Machine Learning manuscript No.
(will be inserted by the editor)

Density-based Weighting for Imbalanced Regression

Michael Steininger · Konstantin Kobs · Padraig Davidson · Anna Krause · Andreas Hotho

Received: 22 November 2020 / Revised: 23 April 2021 / Accepted: 16 June 2021

Abstract In many real world settings, imbalanced data impedes model performance of learning algorithms, like neural networks, mostly for rare cases. This is especially problematic for tasks focusing on these rare occurrences. For example, when estimating precipitation, extreme rainfall events are scarce but important considering their potential consequences. While there are numerous well studied solutions for classification settings, most of them cannot be applied to regression easily. Of the few solutions for regression tasks, barely any have explored cost-sensitive learning which is known to have advantages compared to sampling-based methods in classification tasks.

In this work, we propose a sample weighting approach for imbalanced regression datasets called *DenseWeight* and a cost-sensitive learning approach for neural network regression with imbalanced data called *DenseLoss* based on our weighting scheme. DenseWeight weights data points according to their target value rarities through kernel density estimation (KDE). DenseLoss adjusts each data point's influence on the loss according to DenseWeight, giving rare data points more influence on model training compared to common data points. We show on multiple differently distributed datasets that DenseLoss significantly improves model performance for rare data points through its density-based weighting scheme. Additionally, we compare DenseLoss to the state-of-the-art method SMOGN, finding that our method mostly yields better performance. Our approach provides more control over model training as it enables us to actively decide on the trade-off between focusing on common or rare cases through a single hyperparameter, allowing the training of better models for rare data points.

Keywords Imbalanced regression · Cost-sensitive learning · Sample weighting · Kernel-density estimation · Supervised learning.

This is a post-peer-review, pre-copyedit version of an article published in Machine Learning. The final authenticated version is available online at: https://dx.doi.org/10.1007/s10994-021-06023-5

Chair of Computer Science X, University of Würzburg, Germany
E-mail: {steininger,kobs,davidson,anna.krause,hotho}@informatik.uni-wuerzburg.de

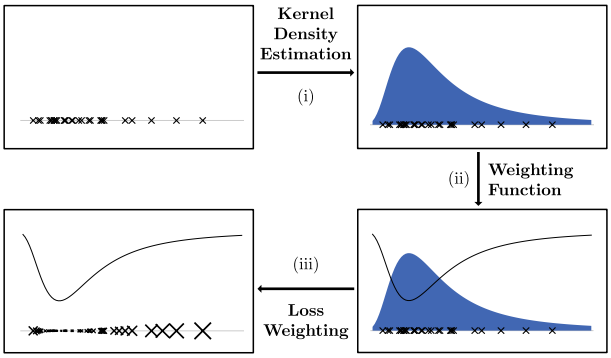


Fig. 1 Given the target values of all training examples, we (i) compute a kernel density estimation (KDE) that approximates the target value distribution, (ii) calculate a weighting function from the resulting probability density function, and (iii) weight the loss for each data point in the training procedure

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 cannot 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 classifi-

cation tasks suggests that exploring this direction for imbalanced regression could also lead to better methods in this domain (Krawczyk 2016).

In this paper, we propose a 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. Our approach is visualized in Figure 1: (i) We approximate 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.

Our contributions are as follows: (i) We propose DenseWeight, a sample weighting approach for regression with imbalanced data. (ii) We propose DenseLoss, a cost-sensitive learning approach based on DenseWeight for neural network regression models with imbalanced data. (iii) We analyze DenseLoss's influence on performance for common and rare data points using synthetic data. (iv) We compare DenseLoss to the state-of-the-art imbalanced regression method SMOGN, finding that our method typically provides better performance. (v) We apply DenseLoss to the heavily imbalanced real world problem of downscaling precipitation, showing that it is able to significantly improve model performance in practice.

2 Related Work

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. In contrast to Kamalov (i) we use KDE to measure rarity on a continuous target domain and not to model features, (ii) we do not generate samples, and (iii) we devise our method for regression. 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 propose 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

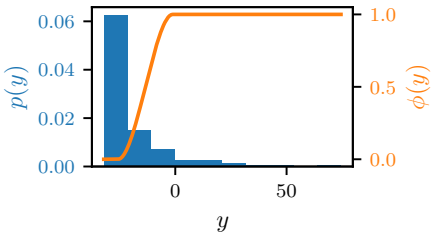


Fig. 2 SmoteR and SMOGN's relevance function ϕ for pareto-distributed data

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 proposal 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 t_R and a relevance function ϕ. They use an automatic method for obtaining ϕ based on box plot statistics through which specific control points on the target domain are obtained. Each control point is a tuple $(y, \phi(y), \phi'(y))$, where $\phi'(y)$ — the derivative of relevance $\phi(y)$ — is always set to 0, since control points are assumed to be local extrema of relevance. The relevance function ϕ is then defined with piecewise cubic Hermite interpolation through these control points (Ribeiro 2011). Figure 2 shows a resulting ϕ for data following a pareto distribution. This automatic method for obtaining ϕ 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 ($\phi(y) > t_R$) 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 undersampled. 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 propose an algorithm-level, cost-sensitive method for imbalanced regression called DenseLoss using our density-based weighting scheme DenseWeight. 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 does 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 Method

In this section we introduce DenseWeight, our proposed sample weighting approach for imbalanced datasets in regression tasks, and DenseLoss, our cost-sensitive learning approach for imbalanced regression problems based on DenseWeight.

3.1 DenseWeight

Our 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 we base our weighting directly on the target distribution's density function instead of box plot statistics (Branco et al. 2017). We call our density-based weighting scheme 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 α 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.

3.1.1 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 $Y = \{y_1, y_2, \ldots, y_N\}$, we approximate it with KDE, which is a non-parametric approach to estimating a density function (Silverman 1986):

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

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:

$$p'(y) = \frac{p(y) - min(p(Y))}{max(p(Y)) - min(p(Y))}\,, \tag{2}$$

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.

3.1.2 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 ϵ. Initially, we define a basic weighting function:

$$f'_w(\alpha, y) = 1 - \alpha p'(y)\,. \tag{3}$$

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 α 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:

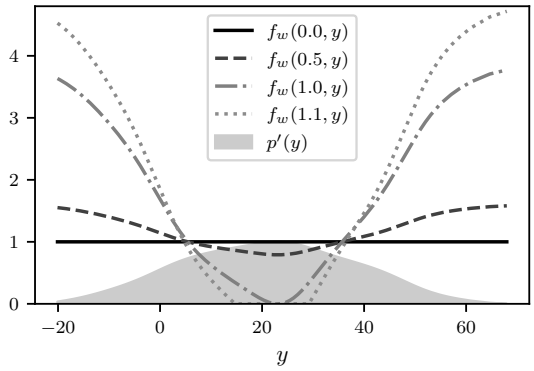


Fig. 3 DenseWeight for data sampled from a Gaussian distribution. With $\alpha = 0$ each sample's weight is 1. Higher α stretches the function, emphasizing density differences. For $\alpha > 1$ (neglecting ϵ) the function is partly clipped to avoid negative weights

$$f''_w(\alpha, y) = max(1 - \alpha p'(y), \epsilon)\,. \tag{4}$$

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 α is scaling all gradients without any normalization. Changing α 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:

$$f_w(\alpha, y) = \frac{f''_w(\alpha, y)}{\frac{1}{N}\sum_{i=1}^{N} f''_w(\alpha, y_i)} = \frac{max(1 - \alpha p'(y), \epsilon)}{\frac{1}{N}\sum_{i=1}^{N}(max(1 - \alpha p'(y_i), \epsilon))}\,. \tag{5}$$

Figure 3 visualizes DenseWeight for a Gaussian distributed target variable. With increasing α, 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 α 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.

3.2 DenseLoss

In this work we focus on neural networks due to their broad applicability to both simple and complex regression problems through the use of either relatively small

multilayer perceptrons (MLPs) or large deep learning neural networks, respectively. Neural networks are typically optimized with gradient descent optimization algorithms that, given model estimates $\hat{Y} = \{\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_N\}$, 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:

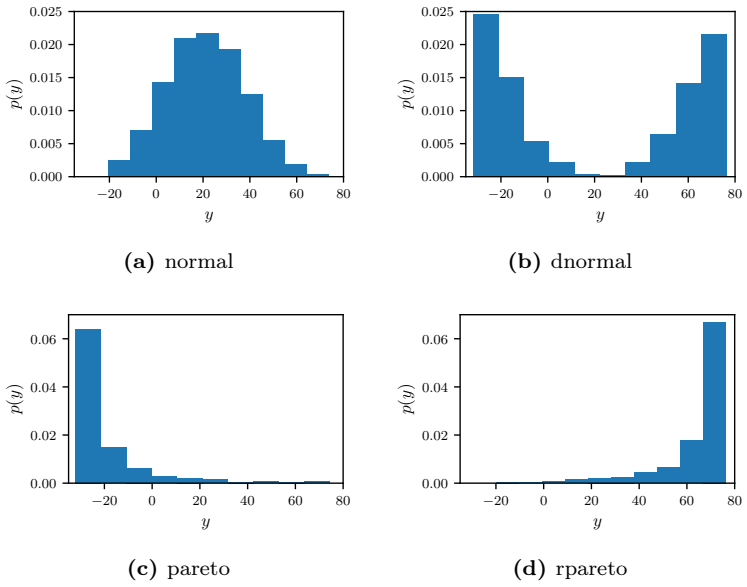
$$L_{DenseLoss}(\alpha) = \frac{1}{N} \sum_{i=1}^{N} f_w(\alpha, y_i) \cdot M(\hat{y}_i, y_i)\,. \tag{6}$$

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. Models trained with DenseLoss are expected to typically perform better for rare cases compared to models trained with uniform sample weights, as we show next.

4 Experiments

We evaluate DenseWeight and DenseLoss with three experiments: a case study on synthetic data, a comparison to the state-of-the-art, and an application to a real world task. First, we examine with synthetic datasets how DenseLoss behaves for different α values and different distribution characteristics, validating that DenseLoss is working as designed. Second, we compare DenseLoss to the state-of-the-art imbalanced regression method SMOGN, showing that our algorithm-level method can typically provide better performance for rare data points than SMOGN's data-level approach. Finally, we apply DenseLoss to the real world task statistical downscaling of precipitation, proving that it can also work for larger datasets and more complex neural network architectures.

For all experiments, we use the library KDEpy's convolution-based KDE implementation FFTKDE. It provides fast density estimation that can, however, only be evaluated on an equidistant grid (Odland 2019). Thus, for each training dataset we span a grid over the target range and assign each data point the density of the closest grid point. We use an equidistant grid with 4096 points, which is 4 times KDEpy's default resolution, to avoid potential negative effects on our method due to low KDE accuracy. In general, the quality of the resulting density function with respect to the real target distribution can be limited by low quality training data with noisy outliers. While we did not encounter such problems in this work, careful data cleaning and tuning of the KDE may improve this for such datasets. To provide a small, positive value to DenseLoss's clipping constant ϵ we set it to 10^{-6} for all experiments. When we report



(a) normal **(b)** dnormal

(c) pareto **(d)** rpareto

Fig. 4 Distribution of the target variable for each synthetic dataset

significantly different results for the experiments, the statistical significance is calculated for the metrics on test datasets with the Wilcoxon signed-rank test (Wilcoxon 1945) and a significance level of 0.05. Our experiments' code and data is available[1].

4.1 Case Study with Synthetic Data

In this case study, we aim to validate the expectation that models trained with DenseLoss achieve improved performance in underrepresented parts of the dataset compared to a regular training procedure. To this end, we use four synthetic datasets with varying characteristics: two heavy-tailed datasets, following a pareto (*pareto*) and a reversed pareto distribution (*rpareto*), respectively. Furthermore, we use a Gaussian dataset (*normal*) and a dataset built from two Gaussians with a sparse middle area (*dnormal*). Figure 4 shows their target distributions. We train models with DenseLoss and different α values to gain insight into the practical effects of different degrees of DenseWeight.

4.1.1 Dataset Creation

We use an MLP as a random function to generate synthetic datasets. This guarantees that the function can be learned again by an MLP in theory. Our network's parameters are initialized with a standard Gaussian distribution. This network is provided with 200 000 sets of 10 features each. The features are also drawn from a standard Gaussian distribution. The network consists of 3 hidden layers with 10 neurons each and ReLU (Nair and Hinton 2010) activation. The final hidden layer is connected to a

[1] https://github.com/SteiMi/density-based-weighting-for-imbalanced-regression

single neuron with linear activation to obtain target values for a regression task. From the resulting 200 000 data points 10 000 were sampled in such a way that there are uniformly distributed target values. This uniform dataset's target values range from −32.13 to 76.42. Then, for each dataset a probability density function is defined corresponding to the desired target distribution. 1000 data points are sampled from the uniform dataset weighted by the samples' desired densities, creating the datasets *pareto*, *rpareto*, *normal*, and *dnormal*. Figure 4 visualizes their target variable distributions.

Each dataset is split randomly in a training (60 %), validation (20 %), and test (20 %) set. The resulting splits are inspected to confirm that their target variables are similarly distributed. Otherwise it would be possible that sparsely sampled ranges in the target variable are not represented in a split through unfortunate random sampling.

4.1.2 Experimental Setup

To illustrate how DenseLoss affects model performance for underrepresented parts of datasets based on our weighting scheme DenseWeight, we conduct a parameter study to examine the effects of different α values. Therefore, we train models with α values ranging from 0.0 to 2.0 with steps of 0.1. To strengthen confidence in the results of this experiment we train 20 model instances per α which are used for testing statistical significance with the Wilcoxon signed-rank test and a significance level of 0.05.

The MLP used is structurally equal to the data generator network. Thus, this model also consists of 3 hidden layers with 10 neurons each and ReLU activation as well as one neuron with linear activation for the output layer. Instead of initializing parameters from a standard Gaussian distribution, we use Kaiming Uniform initialization (He et al. 2015). DenseLoss is the loss function used in conjunction with the metric mean squared error (MSE). The model is trained with Adam optimization (Kingma and Ba 2014), a learning rate of 10^{-4}, and a weight decay coefficient of 10^{-9}. Training is run for at most 1000 epochs, but it is stopped early if the validation loss is not improving for 10 epochs in a row. This improves generalization performance (Prechelt 1998).

4.1.3 Results

To evaluate model performance for separate parts of the target domain, we bin the test data points based on their target value. Each bin spans 20 % of the target variable's range in the test set. We rank these bins per dataset by the number of data points. The bin with the fewest (most) samples has bin rank 1 (5) and is called the least (most) common bin. This allows performance comparisons between similarly rare bins over all datasets. We calculate the root mean squared error (RMSE) and mean absolute error (MAE) for each individual model instance of the 20 instances per tested configuration.

Our MLP without DenseLoss achieves on average over the 20 runs RMSEs (MAEs) between 3.53 (2.70) and 6.75 (5.47) for the most common bins, i.e. bin rank 5, and between 6.68 (6.26) and 27.10 (26.74) for the rarest bins, i.e. bin rank 1, across the synthetic datasets. We find that DenseLoss with, for example, $\alpha = 1.0$ improves average RMSE (MAE) for the rarest bins by between 1.21 (1.48) and 7.02 (7.00) while increasing it for the most common bins by between 1.12 (0.90) and 1.68 (1.49).

Figure 5 visualizes the mean RMSE of models trained with different α values over all synthetic datasets for different bin ranks. DenseLoss typically improves performance in sparsely sampled bins (bin ranks 1 to 3) with a suitable α value. As expected, DenseLoss tends to reduce performance for bins with many samples (bin ranks 4 and

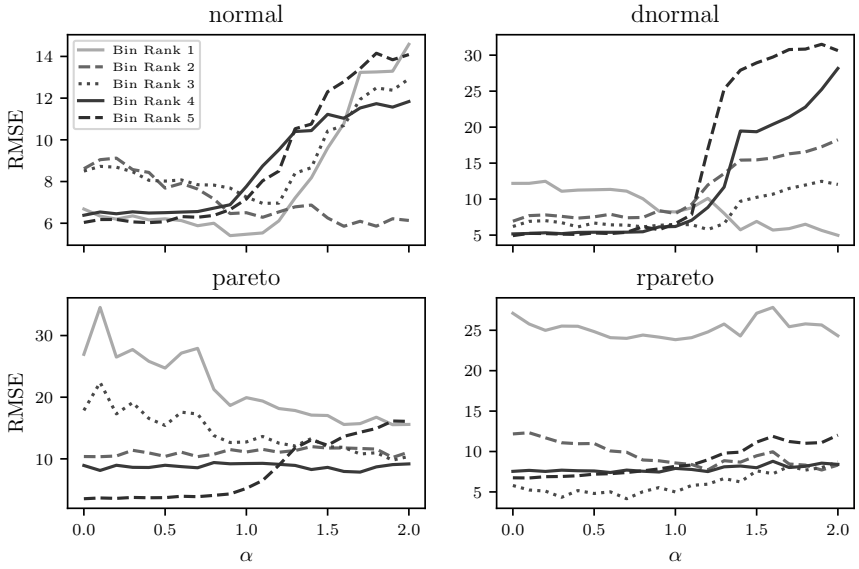


Fig. 5 Mean RMSE per α and bin rank over the synthetic datasets. Bins are ranked in each test set by sample size. Bins with rank 1 (5) contain the fewest (most) samples

5). We find that most α values greater than 0 lead to improvements in rare bins. For example, for *pareto* all tested configurations with $\alpha \geq 0.8$ yielded improvements in the rarest bin and the same is true for all runs with $\alpha \geq 0.2$ for *dnormal*. For *rpareto* all runs with DenseLoss enabled ($\alpha > 0.0$) improved the rarest bin except for $\alpha = 1.5$ and $\alpha = 1.6$, where performance dropped slightly. *normal*'s rarest bin is improved with $0.1 \leq \alpha \leq 1.2$, which is discussed in the next paragraph. As described at the beginning of Section 4 we conduct statistical significance tests to strengthen confidence in our results. When considering $\alpha = 1.0$, which seems to provide good performance for rare data points across all datasets, we find that the performance for the rarest bin has improved significantly compared to not using DenseLoss ($\alpha = 0.0$) for each dataset. Bin rank 2 is improved significantly with $\alpha = 1.0$ for *normal* and *rpareto*, while bin rank 3 is significantly better for *normal* and *pareto*. We also see with $\alpha > 1.0$ that the performance for the most common bin deteriorates considerably for *normal* and *dnormal*, as the weight of more and more of these data points is pushed towards ϵ. This effect is also noticeable in the other bin ranks albeit with reduced strength the rarer the bins get. Interestingly, this performance degradation seems less pronounced for both pareto datasets. We find very similar results with regards to the metric MAE.

Figure 6 shows detailed results for datasets *normal* and *pareto*. Bins are identified by bin rank and ordered to correspond to the dataset's distribution plot at the top, thus visualizing RMSE and density from the lowest (left-most bin) to the highest target values (right-most bin). Setting α to around 1 provides improved performance for rare target ranges while only slightly reducing performance for common target ranges. For example, with $\alpha = 1.0$ in *pareto* we observe an increase in RMSE of 1.68 in the most common bin with rank 5 and a drop in RMSE of 7.02 in the least common bin with rank 1. In general, error for samples in rare target ranges tends to decrease with increasing

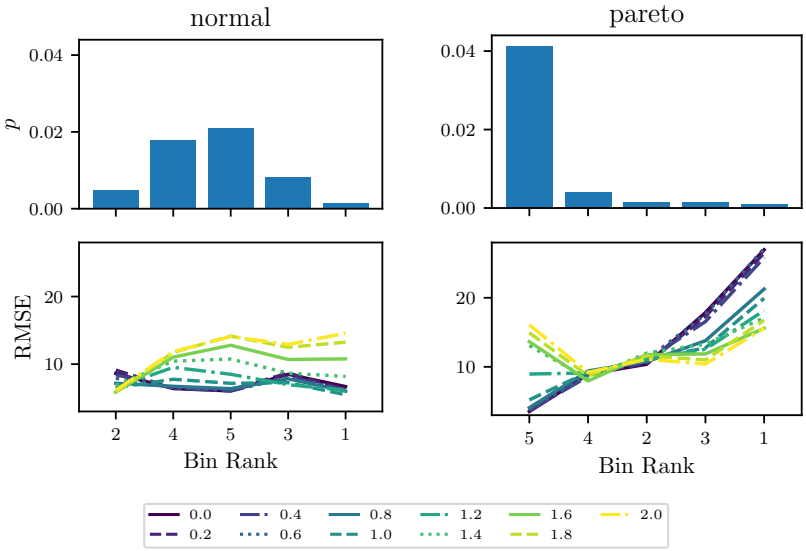


Fig. 6 Mean RMSE per test bin over 20 runs for datasets *normal* (left) and *pareto* (right). Bar charts show the density per bin in the test set. Line plots visualize the mean RMSE per test bin for the α values shown in the box at the figure's bottom

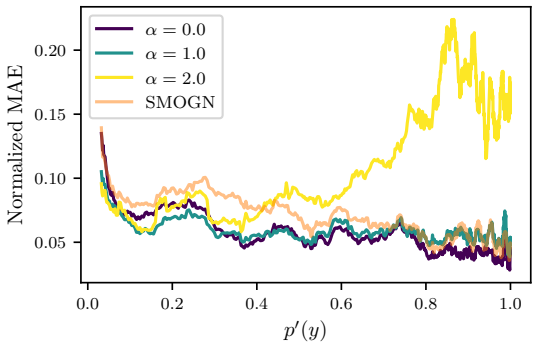


Fig. 7 Normalized MAE for test samples from all synthetic datasets per normalized density. Graph is smoothed via moving mean (window size 30) to ease interpretability

α while performance in common target ranges mostly deteriorates. For *normal*'s rarest bin with rank 1 too large α values ($\alpha \geq 1.4$) also show performance degradation. We hypothesize that this can occur when the target range in the training set has very few data points and the neighboring, more common data points are assigned weights close to 0. In this case the model seems to struggle to learn a general function for the higher target ranges, because of the effectively small number of samples there.

Given the continuous nature of regression datasets it is also interesting to regard the performance over the datasets' target domains. To account for different distributions

among the datasets for this evaluation we calculate the normalized density per test data point's target value (as defined in Equation (2)) through KDE (same parameters as for DenseWeight's KDE) on the target variable of its respective test dataset. In contrast to before, we do not use this normalized density to weight samples or train models but instead use it as a dataset-independent metric for each sample's rarity within its test dataset. This rarity thus provides us with a dataset-independent proxy of the target variable domains. It is independent from the rarity used during model training and does not influence the estimates for the test samples. Also, we calculate the MAE over the 20 runs for each test data point of each dataset. To enable a continuous evaluation over all datasets we normalize the MAE via division by the difference between the maximum and the minimum value of its respective test dataset's target variable. The normalized MAE in conjunction with the normalized densities allow us to plot Figure 7 which visualizes the normalized MAE depending on the data point rarity across all datasets for regular training ($\alpha = 0.0$), DenseLoss ($\alpha = 1.0$ and $\alpha = 2.0$), and also the state-of-the-art imbalanced regression method SMOGN. To account for the high variability and to improve interpretability we smoothed the plot by applying a moving mean with a windows size of 30 data points over the 800 total test data points. We find that DenseLoss with both $\alpha = 1.0$ and $\alpha = 2.0$ typically reduces error for very rare samples ($\sim p'(y) < 0.15$). Performance with $\alpha = 2.0$ deteriorates considerably for more common data points ($\sim p'(y) > 0.4$) while performance of $\alpha = 1.0$ remains close to $\alpha = 0.0$ up until around $p'(y) > 0.75$ where a gap emerges.

While this experiment mainly analyzes DenseLoss in a controlled manner we also applied SMOGN to our synthetic datasets, finding mostly better performance for $\alpha = 1.0$ than SMOGN, when applying SMOGN as described in Section 4.2. Rare parts in *pareto* and *rpareto* were identified automatically; rare parts in *normal* and *dnormal* were identified manually, since the automatic method wrongly deemed all samples relevant. For *normal* and *dnormal* we used the control points $(-10, 1, 0), (20, 0, 0), (50, 1, 0)$ and $(0, 0, 0), (20, 1, 0), (50, 0, 0)$, respectively. Resulting relevance functions are visualized in the Appendix. Since SMOGN's automatic method for obtaining ϕ only works for datasets where rare values are also extreme, it is not suited for *dnormal*. With our manual control points it is still not ideal as it incorrectly deems low target values as relevant, but it is substantially better than considering all data points relevant. *normal*'s manual ϕ shows no such issues. When considering binned evaluation we find that DenseLoss with $\alpha = 1.0$ performs significantly better than SMOGN for the rarest bin on all datasets except *pareto*.

This experiment confirms that DenseLoss allows shifting a model's focus to rarer cases away from the cases it would have focused on with regular training. Inspecting the model performance across the target range with varying α values enables an informed choice for the trade-off between performance in common and rare cases. Thus, DenseLoss provides additional control over model training, allowing to fit models with better performance for rare data points.

4.2 Comparison with State-of-the-Art

SMOGN can currently be considered the state-of-the-art method for imbalanced regression, as it has shown to be better than the other available method SmoteR (Branco et al. 2017). SMOGN's authors present 20 imbalanced datasets in their paper. We apply both SMOGN and DenseLoss to those datasets and compare model performances.

Neural networks trained without applying any method for imbalanced data are used as a baseline. To this end we apply both methods and the baseline to the 20 imbalanced datasets from SMOGN's test section (Branco et al. 2017). We obtain the data from their repository[2]. See the Appendix for an overview. We also compared DenseLoss with SMOGN using DenseWeight for its relevance function in the Appendix, finding similar results as presented in the following, where we compare DenseLoss to SMOGN using its default relevance function.

4.2.1 Experimental Setup

As with the synthetic data, we randomly split each dataset in a training (60 %), a validation (20 %), and a test (20 %) set. Considering the small size of some of the datasets, we inspect the splits to confirm that they are similarly distributed and redo the random split if the distributions are too different.

Models trained with DenseLoss use $\alpha = 1.0$. For SMOGN we use the python package *smogn* (Kunz 2019). Since SMOGN's authors also aim to increase performance for rare data points on these datasets we apply the same hyperparameters as they did in their paper: Rare target values are determined by their automatic method (Ribeiro 2011) as described in Section 2. Just as SMOGN's authors, we consider target values rare where the relevance function yields more than 0.8. SMOGN oversamples data points with rare target values to obtain a more balanced distribution. For oversampling SMOGN is set to consider the 5 nearest neighbor samples. The amount of Gaussian noise added for oversampling (i.e. perturbation) is set to 0.01. We use the same MLP architecture and hyperparameters as described in Section 4.1. Additionally, we repeat the experiment with the same hyperparameters but different MLP topologies, namely a deeper model (4 hidden layers with 10 neurons each), a shallower model (2 hidden layers with 10 neurons each), a wider model (3 hidden layers with 20 neurons each), and a narrower model (3 hidden layers with 5 neurons each), to confirm that our results are not due to a specific network architecture. We find very similar results for all architectures and therefore only report detailed results for one topology (3 hidden layers with 10 neurons each) for brevity. Models are trained and evaluated 20 times per dataset and method to test statistical significance with the Wilcoxon signed-rank test and a significance level of 0.05.

4.2.2 Results

As in Section 4.1, we split each test dataset into 5 equidistant bins and rank the bins by the number of samples. Metrics RMSE and MAE are calculated for each bin.

Figure 8 visualizes the number of dataset wins of DenseLoss, SMOGN, and the baseline (None) per bin rank over the 20 datasets for the metric RMSE. Due to some datasets' small sizes there are some bins without data points in the test set. This results in the bars for rank 1 and 2 not containing 20 wins, since no winner can be found for empty bins. The results show for the rarest bins (bin rank 1) that DenseLoss provides the best performance for 8 datasets while SMOGN only performs best on 3 datasets and applying no method is best for only 2 datasets. DenseLoss has the highest number of significant dataset wins against both methods in this rarest bin rank but also in bin ranks 2 to 4. For bin ranks 1 to 4, DenseLoss wins more than half of the datasets, with

[2] https://github.com/paobranco/SMOGN-LIDTA17

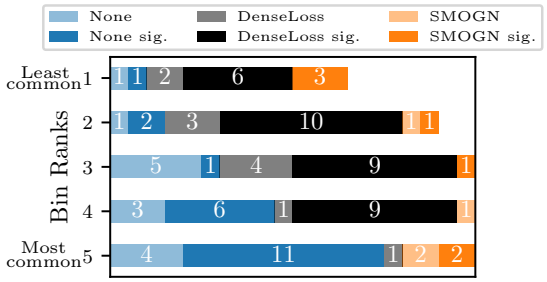


Fig. 8 Number of datasets won per method for each bin based on RMSE. Bins are ranked within each test dataset according to the number of data points. Bins with rank 1 (5) contain the fewest (most) samples. Each bar section shows the number of datasets won by a method at that bin rank. When a method's wins are denoted as "sig." they are significant with regards to both other methods. 5 test datasets had a bin without data points and 2 test datasets had 2 bins without samples. Because of this the bars for bin rank 1 and 2 are smaller as no winner can be determined for empty bins

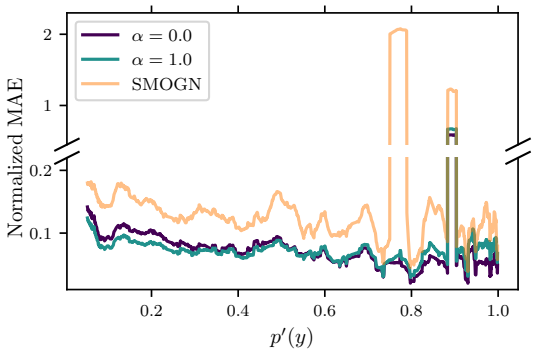


Fig. 9 Normalized MAE for test samples from all 20 datasets per normalized density. Graph is smoothed via moving mean (window size 300) to ease interpretability

most wins being statistically significant against the baseline and SMOGN. Only for bin rank 5 with the most samples, it is typically best to apply no method for imbalanced data. This, however, is expected, as the usual training method is biased towards common target values. We found very similar results for the metric MAE. Repeating this experiment with the other network architectures introduced in Section 4.2.1 further confirms these findings, as is shown in the Appendix. These results suggest that DenseLoss typically provides better performance for rare data points in comparison to the state-of-the-art imbalanced regression method SMOGN.

Similarly as described in Section 4.1.3 we analyze the performance over the datasets' target variable domains in a continuous manner. Thus, we visualize the normalized MAE per data point rarity across all datasets for regular training ($\alpha = 0.0$), DenseLoss ($\alpha = 1.0$), and SMOGN in Figure 9. To account for the high variability and to improve interpretability we smooth the plot using a moving mean with a windows size of 300

samples over the 7188 total test samples. Similarly as in the bin-wise evaluation, we find on average lower error with DenseLoss for rarer data points ($\sim p'(y) < 0.5$) compared to using no imbalanced regression method. Normalized MAE is improved by roughly 10 % for rare data points with $p'(y) < 0.3$ while the error increases with larger densities. SMOGN seems to not work well on average over all datasets even though we used the same datasets with the same hyperparameters as the original SMOGN authors used in their work. We find high variability in SMOGN's performance across the datasets with it working well for some datasets (e.g. cpuSm or acceleration) but considerably worse on most others, leading to relatively large normalized MAE regardless of density. Also note the two outlier segments in the plot showing high MAE that stem from one sample each of the dataset availPwr. Almost all models estimate extremely large values for these two samples, likely due to an unusually high feature value, leading to very large MAE for all moving mean windows that include these samples.

4.3 Statistical Downscaling of Precipitation

To show that DenseLoss can work for larger datasets and more complex neural network architectures, we apply it to the real world task statistical downscaling of precipitation. Its objective is to generate local scale precipitation projections based on spatially coarse precipitation projections stemming from Earth System Models. This can be learned based on high-resolution historical climate observations (Vandal et al. 2017).

A model that does statistical downscaling of precipitation is DeepSD (Vandal et al. 2017). It uses super-resolution convolutional neural networks to improve the resolution of precipitation data. The model is supplied with a map showing daily precipitation at a low spatial resolution. This map is similar to an image where each pixel contains precipitation data for a specific real world area. Additionally, the model is provided with a high-resolution elevation map whose pixels are aligned with the precipitation map, so that any pixel in one map represents the same area as the corresponding pixel in the other map. This information helps the model to take topography as a known influence into account (Daly et al. 2008). DeepSD's authors use the PRISM dataset (Daly et al. 2008) for precipitation data over the Continental United States and elevation data from the GTOPO30 dataset (U.S. Geological Survey 1996). Commonly, there are far less rainy days than dry days at most locations (see Appendix). Yet, especially high precipitation events are interesting as they could have considerable consequences like flooding. Thus, we apply DenseLoss to this real world task in order to improve model performance especially for these rare and extreme events. To this end, we conduct a study for the DenseLoss's α, investigating the influence of α on model performance.

4.3.1 Experimental Setup

For our study, we modified DeepSD's code[3] to include DenseLoss. As such, we use three convolutional layers with 64, 32, and 1 filters and kernel sizes of 9, 1, and 5, respectively. Model training minimizes the MSE with a batch size of 200 and the Adam optimizer with a learning rate of 10^{-4} for the first two layers and 10^{-5} for the last layer. Precipitation data is split into a training (years 1981 to 2005) and a test set (years 2006 to 2014). In contrast to the hyperparameter values described in

[3] https://github.com/tjvandal/deepsd

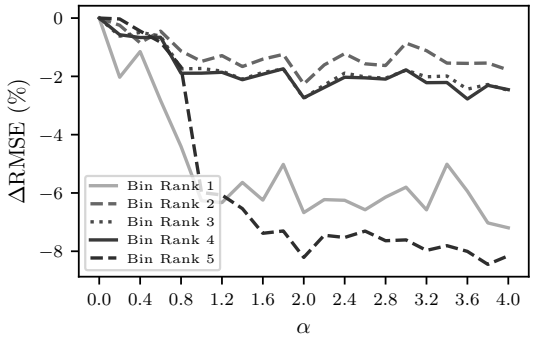


Fig. 10 Change in mean RMSE with respect to not using DenseLoss ($\alpha = 0.0$) per α for each bin rank in PRISM's test set. Bins are ranked within the test dataset according to the number of samples. The Bin with rank 1 (5) contain the fewest (most) samples

the DeepSD paper, we trained for 10^5 instead of 10^7 epochs. This saves computation time as we found no further reduction in training loss when training longer. These are still many epochs but it is necessary given the relatively low learning rates used by DeepSD. We train DeepSD to downscale from 128 km to 64 km resolution. The study tests α values from 0.0 to 4.0 with steps of 0.2. Compared to Section 4.1 we extend this range to assess at which α performance plateaus considering we have found continuous performance gains up to $\alpha = 2.0$ here. DeepSD is trained 20 times per α with different random model initializations to test statistical significance with the Wilcoxon signed-rank test and a significance level of 0.05 (Vandal et al. 2017).

4.3.2 Results

As before we split the test dataset into 5 equidistant bins, rank the bins by the number of samples and calculate RMSE and MAE for each bin.

Figure 10 visualizes the change of mean RMSE in percent with respect to regular training ($\alpha = 0.0$) in all bin ranks for models trained with different α. E.g. a ΔRMSE of $-8\,\%$ indicates an $8\,\%$ lower RMSE compared to not using DenseLoss. Interestingly, DenseLoss does not only improve performance for rare samples (e.g. bin rank 1) but also for common values (e.g. bin rank 5) here. Improvement is most pronounced in the most common and the rarest bin. This suggests that the enormous over-representation of samples with precipitation close to 0 mm may also negatively affect performance for these same very common data points. DenseLoss reduces their influence, effectively reducing the over-representation which in turn seems to lead to better performance for common samples. Performance improves with increasing α before plateauing for $\sim \alpha \geq 2.0$. Our tests for statistical significance show that for each $\alpha \geq 0.8$ performance improvements compared to $\alpha = 0.0$ are significant for all bin ranks.

As in the previous experiments (e.g. Section 4.1.3) we also analyze the performance over the target variable domain in a continuous manner. Thus, we visualize the normalized MAE per sample rarity for regular training ($\alpha = 0.0$) and DenseLoss ($\alpha > 0.0$) in Figure 11. To improve interpretability we smooth the plot by applying a moving mean with a window size of 300 000 data points over the 6 143 403 test samples. We

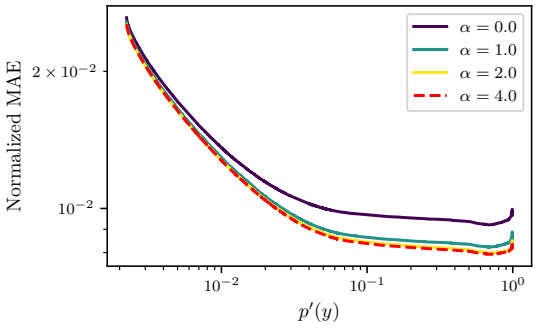


Fig. 11 Normalized MAE for PRISM test samples per normalized density. Graph is smoothed via moving mean (window size 300 000) and logarithmic for interpretability

see again that DenseLoss improves estimates for both rare (left side) and even more so for common samples (right side) here. Performance improvements tend to increase with larger α but only marginally above $\alpha = 2.0$.

In this experiment, we observe a different behavior of DenseLoss than before. Here, DenseLoss is able to improve performance across the complete target variable range instead of trading performance between common and rare samples. We hypothesize that DeepSD's capacity is large enough to learn a good function for both rare and common data points at once, while smaller models might lack the capacity for this. DenseLoss seems to allow this model to converge to an overall better solution.

5 Discussion

In this work, we have shown that DenseWeight and DenseLoss can help to improve model performance for rare data points. However, there are still aspects to discuss.

While DenseWeight can theoretically be used with any algorithm that supports sample weights, we only evaluated it with neural networks using DenseLoss. We expect to see similar results for other algorithms but we did not test this assumption here.

We compared our approach to SMOGN in Section 4.2 but not in the last experiment as we found it to be computationally infeasible. SMOGN's oversampling algorithm calculates the distance between all data points where a data point is the precipitation at one location at one time. Using the available implementation we found through initial testing that this would take years with any hardware available to us.

We did not systematically check whether the architectures used in the first two experiments generalize well but we expect decent generalization performance due to our use of early stopping. Model training is stopped when the validation loss stops improving, which inhibits overfitting. Its effectiveness shows in spot checks where we found no model with substantially higher training than test or validation performance.

Our approach introduces a new hyperparameter α, controlling the strength of density-based weighting, which must be set appropriately. While we find that setting $\alpha = 1.0$ typically provides good performance for rare samples, there can be better choices. With a validation dataset and a suitable goal it is possible to optimize α, however defining a goal is often not trivial in an imbalanced regression setting. It requires

domain knowledge to define which data points are rare and important. If this knowledge is available one could simply search for the α that minimizes the MSE on these rare and important data points to achieve optimal performance for a specific domain.

For the data splits in Section 4.2 we manually confirmed whether the splits are similarly distributed. Random splitting was not able to reliably produce splits with similar distributions given the small sizes of some datasets. While we do not believe this to influence the results, a more automatic method to this would be more objective. One could perhaps try to maximize a distribution similarity score and stop redoing splits if a certain threshold is exceeded but we did not implement this in this work.

6 Conclusion

In this work, we have proposed our sample weighting approach for imbalanced regression DenseWeight and our cost-sensitive learning method DenseLoss, tackling the problem of imbalanced regression for neural networks based on DenseWeight. We show that our approach can improve model performance for rare data points with synthetic datasets, specifically designed to represent different kinds of data distributions. Extensive hyperparameter studies for each dataset provide insight and intuition for how DenseWeight and DenseLoss's α controls a model's focus on rare in comparison to common data points. Experiments on 20 datasets show that DenseLoss typically outperforms the sampling-based method SMOGN. Applying DenseLoss to statistical downscaling of precipitation, we demonstrate its benefits on a real world task and discuss its potential for higher capacity models.

Future work includes examining ensemble approaches for DenseLoss which combine models trained with different α. Depending on α, each model is an expert in different target variable ranges. A meta-model could learn which ensemble member is likely to perform best based on a given sample's features which may lead to nearly optimal performance not only for rare samples but across the whole target range. Furthermore, it is interesting to assess the relation between model capacity and performance across the target domain with DenseLoss, following the intuition that large enough models might be able to learn a function that consistently works well for both rare and common data points. Additionally, ideas which are already established for cost-sensitive learning in imbalanced classification settings could be transferred for regression tasks. Examples include weighting based on the effective number of samples in a target region (Cui et al. 2019) or incorporating sample difficulty in the weighting (Dong et al. 2017).

References

Branco, P., Ribeiro, R. P., & Torgo, L. (2016a). "UBL: an R package for utility-based learning". In: *arXiv preprint arXiv:1604.08079*.

Branco, P., Torgo, L., & Ribeiro, R. P. (2016b). "A survey of predictive modeling on imbalanced domains". In: *ACM Computing Surveys (CSUR)* 49.2, pp. 1–50.

— (2017). "SMOGN: a Pre-processing Approach for Imbalanced Regression". In: *LIDTA*.

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). "SMOTE: synthetic minority over-sampling technique". In: *JAIR* 16, pp. 321–357.

Chen, Y.-C. (2017). "A tutorial on kernel density estimation and recent advances". In: *Biostatistics & Epidemiology* 1.1, pp. 161–187.

Cui, Y., Jia, M., Lin, T.-Y., Song, Y., & Belongie, S. (2019). "Class-balanced loss based on effective number of samples". In: *CVPR 2018*, pp. 9268–9277.

Daly, C. et al. (2008). "Physiographically sensitive mapping of climatological temperature and precipitation across the conterminous United States". In: *International Journal of Climatology* 28.15, pp. 2031–2064.

Dong, Q., Gong, S., & Zhu, X. (2017). "Class rectification hard mining for imbalanced deep learning". In: *ICCV 2017*, pp. 1851–1860.

Grinstead, C. M., & Snell, J. L. (2012). *Introduction to probability*. AMS.

He, H., Bai, Y., Garcia, E. A., & Li, S. (2008). "ADASYN: Adaptive synthetic sampling approach for imbalanced learning". In: *IJCNN 2008*. IEEE, pp. 1322–1328.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". In: *ICCV 2015*.

Hernández-Orallo, J. (2013). "ROC curves for regression". In: *Pattern Recognition* 46.12, pp. 3395–3411.

— (2014). "Probabilistic reframing for cost-sensitive regression". In: *TKDD* 8.4.

Huang, C., Li, Y., Change Loy, C., & Tang, X. (2016). "Learning deep representation for imbalanced classification". In: *CVPR 2016*, pp. 5375–5384.

Kamalov, F. (2020). "Kernel density estimation based sampling for imbalanced class distribution". In: *Information Sciences* 512, pp. 1192–1201.

Kingma, D. P., & Ba, J. (2014). "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980*.

Krawczyk, B. (2016). "Learning from imbalanced data: open challenges and future directions". In: *Progress in Artificial Intelligence* 5.4, pp. 221–232.

Kunz, N. (2019). *smogn*. [Online; version 0.1.2]. URL: https://git.io/JOWoK.

Nair, V., & Hinton, G. E. (2010). "Rectified linear units improve restricted boltzmann machines". In: *ICML 2010*, pp. 807–814.

Odland, T. (2019). *KDEpy*. [Online; version 1.0.10]. URL: https://git.io/JOWrM.

Prechelt, L. (1998). "Early stopping-but when?" In: *Neural Networks: Tricks of the trade*. Springer, pp. 55–69.

Ribeiro, R. P. (2011). "Utility-based Regression". PhD thesis. University of Porto.

Ribeiro, R. P., & Moniz, N. (2020). "Imbalanced regression and extreme value prediction". In: *Machine Learning* 109.9, pp. 1803–1835.

Silverman, B. W. (1986). *Density Estimation for Statistics and Data Analysis*. Vol. 26. CRC Press.

Sun, Y., Wong, A. K., & Kamel, M. S. (2009). "Classification of imbalanced data: A review". In: *IJPRAI* 23.04, pp. 687–719.

Torgo, L., Ribeiro, R. P., Pfahringer, B., & Branco, P. (2013). "Smote for regression". In: *Portuguese conference on artificial intelligence*. Springer, pp. 378–389.

U.S. Geological Survey (1996). *GTOPO30*. URL: https://doi.org/10.5066/F7DF6PQS.

Vandal, T., Kodra, E., Ganguly, S., Michaelis, A., Nemani, R., & Ganguly, A. R. (2017). "Deepsd: Generating high resolution climate change projections through single image super-resolution". In: *KDD 2017*, pp. 1663–1672.

Wang, Y.-X., Ramanan, D., & Hebert, M. (2017). "Learning to model the tail". In: *NIPS 2017*, pp. 7029–7039.

Wilcoxon, F. (1945). "Individual Comparisons by Ranking Methods". In: *Biometrics Bulletin* 1.6, pp. 80–83. ISSN: 00994987. URL: http://www.jstor.org/stable/3001968.

Zhao, H., Sinha, A. P., & Bansal, G. (2011). "An extended tuning method for cost-sensitive regression and forecasting". In: *Decision Support Systems* 51.3.

A Appendix

In the Appendix we present some additional details and results of our work.

Figure 12 visualizes detailed results for datasets *dnormal* and *rpareto* of our experiment with synthetic data. Table 1 lists the datasets used in our comparison to the state-of-the-art with their respective sizes. Figure 13 visualizes the relevance functions for SMOGN on the synthetic datasets. Figure 14 shows the number of datasets won per method in our comparison with the state-of-the-art for additional network architectures. Figure 15 depicts the highly skewed precipitation distribution of the PRISM dataset used in our experiment for statistical downscaling of precipitation.

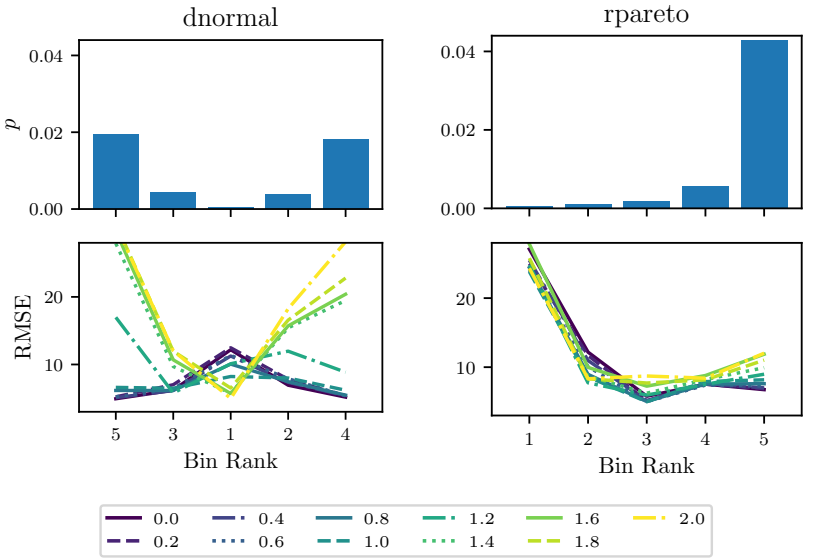
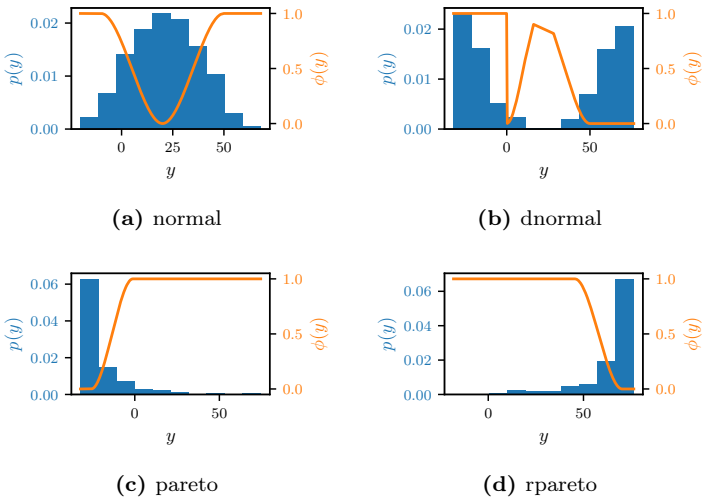


Fig. 12 Mean RMSE per test bin over 20 runs for datasets *dnormal* on the left and *rpareto* on the right. The top charts show the density per bin in the respective test dataset, visualizing the target variable's distribution. The line plots below visualize the mean RMSE per test bin for the α values shown in the box at the figure's bottom

Table 1 Datasets with imbalanced target values and their sizes

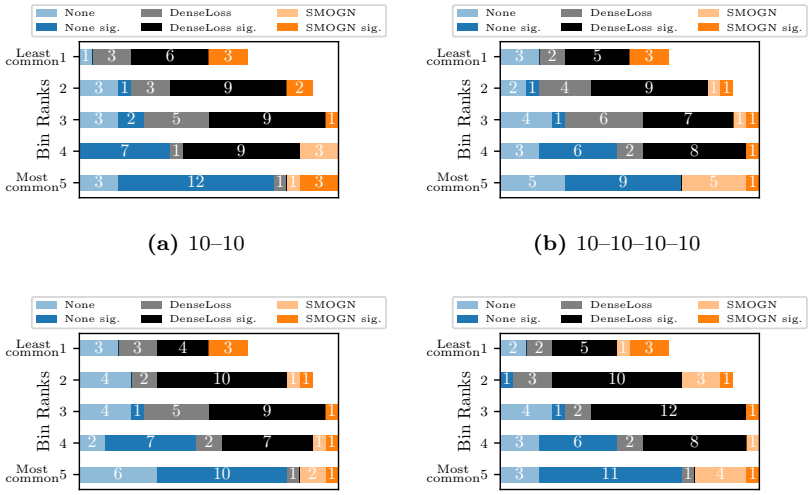
Dataset	N	Dataset	N	Dataset	N	Dataset	N
a1	198	a6	198	availPwr	1802	dAiler	7129
a2	198	a7	198	bank8FM	4499	fuelCons	1764
a3	198	Abalone	4177	boston	506	machineCpu	209
a4	198	acceleration	1732	ConcrStr	1030	maxTorq	1802
a5	198	airfoild	1503	cpuSm	8192	servo	167



(a) normal **(b)** dnormal

(c) pareto **(d)** rpareto

Fig. 13 SMOGN's relevance function ϕ for the synthetic datasets



(a) 10–10 **(b)** 10–10–10–10

(c) 20–20–20 **(d)** 5–5–5

Fig. 14 Number of datasets won per method and bin based on RMSE with different MLP architectures. Subcaptions indicate the number of hidden layers and neurons per hidden layer. E.g., 10–10 represents an MLP with two hidden layers each having 10. Bins are ranked in each test dataset according to sample size. Bins with rank 1 (5) contain the fewest (most) samples. Bar section's show the number of datasets won by a method at that bin rank. Wins denoted as "sig." are significant with regards to both other methods. 5 test datasets had an empty bin and 2 test datasets had 2 empty bins. Thus, the bars for bin rank 1 and 2 are smaller as there is no winner for empty bins

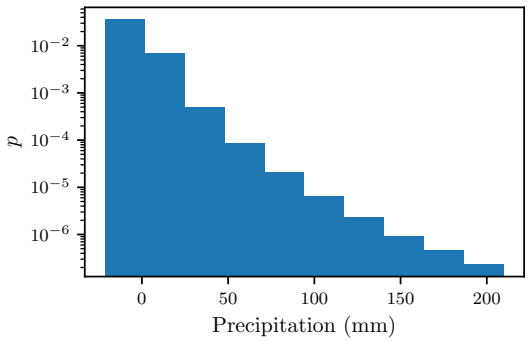


Fig. 15 Distribution of precipitation in the PRISM dataset over all cells and all days from 1981 to 2005. Note that the y-axis is logarithmic. Negative precipitation values may stem from an interpolation method used in the original work, but we decided not to clean the data to stay consistent with previous work

A.1 SMOGN with DenseWeight

The results in this work show that DenseLoss typically outperforms SMOGN. However, it is not clear to which extent the performance differences stem from the different measures of data point rarity or from the methodological differences between resampling and cost-sensitive learning. We therefore adapt SMOGN to use DenseWeight as its relevance function and we repeat the experiments involving SMOGN. We call SMOGN with DenseWeight *SMOGN-DW* in the following.

A.1.1 Experimental Setup

SMOGN identifies the rarity of each data point through a relevance function $\phi : Y \mapsto [0, 1]$ which is obtained through an automatic method based on box plot statistics by SMOGN's authors (Ribeiro 2011; Branco et al. 2017). This relevance function is similar to DenseWeight in that both aim to measure the rarity of a data point. In order to use DenseWeight as a relevance function we normalize the weights of all training set data points to a range between 0 and 1. We set DenseWeight's α to 1 which is the same value used for DenseLoss in the comparisons with SMOGN. We implement SMOGN with DenseWeight by expanding the existing python implementation *smogn* (Kunz 2019). This expanded *smogn* package is also available in our online repository. All other aspects of the experimental setup remain as described in Section 4.

A.1.2 Results

Figure 16 shows the normalized MAE depending on test data point rarity over the synthetic datasets (as in Figure 7) now also with SMOGN-DW. We see that SMOGN and SMOGN-DW perform very similarly on these synthetic datasets and that DenseLoss still tends to provide better performance for more rare data points. For the continuous results over the twenty datasets from Section 4.2 we see in Figure 17 that SMOGN and SMOGN-DW also show mostly similar performance, with lower normalized MAE for data points with $\sim 0.7 > p'(y) > 0.4$ for the latter. DenseLoss still seems to provide better performance than SMOGN-DW and SMOGN.

Figure 18 shows the number of dataset wins (as in Figure 8) but now with SMOGN-DW instead of regular SMOGN and for all evaluated MLP architectures. DenseLoss still has the highest number of significant dataset wins against both methods and almost always wins more

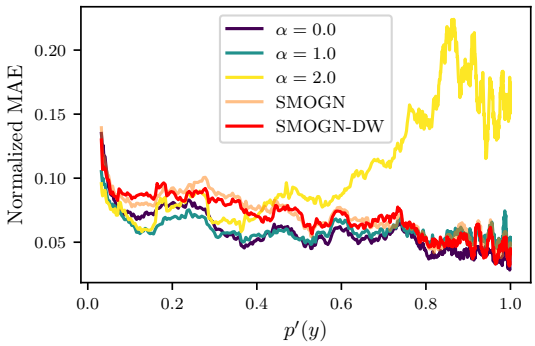


Fig. 16 Normalized MAE for test samples from all synthetic datasets per normalized density. Graph is smoothed via moving mean (window size 30) to ease interpretability

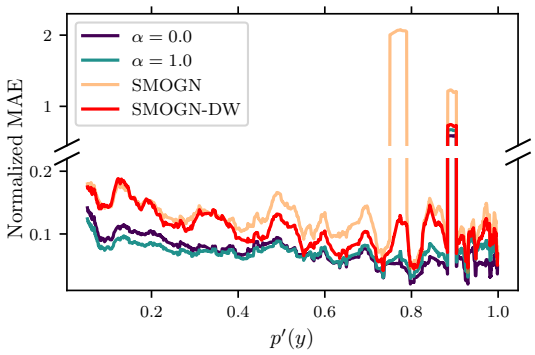
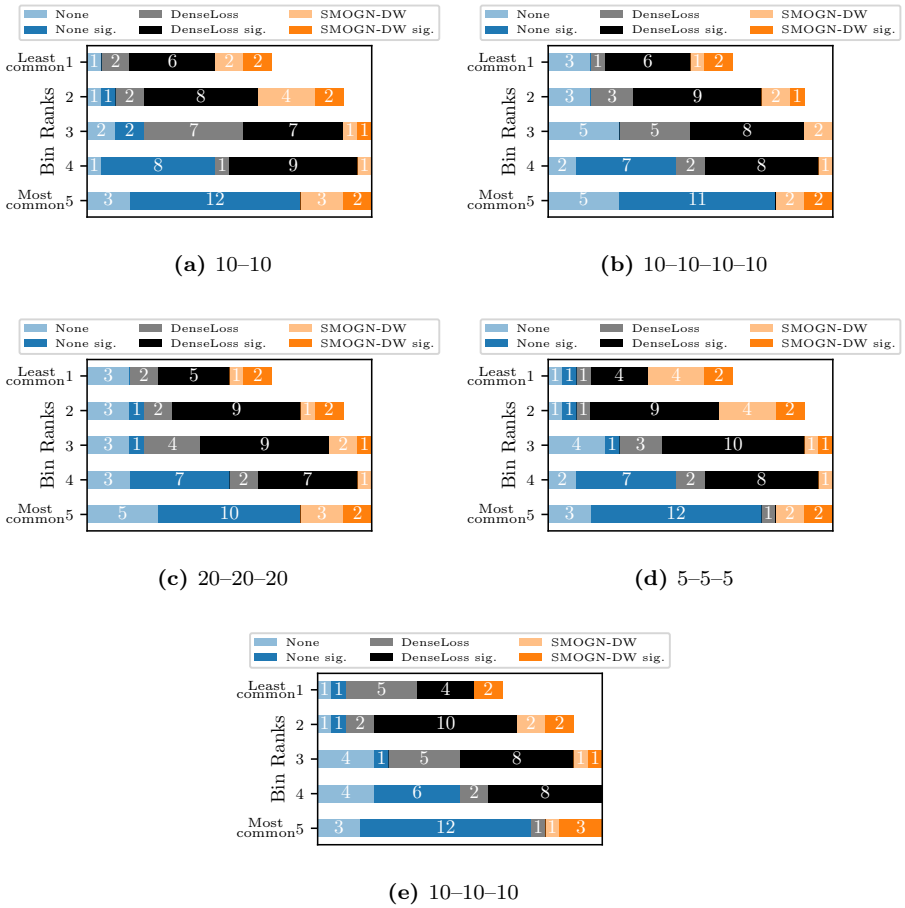


Fig. 17 Normalized MAE for test samples from all 20 datasets per normalized density. Graph is smoothed via moving mean (window size 300) to ease interpretability

than half of the datasets for bin ranks 1 to 4. Only bin rank 1 with architecture 5–5–5 shows one more SMOGN-DW win than the DenseLoss wins but even there DenseLoss has more significant wins. When comparing these results with the dataset wins of regular SMOGN in Figures 8 and 14 we see that the performance difference between SMOGN and SMOGN-DW is rather small with SMOGN-DW occasionally competing slightly better.

Since using the same measure of rarity for both SMOGN and DenseLoss does not improve SMOGN's performance considerably, we can conclude that most of the performance difference seems to stem from the methodological differences between resampling and cost-sensitive learning. Using DenseWeight as a relevance function for SMOGN seems to provide slight improvements compared to the relevance function used by SMOGN's authors but not enough to close the gap to DenseLoss.



(a) 10–10 **(b)** 10–10–10–10

(c) 20–20–20 **(d)** 5–5–5

(e) 10–10–10

Fig. 18 Number of datasets won per method and bin based on RMSE with different MLP architectures. Subcaptions indicate the number of hidden layers and neurons per hidden layer. E.g., 10–10 represents an MLP with two hidden layers each having 10. Bins are ranked in each test dataset according to sample size. Bins with rank 1 (5) contain the fewest (most) samples. Bar section's show the number of datasets won by a method at that bin rank. Wins denoted as "sig." are significant with regards to both other methods. 5 test datasets had an empty bin and 2 test datasets had 2 empty bins. Thus, the bars for bin rank 1 and 2 are smaller as there is no winner for empty bins