DEEP IMBALANCED REGRESSION FOR VENTILATOR PRESSURE

Nguyen Ngoc Ha My$^{1\,[0000-1111-2222-3333]}$, Nguyen Thi Phuong Thao$^{2,3\,[1111-2222-3333-4444]}$, Nguyen Pham Hong Duyen$^{3,4\,[2222--3333-4444-5555]}$, and Tran Thi Thu Ha$^{4\,[3333-4444-5555-6666]}$

Department of Information Science and Engineering, University of Information Technology, Vietnam National University of Ho Chi Minh City, Vietnam
20521623@gm.uit.edu.vn, 20521936@gm.uit.edu.vn, 20520477@gm.uit.edu.vn and 20521273@gm.uit.edu.vn

Abstract. This paper presents deep imbalanced regression approaches to simulate the pressure of a mechanical ventilator. In many real-world settings, imbalanced data impedes model performance of learning algorithms, like neural networks, mostly for rare cases. Ventilator pressure data, which was collected from a modified open-source ventilator exhibits imbalance distribution, where certain target values have significantly fewer observations. This paper exploits recent studies and provides a simulator based on a deep imbalance regression model to predict the airway pressure in the respiratory circuit during the inspiratory phase of a breath given a time series of control parameters and lung attributes. These approaches demonstrate the effectiveness of deep imbalance regression in tracking pressure waveforms compared to traditional deep learning models.

Keywords: Ventilator pressure · Deep Sequence model · Imbalanced regression · Weighting sample · Cost-sensitive learning.

1 Introduction

Many machine learning algorithms, like neural networks, typically expect roughly uniform target distributions (Cui et al. 2019; Krawczyk 2016; Sun et al. 2009). In the case of classification, that means that there are similar numbers of examples per class. For regression, there should be a similar density of samples across the complete target value range. However, many datasets exhibit skewed target distributions with target values in certain ranges occurring less frequently than others. Consequently, models can become biased, leading to better performance for common cases than for rare cases (Cui et al. 2019; Krawczyk 2016). This is particularly problematic for tasks where these rare occurrences are of special interest. Examples include precipitation estimation, where extreme rainfall is rare but can have dramatic consequences, or fraud detection, where rare fraudulent events are supposed to be detected.
There are many solutions to this problem for classification tasks including resampling strategies (Chawla et al. 2002; He et al. 2008) and cost-sensitive learning approaches (Cui et al. 2019; Huang et al. 2016; Wang et al. 2017). However, these can

not be applied easily to regression tasks because of the inherent differences between continuous and discrete, nominal target values. Typical solutions to data imbalance require a notion of rarity or importance for a data point in order to know which data points to over- and undersample or which data points to weight more strongly. It is harder to define which values are rare for regression tasks in comparison to classification tasks, since one cannot simply use class frequencies (Branco et al. 2017). Only few works explore methods improving model performance for rare cases in regression settings, mostly proposing sampling-based approaches (Branco et al. 2017; Krawczyk 2016; Torgo et al. 2013). These can have disadvantages in comparison to cost-sensitive methods since the creation of new data points via oversampling of existing data points may lead to overfitting as well as additional noise, while undersampling removes information (Cui et al. 2019; Dong et al. 2017). The success of cost-sensitive learning for imbalanced classification tasks suggests that exploring this direction for imbalanced regression could also lead to better methods in this domain (Krawczyk 2016).
In this paper, we applied two sample weighting approaches for imbalanced regression datasets called Ventilator Pressure and, based on this, a cost-sensitive learning method for imbalanced regression with neural networks called DenseLoss and Label Distribution Smoothing. Both techniques assign each data point in the training set a weight according to their weighting schemes, increasing the influence of rare data points on the loss and the gradients. After experiments, we compared the impacts of these weighting methods on Ventilator Pressure data to the final results and showed that it can significantly improve model performance in practice.

1.1 Data Description

The dataset was collected from a modified open-source ventilator connected to an artificial bellows test lung (IngMar, 2020) via a respiratory circuit (Suo et at., 2021). Fig. 1 below illustrates the circuit, with two control inputs highlighted in green and the state variable (airway pressure) to predict in blue.
The dataset has eight features with over six million rows including 75,000 training breaths and 50,000 test breaths. It contains details regarding lung attributes, the input and output valves of the machinery, and the pressure inside the lungs connected to the machine.
The target attributes are the control u, pressure over time and a range of lung attributes (R, C). Each time series represents an approximately 3-second breath. The dataset is organized such that each row is a time step in a breath and gives the two control signals, the resulting airway pressure, and relevant attributes of the lung.
The first control input u_in is a continuous variable from 0 to 100 representing the percentage the inspiratory solenoid valve is open to let air into the lung (i.e., 0 is completely closed and no air is let in and 100 is completely open). The second control input u_out is a binary variable representing whether the exploratory valve is open (1) or closed (0) to let air out.
The breaths were recorded in a mechanical lung with different resistance and capacity properties, which were encoded in provided R and C variables. The first parameter R represents the change of pressure per variation in flow (air volume per time)

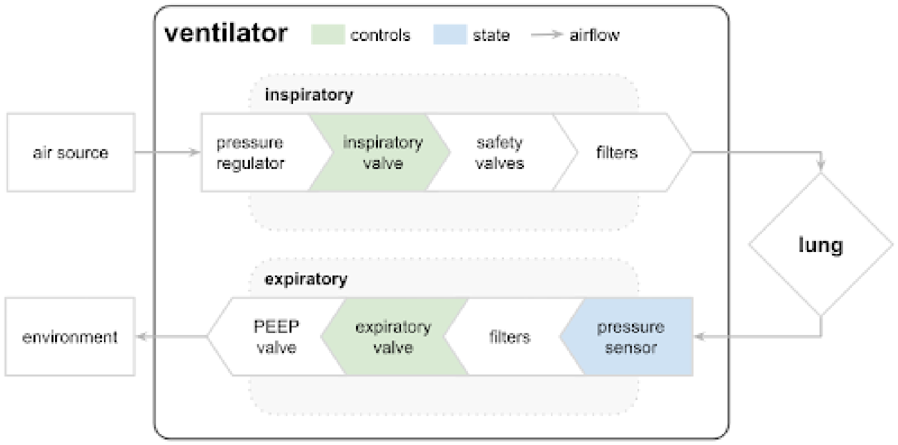

Fig. 1. A simplified respiratory circuit shows the airflow through the inspiratory pathway, entering and exiting the lung, and exiting through the expiratory pathway.

in cmH2O/L/S. The second parameter C is the capacity of the lung to expand, representing the change in volume per change in pressure in mL/cmH2O. In the case of the Covid-19, the values of R C most often found in patients were R equal to 20 and C among these values [10, 20, 50] (Haudebourg et al., 2020).

2 Related works

Imbalanced data can in principle be tackled with data-level methods, algorithm-level methods, or a combination of both (Krawczyk 2016). Data-level methods typically over- and or undersample subsets of a dataset to balance the distribution. Algorithm-level methods modify existing learning algorithms to better cope with imbalanced data.

There are many solutions to data imbalance for classification tasks. Data-level methods for classification often create new samples for rare classes (oversampling) and/or remove samples of common classes (undersampling). Notable examples include ADASYN (He et al. 2008) and SMOTE (Chawla et al. 2002). Recently, KDE was used to estimate the feature distribution of minority classes (Kamalov 2020). New minority class samples are generated using the estimated feature distribution. Algorithm-level methods for classification typically involve cost-sensitive learning, where the loss of samples with rare classes is emphasized in the overall loss (Cui et al. 2019). Weighting is often based on the inverse class frequency as a measure of rarity (Huang et al. 2016; Wang et al.2017). We apply a conceptually similar method, but for regression instead of classification. The continuous target variable of regression tasks makes it harder to determine a single sample's rarity, preventing simple adaptations of existing cost-sensitive learning approaches (Branco et al. 2017). While there is work on cost-sensitive learning for regression models, these approaches assign different costs to over- and underestimation respectively, regardless of a data point's rarity (Zhao et al. 2011; Hernández-Orallo 2013). However, we are interested in exploring how cost-sensitive learning can be used to solve the problem of imbalanced datasets for regression tasks, for which only few works exist. There is a cost-sensitive post-processing technique called probabilistic reframing which adjusts estimates of previously built models to different contexts (Hernández-Orallo 2014). It would be feasible to apply this to imbalanced domains but it was not evaluated for this yet (Branco et al. 2016b). A cost-sensitive method for obtaining regression tree ensembles biased according to a utility function is ubaRules (Ribeiro 2011) which is mostly used to estimate extreme values as accurately as possible. It is specific to regression tree ensembles while our application is designed for—but not restricted to—the use with neural networks. A metric that takes both rare, extreme samples and common samples into account for evaluating a model's ability to predict extreme values is SERA (Ribeiro and Moniz 2020). SERA can be considered a loss function that is used for model selection and hyperparameter optimization but it is not incorporated in a learning method like DenseLoss.

Despite the lack of cost-sensitive approaches, there are sampling-based data-level methods which are applied during data pre-processing. One approach is SMOTE for regression (SmoteR) (Torgo et al. 2013), which is based on the original SMOTE method for classification (Chawla et al. 2002). It combines undersampling of common data points and oversampling of rare cases, in order to create a more balanced distribution. The authors adjust SMOTE to work for regression domains by binning data points into relevant and irrelevant partitions using a relevance threshold tR and a relevance function φ. They use an automatic method for obtaining based on box plot statistics through which specific control points on the target domain are obtained. Each control point is a tuple (y, $\varphi(y)$, $\varphi'(y)$), where $\varphi'(y)$ —the derivative of relevance $\varphi(y)$—is always set to 0, since control points are assumed to be local extrema of relevance. The relevance function is then defined with piecewise cubic Hermite interpolation through these control points (Ribeiro 2011). This automatic method for obtaining assumes that extreme values are rare, which is in contrast to our work, where rare values are automatically detected without such assumptions. Data points marked as relevant($\varphi(y) > t_R$) are oversampled, creating new synthetic cases via interpolation of features and target values between two relevant data points. Irrelevant data points are undersampled.

The SMOGN (Branco et al. 2017) algorithm builds on SmoteR and combines it with oversampling via Gaussian noise. For the latter, normally distributed noise is added to the features and the target value of rare data points, creating additional, slightly altered replicas of existing samples (Branco et al. 2016a). Rare data points are identified using the same method for obtaining a relevance function φ used by SmoteR. SMOGN iterates over all rare samples and selects between SmoteR's interpolation based oversampling and Gaussian noise based oversampling depending on the distance to the k-nearest neighbors. For small distances, SmoteR's interpolation is applied, since interpolation is deemed more reliable for close samples. Other rare data points are oversampled with Gaussian noise. Common data points are randomly un-

dersampled. The authors report improvements compared to SmoteR (Branco et al. 2017). Because of this and a lack of other methods, SMOGN can be considered the state-of-the-art.
In contrast to these data-level methods, we apply an algorithm-level, cost-sensitive method for imbalanced regression called DenseLoss and Label Distribution (LDS) using density-based weighting scheme DenseWeight and kernel distribution to perform explicit distribution smoothing in the label space. The concept of weighting data points based on the target value distribution is already present in prior work, e.g. in the automatic method for obtaining relevance functions used by SMOGN, or in SERA. However, DenseWeight and LDS do not make assumptions about which cases are rare since it determines relative rarity with a density function. Contrary to SmoteR and SMOGN, DenseLoss does not explicitly change the dataset, e.g. by creating new samples.

3 Methods

3.1 Data Processing

Feature Engineering Features are divided into 3 initial features [time_step, u_in, u_out] and 85 designed features. original features, all features except R and C were used in their primary form, and one-hot encoded R and C were used to display the type information of R and C for the model.
We designed more than 70 features to use existing features, faster model convergence, and low MAE scores. Besides statistical data quality checks, such as multicollinearity and serial correlation are performed without noticeable results or even harm to the model. Most of the features have been discussed in the contest forum.

Data Normalization We used RobustScaler to shrink the dataset, save resources, increase the learning speed of the neural network, and limit the effect of outliers. RobustScaler removes the median and scales the data according to the quantile range. We used comprehensive quantile range of [20,80] to reduce variance. Centering and scaling happen independently on each feature by computing the relevant statistics on the samples in the training set. Median and quantile range are then stored to be used on later data using the transform method.
Its formula is as follows:

$$\frac{x_i - Q_1(x)}{Q_3(x) - Q_1(x)}$$

3.2 Model Architecture

Layer BiLSTM is a sequence processing model that consists of two LSTMs: one taking the input in a forward direction, and the other in a backwards direction. BiLSTMs effectively increase the amount of information available to the network, improving the context available to the algorithm (e.g. knowing what words immediately follow and precede a word in a sentence).

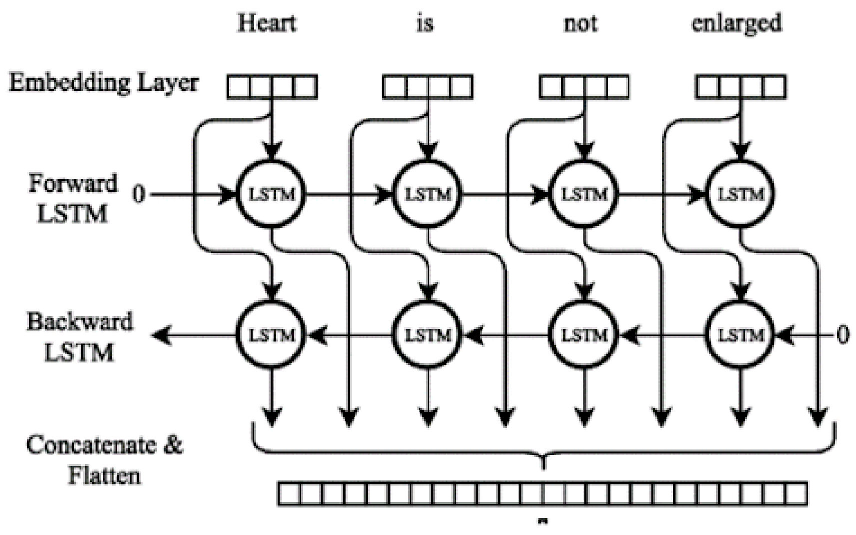

Fig. 2. The process of BiLSTM Model

ResBiLSTM extracts semantic information in specific paths, with 4 bidirectional LSTM layers with [1028, 512, 256,128] units and the last 2 layers with [218,1] units. The near-end layer is exponential linearity (SELU) enabled and scaled to solve the vanishing gradient problem, and even in the presence of noise and perturbations the learning is powerful. The target pressures are so dependent on the previous that adding LSTMs is necessary.we add skip connections by feeding the l-th layer output as the l+2-th layer input, thus avoiding performance degradation.

3.3 Label Distribution Smoothing (LDS)

Problem Setting Let $\{(x_i, y_i)\}_{i=1}^{N}$ be a training set, where $x_i \in R^d$ denotes the input and $y_i \in R$ is the label, which is a continuous target. We introduce an additional structure for the label space Y, where we divide Y into B groups (bins) with equal intervals, i.e., [y0, y1), [y1, y2), . . . , [yB1, yB). Throughout the paper, we use $b \in B$ to denote the group index of the target value, where $B = \{1, ..., B\} \subset Z^+$ is the index space. In practice, the defined bins reflect a minimum resolution we care for grouping data in a regression task. For instance, in age estimation, we could define δy , $yb + 1 - yb = 1$, showing a minimum age difference of 1 is of interest. Finally, we denote $z = f(x;\theta)$ as the feature for x, where $f(x;\theta)$ is parameterized by a deep neural network model with parameter θ. The final prediction y^ is given by a regression function g(·) that operates over z.

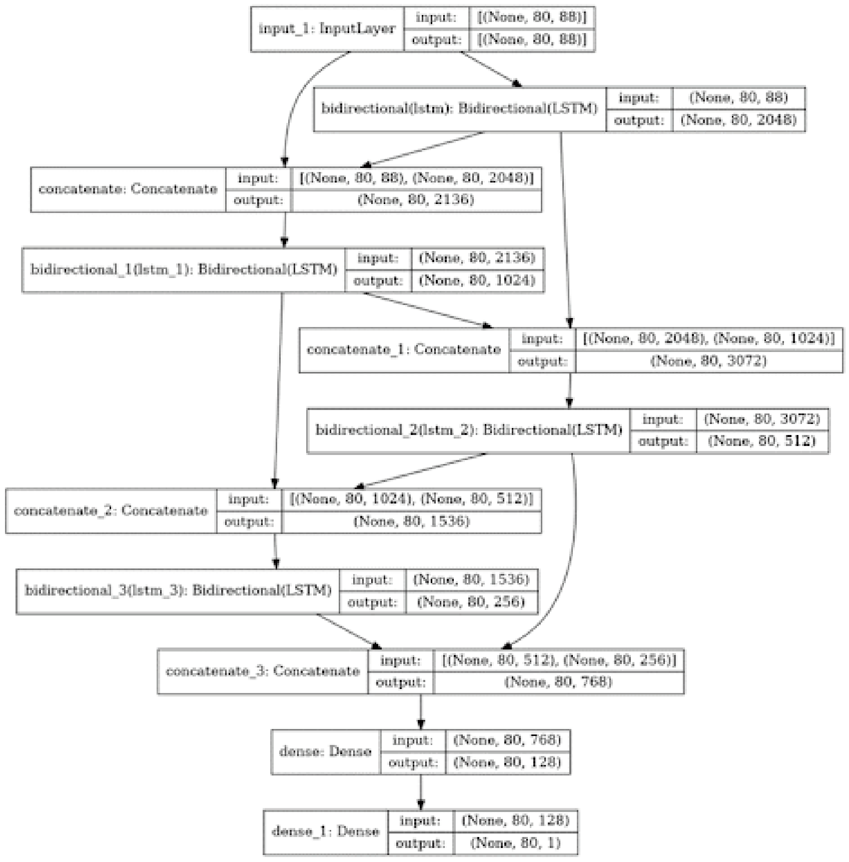

Fig. 3. Summary of BiLSTM model

LDS for Imbalanced Data Density Estimation. LDS convolves a symmetric kernel with the empirical density distribution to extract a kernel-smoothed version that accounts for the overlap in information of data samples of nearby labels. A symmetric kernel is any kernel that satisfy $k(y, y') = k(y', y)$ and $\bigtriangledown_y k(y, y') + \bigtriangledown'_y k(y', y) = 0$, $\forall y, y' \in Y$. Note that a Gaussian or a Laplacian kernel is a symmetric kernel, while k(y, y') = yy' is not. The symmetric kernel characterizes the similarity between target values y' and any y w.r.t. their distance in the target space. Thus, LDS computes the effective label density distribution as:

$$\tilde{p}(y') \int_y k(y, y') p(y).dy$$

With p(y) is the number of appearances of label of y in the training data, and $\tilde{p}(y')$ is the effective density of label y'.

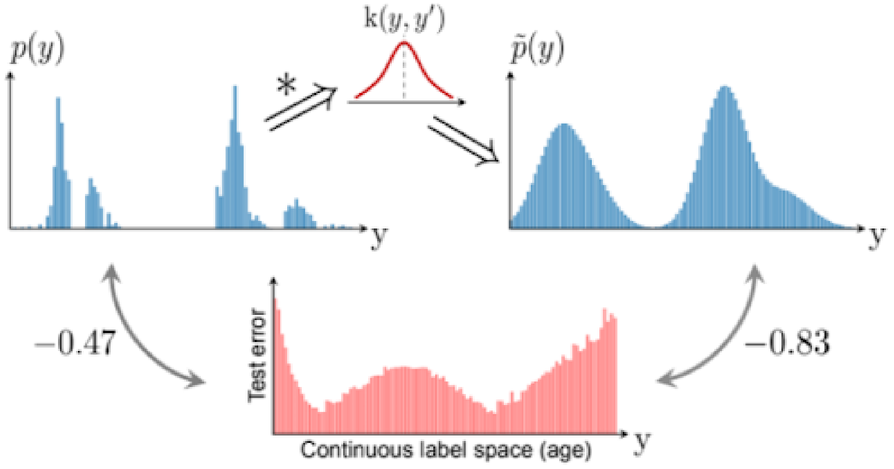

Fig. 4. Label distribution smoothing (LDS) convolves a symmetric kernel with the empirical label density to estimate the effective label density distribution that accounts for the continuity of labels.

Fig. 4 illustrates LDS and how it smooths the label density distribution. Further, it shows that the resulting label density computed by LDS correlates well with the error distribution (0.83). This demonstrates that LDS captures the real imbalance that affects regression problems.
Now that the effective label density is available, techniques for addressing class imbalance problems can be directly adapted to the Deep Imbalanced Regression context. In this case, we use re-weighting method, where we re-weight the loss function by multiplying it by the inverse of the LDS estimated label density for each target.

```
**Algorithm** We have the Algorithm of LDS:
**Input:** Training set D = $\{(x_i, y_i)\}_{i=1}^{N}$, bin size $\Delta b$, symmetric kernel distribution k(y, y').
Calculate the empirical label density distribution p(y) based on $\Delta b$ and D.
Calculate the effective label density distribution $\tilde{p}(y') \int_y k(y, y') p(y).dy$.
/* Example: Combine LDS with loss inverse re-weighting */
**for all** $(x_i, y_i) \in D$ **do**
    Compute weight for each sample as $w_i = \frac{c}{\tilde{p}(y_i)} \propto \frac{1}{\tilde{p}(y_i)}$ (constant c as scaling factor)
**end for**
**for** number of training iterations **do**
    Sample a mini-batch $\{(x_i, y_i, w_i)_{i=1}^{m}\}$ from D
    Forward $\{x_i\}_{i=1}^{m}$ and get corresponding predictions $\{y_i\}_{i=1}^{m}$
    Do one training step using the weighted loss $\frac{1}{m}\sum_{i=1}^{m} w_i \mathcal{L}(\hat{y}_i, y_i)$
**end for**
```

3.4 DenseWeight

We apply another sample weighting approach for imbalanced regression datasets called DenseWeight and, based on this, a cost-sensitive learning method for imbalanced regression with neural networks called DenseLoss. This is visualized in Fig. 5: (i) It approximates the density function of the training target values using KDE. (ii) The resulting density function forms the basis for calculating DenseWeight's weighting function. (iii) DenseLoss assigns each data point in the training set a weight according to DenseWeight, increasing the influence of rare data points on the loss and the gradients. We introduce a single, easily interpretable hyperparameter, which allows us to configure to which extent we shift a model's focus towards rare regions of the target variable's distribution.

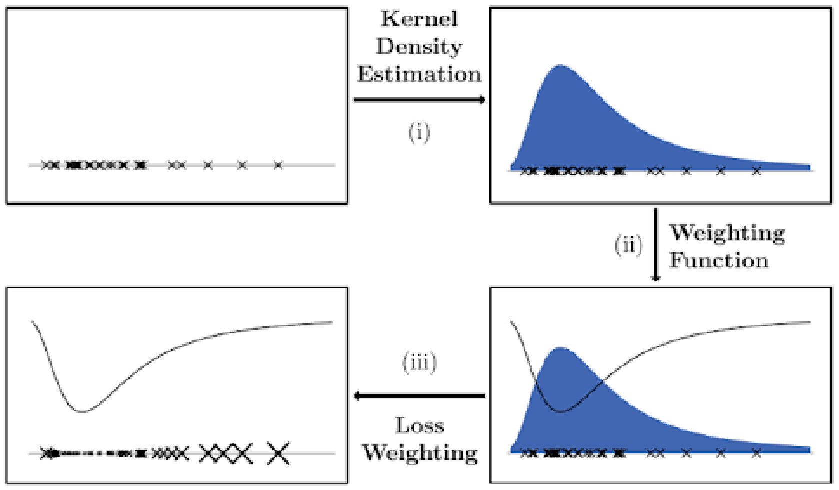

Fig. 5. Given the target values of all training examples, (i) compute a kernel density estimation (KDE) that approximates the target value distribution, (ii) calculate a weighting function from the resulting probability density function, and (iii) weight the loss for each data point in the training procedure.

DenseWeight The goal is to weight individual data points based on the rarity of their target values. Thus, we want to calculate a weight for each sample inversely proportional to the probability of the target value's occurrence. This is similar to the relevance functions used by the resampling approach SMOGN but DenseWeight base its weighting directly on the target distribution's density function instead of box plot statistics (Branco et al. 2017). This density-based weighting scheme is called DenseWeight. We design its weighting function f_w so that the degree of weighting can be controlled by a hyperparameter $\alpha \in [0, \infty)$ with the following properties.

P.1 Samples with more common target values get smaller weights than rarer samples.

P.2 f_w yields uniform weights for $\alpha = 0$, while larger values further emphasize the weighting scheme. This provides intuition for the effects of α.

P.3 No data points are weighted negatively, as models would try to maximize the difference between estimate and true value for these data points during training.

P.4 No weight should be 0 to avoid models ignoring parts of the dataset.

P.5 The mean weight over all data points is 1. This eases applicability for model optimization with gradient descent as it avoids influence on learning rates.
These weights can theoretically be applied to any type of machine learning model that allows for sample weighting to allow fitting models better suited for the estimation of rare cases. We will use them for our cost-sensitive imbalanced regression approach for neural networks DenseLoss in this work. Next, we define how the rarity of a data point is measured, before designing the weighting function f_w with these properties.

Measure of rarity In order to weight data points based on the rarity of their target values, we need a measure of rarity for f_w . To this end we want to determine the target variable's density function p. Values of density functions can be interpreted as relative measures of density, allowing the distinction between rare and common value ranges (Grinstead and Snell 2012). To obtain density function p for a dataset with N data points and target values Y = {y1, y2, ... , yN}, DenseWeight approximates it with KDE, which is a non-parametric approach to estimating a density function (Silverman 1986):

$$p(y) = \frac{1}{Nh} \sum_{i=1}^{N} K(\frac{y - y_i}{h})$$

With kernel function K and bandwidth h. Literature shows that the choice of kernel function is rather unimportant for KDE with only small differences between common kernel functions (Chen 2017), which is why we use Gaussian kernels. For bandwidth selection, we found that, in practice, the automatic bandwidth selection method Silverman's rule (Silverman 1986) produces density functions which follow the distributions well for the datasets used in this work. KDE allows calculating a density value per data point. Since it does not affect relative density information, we can normalize all data points' density values in the training set to a range between 0 and 1:

$$p'(y) = \frac{p(y) - min(p(Y))}{max(p(Y)) - min(p(Y))}$$

Where p(Y) is the element-wise application of p to Y.

This normalized density function $p' \in [0,1]$ provides intuitively interpretable values. For example, the data point in the most densely populated part of Y is assigned a value of 1, while the data point in the most sparsely populated part of Y is assigned a value of 0. Note that this normalization does not work for completely uniform data but there is no reason to apply DenseWeight with uniformly distributed data anyways.

Weighting function In this section, we introduce DenseWeight 's final weighting function f_w in a step wise manner. To this end, we use the normalized density function p, hyperparameter α, and a small, positive, real-valued constant . Initially, we have a basic weighting function:

$$f'_w(a, y) = 1 - ap'(y)$$

This function already satisfies properties P.1 and P.2, since p' yields larger values for rare data points compared to more common data points and α scales p, controlling the strength of density-based weighting. Setting $\alpha = 0$ has the intuitive effect of disabling density-based weighting, while $\alpha = 1$ leads to the most common data point's weight reaching 0 in this basic weighting function. Accordingly, all weights are positive for $\alpha < 1$, while $\alpha > 1$ leads to negative weights for the most common data points. The defined behavior of the values 0 and 1 provides intuition for the choice of sensible values. However, there are still desired properties which f'_w does not satisfy. For example, we want to avoid negative and 0 weights as described in properties P.3 and P.4. To this end, we clip f'_w at the small, positive, real valued constant ϵ:

$$f"_w(a, y) = max(1 - ap'(y), \epsilon)$$

Function f''_w satisfies all desired properties except for P.5. Using it for weighting a cost- sensitive model optimization approach based on gradient descent like DenseLoss would influence the learning rate since is scaling all gradients without any normalization.
Changing would also require a different learning rate if the magnitude of model parameter changes is to stay consistent. Finding a sensible learning rate would be tedious. Dividing f''_w by its mean value over all data points of the training set corrects this. The mean weight becomes 1, preventing a change in the average gradients magnitude. This leads us to DenseWeight's weighting function f_w :

$$f_w(a, y) = \frac{f"_w(a, y)}{\frac{1}{N}\sum_{i=1}^{N} f"_w(a, y_i)} = \frac{max(1 - ap'(y), \epsilon)}{\frac{1}{N}\sum_{i=1}^{N} (max(1 - ap'(y_i), \epsilon))}$$

Fig. 5 visualizes DenseWeight for a Gaussian distributed target variable. With increasing , weight differences between common and rare data points are emphasized more strongly. Setting $\alpha = 1$ yields a weighting function that barely reaches ϵ for the most common data points. To push more of the common data points towards a weight of ϵ, α can be increased beyond 1.
The most suitable α value for a specific task can be found by conducting a hyperparameter study. DenseLoss 's allows for easy adjustment of the trade-off between

focusing on common or rare parts of a dataset. Thus, there needs to be a definition (at least implicitly) for the meaning of performance regarding the task at hand, making it impossible to give a general rule for an optimal α.

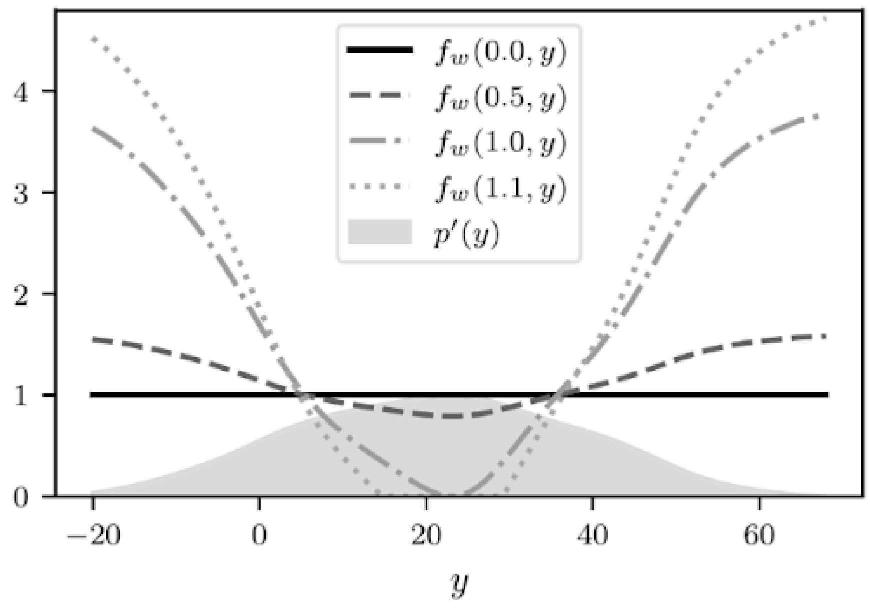

Fig. 6. DenseWeight for data sampled from a Gaussian distribution. With $\alpha = 0$ each sample's weight is 1. Higher stretches the function, emphasizing density differences. For $\alpha > 1$ (neglecting ϵ) the function is partly clipped to avoid negative weights.

3.5 Custom loss function

Given model estimates Ŷ = {ŷ1, ŷ2, ... , ŷN}, aim to minimize a metric M that is incorporated into a loss function L for which we can apply sample weighting. When combining DenseWeight and sample weighting for loss functions we obtain a cost-sensitive approach for regression with imbalanced datasets, which we call DenseLoss:

$$L_{DenseLoss}(\alpha) = \frac{1}{N}\sum_{i\ =1}^{N} f_w(a, y_i).M(\hat{y}_i, y_i)$$

Weighting the loss per sample with DenseWeight affects the gradients' magnitude calculated based on each sample. Rarer samples yield larger gradients than more common samples even when the model's estimates are equally good according to the chosen metric. Thus, the gradients focus more on achieving best possible estimates for rare samples than for common samples. When updating model parameters with these gradients, this leads to models better suited for estimating rare samples. Similarly to cost-sensitive imbalanced classification methods weighting samples according to the inverse class frequency (Cui et al. 2019), DenseLoss is also cost-sensitive as it adapts the cost for rare samples in comparison to common samples according to the weights assigned by DenseWeight. In contrast to SMOGN, the state-of-the-art method for imbalanced regression, our approach works at the algorithm-level instead of the data-level. Weighting a loss function with DenseWeight is a very flexible approach in principle as it allows for optimization using any gradient descent optimization algorithm and any metric.

4 Experiment

4.1 Experimental Setup

The solution is implemented with Scikit-learn and TensorFlow to process the data and build the model. The model's initial hyperparameters are mostly in default state or taken for the original papers. We use Adam as an optimizer and ReduceLROnPlateau as a learning rate scheduler. We determine when to stop training with an early stopping based on the MAE score on the validation set and use dropout to regularize a model. Finally, we evaluate the results according to the mean absolute error.

4.2 Evaluation and Metrics

Following (Liu et al.,2019), we divide the target space into three disjoint subsets: many-shot region (bins with over 100 training samples), medium-shot region (bins with 20100 training samples), and few-shot region (bins with under 20 training samples), and report results on these subsets, as well as overall performance. For metrics, we use common metrics for regression, such as the mean-average-error (MAE) and R2-score.
Vanilla model: We use the term VANILLA to denote a model that does not include any technique for dealing with imbalanced data. To combine the vanilla model with LDS, we re-weight the loss function by multiplying it by the inverse of the LDS estimated density for each target bin. For DenseWeight, we multiply the loss by DenseWeight.

Metrics	MAE				MSE			
Shot	All	Many	Median	Few	All	Many	Median	Few
VANILL A	**0.660**	**0.243**	0.854	1.891	**1.253**	**0.111**	1.751	5.948
VANILL A + LDS	0.942	0.351	1.229	2.204	2.504	0.301	3.539	8.579
VANILL A + Dense Weight ($\alpha = 1$)	1.002	0.600	1.160	3.345	2.569	0.559	3.222	19.716
VANILL A + Dense Weight ($\alpha = 2$)	0.742	0.334	**0.785**	**1.231**	1.436	0.287	**1.577**	5.110

Comparison of MAE results for different models and shots.

Our experiments show that LDS has no effect to reduce MAE compared to VANILLA model. The improvement of DenseWeight method depends on the way we set parameter appropriately to find the most suitable weights for Ventilator Pressure dataset. However, Density-based Weighting for imbalanced regression method sacrificed the overall result to reduce results on few shot and that should be applied in cases where rare target values are significantly meaningful and important for the application.

5 Conclusion

We introduce the Deep Imbalanced Regression tasks that learn from Ventilator Pressure data with continuous targets and apply two simple and easy-to-use algorithms for weighting imbalance data. We also present a simulator based on ResBiLSTM to combine with weighting sample methods and predict ventilator pressure. We found some drawbacks of these techniques to Ventilator Pressure data. Nevertheless, we believe that if we find the most effective way to calculate the loss for imbalanced data with weights or another method to find weights for the specific dataset, we can generate a better model to predict rare values in imbalanced data without sacrificing overall results.

References

1. Author, Yuzhe Yang, Kaiwen Zha, Ying-Cong Chen, Hao Wang, Dina Katabi, Delving into Deep Imbalanced Regression, 13 May 2021
2. Author, Michael Steininger1, Konstantin Kobs1, Padraig Davidson1, Anna Krause1, Andreas Hotho1, Density-based weighting for imbalanced regression, 2021
3. Author, Paula Branco, Luís Torgo, Rita P. Ribeiro, SMOGN: a Pre-processing Approach for Imbalanced Regression, 2017
4. Author, Abdelghani Belgaid, Deep Sequence Modeling for Pressure Controlled Mechanical Ventilation, March 4, 2022.
5. Author, Jiawei Ren1, Mingyuan Zhang1, Cunjun Yu2, Ziwei Liu1, Balanced MSE for Imbalanced Visual Regression.
6. Liu, Z., Miao, Z., Zhan, X., Wang, J., Gong, B., and Yu, S. X. Large-scale long-tailed recognition in an open world. In CVPR, 2019.