Applied Intelligence
https://doi.org/10.1007/s10489-024-05608-6

# WSMOTER: a novel approach for imbalanced regression

**Luís Camacho**[1] · **Fernando Bacao**[1]

Accepted: 10 June 2024


**Abstract**
Although the imbalanced learning problem is best known in the context of classification tasks, it also affects other areas of learning algorithms, such as regression. For regression, the problem is characterized by the existence of a continuous target variable domain and the need for models capable of making accurate predictions about rare events. Furthermore, such rare events with a real-value target are often the ones with greater interest in having models that can predict them. In this paper, we propose the novel approach WSMOTER (Weighting SMOTE for Regression) to tackle the imbalanced regression problem, which, according to the experimental work we present, outperforms currently available solutions to the problem.

**Keywords** Imbalanced · Regression · Data-level · Supervised learning

## 1 Introduction

Many real-world applications need the prediction of rare and extreme values in the context of forecasting problems in which the target variable domain is continuous [1]. Examples of such applications from a diverse set subject areas include: hydrology [2], environmental science [3] and urban studies [4], among many others.

In the first mentioned example, Snieder et al. [2] evaluated the performance of machine learning models in the prediction of high flows in flood warning systems. In the second example, Saha et al. [3] used models to make predictions of nitrous oxide ($N_2O$) emissions from agricultural activity, which is a gas that impacts global warming. Ranacher et al. [4] presented a model for estimating urban road traffic, namely the energy efficiency of car movement patterns.

Although these real-life examples come from a wide range of different fields, they all have some characteristics in common. These applications require regression modelscapable of making accurate predictions regarding rare events. This ability is what most interests end users. For example, in the scope of flood warning systems, end users are interested in forecasting high flow events. Considering the observed flow data, there are many more examples of low-flow data than high-flow data, hence, the occurrence of high flows is a rare event [2]. This skewed distribution of the observed data is an issue for machine learning algorithms as it hinders learning, the problem being known by imbalanced learning [5, 6].

The research community has already introduced some proposals to tackle the imbalanced learning problem concerning regression tasks, such as Torgo et al. [7], Branco et al. [8], and Steininger et al. [9]. However, solutions to this problem are still scant, and there are far fewer alternatives for imbalanced regression than for imbalanced classification [10]. Thus, the development of new algorithms that allow getting better predictions of rare events is an important task.

Our main contributions are:

- We introduce WSMOTER, a data-level approach to tackle the imbalanced regression problem.
- WSMOTER is lined up with the nature of regression tasks, and this differentiates it from other solutions, namely: there are no arbitrary thresholds that split the domain into partitions of rare data points and partitions of common data points. The target variable domain is continuous, and each data point is treated individually in this domain.

✉ Luís Camacho
d20190051@novaims.unl.pt

Fernando Bacao
bacao@novaims.unl.pt

[1] NOVA Information Management School (NOVA IMS), Universidade Nova de Lisboa, Campus de Campolide, Lisboa 1070-312, Portugal

Published online: 26 June 2024

- WSMOTER was compared with state-of-the-art currently available solutions.

This article has the following sections. Section 2 defines the imbalanced regression problem, and Section 3 reviews related work. Section 4 introduces our proposal to improve the performance of the models. Section 5 describes the research methodology, and Section 6 shows the results. Lastly, Section 7, presents the conclusions.

## 2 Problem definition

In supervised machine learning, regression problems are solved through algorithms that produce models from a training set of real-value examples.

Thus, given a training set $\{(x_i, y_i)\}$ with $i = 1, ..., N$ and where $N$ is the number of examples, $x_i$ is a feature vector, and $y_i$ is the target; the goal is to find a model $f$ such that $f(x_i)$ approaches $y_i$. The model can then be used to make predictions for unlabeled examples.

Being a regression problem, the target variable domain is continuous. If this domain contains ranges of most interest to end-users, for which there are few examples in the training set whose label matches them, then we can say that an imbalanced regression problem is faced. Below, we present the problem definition:

**Definition 1** *An imbalanced regression problem is meant a regression task where the following two conditions occur cumulatively:*

- *The target distribution of the training set examples is skewed. There are ranges of the target variable domain in which the density of samples is lower.*
- *There is a particular interest in the model's prediction ability regarding examples whose target belongs to low-density ranges of the domain.*

The simultaneous occurrence of the two conditions referred to in the definition becomes a problem because algorithms are designed to minimize the error, and as there are few examples in the training set of those that most interest the user, then these have little influence on error minimization, leading to a bias of the algorithm to examples of denser ranges [11].

Finally, it should also be noted that in many real-life applications, the problem of imbalanced regression manifests itself towards the extremes of the distribution of the target values, this being the range of most interest. Moreover, the examples whose target value is extreme are scarce. Therefore, for this type of application, a model capable of accurately predicting extreme events is needed. For example, improving the accuracy of high flow forecasts is paramount in the real-life application of flow forecasting in riverine flood warning systems.

## 3 Related work

Improving the predictive ability of machine learning models is the main purpose of research in imbalanced regression. However, before developing new algorithms with better performance, it is necessary to identify the rare instances of the data set. In classification tasks, the rare instances belong to minority classes, i.e., classes with fewer examples. Regarding regression, there is a target variable domain that is continuous rather than a finite set of classes. As such, the definition of rare instances cannot be done the same way as in classification and a mechanism to measure the rarity of the examples is needed.

Two different ways have been used in the literature. Steininger et al. [9] proposed an approach, called DenseWeight, to measure the rarity of each existing data point according to the density of the range of the target variable domain to which its target value belongs. Thus, the data points are weighted according to their importance from the point of view of their target values. The proposed approach is based on the density function obtained using kernel density estimation. Sadouk et al. [12] also used kernel density estimation, such as in DenseWeight, but different procedures were used. With these methods, there are not made assumptions about the location of the rare data points since these can be in any range of the target variable domain.

Another approach is the proposal of Ribeiro [13], that uses boxplot statistics to get a relevance function. This function then assigns a relevance value to each data set example. The rare are those whose relevance is above the relevance threshold indicated by the end-user.

This notion of rare instance and the relevance function that gave rise to it, are appropriate for a concept of rarity that associates the rare points to the extremes of the distribution of the target values. While other ranges of the target variable domain might be regions of low density of examples in which the end-user is interested, on the other hand, it is also true that for many real-world applications, the instances that matter are examples that combine rarity with extreme events. For example, in [14], machine learning methods were used to predict large wildfires. Since forest fire records show that the number of large fires is much lower than the one of typical fires, this is an example where rare events coincide with extreme values. Furthermore, large wildfires are the most relevant due to the consequences that may arise from them.

Returning to the topic of obtaining better performing machine learning models in predicting rare cases, different methods to solve the problem have been already proposed by the research community, such as: data-level solutions,

cost-sensitive learning, the introduction of modifications at the algorithm level, and ensemble-based solutions combined with other techniques.

The idea of cost sensitive learning in the context of imbalanced learning is to adapt the loss function, assigning different costs to errors depending on the class the instances belong to [15]. Cost-sensitive learning often takes the form of a training set weighting scheme so that rare instances can increase their importance through a greater penalty for errors on these instances during training [15].

DenseLoss [9] is one example of a cost sensitive solution, among the few existing alternatives to the imbalanced regression problem. This approach is a combination of DenseWeight with sample weighting for loss functions. DenseWeight is used to measure the rarity of the instances according to their target values distribution, resulting in the assignment of weights to the instances, which are then incorporated into the loss function. Steininger et al. [9] only used neural networks in his work, although the concept can be used in other algorithms.

By ensemble learning is meant the combined use of multiple learners. One well-known example of this kind of method is boosting. It consists of a sequential training process in which each learner is trained with the aim of correcting the mistakes of the previous ones [16]. Adaboost [17] is a representative algorithm of a boosting method, being considered one of the most influential [16]. However, using ensemble learning does not solve the imbalance problem by itself, but only when used in conjunction with some of the other techniques [18]. Two possibilities are: the introduction of a data level solution or the use of costs [16].

Data-level approaches are popular methodologies to deal with imbalanced data sets in classification tasks [19]. Therefore, it is not surprising that some proposals have also emerged to tackle imbalanced regression tasks, although in much smaller numbers compared to the number of existing options for imbalanced classification.

Torgo et al. [7] adapted two data-level methods used in classification tasks. The first proposal is to apply Random Undersampling to the normal instances of the training set. These are the instances whose relevance value, given by the relevance function proposed by Ribeiro [13], is below the threshold designated by the end-user. The purpose is to balance the number of rare and normal instances. The second proposal is SmoteR, which is an adaptation to the regression of the well-known SMOTE algorithm [20]. Synthetic instances are generated from the examples whose relevance value is above the threshold indicated by the end-user. One of the main modifications implemented is related with the label of the model's predictions, which in the case of regression is a real value rather than a minority class of classification problems. The proposed solution is based on the Euclidean distances of the synthetic instance to the two seed examples used to generate it. On the other hand, instances whose relevance is below the threshold are randomly undersampled.

Despite the pioneering role played by SMOTE, several limitations motivated the appearance of other proposals in classification problems [21]. In the same way, SmoteR algorithm can be seen as a first attempt to use a SMOTE-based technique in regression [1].

Branco et al. [22] introduced SMOGN. This method is a combination of SmoteR with the introduction of Gaussian Noise. The latter generates synthetic instances through the addition of noise to already existing rare cases.

Other solutions to imbalanced regression at data-level include yet proposals such as [8] and [23]. However, almost all of the data-level proposals are based on a concept of splitting the data set into partitions of rare cases and partitions of common cases, which are then used to perform oversampling or undersampling. Moreover, the line that separates both partitions is often based on an arbitrary threshold defined by the user.

## 4 The proposed method

### Motivation

The basic idea that inspired us to propose WSMOTER was to offer a solution for imbalanced regression more in tune with the nature of regression problems. Bearing this in mind, concepts based on splitting the data set into partitions were avoided. Moreover, the use of any arbitrary thresholds to distinguish rare from common instances was also not welcome. Finally, there was always a concern to improve the performance of models trained on data resampled with WSMOTER.

Intuitively, the main idea of WSMOTER is as follows: each real instance belonging to the training set can be used as a seed to generate synthetic instances according to the rule, the greater the rarity in the target domain, the greater the probability of being selected. Once a first seed has been selected, it is necessary to select one more, as two instances are needed to generate a synthetic instance. The original SMOTE only needs to choose one of the neighbors of the first seed, in the minority class. Since there is no minority class neither a partition where to search, the selection process of the second seed has to be different. WSMOTER searches for the second seed in the neighborhood of the first seed, also considering the neighborhood from the point of view of the target values, beyond the feature space.

WSMOTER is a data-level solution for imbalanced regression derived from the known SMOTE preprocessing algorithm [20]. Since the original SMOTE was designed to imbalanced classification problems, then important mod-

ifications need to be introduced to make it suitable for imbalanced regression problems. The next paragraph enumerates the key ideas.

### WSMOTER key ideas

- Original SMOTE is applied to instances from minority classes. Since in regression there are no classes but a continuous target domain, a mechanism to assign a value to each instance to express its rarity is needed. DenseWeight was incorporated into WSMOTER to provide to this preprocessing algorithm with such a mechanism.
- In the selection process of the already existing instances to use as seeds of synthetic instances, the original SMOTE only has to randomly pick up instances from a minority class. WSMOTER considers the instances' rarity as weights. This strategy allows higher weight instances to have a greater chance of being selected as they are considered rarer.
- Regarding the selection of the second seed, WSMOTER first searches in the neighborhood of the first seed from the point of view of the target values, and then the feature space.
- After the selection of the two seeds, the process of instances' interpolation of the original SMOTE is used.
- The assignment of a target value to the generated instance is a trivial task for classification problems but not for regression problems. The original SMOTE automatically assigns to the new instance the minority class of its seeds. In the context of regression, a numeric value is expected rather than a class since the domain of the target variable is continuous. For this issue, it was adopted the same solution of SmoteR. The target value is the weighted average of the target values by the Euclidean distance of the new instance to its seeds. Thus, the closer the new instance is to one of its seeds, the closer its target value is to the value of this seed.

Despite sharing the same procedure to assign a target value to a new synthetic instance, SmoteR and WSMOTER are clearly distinct. SmoteR uses an arbitrary threshold to separate the target domain into partitions of common instances and partitions of rare instances, to then use the partitions of rare to generate synthetic instances. In WSMOTER, every instance can be used to generate new instances and what distinguishes them is that some are more likely to be selected than others. Also, the target values of the instances are considered in the selection process, as referred in the previous bullet points.

Once the main ideas are presented, it is necessary to provide some comments as well as more details on how the ideas were implemented. The next enumeration follows the order of the WSMOTER key ideas bullet points.

1. Assigning a value to each instance to express its rarity could be done with either of the two techniques described in Section 3. We decided to use DenseWeight because this technique makes no assumptions about the location of the rare instances and in case the rare ones coincide with the extremes, they will be also identified and will receive the appropriate weight.
2. The selection process is done with replacement.
   Before the final assignment of the weights to respective instances, the weights are further normalized by dividing each rarity value by the total sum of the rarity values. The normalized value is then the final weight assigned to each instance.
3. Finding another real instance to pair with the one that had already been selected is somehow a difficult task because there is no minority class neither a partition where to look for. Also, it is not a good idea to look for neighbors in the entire feature space as it would be computationally demanding and without guaranteed effectiveness. The procedure to accomplish this item is as follows:
   (a) the values of the entire target variable domain are sorted and stored in a new variable.
   (b) the new variable is used to look up for the position of the target value of the first seed.
   (c) from this position, a shift of $k$ positions is made to the right. For first seeds with a target value below the median, the shift is made to the left.
   (d) the target value of the new position is picked up .
   (e) the search process of the neighbors on the feature space is limited to instances whose target values are between the target value of the first seed and the target value founded in the previous step.
   (f) one of the neighbors is randomly selected from among those that respect the condition referred to in the previous step.
4. Once the selection phase is completed, the generation of new instances begins. Here the general process of interpolation of the original SMOTE is used.
5. Finally, a numeric value is assigned to the new instance according to the described process in the WSMOTER key ideas paragraph.

### Remarks

Regarding the item (c) of the enumeration, we used $k = 10$ in our work. Although other values can be used, it is wise not to choose numbers that are too small or too large. A small number would narrow the search of neighbors and a

big number would make the search process unnecessarily slower.

Still in the item (c), at the upper extreme of the ordered version of the target values, the last $k$ values do not have enough values on their right. For these cases, are also used values located on the left of the target value of the first seed. At the lower extreme, there is an analogous situation.

**Algorithm 1** WSMOTER.

**Input:** $X, Y, \alpha, N, K$
// $\alpha = 1$ : DenseWeight hyperparameter
// $N$ : the final number of instances, in the shape $(N_{<median}, N_{>=median})$
// $K$ : number of neighbors
**Output:** $X_{resampled}, Y_{resampled}$

**function** DENSE($Y$)
// return rarity of each $y \in Y$
return $DenseWeight_{\alpha}(Y)$
**end function**

**function** MAKESAMPLES($X, Y, Y_{weights}, kneigh(X), Ni$)
$weights \leftarrow \frac{Y_{weights}}{\sum_{i=1}^{n} Y_{weights}(i)}$
// $n : |Y_{weights}|$
$S_1 \leftarrow$ randomly select $Ni$ instances of $X$ according to the weights
$S_2 \leftarrow$ for each $x \in S_1$ randomly choose one of its kneigh($x$) neighbors
$X_{new} \leftarrow$ GenerateSyntheticInstances($S_1, S_2$)
// $X_{new}$ is generated following the general procedure of the SMOTE
$Y_{new} \leftarrow$ weighted average of the target values of $S_1$ and $S_2$
return $< X_{new}, Y_{new} >$
**end function**

$X_{resampled} = X$
$Y_{resampled} = Y$
$Y_{weights} \leftarrow$ DENSE($Y$)
**for** $< X, Y >$ in $\{< X, Y >_{y_{med}=1}, < X, Y >_{y_{med}=2}\}$ **do**
$Y_{weights} \leftarrow Y_{weights}(Y)$
$Y_{sorted} \leftarrow$ sort($Y$)
**for** $< x, y > \in < X, Y >$ **do**
$y_{pos} \leftarrow$ position of $y$ in $Y_{sorted}$
// $k$ : shift on the $Y_{sorted}$
**if** $y_{med} = 1$ **then**
$interval \leftarrow [Y_{sorted}(y_{pos} - k), y]$ if $y_{pos} - k >= 0$ else $Y_{sorted}[: k]$
**else**
$n \leftarrow |Y_{sorted}|$
$interval \leftarrow [y, Y_{sorted}(y_{pos} + k)]$ if $y_{pos} + k < n$ else $Y_{sorted}[-k :]$
**end if**
kneigh($x$) $\leftarrow$ determine $K$ neighbors among instances whose target value $\in interval$
**end for**
$< X_{new}, Y_{new} > \leftarrow$ MAKESAMPLES($X, Y, Y_{weights}, kneigh(X), Ni$)
// $Ni$ : number of instances to generate
$< X_{resampled}, Y_{resampled} > \leftarrow < X_{resampled} \cup X_{new}, Y_{resampled} \cup Y_{new} >$
**end for**

The pseudo-code of WSMOTER can be found in Algorithm 1.

## 5 Research methodology

It is important to compare WSMOTER's performance with state-of-the-art solutions like DenseLoss, which is a combination of the DenseWeight method with sample weighting for loss functions on neural networks. As DenseLoss has outperformed the data-level solution SMOGN in an experimental study [9] and the authors of the SMOGN have already stated that SMOGN present advantages when compared to the singular use of either SmoteR or the introduction of Gaussian Noise, it seems that the most pertinent comparison is between WSMOTER and the two methods: DenseLoss and a Multi-layer Perceptron regressor as baseline. This is our first experiment.

We proceed with a second experiment to confirm that the results obtained can be generalized to other algorithms, resorting to the Random Forest algorithm. The approaches in comparison were: the single use of Random Forest as baseline; cost-sensitive Random Forest; Adaboost with sample weighting and Decision Tree as base estimator, and the combination of WSMOTER with Random Forest.

### 5.1 Data sets

Thirty-six data sets were used to assess the performance of WSMOTER. These include all the 20 real data sets used in paper [9] to compare DenseLoss against SMOGN and other 16 datasets, mainly from the Keel repository [24]. Many of these data sets were also used in other papers.

The data sets used in the study have different characteristics regarding the number of instances, features and rare cases. Of the 36 data sets, 15 include nominal and numeric features and the remaining have only numeric features. Table 1 presents the characteristics of all data sets.

### 5.2 Metrics

Two preliminary remarks have to be made about the metrics used in imbalanced scenarios. Firstly, the unique use of standard metrics such as Mean Absolute Error and Mean Squared Error is not the most adequate to imbalanced regression, as these metrics are only focused on the extent of errors and do not consider their location [11]. Secondly, due to the limitation mentioned in the previous remark, more suitable metrics for imbalanced regression have been proposed in the literature. However, there is no benchmark metric in the field of imbalanced regression. Different works have used different metrics, which somehow makes it difficult to analyze the results.

For example, the $F_1$-measure for regression has been one of the most used in previous works about imbalanced regression. However, this metric has also some limitations, namely the reliance on an arbitrary user-defined threshold to distin-

**Table 1** Data sets characteristics: number of instances and features

| Data set | Instances | Features | Data set | Instances | Features |
|---|---|---|---|---|---|
| a1 | 198 | 12 | cpuAct | 8192 | 22 |
| a2 | 198 | 12 | cpu_small | 8192 | 13 |
| a3 | 198 | 12 | Delta_ailerons | 7129 | 6 |
| a4 | 198 | 12 | Delta_elv | 9517 | 7 |
| a5 | 198 | 12 | ele-1 | 495 | 3 |
| a6 | 198 | 12 | Elevators | 16599 | 19 |
| a7 | 198 | 12 | ForestFires | 517 | 13 |
| Abalone | 4177 | 9 | Fuel_cons_country | 1764 | 38 |
| Acceleration | 1732 | 15 | Heat | 7400 | 12 |
| Ailerons | 13750 | 41 | House | 22784 | 17 |
| Airfoild | 1503 | 6 | Kinematics32fh | 8192 | 33 |
| AutoPrice | 205 | 26 | MachineCpu | 209 | 7 |
| Available_power | 1802 | 16 | Maximal_torque | 1802 | 33 |
| Bank8FM | 4499 | 9 | Mortgage | 1049 | 16 |
| Boston | 506 | 14 | Puma32h | 8192 | 33 |
| California | 20640 | 9 | Servo | 167 | 5 |
| Compactiv | 8192 | 22 | Treasury | 1049 | 16 |
| ConcreteStrength | 1030 | 9 | Wankara | 1609 | 10 |

guish the examples whose true or predicted values match a rare case. Furthermore, it does not consider the examples that neither the true value nor the prediction corresponds to a rare case [10].

Ribeiro and Moniz [10] proposed the evaluation metric SERA (Squared error-relevance area) to counter these limitations. The idea behind SERA is to have a metric focused on errors of highly relevant examples but considering the entire domain for evaluating the model. No arbitrary threshold is needed to distinguish rare from normal cases, but a relevance definition to measure the relevance of each instance is required.

Steininger et al. [9] evaluates different parts of the target domain. Firstly, the range of the target values is divided in 5 bins of equal length. Secondly, to each bin is assigned a rank between 1 and 5 according to the number of instances and hence, the bin with fewest instances has rank 1, the bin with the second smallest number of instances has rank 2 and so on until rank 5. Thereafter, Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) are calculated per bin.

Since in our work we make a comparison with DenseLoss and other cost-sensitive solutions without making any assumptions about the location of rare cases, we decided to use the same metric to ensure a fair comparison, and to avoid the results from being biased due to the use of different measures of instance rarity. We also used the RMSE metric despite its limitations for the purpose of analyzing the results from the point of view of a standard metric.

#### Remark

The use of bins only aims to evaluate the model in different ranges of the domain. WSMOTER does not use these bins at any time for instance selection or model training.

The distribution of the instances per bin according to their target values can be seen in Table 2. From this table, it is possible to assign a rank to each bin, in order to implement the metric defined in [9]. For example, to the bins of the data set 'delta_ailerons', were assigned ranks 1, 4, 5, 3, 2, respectively. In a small number of cases, there was a tie in the number of instances in two or more bins of a data set such as in data set 'a6'. In all those cases, it was assigned the best rank to the tied bin on the right, by no particular criteria other than the need to break ties. Anyway, this only happened in a small number of cases and all the approaches compared in our study faced the same condition.

### 5.3 Algorithms

WSMOTER is an oversampler, which means that it is an algorithm that modifies the training set by adding instances. In order to be possible to evaluate the performance of WSMOTER, a machine learning algorithm is then needed to train on the data. Two different algorithms were used, namely: Random Forest regressor (RF) [25] from the Scikit-Learn library [26] and Multi-layer Perceptron regressor (MLP) from the Keras API for TensorFlow.

**Table 2** Number of instances per bin in the 36 data sets

| Data set | bin1 | bin2 | bin3 | bin4 | bin5 |
|---|---|---|---|---|---|
| a1 | 135 | 27 | 19 | 11 | 6 |
| a2 | 161 | 27 | 8 | 1 | 1 |
| a3 | 164 | 18 | 13 | 1 | 2 |
| a4 | 191 | 5 | 0 | 1 | 1 |
| a5 | 159 | 23 | 10 | 4 | 2 |
| a6 | 170 | 17 | 7 | 2 | 2 |
| a7 | 179 | 10 | 4 | 1 | 4 |
| Abalone | 448 | 3036 | 557 | 129 | 7 |
| Acceleration | 449 | 1063 | 199 | 17 | 4 |
| Ailerons | 15 | 147 | 1098 | 6034 | 6456 |
| Airfoild | 53 | 231 | 486 | 597 | 136 |
| AutoPrice | 128 | 54 | 9 | 11 | 3 |
| Available_power | 1121 | 550 | 93 | 32 | 6 |
| Bank8FM | 2684 | 1103 | 514 | 170 | 28 |
| Boston | 76 | 236 | 125 | 38 | 31 |
| California | 4489 | 7870 | 4568 | 1991 | 1722 |
| Compactiv | 294 | 0 | 100 | 1299 | 6499 |
| ConcreteStrength | 178 | 337 | 308 | 152 | 55 |
| cpuAct | 294 | 0 | 100 | 1299 | 6499 |
| cpu_small | 294 | 0 | 100 | 1299 | 6499 |
| Delta_ailerons | 8 | 661 | 6034 | 414 | 12 |
| Delta_elv | 15 | 642 | 7853 | 989 | 18 |
| ele-1 | 267 | 177 | 34 | 14 | 3 |
| Elevators | 13722 | 2261 | 518 | 87 | 11 |
| ForestFires | 514 | 1 | 0 | 1 | 1 |
| Fuel_consumption_country | 361 | 1092 | 272 | 34 | 5 |
| Heat | 4570 | 2280 | 451 | 86 | 13 |
| House | 20416 | 1819 | 369 | 98 | 82 |
| Kinematics32fh | 120 | 2014 | 4229 | 1669 | 160 |
| MachineCpu | 185 | 15 | 6 | 1 | 2 |
| Maximal_torque | 1102 | 577 | 105 | 12 | 6 |
| Mortgage | 402 | 353 | 158 | 82 | 54 |
| Puma32h | 433 | 1771 | 3973 | 1646 | 369 |
| Servo | 133 | 6 | 12 | 14 | 2 |
| Treasury | 514 | 379 | 73 | 53 | 30 |
| Wankara | 33 | 400 | 484 | 428 | 264 |

For the implementation of WSMOTER, we used the code of SMOTE and SMOTENC [27] as baseline, introducing all the required modifications in the code to make the implementation of our approach viable, following all the guidelines described in section 4. In our experiments, an oversampling rate of 200% was used for WSMOTER.

The performance of WSMOTER was compared against other approaches in two different experiments. One of those approaches is DenseLoss, a combination of the method DenseWeight with sample weight for loss functions. This combination was accomplished through the Keras API. The implemented neural network is completely in accordance with the description done in paper [9], namely: three hidden layer with ten neurons, Kaiming uniform initialization, ReLU activation, one output layer with linear activation, Adam optimization, learning rate of 0.0001, weight decay coefficient of $10^{-9}$ and 1000 epochs of train with early stopping.

Regarding the Adaboost algorithm, it was used the implementation available in the Scikit-Learn library [26].

For the sake of fairness and to avoid bias in the results, in all the approaches used in the experiments that require to measure the data points rarity, it was used the DenseWeight [9] method. The parameter $\alpha$ of the DenseWeight was defined as one because paper [9] stated that it improves the model performance.

In our work, we also used the Python libraries SciPy [28] and statsmodels [29].

### 5.4 Experimental framework

In order to proceed with the experimental study, two repetitions of 5-fold cross-validation were performed. The procedure is briefly the following: after splitting the data into five folds, one of them is used as a test set and the remaining four compose the training set. Since the five folds are used as the test set, a model is trained and tested five times to complete one cross-validation.

The process is then repeated, splitting the data differently. The results presented in Section 6 are the average of the results obtained in all cycles of training and testing.

## 6 Results

Tables 3 and 4 show the performance of the models in experiment 1. All approaches were performed with neural networks, but they are obviously different from each other. WSMOTER generates synthetic instances based on the rarity of the instances already existing in the training set and DenseLoss uses the rarity of those already existing instances as weights in a cost-sensitive learning. Table 3 shows the number of wins of each approach. Note that the highest pos-

**Table 3** Number of wins in experiment 1

| Metric | MLP | DenseLoss | WSMOTER + MLP |
|---|---|---|---|
| RMSE_bin_rank_1 | 1 | 9 | **21** |
| RMSE_bin_rank_2 | 0 | 17 | **19** |
| RMSE_bin_rank_3 | 0 | 16 | **20** |
| RMSE_bin_rank_4 | 2 | **18** | 16 |
| RMSE_bin_rank_5 | 17 | 1 | **18** |
| RMSE | 12 | 2 | **22** |

**Table 4** Results for bin with rank 1 in experiment 1

| Data set | MLP | DenseLoss | WSMOTER + MLP |
|---|---|---|---|
| a1 | 70.45617 | 60.87134 | **51.897995** |
| a2 | 65.57555 | **59.638412** | 62.14405 |
| a3 | 20.671211 | **14.127478** | 18.118637 |
| a4 | * | * | * |
| a5 | 36.644947 | 31.584772 | **29.13421** |
| a6 | 60.056953 | 50.19655 | **49.41433** |
| a7 | 19.776226 | **16.35384** | 17.375711 |
| Abalone | 10.29182 | **9.236755** | 10.082174 |
| Acceleration | 16.643904 | 15.711986 | **8.413387** |
| Ailerons | 0.001274 | **0.001088** | 0.001135 |
| Airfoild | 8.54686 | 7.557911 | **4.957806** |
| AutoPrice | 42558.055 | 42569.156 | **38283.902** |
| Available_power | 136.37964 | 153.22499 | **60.411945** |
| Bank8FM | **0.089678** | 0.090615 | 0.203755 |
| Boston | 32.89584 | 35.6153 | **10.792629** |
| California | 168063.75 | **142983.58** | 161088.48 |
| Compactiv | * | * | * |
| ConcreteStrength | 35.62862 | 34.911743 | **14.072458** |
| cpuAct | * | * | * |
| cpu_small | * | * | * |
| Delta_ailerons | 0.001064 | 0.001056 | **0.001032** |
| delta_elv | 0.006662 | **0.00577** | 0.006348 |
| ele-1 | 6319.385 | 6359.4834 | **3814.4775** |
| Elevators | 0.020887 | 0.019727 | **0.017126** |
| ForestFires | * | * | * |
| Fuel_cons_country | 6.983841 | 6.281194 | **4.542336** |
| Heat | 109.99745 | 92.425446 | **18.160637** |
| House | 392741.94 | 372473.84 | **368280.4** |
| Kinematics32fh | 0.236707 | **0.184368** | 0.186712 |
| MachineCpu | 858.1273 | 857.6991 | **477.28766** |
| Maximal_torque | 346.0211 | 331.36664 | **75.92478** |
| Mortgage | 0.509043 | 0.495461 | **0.330566** |
| Puma32h | 0.054425 | **0.037516** | 0.061929 |
| Servo | 5.53902 | 4.301757 | **3.440785** |
| treasury | 1.086821 | 1.099883 | **0.599642** |
| Wankara | 1.882654 | 1.865015 | **1.255374** |

* Data sets for which there are not instances belonging to the bin with rank 1

sible number of wins is 36, the number of data sets used in the study. However, the number of wins for bin with rank 1 is only 31 because 5 data sets do not contain any representative and therefore there cannot be a winner. The best result is marked in bold in the table as well as in the remaining tables of our work.

In Table 4 we see the results regarding the bin with rank 1 and although WSMOTER did not always win, as we have seen in Table 3, what deserves to be highlighted is the significant difference obtained in favor of WSMOTER in some data sets, such as in 'ele1', 'maximal_torque' or 'servo', among others.

The results in the various ranked bins show that until bin rank 4, DenseLoss and WSMOTER almost always outperformed MLP with the exception of a residual number of cases. Therefore, these results validate the adoption of solutions in favor of rare data points. In the comparison between DenseLoss and WSMOTER we see a clear advantage for WSMOTER in the bin rank 1, and a slight advantage in the remaining. However, note that bin rank 1 is the one with most rare cases. In the bin rank 5, there is almost a tie with the MLP approach, which is not surprising since better performance with respect to rare cases is often achieved at the expense of performance with normal cases, which has already been reported in the literature.

The results presented in Tables 3 and 4 are complemented with the Table 5. The results were obtained as follows. Firstly, for each metric, ranking scores were assigned to the three methods: MLP, DenseLoss and WSMOTER, for each data set, in the following way: to the best result was assigned rank 1, the second-best result was assigned rank 2, and the third place was assigned rank 3. Finally, the ranking scores assigned to the methods were averaged. The standard deviation is presented in parentheses.

Table 5 shows that WSMOTER had the best average ranking score in the two most relevant bins, which did not happen in bin rank 5 as already expected. Surprisingly, WSMOTER achieved the best score regarding the RMSE metric, which seems to indicate that the gains in the bins with most rare cases compensated the degradation of results in the bin with most common cases and even when there is no special concern with rare cases it is useful to use WSMOTER.

Regarding the experiment 2, once again, WSMOTER seems to dominate the most relevant bins, i.e. the ones that have higher ranks. Up to bin rank 3, WSMOTER is the one with more wins as we can see in Table 6. Furthermore, WSMOTER had the best average ranking score in the two most relevant bins. In Table 7 we can see the mean ranking across the data sets.

In a general analysis of the results of the second experiment, no significant differences were detected in relation to the first, due to change to the Random Forest algorithm.

**Table 5** Mean ranking across the data sets in experiment 1

| Metric | MLP | DenseLoss | WSMOTER + MLP |
|---|---|---|---|
| RMSE_bin_rank1 | 2.74 (0.51) | 1.87 (0.66) | **1.39**(**0.61**) |
| RMSE_bin_rank2 | 2.69 (0.46) | 1.69 (0.74) | **1.61**(**0.72**) |
| RMSE_bin_rank5 | **1.56** (0.55) | 2.83 (0.44) | 1.61 (0.68) |
| RMSE | 1.75 (0.60) | 2.75 (0.55) | **1.50**(**0.69**) |

**Table 6** Number of wins in experiment 2

| Metric | Random forest | Cost-sensitive Random forest | AdaBoost | WSMOTER + Random forest |
|---|---|---|---|---|
| RMSE_bin_rank1 | 2 | 2 | 5 | **22** |
| RMSE_bin_rank2 | 4 | 5 | 8 | **19** |
| RMSE_bin_rank3 | 8 | 4 | 10 | **14** |
| RMSE_bin_rank4 | 6 | 8 | **12** | 10 |
| RMSE_bin_rank5 | **18** | 17 | 0 | 1 |
| RMSE | 13 | **14** | 1 | 8 |

**Table 7** Mean ranking across the data sets in experiment 2

| Metric | Random forest | Cost-sensitive Random forest | AdaBoost | WSMOTER + Random forest |
|---|---|---|---|---|
| RMSE_bin_rank1 | 2.77 (0.83) | 2.71 (0.85) | 3.06 (1.19) | **1.45** (0.80) |
| RMSE_bin_rank2 | 2.53 (0.83) | 2.78 (0.97) | 2.97 (1.28) | **1.72** (0.90) |
| RMSE_bin_rank5 | **1.51**(**0.49**) | 1.56 (0.56) | 3.94 (0.23) | 2.99 (0.42) |
| RMSE | **1.82**(**0.69**) | 1.88 (0.84) | 3.83 (0.55) | 2.47 (0.93) |

**Table 8** Results for Friedman's test in experiment 1

| Metric | p-value | Significance |
|---|---|---|
| RMSE_bin_rank_1 | 4.505020e−07 | True |
| RMSE_bin_rank_2 | 2.079603e−06 | True |
| RMSE_bin_rank_5 | 6.997046e−09 | True |
| RMSE | 1.444980e-07 | True |

**Table 9** Results for Friedman's test in experiment 2

| Metric | p-value | Significance |
|---|---|---|
| RMSE_bin_rank_1 | 2.734518e−06 | True |
| RMSE_bin_rank_2 | 2.087139e−04 | True |
| RMSE_bin_rank_5 | 1.371382e−19 | True |
| RMSE | 2.548856e-12 | True |

**Table 10** Holm's method adjusted p-values in experiment 1

| Metric | MLP | DenseLoss |
|---|---|---|
| RMSE_bin_rank_1 | 1.921e−07 | 5.678e−02 |
| RMSE_bin_rank_2 | 8.606e−06 | 7.237e−01 |
| RMSE_bin_rank_5 | 1.e+00 | 4.31e−07 |
| RMSE | 2.888e−01 | 2.275e−07 |

**Table 11** Holm's method adjusted p-values in experiment 2

| Metric | Random forest | Cost-sensitive Random forest | AdaBoost |
|---|---|---|---|
| RMSE_bin_rank_1 | 1.1e−04 | 1.248e−04 | 2.614e−06 |
| RMSE_bin_rank_2 | 8.113e−03 | 1.045e−03 | 1.198e−04 |
| RMSE_bin_rank_5 | 1.e+00 | 1.e+00 | 4.908e−03 |
| RMSE | 1.e+00 | 1.e+00 | 2.313e−05 |

WSMOTER was not always the best approach but clearly outperformed the remaining alternatives in the best-ranked bins.

The non-parametric Friedman test and the Holm's post-hoc method were used to statistically validate the results [30].

Tables 8 and 9 show the results of the application of the non-parametric Friedman test in experiments 1 and 2, respectively. The null hypothesis tested is that all approaches used are equivalent, which is rejected at a significance level of 0.05 in all cases.

Finally, Holm's post-hoc method was applied to compare WSMOTER with each of the other approaches. The null hypothesis tested is that WSMOTER does not performs better than each of those approaches.

In the experiment 2, the null hypothesis tested was always rejected at a significance level of 0.05 for RMSE_bin_rank_1 and RMSE_bin_rank_2, whereas for RMSE_bin_rank 5 and RMSE metric this only happened with the Adaboost algorithm. For the experiment 1, the null hypothesis tested was rejected for RMSE_bin_rank_1 and RMSE_bin_rank_2 regarding the MLP approach but unfortunately, it was not rejected regarding DenseLoss. However, it is required to emphasize that it was due to a very small difference in the adjusted p-value and the use of just a slightly higher significance level leads to a different result in the bin rank 1. The results are shown in Tables 10 and 11.

## 7 Conclusion

The approach WSMOTER is presented in this paper, which is suitable for imbalanced regression. WSMOTER is based on the SMOTE algorithm, but modifications were introduced to make it proper for imbalanced regression problems. These modifications are focused on the selection process of the examples that are used to generate new instances.

The approach used fits in a much better way to the nature of regression problems, treating each data point individually and avoiding discretization processes.

This kind of method has the advantage of not being specific to a problem or a machine learning algorithm, and can be universally used, which is not the case with methods at the algorithm-level [31, 32].

Regarding limitations, WSMOTER is not computationally demanding by itself but, as any other oversamplers, WSMOTER increases the size of the training set leading to an increase in the time required to train a model [33]. Therefore, WSMOTER is a solution with greater potential if applied to data sets of up to medium size.

WSMOTER was assessed and compared with currently available solutions in an empirical study using 36 data sets. The results show that it has outperformed the other solutions, and which is a better alternative for facing regression problems with imbalanced data sets.

**Author Contributions** Luís Camacho: Conceptualization, Methodology, Software, Validation, Writing - original draft and Writing - review & editing; Fernando Bacao: Conceptualization, Methodology, Validation and Writing - review & editing.

**Funding** Open access funding provided by FCT|FCCN (b-on).

**Availability of Data and Materials** All data sets used are publicly available.

**Code Availability** The source code is available at https://drive.google.com/drive/folders/1h6Q5sKB5bnqk0Kh01sH1uc6LTp4KSPbb?usp=sharing.

## Declarations

**Conflict of Interest/Competing Interests** The authors have no relevant financial or non-financial interests to disclose.

**Ethics Approval** Not applicable.

**Consent to Participate** Not applicable.

**Consent for Publication** Not applicable.



## References

1. Krawczyk B (2016) Learning from imbalanced data: open challenges and future directions. Prog Artif Intell 5:221–232. https://doi.org/10.1007/s13748-016-0094-0
2. Snieder E, Abogadil K, Khan UT (2021) Resampling and ensemble techniques for improving ann-based high-flow forecast accuracy. Hydrol Earth Syst Sci 25(5):2543–2566. https://doi.org/10.5194/hess-25-2543-2021
3. Saha D, Basso B, Robertson GP (2021) Machine learning improves predictions of agricultural nitrous oxide ($N_2O$) emissions from intensively managed cropping systems. Environ Res Lett 16(2):024004. https://doi.org/10.1088/1748-9326/abd2f3
4. Ranacher P, Brunauer R, Van der Spek SC, Reich S (2016) A model to estimate and interpret the energy-efficiency of movement patterns in urban road traffic. Comput Environ Urban Syst 59:152–163. https://doi.org/10.1016/j.compenvurbsys.2016.06.006

5. He H (2013) Introduction. In: Imbalanced learning: foundations, algorithms, and applications, pp 1–12. John Wiley & Sons, New Jersey. Chap. 1. https://doi.org/10.1002/9781118646106.ch1
6. He, H., Garcia, E.A. (2009) Learning from imbalanced data. IEEE Trans on Knowl and Data Eng 21(9), 1263–1284 https://doi.org/10.1109/TKDE.2008.239
7. Torgo L, Branco P, Ribeiro RP, Pfahringer B (2015) Resampling strategies for regression. Expert Syst 32(3):465–476. https://doi.org/10.1111/exsy.12081
8. Branco P, Torgo L, Ribeiro RP (2019) Pre-processing approaches for imbalanced distributions in regression. Neurocomputing 343:76–99. https://doi.org/10.1016/j.neucom.2018.11.100
9. Steininger M, Kobs K, Davidson P, Krause A, Hotho A (2021) Density-based weighting for imbalanced regression. Mach Learn 110:2187–2211. https://doi.org/10.1007/s10994-021-06023-5
10. Ribeiro RP, Moniz N (2020) Imbalanced regression and extreme value prediction. Mach Learn 109:1803–1835. https://doi.org/10.1007/s10994-020-05900-9
11. Branco P, Torgo L, Ribeiro RP (2016) A survey of predictive modeling on imbalanced domains. ACM Comput Surv 49(2) https://doi.org/10.1145/2907070
12. Sadouk L, Gadi T, Essoufi EH (2021) A novel cost-sensitive algorithm and new evaluation strategies for regression in imbalanced domains. Expert Syst 38(4):12680. https://doi.org/10.1111/exsy.12680
13. Ribeiro RPA (2011) Utility-based regression. PhD thesis, Faculty of Sciences University of Porto, Porto
14. Pérez-Porras, FJ, Triviño-Tarradas P, Cima-Rodríguez C, Meroño-de-Larriva JE, García-Ferrer A, Mesas-Carrascosa FJ (2021) Machine learning methods and synthetic data generation to predict large wildfires. Sensors 21(11) https://doi.org/10.3390/s21113694
15. Fernández A, García S, Galar M, Prati RC, Krawczyk B, Herrera F (2018) Cost-Sensitive learning, pp 63–78. Springer, Cham. https://doi.org/10.1007/978-3-319-98074-4_4
16. Liu XY, Zhou ZH (2013) Ensemble methods for class imbalance learning. In: Imbalanced Learning, pp. 61–82. John Wiley & Sons, Ltd, New Jersey. Chap. 4. https://doi.org/10.1002/9781118646106.ch4
17. Freund Y, Schapire RE (1997) A decision-theoretic generalization of on-line learning and an application to boosting. J Comput Syst Sci 55(1):119–139. https://doi.org/10.1006/jcss.1997.1504
18. Fernández A, García S, Galar M, Prati RC, Krawczyk B, Herrera F (2018) Ensemble learning, pp 147–196. Springer, Cham. https://doi.org/10.1007/978-3-319-98074-4_7
19. Hoens TR, Chawla NV (2013) Imbalanced datasets: From sampling to classifiers. In: Imbalanced learning: foundations, algorithms, and applications, pp 43–59. John Wiley & Sons, New Jersey. Chap. 3. https://doi.org/10.1002/9781118646106.ch3
20. Chawla NV, Bowyer KW, Hall LO, Kegelmeyer WP (2002) Smote: synthetic minority over-sampling technique. J Artif Int Res 16(1):321–357
21. Douzas G, Bacao F (2019) Geometric smote a geometrically enhanced drop-in replacement for smote. Inf Sci 501:118–135. https://doi.org/10.1016/j.ins.2019.06.007
22. Branco P, Torgo L, Ribeiro RP (2017) SMOGN: a pre-processing approach for imbalanced regression. In: Luís Torgo, P.B., Moniz, N. (eds.) Proceedings of the first international workshop on learning with imbalanced domains: theory and applications. Proceedings of Machine Learning Research, vol 74, pp 36–50. PMLR
23. Camacho L, Douzas G, Bacao F (2022) Geometric smote for regression. Expert Syst Appl 193:116387. https://doi.org/10.1016/j.eswa.2021.116387
24. Alcalá-Fdez J, Fernandez A, Luengo J, Derrac J, García S, Sánchez L, Herrera F (2011) Keel data-mining software tool: data set repository, integration of algorithms and experimental analysis framework. J Mult Valued Log Soft Comput 17(2–3):255–287
25. Breiman L (2001) Random forests. Mach Learn 45:5–32. https://doi.org/10.1023/A:1010933404324
26. Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay E (2011) Scikit-learn: machine learning in python. J Mach Learn Res 12:2825–2830
27. Lemaître G, Nogueira F, Aridas CK (2017) Imbalanced-learn: a python toolbox to tackle the curse of imbalanced datasets in machine learning. J Mach Learn Res 18(1):559–563
28. Virtanen P, Gommers R, Oliphant TE, Haberland M, Reddy T, Cournapeau D, Burovski E, Peterson P, Weckesser W, Bright J, van der Walt SJ, Brett M, Wilson J, Millman KJ, Mayorov N, Nelson ARJ, Jones E, Kern R, Larson E, Carey CJ, Polat İ, Feng Y, Moore EW, VanderPlas J, Laxalde D, Perktold J, Cimrman R, Henriksen I, Quintero EA, Harris CR, Archibald AM, Ribeiro AH, Pedregosa F, van Mulbregt P, SciPy 1.0 Contributors, (2020) SciPy 1.0: Fundamental algorithms for scientific computing in Python. Nat Methods 17:261–272. https://doi.org/10.1038/s41592-019-0686-2
29. Seabold S, Perktold J (2010) statsmodels: econometric and statistical modeling with python. In: 9th Python in science conference
30. Demšar J (2006) Statistical comparisons of classifiers over multiple data sets. J Mach Learn Res 7:1–30
31. Douzas G, Bacao F, Last F (2018) Improving imbalanced learning through a heuristic oversampling method based on k-means and smote. Inf Sci 465:1–20. https://doi.org/10.1016/j.ins.2018.06.056
32. Galar M, Fernandez A, Barrenechea E, Bustince H, Herrera F (2012) A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 42(4):463–484 https://doi.org/10.1109/TSMCC.2011.2161285
33. Weiss GM (2013) Foundations of imbalanced learning. In: Imbalanced learning: foundations, algorithms, and applications, pp 13–41. John Wiley & Sons, New Jersey. Chap. 2. https://doi.org/10.1002/9781118646106.ch2