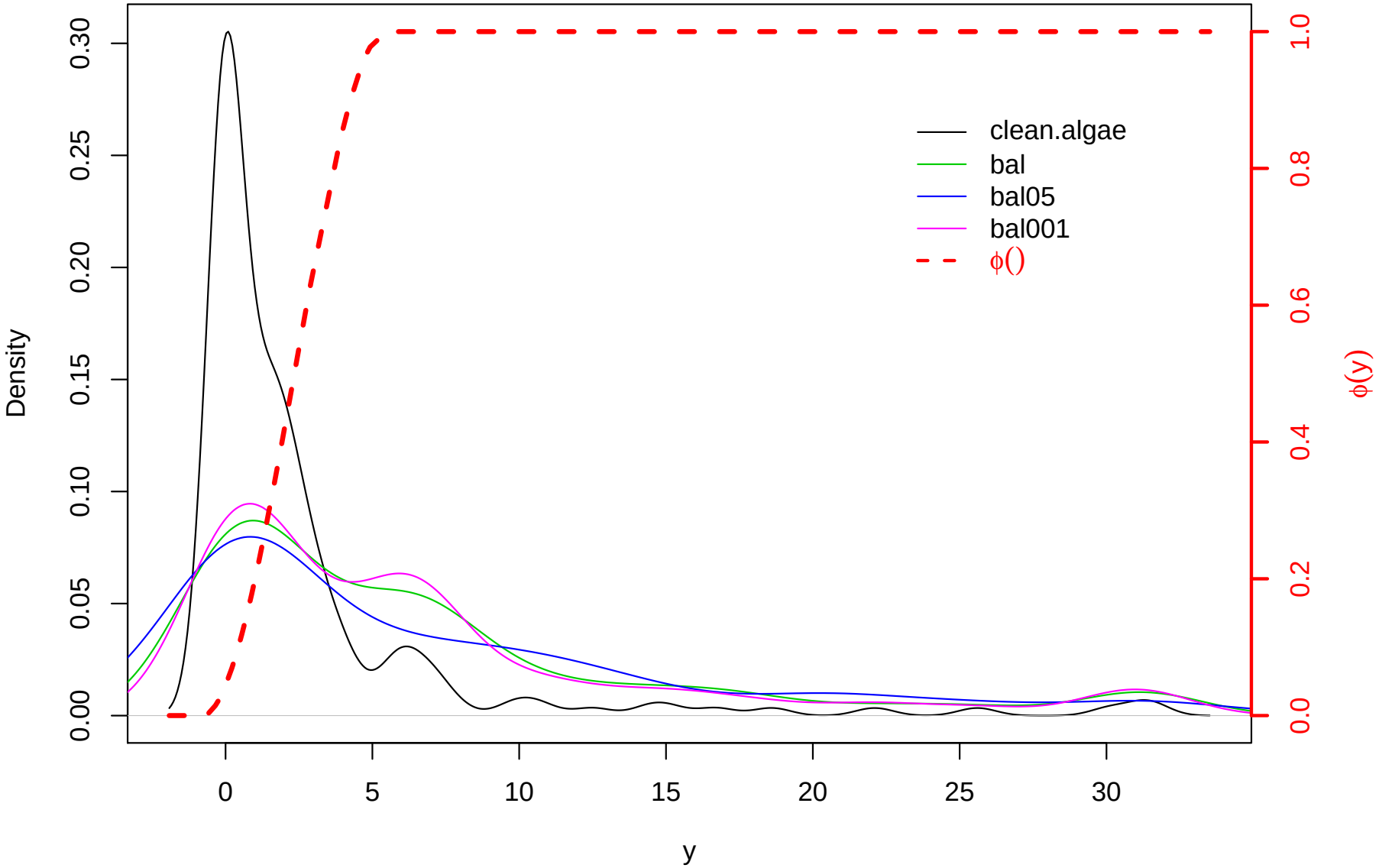


Further examples with Gaussian Noise

```
# the default uses the value of 0.1 for "pert" parameter
bal <- GaussNoiseRegress(a7~., clean.algae, thr.rel=thr.rel,
                         C.perc="balance")

# try two different values for "pert" parameter
bal05 <- GaussNoiseRegress(a7~., clean.algae, thr.rel=thr.rel,
                         C.perc="balance", pert=0.5)
bal001 <- GaussNoiseRegress(a7~., clean.algae, thr.rel=thr.rel,
                         C.perc="balance", pert=0.01)
```

The impact of changing the parameter pert



SmoteR - SMOTE for Regression Tasks

- Once again the relevance function and the relevance threshold determine which are the relevant and the unimportant cases.
- The algorithm combines an oversampling strategy by interpolation of relevant examples with a random undersampling approach.
- A similar procedure to the original Smote algorithm is used to generate new examples
- For the target variable value of the new examples a weighted average of the values of target variable of the two seed examples is used.
- The weights are calculated as an inverse function of the distance of the generated case to each of the two seed examples.

L. Torgo; R. Ribeiro; B. Pfahringer and P. Branco (2013): SMOTE for Regression, in 16th Portuguese Conference on Artificial Intelligence, EPIA 2013, pp. 378-389. LNAI - Springer

Using the SmoteR Algorithm

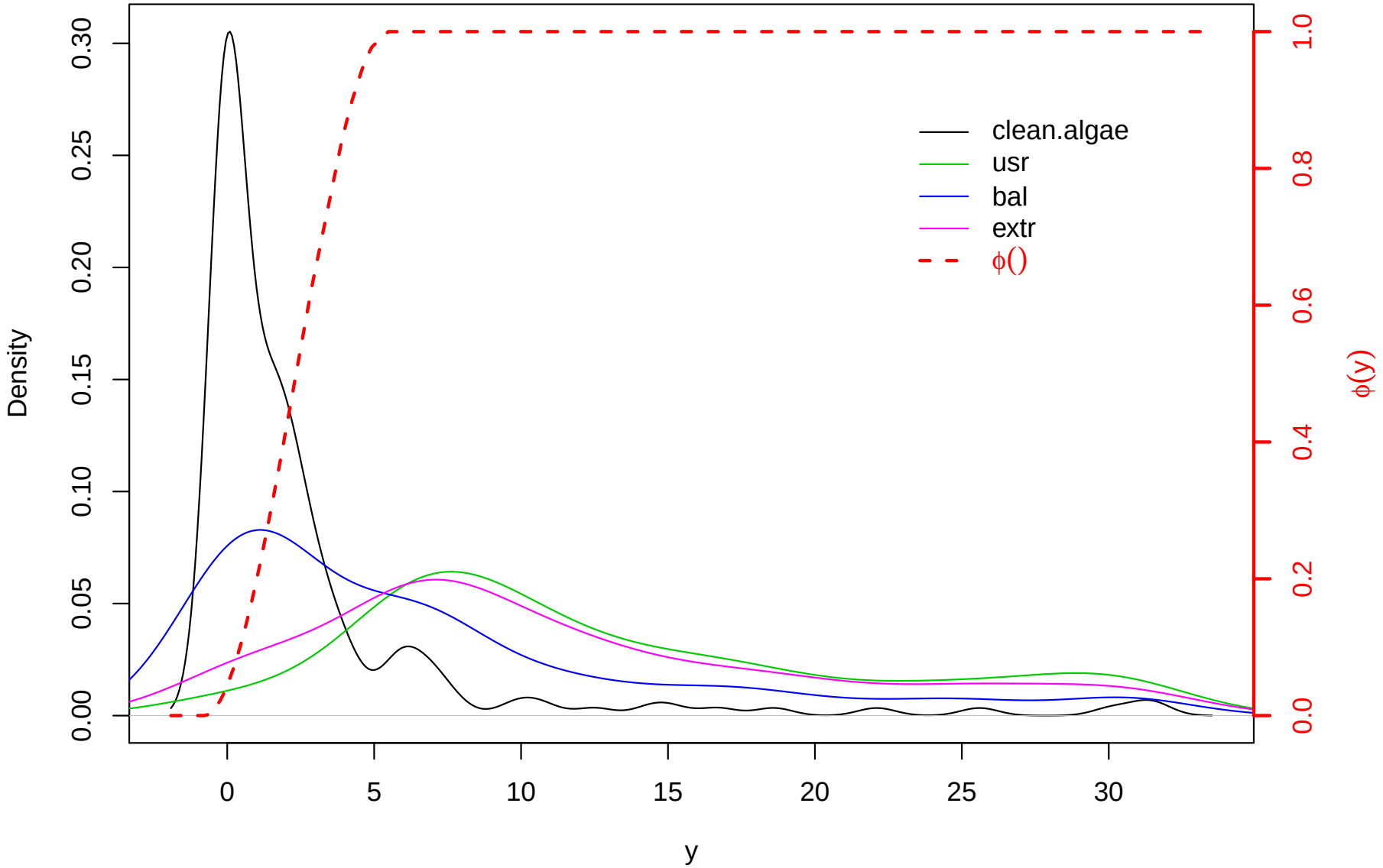
```
# we have two ranges: the first must be undersampled and the second oversampled.
# Thus, we can chose the following percentages:
thr.rel <- 0.8
C.perc  <- list(0.1, 8)

# using these percentages and the relevance threshold of 0.8 with all the
# other parameters default values
usr <- SmoteRegress(a7~., clean.algae, thr.rel=thr.rel, C.perc=C.perc,
                    dist="HEOM")

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

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

The impact of the parameters on SmoteR



WEighted Relevance-based Combination Strategy (WERCS)

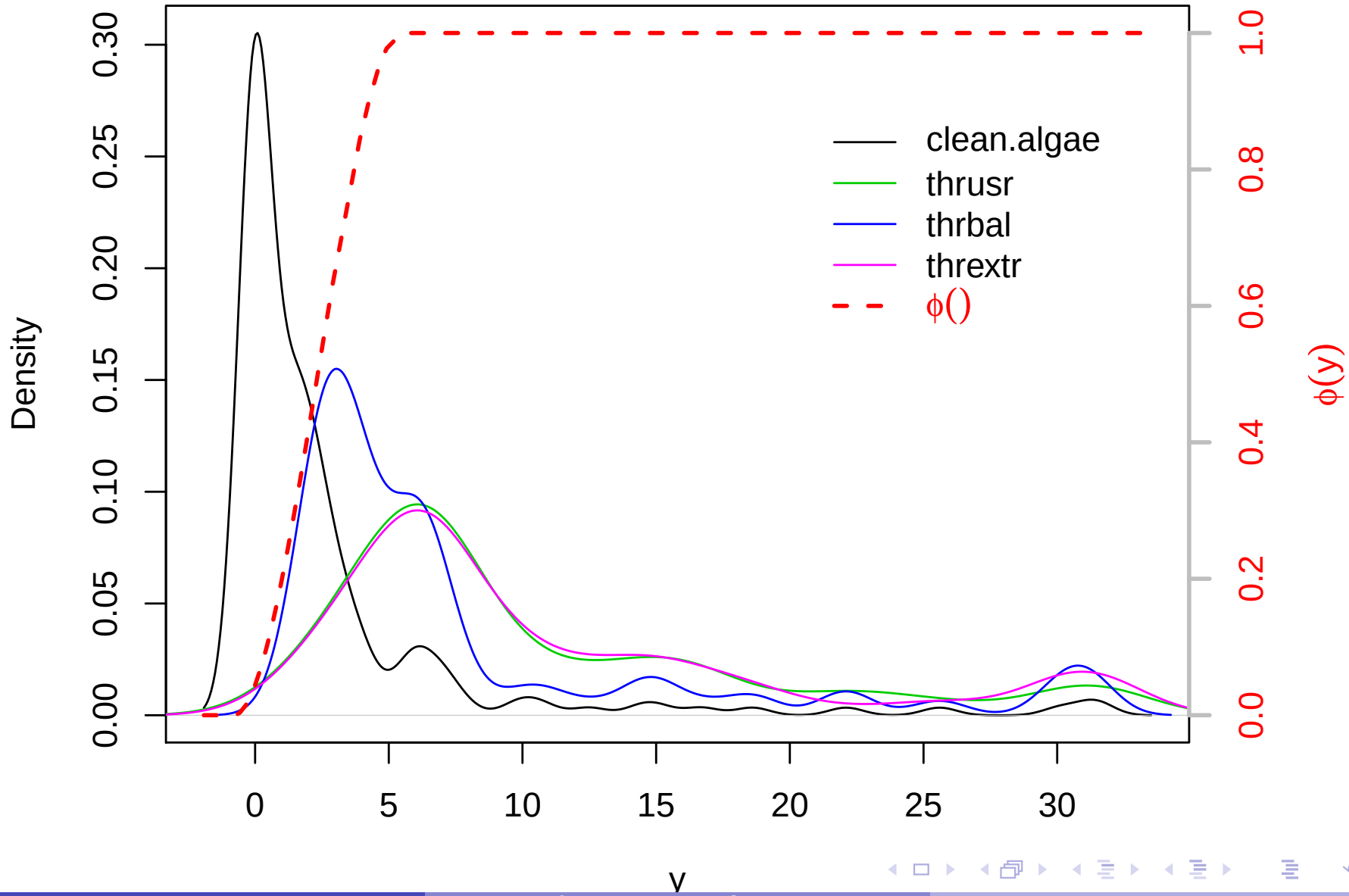
- WERCS key idea is to use the relevance function scores as probabilities for resampling the examples.
- Examples are selected for being either removed or added as replicas using these probabilities.
- In oversampling examples with higher relevance have higher probability of being replicated.
- In undersampling examples are randomly selected to be removed with probability $1 - \phi(y)$, i.e, the higher the relevance value of an example, the lower will be the probability of being removed.
- User is not required to set a relevance threshold

WERCS in UBL

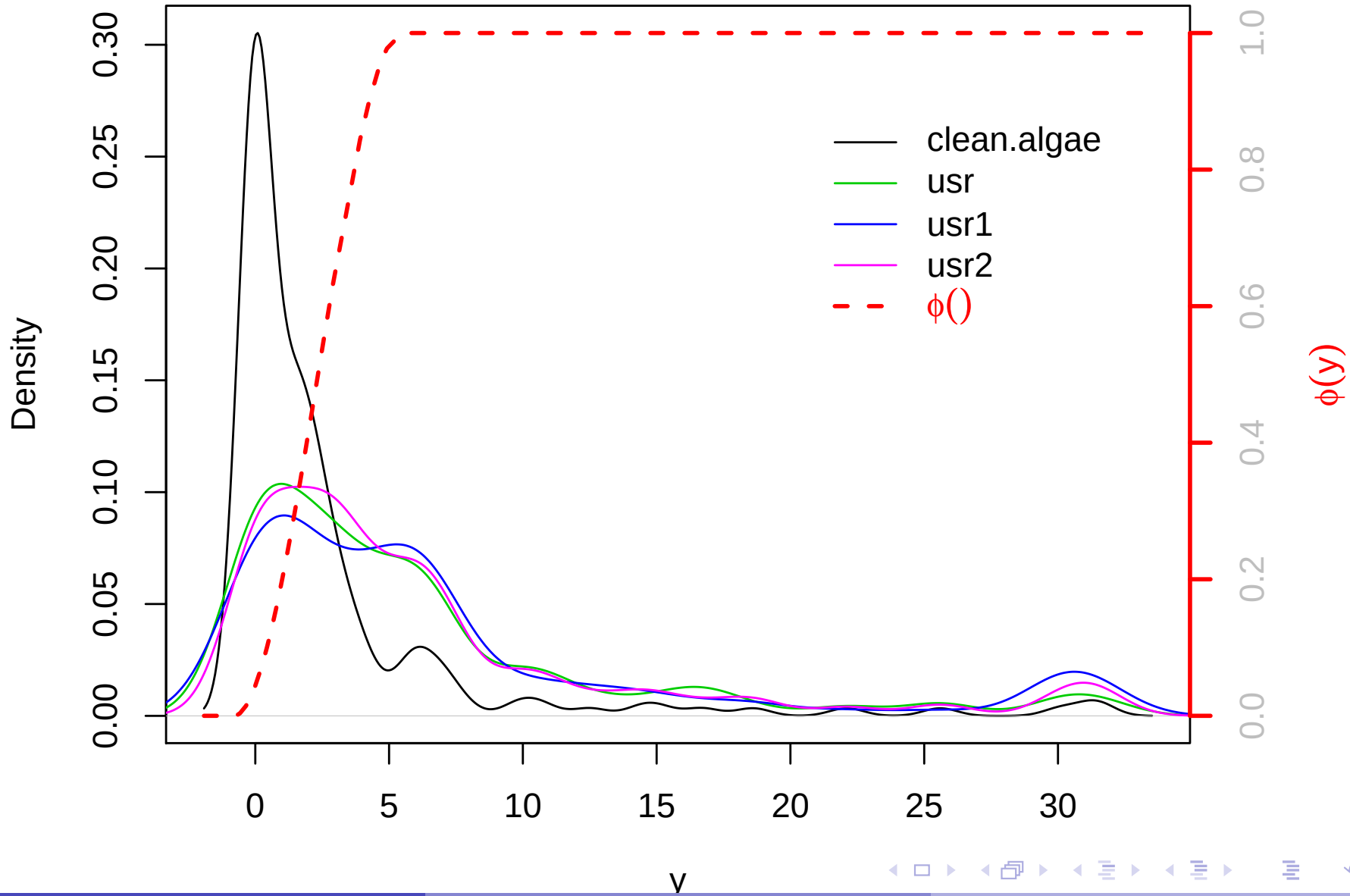
```
# using importance sampling with threshold definition
C.perc=list(0.2,6)
thrusr <- ImpSampRegress(a7~., clean.algae, thr.rel=0.8, C.perc=C.perc)
thrbal <- ImpSampRegress(a7~., clean.algae, thr.rel=0.8, C.perc="balance")
thrextr <- ImpSampRegress(a7~., clean.algae, thr.rel=0.8, C.perc="extreme")

# importance sampling without threshold
usr <- ImpSampRegress(a7~., clean.algae)
usr1 <- ImpSampRegress(a7~., clean.algae, U=0.9, O=0.2)
usr2 <- ImpSampRegress(a7~., clean.algae, U=0.2, O=1)
```

The impact of the parameters on WERCS - 1



The impact of the parameters on WERCS - 2

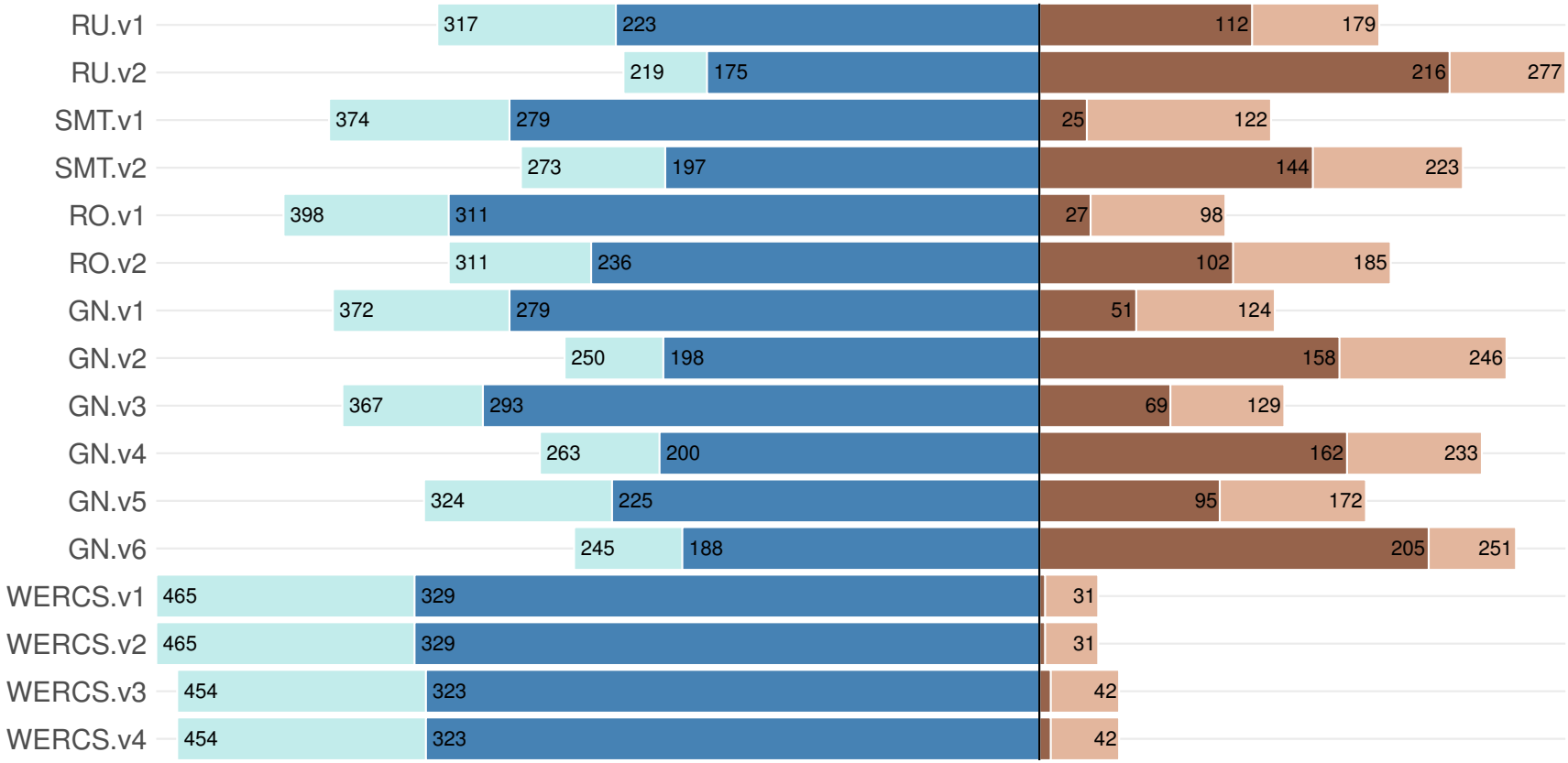


Experimental Analysis of the Methods

- Resampling strategies considered:
  - No resampling - original data (the baseline)
  - 2 variants of Random Undersampling (RU)
  - 2 variants of Random Oversampling (RO)
  - 2 variants of SmoteR (SMT)
  - 6 variants of Gaussian Noise (GN)
  - 4 variants of WERCS
- Several regression algorithms (31)
  - 1 LM + 8 NNET variants + 4 MARS variants + 12 SVM variants + 6 RF variants
- 15 regression data sets with different characteristics
- **A total of 7905 ($15 \times 31 \times 17$) alternatives**
- The values of F_1 for regression were estimated by means of 2×10 - fold cross validation process and the statistical significance of the observed paired differences was measured using the non-parametric Wilcoxon Sign Rank test

A summary of the results

Total number of wins (left - blue) and losses (right - brown) of sampling strategies, against the baseline of using the original imbalanced data set.



Summary/Conclusions/Recommendations

- Random forests (RF) with either **RO**, **GN** or **WERCS** achieved the best performance in 10 out of 15 data sets
- **WERCS** achieved the most consistent performance when compared to using the original data.
- **WERCS** is also more "user-friendly" as it does not require setting a relevance threshold
- It is also computationally more efficient as it does not involve generating new cases (just replicating) and it does not increase the training set sizes

Resampling Strategies for Regression

Paula Branco, **Luis Torgo**, Rita Ribeiro and Bernhard Pfahringer

Departamento de Ciência de Computadores
Faculdade de Ciências/Universidade do Porto

January, 2017