Received: January 17, 2023. Revised: May 10, 2023. 150

International Journal of
Intelligent Engineering & Systems

http://www.inass.org/

A Cost-Sensitive Approach applied on Shallow and Deep Neural Networks for Classification of Imbalanced Data

A Lamyaa Sadouk[1]* Taoufiq Gadi[1], El Hassan Essoufi[1] Mohamed Elhassan Bassir[2]

[1]*Faculty of Science and Technology Settat, University Hassan I, Morocco.*
[2]*National Schools of Applied Sciences Berrechid, University Hassan I, Morocco.*
* Corresponding author's Email: lamyaa.sadouk@gmail.com

Abstract: In this study, we propose a cost-sensitive learning approach applied on neural networks to deal with classification under imbalanced domains. Our approach is able to automatically learn robust features for both frequent and rare classes by automatically assigning misclassification penalties to each class based the frequency of occurrence of that class. This approach is investigated in the context of shallow networks (multi-layer perceptrons) and deep networks (convolutional neural networks). Moreover, it offers not only a better convergence but also a faster convergence since it can boost optimization by increasing weight gradients which are getting small due to their fitting to the frequent classes. Extensive experiments were carried out on one- and two-dimensional datasets. Running experiments using several loss functions showed the efficacity of our approach on loss functions which do not have probability estimates. Additionally, our approach achieved a good performance compared to common undersampling and oversampling methods as well as models based on generative adversarial networks.

Keywords: Cost-sensitive learning, Data imbalance, Loss functions, Multilayer perceptrons, Convolutional neural networks.

1. Introduction

Recently, growing advances in science and technology have resulted in the availability of data and its expansion in several fields. The presence of this massive volume of data contributes to the development of data mining tasks, but also brings in some challenges such as data imbalance. Even though the amount of annotated data has increased, these data have imbalanced distributions, which means that frequent classes (also denoted as negative or majority classes) are abundant while others (referred to as positive or minority classes) only have limited representations.

And, in real-world scenarios, several imbalanced classification tasks focus on correctly classifying the minority class. One example is the medical diagnosis of a particular disease (such as tumor diagnosis) where this latter is dangerous and should still be detected even though it occurs rarely. Given such imbalanced data at training, standard algorithms produce trained models that tend to learn more from instances belonging to the more observed or over-represented class than from those belonging to the less observed or under-represented class since they are unable to learn proper decision rules, resulting in imbalanced and erroneous performance on test data. In other words, imbalanced data causes trained classification models to be biased towards frequent classes. However, to ensure an efficient learning from existent models, it is important to obtain a good accuracy within both frequent and rare class instances.

In this work, we attempt to solve the issue of bad classification within imbalanced datasets by proposing a cost-sensitive learning strategy which improves the sensitivity of multi-class neural networks toward minority classes. The key contributions of this study are as follows.

1) An approach to train shallow and deep neural networks (such as multi-layer perceptrons and convolutional neural networks) with imbalanced

datasets, thus handling classification of 1-, 2-, and 3-dimentional input datasets.
2) A boosting technique in optimization which makes a faster NN learning and thus a faster convergence.
3) A method which works with all loss functions which do not have a probability estimate.
4) An empirical evaluation of the proposed approach against the state of the art.

The remainder of this paper is organized as follows. We briefly discuss the related work in the next section. Afterwards, we introduce and describe our proposed cost-sensitive learning algorithm (Section 3.1), analyze which loss functions are the most efficient when applied to the cost-sensitive strategy (Section 3.2), and discuss the convergence speed once the cost-sensitive technique is applied (Section 3.3). Implementation details including the description of datasets as well as NN hyper-parameters are provided in Section 4. Then, experiments and results are summarized in Section 5. Finally, a conclusion is given in Section 6.

2. Related work

There is a large literature on solving the imbalanced data problem. Existing approaches can be classified into algorithm-level and data-level approaches.

Data-level methods. The data-level techniques, also called "re-sampling techniques", do not affect the learning algorithm itself but rather modify the data distribution to solve the imbalanced classification problem. These techniques can group into four categories. Over-sampling is one of them and its methods consist of increasing the quantity of minority class instances by replicating them [1-5], or synthesizing new ones using the SMOTE technique [1] or one of its variants (Borderline SMOTE [2], Safe-level SMOTE [3], Local neighborhood SMOTE [4], ADASYN [5], etc.) to balance the data distribution. Nonetheless, these methods are criticized for the over-fitting problem [6] and for their inefficiency when dealing with high-dimensional data.

Another direction to balance the data distribution is the under-sampling category. Standard under-sampling consists of randomly under-sampling the majority class instances to construct the balanced positives and negatives. More advanced under-sampling methods include one-sided selection (OSS) [7] and nearest neighbor cleaning rule [8]. However, such methods are only suitable for training low dimensional, highly structured and size limited data (which is often in vector form) and do not behave satisfactory for high-dimensional, unstructured and large datasets.

A novel attempt to balance data is through dynamic sampling. For instance, the work of [9] proposes to train a deep learning model (convolutional neural network) using dynamic sampling, whereby the performance metric on the reference dataset is utilized to adjust the class distribution of training samples of the next iteration. In other words, more samples of classes with low F1-scores will be selected in the next iteration.

Algorithm-level methods. The goal of such methods is to directly modify the learning procedure to improve the sensitivity of the classifier towards minority classes. To this matter, several learning algorithms/classifiers have been proposed such as cost-sensitive SVM learning [10], neuro-fuzzy modeling [11], an extreme learning machine (ELM) with a weighting based on Adaboost [12], and an ensemble of soft-margin SVMs formed using boosting [13].

In the scope of shallow neural networks also known as multi-layer perceptions (MLP), several cost-sensitive approaches applied to imbalanced problems have been proposed in [14, 16-20]. Such approaches are similar with respect to the loss function which is the L_2 loss (Euclidean loss), with respect to the cost function formulation which is based on the dissociation of the class objectives, and also with respect to the learning rule used namely the extension of the backpropagation algorithm [15]. The peculiarities of these methods are in the strategies used to incorporate costs into the classes. In the study of [14], authors proposed to weight the L_2 loss function of a RBF neural network with the parameter $\gamma_k = \frac{max(N_j)}{N_k}$, where N_k is the number of examples of the kth class, and $max(N_j)$ is the number of examples of the majority class. Authors also argued that the values of γ_k should be gradually diminished along the training. Following the same direction as [14], the work of [16] consists of weighting the same L_2 loss function for a NN defining but this time with $\gamma_k = \frac{1}{N_k}$. Another study [17] which assumes a two-class MLPs, also proposes to intensify the error signal resulting from the target output neuron of the minority class by setting the parameters that control the order of the modified cost function unequal and equal to 2 and 4 for the dominant and rare classes respectively. Another attempt to deal with binary imbalanced classification was made by [18] which incorporates into the loss function parameters $\gamma_1 =$

$\frac{N_2}{N_1+N_2}$ and $\gamma_2 = \frac{N_1}{N_1+N_2}$ to weight positive and negative classes respectively. Meanwhile, the study of [19] proposes an improved Metropolis Hasting (IMH) algorithm to sample absent minority class samples by collecting samples rejected by the majority class approximation process. Those sampled absent minority samples are then provided to neural networks to address the data imbalance problem. In the work of [20], authors combine the Bayesian formulation with threshold-based regression methods, where the decision is based on thresholds over the output of a neural network.

In the scope of deep neural networks, recently, several studies integrated existing class imbalance solutions into deep learning models. The study [21] introduces a cost-sensitive convolutional neural network (CNN) whose class-dependant cost matrix $\mathcal{E}$ (or weight) is optimized along with neural network parameters during the training phase. With $\mathcal{E}_{p,q}$ denoting the misclassification cost of classifying an instance belonging to a class p into a different class q, the goal is to turn $\mathcal{E}_{p,q} = 0$ as $p = q$ during the backpropagation process. On the other hand, a novel approach based on an learned embedding with CNN is addressed by [22] whereby a CNN is trained with instances selected through a new quintuplet sampling scheme and the associated triple-header hinge loss. Another attempt to solve class imbalance with CNNs is made by [23] using a weighted softmax loss function where $\gamma_k = 1 + \frac{max(N_j)-N_k}{\beta * max(N_j)}$, with β a parameter that controls the scaling of the weighted loss (β was chosen to be 20). The study of [24] is an extension of [16] for CNNs for binary classification where authors first propose $\gamma_k = \frac{1}{N_k}$ then propose $\gamma_k = \frac{FE_k}{N_k}$ where $FE_k = \sum_{i=1}^{N_k} \sum_{m=1}^{M} \left(\hat{y}_m^{(i)} - y_m^{(i)}\right)^2$, M being the number of classes.

Hybrid methods. Another way to address class imbalance is to adopt a hybrid (algorithm and data) approach. In [25], a novel oversampling method called Ripple-SMOTE is combined with the weighted loss function suggested in [23]. Also, frameworks based on Generative Adversarial Networks (GAN) were introduced to fuse between data generation of minority class instances and classifier training and reduce misclassifications for minority class instances, such as conditional GAN (cGan) [26], balancing GAN (BAGAN) [27], Generative adversarial minority oversampling (GAMO) [28], oversampling near the borderline GAN (OBGAN) [29].

To handle the class imbalance problem, we introduce a cost-sensitive approach which has the following properties:

- Our approach is able to classify high dimensional data, which is a common issue faced by data-level methods.
- Unlike data-level methods and hybrid methods based on Generative Adversarial Network architectures, our approach does not alter the original data distribution, resulting in a lower computational cost during the training process.
- Our algorithm is rather an extension of cost-sensitive algorithms dealing with binary classification [14,16-18]. It handles multi-class classification tasks by providing an asymmetrical learning of NN via a modified (backpropagation) weight update rule.

3. Methodology

In this section, we introduce the proposed cost-sensitive loss function for classification under imbalanced domains.

3.1 Cost-sensitive learning

Given a highly imbalanced training dataset $\{(\boldsymbol{x}_s, y_s)\}_{s=1}^{S}$ where S is the total number of instances (e.g., samples) in the training set, a neural network is trained by a function $\boldsymbol{\omega}(.)$, under the minimization of our cost-sensitive loss function using backpropagation [15] and stochastic gradient descent. Training this network generates tuned parameters $\boldsymbol{\theta}$ which define a mapping between the input vector $\boldsymbol{x}_s$ and the output y_s, represented by:

$$\hat{y}_s = \omega(\boldsymbol{x}_s, \boldsymbol{\theta}) \tag{1}$$

where $\hat{y}_s$ is the estimated output value.

The training process of a neural network is accomplished through the minimization of an objective function that measures the error between the target class (ground-truth value) and the estimated value (prediction) as follows,

$$\underset{\boldsymbol{\theta}}{\text{minimize}}\, E(\boldsymbol{\theta}) \tag{2}$$

where

$$E(\boldsymbol{\theta}) = \frac{1}{N}\sum_{\boldsymbol{i=1}}^{\boldsymbol{N}} E^{(i)}(\boldsymbol{\theta}) = \sum_{i=1}^{N} l(\mathbf{y}^{(i)}, \hat{\mathbf{y}}^{(i)}) \tag{3}$$

where N is the number of instances per batch, $l(\cdot)$ is the loss function, $\boldsymbol{y}^{(i)}$ is the true label as one-hot encoding and $\hat{\boldsymbol{y}}^{(i)}$ is the predicted output vector e.g.,

the output of the last layer of the network, for an instance i within the batch.

Empirical studies performed with the backpropagation algorithm show that the imbalance problem is due to the contribution to loss function, from the positive classes (e.g., minority classes) in relation to negative classes (e.g., majority classes), where the most contribution to the loss function is produced by negative classes. Therefore, the training process is dominated by these latter.

In order to achieve solutions that are sensitive to the importance of each class e.g., that give equal importance to each class, we propose a cost-sensitive cost function for the parameter estimation of neural networks. Given M classes and N training samples $N = \sum_{m=1}^{M} n_m$ with n_m the number of samples of the m^{th} class, the expression of this cost function is defined as the weighted sum of functionals $E^{(i)}(\boldsymbol{\theta})$ given by,

$$\left(E(\boldsymbol{\theta})\right)^{cost-sens.} = \frac{1}{N}\sum_{m=1}^{M} \lambda(m) \sum_{i=1}^{n_m} l(\mathbf{y}^{(i)}, \hat{\mathbf{y}}^{(i)}) \tag{4}$$

where $\lambda(m)$ is the weighting parameter corresponding to the class m.

The question is how to find the proper parameter $\lambda(m)$ that maximizes classification performance given imbalanced datasets. The solution proposed by [14] was to set $\lambda(m) = \frac{max(n_k)}{n_m}$, where $k = 1, \dots, M$ and $max(n_k)$ is the number of examples of the most frequent class. However, with such a method, $1 \leq \lambda \leq \frac{max(n_k)}{min(n_k)}$ implying that λ does not have a fixed bound and can get very high as $n_m \ll max(n_k)$ (e.g., as the number of instances of the most frequent class is much higher than the one of the rarest class). Thus, this method can yield to an aggressive weighting of the loss function.

Another solution is to come up with a fixed bounded λ. As such, we first define the relevance of a class m as,

$$\emptyset(m) = 1 - \frac{n_m}{max(n_k)} \tag{5}$$

where the relevance $\emptyset(m)$ is a probability ranging from 0 to 1. $\emptyset(m)$ increases as the class m is more positive (more infrequent) and decreases as the class m is more negative (more frequent). For instance, $\emptyset(m) \approx 1$ for $n_m \ll max(n_k)$ and $\emptyset(m) = 0$ for $n_m = max(n_k)$.

Afterwards, the parameter $\lambda(m)$ is obtained,

$$\lambda(m) = 1 + \tau\, \emptyset(m) \tag{6}$$

where τ is the parameter that regulates the weighting of the relevance $\emptyset(m)$, defined as $\tau \geq 1$. Accordingly, the parameter $\lambda(m)$ ranges from 1 to $1 + \tau$. For example, given $\tau = 1$ and an instance i whose target is the most frequent class, then $\lambda(m) = 1$; so no extra weight is attributed to the functional or error $E^{(i)}(\boldsymbol{\theta})$. On the other hand, with an instance i whose target is the most rare class, $\lambda(m) = 2$; so the $E^{(i)}(\boldsymbol{\theta})$ is multiplied by 2.

In order to show the impact of the cost-sensitive approach on the neural network (NN) learning process, let's analyze the cost-sensitive gradient of the loss function. Given the simple functional Eq. (3), let's compute its scalar gradient calculated for a given weight $\theta_{p,k}$ of the network (k being one of the M output nodes located at the last NN layer l and p being one of the input nodes at layer $l-1$), using the standard backpropagation algorithm [15]. The functional gradient with respect to $\theta_{p,k}$ can be written as,

$$\nabla E\left(\theta_{p,k}\right) = \frac{1}{N}\sum_{i=1}^{N} \frac{\partial l(y_k^{(i)}, \hat{y}_k^{(i)})}{\partial \hat{y}_k^{(i)}}\, x_p^{(i)} \tag{7}$$

where $y_k^{(i)}$ and $\hat{y}_k^{(i)}$ are the true label and the predicted NN output respectively at node k for instance i, $\frac{\partial l(y_k^{(i)}, \hat{y}_k^{(i)})}{\partial \hat{y}_k^{(i)}}$ is the partial derivative of the loss with respect to $\hat{y}_k^{(i)}$, and $x_p^{(i)}$ is the value of the input node p at the same instance. Let's note that, in our case, no activation function is present between the sum of weighted input nodes (e.g. the linear combination of inputs) and the loss function. We will further show in section 3.2 that omitting this activation function is responsible for improving the efficiency of our cost-sensitive approach.

On the other hand, given the cost-sensitive functional Eq. (4), its gradient with respect to $\theta_{p,k}$ is given by,

$$\left(\nabla E(\theta_{p,k})\right)^{cost-sens.} = \frac{1}{N}\sum_{i=1}^{N} \lambda^{(i)} \frac{\partial l(y_k^{(i)}, \hat{y}_k^{(i)})}{\partial \hat{y}_k^{(i)}}\, x_p^{(i)} \tag{8}$$

where $\lambda^{(i)} = \lambda\left(m^{(i)}\right)$ for simplicity, with m^i being the target class (label) of instance i, and $\frac{\partial l(y_k^{(i)}, \hat{y}_k^{(i)})}{\partial \hat{y}_k^{(i)}}$ is the standard (pure) partial derivative of that loss function.

Then, using Eq. (6) and Eq. (7), Eq. (8) becomes,

$$\left(\nabla E\left(\theta_{\boldsymbol{p,k}}\right)\right)^{cost-sens.} = \nabla E\left(\theta_{\boldsymbol{p,k}}\right) + \frac{1}{N}\sum_{i=1}^{N} \tau\, \emptyset^{(i)} \frac{\partial l(y_k^{(i)}, \hat{y}_k^{(i)})}{\partial \hat{y}_k^{(i)}}\, x_p^{(i)} \quad (9)$$

where $\emptyset^{(i)} = \emptyset\left(m^{(i)}\right)$, suggesting that $\left(\nabla E(\theta_{\boldsymbol{p,k}})\right)^{cost-sens.}$ is simply $\nabla E(\theta_{p,k})$ (first term of Eq. (9)) plus an extra cost (second term of Eq. (9)) whose value depends on the relevance of instances within the batch. We notice that this extra cost gets higher as the relevance of the instances $i \in \{1, \dots, N\}$ is higher e.g., as the classes of instances are less frequent . And conversely, this extra cost goes to 0 as the classes of instances are more frequent. Consequently, $\left(\nabla E(\theta_{p,k})\right)^{cost-sens.} \gg \nabla E(\theta_{p,k})$ for less frequent instances and $\left(\nabla E(\theta_{p,k})\right)^{cost-sens.} \approx \nabla E(\theta_{p,k})$ for more frequent instances. Hence, more learning occurs at rare instances (positives).

Moreover, along the training process, the impact of the extra cost of the gradient diminishes since $\frac{\partial l(y_k^{(i)}, \hat{y}_k^{(i)})}{\partial \hat{y}_k^{(i)}}$ gets smaller over epochs.

3.2 Efficiency of the cost-sensitive learning based on the loss function used

Now that the cost-sensitive cost function is defined, it is necessary to define the proper loss function for our cost sensitive approach. To do, we lay out a background on different loss functions for classification, then we discuss which ones are the most suitable to our approach.

3.2.1. Background on loss functions for classification

Several loss functions for multiclass classification exist. The most famous ones are listed in Table 1. Knowing that the true label $\mathbf{y}^{(i)}$ of instance i is a one-hot encoding vector, then $y_k^{(i)}$ the true label at a certain node k can be either 0 or 1. Accordingly, graphical representations of loss functions within Table 1 are displayed in Figure.1.a for $y_k^{(i)} = 1$ and in Figure.1.b for $y_k^{(i)} = 0$. From these figures we note that: (i) $hinge$, $hinge_2$, and $hinge_3$ are variants of $mshinge$, $mshinge_2$, and $mshinge_3$ respectively where $\max_{q \neq t} \hat{y}_q^{(i)} = 0$, (ii) in order to visualize the cross-entropy loss, the activation function $\sigma(\cdot)$ is chosen to be the sigmoid function instead of the softmax function.

In our study, we call "probability estimate loss functions" loss functions which are applied to probability estimates $\sigma(\cdot)$ such as softmax or sigmoid functions. In other words, these loss functions are connected to final layer activations (output neurons) which are based on probability estimates. In Table 1, examples of such loss functions are $log \circ \sigma$ and $L_2 \circ \sigma$, while the rest of loss functions are applied to final layer activations based on a linear output.

Table 1. List of loss functions used in our analysis. $\mathbf{y}^{(i)}$ and $\hat{\mathbf{y}}^{(i)}$ represent respectively the true label and the predicted output at the last network layer at instance i. $\cdot_m$ denotes the mth dimension/element of a given vector and $\sigma(\cdot)$ denotes a probability estimate such as the softmax function or the sigmoid function. Full names of proposed loss functions are given in the Appendix.

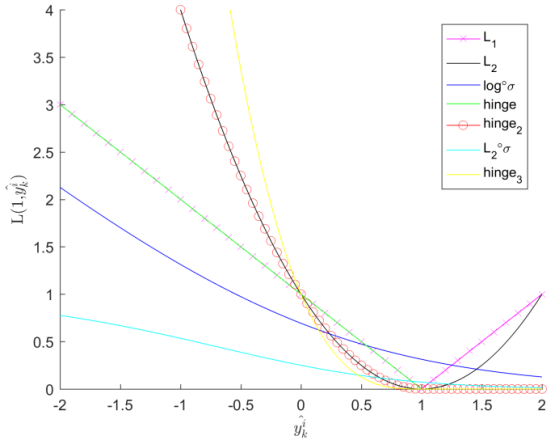
Loss function	Equation $\boldsymbol{l}(\mathbf{y}^i, \hat{\mathbf{y}}^i)$
$\boldsymbol{L_2}$	$\left\|\mathrm{y}^{(i)} - \hat{\mathrm{y}}^{(i)}\right\|_2^2$
$\boldsymbol{L_2 \circ \sigma}$ such that $\boldsymbol{\sigma(\cdot) = sigmoid}$	$\left\|\mathrm{y}^{(i)} - \mathrm{p}^{(i)}\right\|_2^2$ where $\mathrm{p}^{(i)} = \sigma(\hat{\mathrm{y}}^{(i)})$
$\boldsymbol{Mshinge}$	$max\left\{0, 1 + \max_{q \neq t} \hat{\mathrm{y}}_q^{(i)} - \hat{\mathrm{y}}_t^{(i)}\right\}$, $\mathrm{y}_t^{(i)} = 1$
$\boldsymbol{Mshinge_2}$	$max\left\{0, -\left(1 + \max_{q \neq t} \hat{\mathrm{y}}_q^{(i)} - \hat{\mathrm{y}}_t^{(i)}\right)^2\right\}, \mathrm{y}_t^{(i)} = 1$
$\boldsymbol{Mshinge_3}$	$max\{0, \left(1 + \max_{q \neq t} \hat{\mathrm{y}}_q^{(i)} - \hat{\mathrm{y}}_t^{(i)}\right)^3\}$, $\mathrm{y}_t^{(i)} = 1$
$\boldsymbol{log \circ \sigma}$ such that $\boldsymbol{\sigma(\cdot) = softmax}$	$-\sum_m^M \mathrm{y}_m^{(i)} \log\left(p_m^{(i)}\right)$ where $p_m^{(i)} = \sigma(\hat{\mathrm{y}}_m^{(i)}) = \frac{e^{\hat{y}_m^{(i)}}}{\sum_{j=1}^M e^{\hat{y}_j^{(i)}}}$ and M denotes the total number of neurons in the final output layer (e.g., M equals the number of classes)

3.2.2. Cost-sensitive learning applied on loss functions

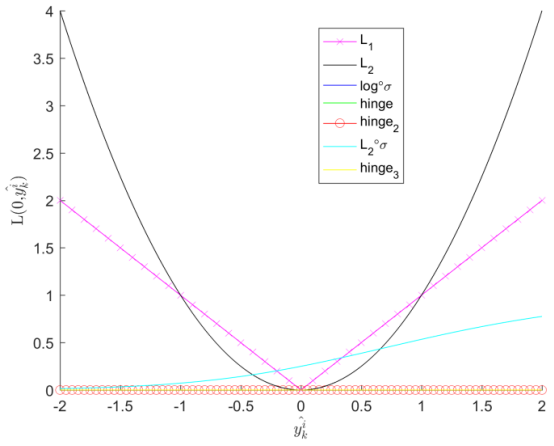
In order to see the impact of the cost-sensitive approach on loss functions, we choose to plot in Fig. 2 (a) and (b) respectively the standard and cost-sensitive partial derivatives of loss functions displayed in Fig. 1 with respect to output neuron.

For the cost-sensitive version of loss functions, we pick an instance i whose class $m^{(i)}$ is very rare such that $\emptyset(m^{(i)}) = 0.9$, and we set $\tau = 1$ to obtain $\lambda(m^{(i)}) = 1.9$. Fig. 2. (a) and Fig. 2 (b) depict respectively $\frac{\partial l(y_k^{(i)}, \hat{y}_k^{(i)})}{\partial \hat{y}_k^i}$ and $\left(\frac{\partial l(y_k^{(i)}, \hat{y}_k^{(i)})}{\partial \hat{y}_k^{(i)}}\right)^{cost-sensitive}$ at $y_k^{(i)} = 1$.

Received: January 17, 2023. Revised: May 10, 2023.

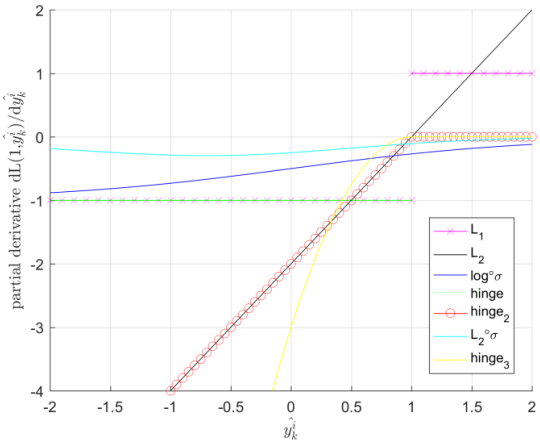


(a)

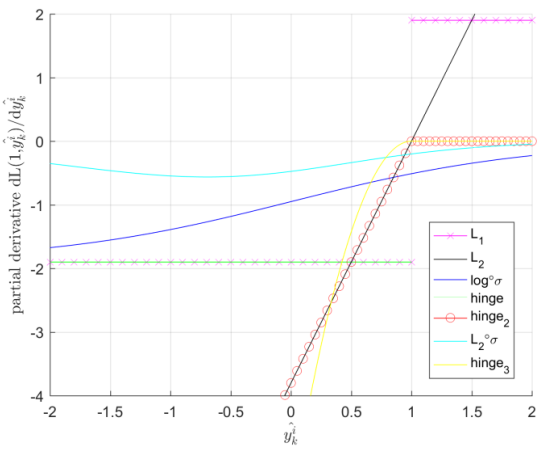


(b)

Figure. 1 Plots (a) and (b) represent different loss functions $l(\mathbf{y}^{(i)}, \hat{\mathbf{y}}^{(i)})$ at an instance i with respect to neuron predicted output $\hat{y}_k^{(i)}$ when $y_k^{(i)} = 1$ and $y_k^{(i)} = 0$ respectively. $\sigma(\cdot)$ denotes the sigmoid function



(a)



(b)

Figure. 2 Plots in (a) and (b) represent respectively standard and cost-sensitive partial derivatives of different losses at instance i with respect to $\hat{y}_k^{(i)}$. Instance i is chosen to have a high relevance $\emptyset(m^i) = 0.9$, and τ is set to 1

By analyzing plots of Fig. 2 (a) and Fig. 2 (b), we observe that, for a prediction $\hat{y}_k^{(i)} = 0$ far from the true label $y_k^{(i)} = 1$, $\frac{\partial l(y_k^{(i)}, \hat{y}_k^{(i)})}{\partial \hat{y}_k^{(i)}}$ is equal to -1, -2, -12 and -2 for $Mshinge$, $Mshinge_2$, $Mshinge_3$, and L_2 respectively, and to -0.5 and -0.25 for $log \circ \sigma$ and $L_2 \circ \sigma$ respectively. This suggests that probability estimate loss functions (e.g., $log \circ \sigma$ and $L_2 \circ \sigma$) produce absolute values of partial derivatives smaller than those of other loss functions.

Meanwhile, after applying the cost sensitive strategy on these loss functions, for the same prediction $\hat{y}_k^{(i)} = 0$, the value of $\left(\frac{\partial l(y_k^{(i)}, \hat{y}_k^{(i)})}{\partial \hat{y}_k^{(i)}}\right)^{cost-sensitive}$ is -1.9, -3.8, -22.8 and -3.8 for $Mshinge$, $Mshinge_2$, $Mshinge_3$, and L_2 respectively, compared to -0.95 and -0.475 for $log \circ \sigma$ and $L_2 \circ \sigma$ respectively. Therefore, $Mshinge$, $Mshinge_2$, $Mshinge_3$, and L_2 loss functions yield relatively high $\left|\left(\frac{\partial l(y_k^{(i)}, \hat{y}_k^{(i)})}{\partial \hat{y}_k^{(i)}}\right)^{cost-sensitive}\right|$, compared to probability estimate loss functions which have relatively low $\left|\left(\frac{\partial l(y_k^{(i)}, \hat{y}_k^{(i)})}{\partial \hat{y}_k^{(i)}}\right)^{cost-sensitive}\right|$.

In general, as the prediction gets further from the true label (such that $\hat{y}_k^{(i)} \ll y_k^{(i)}$ at $y_k^{(i)} = 1$), $\left|\frac{\partial l(y_k^{(i)}, \hat{y}_k^{(i)})}{\partial \hat{y}_k^{(i)}}\right|$ of probability estimate loss functions increases at a slow rate, as opposed to $\left|\frac{\partial l(y_k^{(i)}, \hat{y}_k^{(i)})}{\partial \hat{y}_k^{(i)}}\right|$ of $Mshinge_2$, $Mshinge_3$, and L_2 loss functions. As a result, multiplying partial derivatives by $\lambda(m^i)$ still

 DOI: 10.22266/ijies2023.0831.13

produces a small $\left|\left(\frac{\partial l(y_k^{(i)},\hat{y}_k^{(i)})}{\partial \hat{y}_k^{(i)}}\right)^{cost-sensitive}\right|$ for probability estimate loss functions and a relatively high $\left|\left(\frac{\partial l(y_k^{(i)},\hat{y}_k^{(i)})}{\partial \hat{y}_k^{(i)}}\right)^{cost-sensitive}\right|$ for other loss functions as shown in Fig. 2 (b). Hence, the impact of the cost-sensitive approach will be very sparse for the probability estimate loss functions and relatively high for $Mshinge$, $Mshinge_2$, $Mshinge_3$, and L_2.

3.3 Faster convergence / Boosting network learning

One of the factors responsible for the fast NN convergence is the use of an adaptable learning rate α e.g., an α not too high for the network to diverge and not too low for it to converge too slowly. Below we show that our cost-sensitive approach is able to improve the network convergence via a dynamic learning rate per instance $\alpha^{(i)}$ which depends on the rarity or relevance of that instance.

Using the cost-sensitive gradient Eq. (8), the Stochastic Gradient Descent learning rule is obtained as,

$$\theta_{p,k} := \theta_{p,k} - \frac{1}{n}\sum_{i=1}^{N} \alpha^{(i)}\ \frac{\partial l(\mathbf{y}^{(i)},\hat{\mathbf{y}}^{(i)})}{\partial \hat{y}_k^{(i)}}\ x_p^{(i)} \quad (10)$$

where $\alpha^{(i)} = \alpha\ \lambda^{(i)}$ and $\alpha \le \alpha^{(i)} \le (1+\tau)\alpha$.

So, we can consider α_i as a dynamic learning rate which changes with respect to the rarity or relevance of the instance, by increasing up to $(1+\tau)\alpha$ for the rarest instances and decreasing to the original learning rate α for the most frequent instances.

As such, our cost-sensitive gradient $\left(\nabla E(\theta_{p,k})\right)^{cost-sensitive}$ has the property of boosting learning as it helps the network converge faster. Indeed, our gradient has the property/particularity of increasing small weight updates, which is one of the 2 objectives of the Adam optimizer [30] (whole goal is to make relatively small weight update bigger and relatively high weights updates smaller).

In order to show this property, let's consider training a two-class neuron network (NN) under imbalanced domains using (stochastic gradient descent with) a batch of size 10. By applying the standard cost function Eq. (3), this NN tends to classify negative or frequent instances as negative and positives or rare instances as negatives. For example, given a run composed of 8 negative instances and 2 positives ones, the NN is more likely to correctly classify the 8 negatives as negatives and to wrongly classify the 2 positives as negatives, resulting in a relatively–small average gradient $\left(\nabla E(\theta_{p,k})\right)^{cost-sensitive}$ and thus a small learning. So, more backpropagation runs are needed in order to learn from positive instances. However, employing the cost-sensitive approach produces a higher average error (since the loss of every positive instance is increased), and a higher average gradient, which makes learning from positive instances greater and faster. So, the cost-sensitive loss boosts learning by making small gradients bigger.

4. Experimental study

4.1 Datasets

1D datasets. Experiments were carried out on eight real one-dimensional datasets (e.g., whose input data is a vector) extracted from the UCI Database Repository (http:www.ics.uci.edu/~mlearn). "PID" and "WPBC are used as abbreviations of "Pima Indians Diabetes" and "WP Breast Cancer" respectively. The datasets and some of their characteristics are summarized in Table 2. All these datasets have passed through the following pre-processing steps: categorical attributes were expanded into the corresponding binary vectors, and then each attribute (metric or binary) was normalized to the interval [0, 1]. Furthermore, the "yeast_8l" dataset is simply the "yeast" dataset with class 9 and 10 removed (since these latter contain very few instances, which makes the NN hard to train).

2D dataset: MNIST. MNIST is considered a simple dataset generally used for digits' images classification tasks. It consists of grayscale images of size 28×28 with ten classes corresponding to digits from 0 to 9. The number of instances per class in the original training dataset ranges from 5421 in class 5 (e.g., number "5") to 6742 in class 1 (e.g., number "1"). In our study, we subsample uniformly at random each class to obtain no more than 600 examples par class. This dataset is referred to as "Mnist".

Afterwards, in order to show the performance of our cost-sensitive algorithm, the "Mnist" dataset needs to be imbalanced. To do so, we defined the ratio between the number of examples in majority classes and the number of examples in minority classes as follows, $r = \frac{\max_k\{n_k\}}{\min_k\{n_k\}}$. As such, for r equal to 10, 30, 40 or 50, classes 1 and 3 both have 60, 30, 15 or 12 instances respectively. These datasets are denoted as "Mnist10", "Mnist30", "Mnist40" and "Mnist50" respectively.

Table 2. Characteristics of the 1D datasets.

Dataset	No. of Attributes	No. classes	No. of instances	Class distribution
Ionosphere	34	2	181	126/55
PID	8	2	768	268/500
WPBC	30	2	198	47/151
SPECTFHeart	43	2	267	55/212
Yeast_8l	8	8	1645	464/430/424/163/51/47/35/31
Car	6	4	1728	1210/384/69/65
Satimage	36	6	6435	1533/703/1358/626/707/1508
Thyroid	21	3	7200	166/368/6666
Mnist	28x28	10	6000	600 for all classes
Mnist10	28x28	10	4920	60 for class 1,3 600 for others
Mnist30	28x28	10	4860	20 for class 1,3 600 for others
Mnist40	28x28	10	4850	15 for class 1,3 600 for others
Mnist50	28x28	10	4824	12 for class 1,3 600 for others

4.2 Training and experimental setup

In our study, a multi-layer perceptron with three hidden is used to train the 1D datasets, whereas the 2D datasets are trained using a convolutional neural network (CNN) with two convolutional layers and two fully connected layers. The optimization algorithm used for experiments within Section 5.1 and 5.2 is stochastic gradient descent (SGD) with momentum value of 0.9 and a weight decay of 0.0005. As for experiments of section 5.3, SGD with the same momentum and weight decay is used for training CNNs, whereas the Adam optimizer [30] is used for training MLPs with the exponential decay rate for 1st and 2nd moment estimates β_1 and β_2 set to 0.9 and 0.999 respectively, the offset ε set to 10^{-8}, and a starting learning rate of 0.0001. Hyper-parameters such as the learning rate, batch size and network architecture vary from one dataset to another and are set according to Table 3. Note that learning rates displayed in Table 3 (a) are used when the optimizer is SGD with momentum. On the other hand, when dealing with the Adam optimizer, the learning rate is set to 0.001 for all datasets within Table 3 (a). The dropout rate for CNN is set to 0.5. Training is performed for 15 to 100 epochs, depending on the dataset used and the experimental method employed. We report results with 3-fold cross validation.

As for parameters of the cost-sensitive approach, τ is set to 2 and 50 for the MLP and CNN models respectively. Indeed, as the proposed CNN architecture has more hidden layers than the proposed MLP architectures, the magnitude of the gradients with each subsequent layer of the CNN gets exponentially smaller in the backpropagation process, which results in very slow learning of weights in the CNN lower layers. So, in order for CNN lower layers' weights to be affected by the cost-sensitive technique, the weighting parameter λ needs to be relatively high compared to λ of a MLP.

Table 3. Training hyper-parameters for the 1D datasets in: (a) and the 2D datasets in (b). For these datasets, the MLP used has $n1$, $n2$ and $n3$ neurons in the first, second and third layer respectively and (b) the architecture of the CNN used for the 2D datasets is defined

(a)

Dataset	Learning rate	Batch size	$[\boldsymbol{n1}, \boldsymbol{n2}, \boldsymbol{n3}]$
Ionosphere	0.010	5	500, 50, 2
PID	0.010	5	200, 20, 2
WPBC	0.005	5	500, 50, 2
Yeast_8l	0.001	10	200, 100, 8
Car	0.010	10	150, 75, 4
Satimage	0.010	50	600, 100, 6
Thyroid	0.010	50	350, 70, 3

(b)

Layer	Depth	Kernel size	Stride
Convolution	20	5x5	1
ReLU	20	-	-
Max-pooling	20	2x2	2
Convolution	50	5x5	1
ReLU	50	-	-
Max-pooling	50	2x2	2
Fully connected	500	4x4	1
ReLU	500	-	-
Dropout	500	-	-
Fully connected	10	1x1	1

As for the performance metric, the most widely one for evaluating performance in the context of multiclass classification within neural networks (MLPs or CNNs) is overall accuracy which is the proportion of test examples that were correctly classified. However, this metric has some significant and long acknowledged limitations, particularly in the context of imbalanced datasets. Specifically, when the test set is imbalanced, accuracy will favor classes that are overrepresented in some cases leading to highly misleading assessment. In order to make classification performance of each class equally represented in the evaluation measure, [31] suggested the G-mean as the geometric means of recall values for the bi-class scenario. Expanding this measure to the multiple class scenario was introduced by [32] whereby the G-mean is the geometric means of recall values of every classes as follows,

$$G - mean = (\prod_{i=1}^{m} \quad _i)^{1/m} \quad (11)$$

Where $_i$ is the recall value of class i and m is the number of classes.

As each recall value representing the classification performance of a specific class is equally accounted, G-mean is the proper metric for our study as it is able to measure the balanced performance among imbalanced classes.

5. Experiments and results

Training MLPs on the 1D datasets and CNNs on the 2D datasets is performed using the standard version of loss functions mentioned in Table 1 (Section 3) as well as the cost-sensitive version of these loss functions. These latter include L_2, $Mshinge$, $Mshinge_2$, $Mshinge_3$, $log \circ \sigma$ (where $\sigma(\cdot)$ corresponds to the softmax function), and $L_2 \circ \sigma$ ($\sigma(\cdot)$ being the sigmoid function). However, L_1 loss function was dismissed from this experiment since it does not learn at all due to "jumps" in the NN model caused by its partial derivatives with respect to the predicted network output being either -1 or 1 (as seen in Fig. 2 (a). Each experiment is repeated three times and its mean performance across all three runs is depicted in Table 4. Convergence rates are visualized through learning curves of each experiment (of 1 run only) which are displayed in Fig. 3.

5.1 Effect of cost-sensitive learning on classification performance

From results in Table 4, we observe that applying the cost-sensitive approach on L_2, $Mshinge$, $Mshinge_2$ and $Mshinge_3$ loss functions improves the G-mean performance in overall. Indeed G-mean results for the cost-sensitive version of these loss functions are higher than results for the standard version for all 1D and 2D datasets (except "WB breast cancer" dataset for $Mshinge_3$). For instance, the cost-sensitive approach boosts performance from 0% to 60.69% for the "thyroid" dataset when applied on $Mshinge$, from 0% to 60.66% for the "Yeast_8l" dataset when applied on $Mshinge_2$, and from 0% to 98.41% for the "Mnist50" dataset when applied on L_2. However, we can see that applying the same cost-sensitive approach on $\log \circ \sigma$ loss function seems to decrease performance for several datasets (such as "Ionosphere", "Yeast_8l", "Mnist30", "Mnist40" and "Mnist50") and to increase performance for the rest of the datasets. The same behavior is observed for $L_2 \circ \sigma$ loss function with a decrease in performance for datasets "Ionosphere", "PID", "WB breast cancer", and "Satimage", versus an increase in performance for the other datasets.

Given these observations, the following reflections can be made:

(i) When dealing with L_2, $Mshinge$ and $Mshinge_2$ and $Mshinge_3$ loss functions, our cost-sensitive approach is able to capture more relevant features e.g., features that are shared between positive and negative instances. Indeed, at each instance i, multiplying such loss functions by λ contributes in balancing weights θ between positives and negatives, thus generating better weights. Furthermore, the positive impact of this approach with such loss functions is drawn for both the shallow NN models (MLPs) and deep learning ones (CNNs). Thus, our cost-sensitive approach can be regarded as a reliable technique when applied on loss functions L_2, $Mshinge$, $Mshinge_2$, and $Mshinge_3$.

(ii) However, results convey that our cost-sensitive approach is not suitable for probability estimate loss functions ($\log \circ \sigma$ and $L_2 \circ \sigma$), which confirms the graphical interpretation of section 3.2. Indeed, as $\sigma(\cdot)$ turns predicted outputs into probabilities, partial derivatives of the loss function with respect to this output tend to be small (within the range [0,1]), making partial derivatives of the cost-sensitive loss function small with little impact on NN learning.

(iii) Also, let's note that the non-linearity present within partial derivatives of loss functions $\log \circ \sigma$ and $L_2 \circ \sigma$ is not the property responsible for the inefficiency of the cost-sensitive approach. Indeed, even with the non-linearity of $Mshinge_3$ partial derivative, $Mshinge_3$ along with the cost-sensitive method improves classification performance.

5.2 Effect of cost-sensitive learning on convergence speed

From plots in Fig. 3 (a), (b), (c), (d), (e) and (f) corresponding to learning curves of NNs for $Mshinge$, $Mshinge_2$, $Mshinge_3$, L_2, $L_2 \circ \sigma$ and $log \circ \sigma$ loss functions respectively in the standard and cost-sensitive form, we observe that the cost-sensitive strategy increases the convergence speed for $Mshinge$, $Mshinge_2$, $Mshinge_3$ and L_2 loss functions. As explained in the methodology (Section 3.3), this improvement over the NN convergence speed is due to the coefficient $\lambda(m^{(i)})$ within the gradient which acts like a learning rate magnifier for any positive instance i and boosts the learning

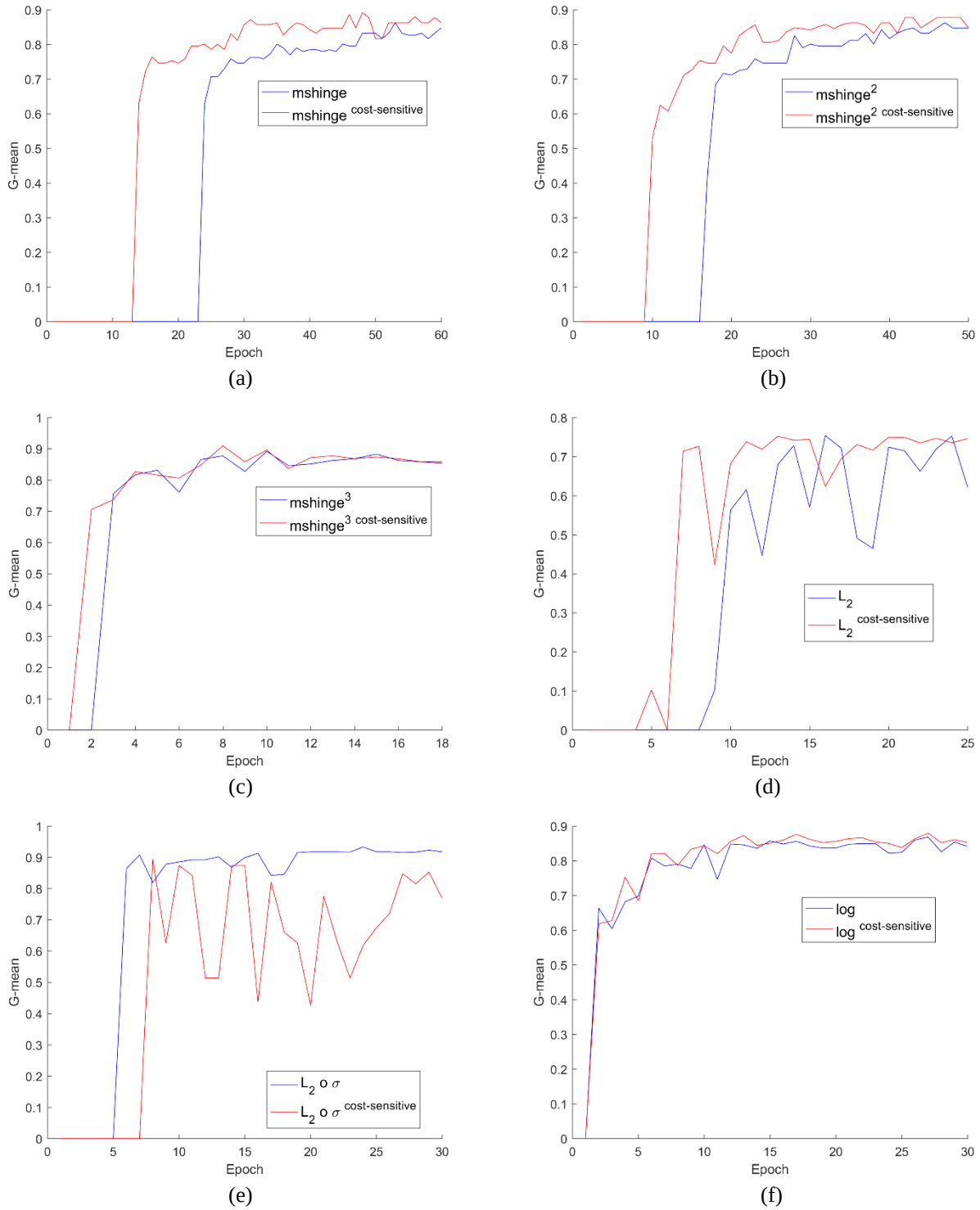


Figure. 3 Learning curves of NNs for different loss functions with and without the cost-sensitive approach applied. Each of the plots is a result of NN training on a specific dataset which is mentioned in parentheses: (a) $mshinge$ ("Ionosphere"), (b) $mshinge_2$ ("Ionosphere"), (c) $mshinge_3$ ("Ionosphere"), (d) L_2 ("PID"), (e) $L_2 \circ \sigma$ ("WPBC") and (f) $log \circ \sigma$ ("Satimage")

process for i. Nonetheless, we notice a different behavior for $log \circ \sigma$ and $L_2 \circ \sigma$ where the cost-sensitive approach either does not affect convergence speed as it is the case for $log \circ \sigma$ Fig. 3 (f) or reduces convergence speed as observed for $L_2 \circ \sigma$ Fig. 3 (e). This confirms our earliest assumption (section 3.2) that the impact of the cost-sensitive technique on loss functions with probability estimates such as $log \circ \sigma$ and $L_2 \circ \sigma$ has a sparse impact on the network learning compared to the large impact given loss functions with no probability estimates.

5.3 Comparative results

An empirical study was conducted to evaluate the effectiveness of the cost-sensitive algorithm in improving the performance of both shallow and deep neural networks (MLPs and CNNs). Our approach was compared with seven benchmark methods: SMOTE [1], Borderline-SMOTE [2], ADASYN [5], simple over-sampling, simple undersampling. the cost-sensitive method ST1 [14], and the generative adversarial network based framework GAMO [28]. The oversampling methods (SMOTE, Borderline-SMOTE and ADASYN) were implemented using the The imbalanced-learn library [33]. As for the configuration of GAMO, the official GAMO library at GitHub was used. A pure neural network, e.g., without any strategy to deal with imbalanced data, was also tested within exactly the same conditions of the other algorithms. ADAM was used as the optimization algorithm for training MLPs (with 1D datasets) while SGD momentum was used for training CNNs (with 2D datasets) since, as stated in [7], ADAM shows large performance improvement over SGD with momentum for MLPs versus marginal improvement for CNNs.

In these experiments, the chosen loss function to conduct all these methods is L_2. Morevoer, each experiment is conducted three times and its mean G-mean classification result is reported in Table 4. From obtained results, our approach competes with state-of-the-art approaches. Furthermore, several observations can be drawn:

(i) Our cost-sensitive method surpasses by far the baseline and undersampling techniques. Indeed, as we undersample, we are able to balance the data for proper neural network training, but we remove training instances which could hold valuable characteristics, thus losing relevant information.

(ii) Although our approach performs less than oversampling methods (simple oversampling, SMOTE, Borderline-SMOTE and ADASYN) for low-dimensional datasets (e.g., 1D datasets), it performs slightly better these methods for high-dimensional datasets (e.g., 2D datasets). For almost all datasets (except "PID" and "Thyroid"), the performance of our approach is higher than the oversampling technique. This is because oversampling generates redundant instances which might cause overfitting of the NN especially if dealing with a complex NN such as CNN.

(iii) Our approach is observed to perform better than the ST1 technique. This can be explained by the fact that ST1 upweights minority class (positive) instances with an unbounded weight which could get very high as $n_m \ll max(n_k)$ (n_m and $max(n_k)$ being respectively the number of minority class instances and the number of the most frequent class instances), thus making the resulting network too biased toward majority classes. On the other hand, our method upweights these instances with a bounded weight that varies from 0 to $\tau + 1$.

(iv) In general, our approach competes with the GAMO framework. Interestingly, our approach performs better than GAMO given very rare instances. Indeed, for Mnist10 and Mnist30, GAMO's performance surpasses the performance of our approach. But, as the number of minority class instances decreases, our approach is observed to perform slightly better than GAMO. This may be due to the fact that a higher weight-updating is given to minority classes as these latter are more infrequent, which contributes to a better learning of these classes and thus to a higher performance.

6. Conclusion

Our approach addresses the class imbalance problem which is commonly encountered when dealing with real-world datasets, by introducing a cost-sensitive strategy applied on neural networks at the training phase. Based on a cost-sensitive error function, its objective is to correctly classify minority classes and favour them as much as the frequent ones by assigning a weighted misclassification cost based on the distribution of classes. By properly weighting the loss function, weight-updating is intensified for the minority class based on the probability of their occurrence. Throughout this paper, results on several popular datasets showed that: (i) our approach has a better convergence than the baseline algorithm and competes with state-of-the-art techniques including oversampling methods and the deep learning method based on generative adversarial networks GAMO, (ii) it also offers a faster convergence by boosting the optimizer when positive instances are present, (iii) it can be applied on shallow and deep neural networks (MLPs and CNNs), which allows classification of imbalanced datasets with 1-,2-, and 3-dimentional inputs. We also show that the cost-sensitive approach is efficient only for loss functions with no probability estimates. Therefore, the cross entropy loss, which is the most commonly used loss function in the majority

Table 4. Mean classification results of neural networks over 3 runs using different loss functions in terms of average values of g-mean (in %). Best rates per loss function and per dataset are in bold. The abbreviation "Stand." stands for "Standard"

	$log \circ \sigma$		$Mshinge$		$Mshinge_2$		$Mshinge_3$		$L_2 \circ \sigma$		L_2	
	Stand.	**Ours**	**Stand.**	**Ours**	**Stand.**	**Ours**	**Stand.**	**Ours**	**Stand.**	**Ours**	**Stand.**	**Ours**
Ionosphere	**90.60**	88.29	86.25	**89.17**	86.25	**87.72**	89.17	**90.93**	**85.29**	82.02	70.26	**81.58**
PID	75.52	**76.38**	75.22	**75.27**	75.50	**76.54**	74.65	**75.77**	**76.08**	32.12	75.77	**75.97**
WBBC	92.83	**93.36**	92.40	**93.19**	92.17	**92.62**	**94.13**	93.98	**93.27**	89.38	82.30	**89.88**
SPECTF_Heart	79.66	**81.18**	**83.68**	**83.68**	80.52	**82.12**	81.32	**83.68**	73.49	**82.02**	80.05	**82.90**
Yeast_8l	**58.72**	58.02	56.36	**59.84**	0.00	**60.66**	53.11	**59.43**	0.00	0.00	57.32	**63.15**
Car	98.01	**100.00**	98.75	**99.63**	98.50	**99.94**	98.67	**99.82**	87.57	**96.52**	96.56	**99.57**
Satimage	87.83	**87.94**	86.86	**88.73**	87.60	**88.94**	87.58	**88.58**	**88.34**	82.74	85.55	**88.12**
Thyroid	51.03	**54.93**	0.00	**60.69**	57.97	**72.26**	50.51	**72.65**	0.00	**41.08**	46.75	**73.14**
Mnist10	94.88	**95.15**	94.78	**96.55**	94.96	**95.18**	95.18	**95.92**	90.74	**96.21**	91.76	**96.27**
Mnist30	**92.00**	90.94	91.95	**93.45**	91.89	**92.50**	90.31	**90.75**	90.43	**92.36**	90.43	**92.64**
Mnist40	97.77	**98.50**	95.81	**98.41**	87.81	**97.13**	96.80	**98.52**	0.00	**97.81**	0.00	**98.41**
Mnist50	**98.70**	98.11	97.89	**98.70**	**98.66**	98.29	98.61	**98.67**	0.00	**97.05**	0.00	**98.43**

Table 4. Comparative results between different methods in terms of the G-mean performance metric (in %).

Dataset	L_2	**SMOTE [1]**	**Borderline-SMOTE** [2]	**ADASYN** [5]	**Over-sampling**	**Under-sampling**	**Ours**	**ST1** [14]	**GAMO** [28]
PID	0.760	0.789	0.762	0.737	0.786	0.754	**0.772**	0.762	0.760
WBBC	0.928	0.931	0.943	0.939	0.926	0.926	0.943	0.931	**0.948**
yeast_8l	0.590	0.640	0.640	0.571	0.620	0.559	0.630	0.625	**0.641**
thyroid	0.824	0.932	0.942	0.938	0.937	0.776	0.864	0.931	**0.941**
car	0.987	0.980	0.976	0.984	0.996	0.940	**0.996**	0.993	0.985
satimage	0.893	**0.946**	0.940	0.930	0.924	0.913	0.935	0.898	0.941
mnist10	0.974	0.988	**0.990**	0.981	0.980	0.939	0.988	0.963	0.989
mnist30	0.960	0.989	0.986	0.847	0.906	0.888	0.990	0.962	**0.991**
mnist40	0.941	0.988	0.990	0.976	0.959	0.837	**0.996**	0.976	0.992
mnist50	0	0.991	0.990	0.993	0.985	0.772	**0.994**	0.941	0.991

of studies, is not a good choice as it comes to classifying imbalanced datasets with our cost-sensitive approach.

Conflicts of interest

The authors declare no conflict of interest.

Author contributions

Supervision, Taoufiq Gadi and El Hassan Essoufi; Review and editing, Mohamed Elhassan Bassir.

Acknowledgments

This research received no specific grant from any funding agency.

Data availability

The data that support the findings of this study are openly available at the URL:

https://github.com/lsadouk/imbalanced_classification

References

[1] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "{SMOTE}: synthetic minority over-sampling technique", *Journal of Artificial Intelligence Research*, Vol. 16, pp. 321–357, 2002.

[2] H. Han, W. Y. Wang, and B. H. Mao, "Borderline-SMOTE: A New Over-Sampling Method in Imbalanced Data Sets Learning", In: *Proc. of International Conference on Intelligent Computing*, pp. 878–887, 2005.

[3] C. Bunkhumpornpat, K. Sinapiromsaran, and C. Lursinsap, "Safe-Level-SMOTE: Safe-Level-Synthetic Minority Over-Sampling TEchnique for Handling the Class Imbalanced Problem", In: *Proc. of the 13th Pacific-Asia Conference on Advances in Knowledge Discovery and Data*

Mining, pp. 475–482, 2009.

[4] T. MacIejewski and J. Stefanowski, "Local neighbourhood extension of SMOTE for mining imbalanced data", In: *Proc. of IEEE SSCI 2011: Symposium Series on Computational Intelligence - CIDM 2011: 2011 IEEE Symposium on Computational Intelligence and Data Mining*, pp. 104–111, 2011.

[5] H. He, Y. Bai, E. A. Garcia, and S. Li, "ADASYN: Adaptive synthetic sampling approach for imbalanced learning", In: *Proc. of the International Joint Conference on Neural Networks*, pp. 1322–1328, 2008.

[6] J. Song, Y. Shen, Y. Jing, and M. Song, "Towards Deeper Insights into Deep Learning", In: *Proc. of the 25th international conference on Machine learning*, Vol. 2, pp. 674–684, 2017.

[7] M. Kubat and S. Matwin, "Addressing the curse of imbalanced training sets: One sided selection", In: *Proc. of the 14th International Conference on Machine Learning*, pp. 179–186, 1997.

[8] J. Laurikkala, "Improving Identification of Difficult Small Classes by Balancing Class Distribution", In: *Proc. of Conference on Artificial Intelligence in Medicine in Europe*, pp. 63–66, 2001.

[9] S. Pouyanfar, Y. Tao, A. Mohan, H. Tian, A. S. Kaseb, and K. Gauen, "Dynamic Sampling in Convolutional Neural Networks for Imbalanced Data Classification", In: *Proc. of IEEE 1st Conference on Multimedia Information Processing and Retrieval*, no. June, pp. 112–117, 2018.

[10] D. Zhang, Yong and Wang, "A cost-sensitive ensemble method for class-imbalanced datasets", *Abstr. Appl. Anal.*, Vol. 2013, 2013.

[11] M. Gao, X. Hong, and C. J. Harris, "Construction of neurofuzzy models for imbalanced data classification", *IEEE Transactions on Fuzzy Systems*, Vol. 22, No. 6, pp. 1472–1488, 2014.

[12] K. Li, X. Kong, Z. Lu, L. Wenyin, and J. Yin, "Boosting weighted ELM for imbalanced learning", *Neurocomputing*, Vol. 128, pp. 15–21, 2014.

[13] B. X. Wang and N. Japkowicz, "Boosting support vector machines for imbalanced data sets", *Knowledge and Information Systems*, Vol. 25, No. 1, pp. 1–20, 2010.

[14] R. Alejo, V. Garcia, J. M. Sotoca, R. A. Mollineda, and J. S. Sanchez, "Improving the performance of the RBF neural networks trained with imbalanced samples", *Computational and Ambient Intelligence*, Vol. 4507, pp. 162–169, 2007.

[15] D. E. Rumelhart, J. L. McClelland, C. Asanuma, F. H. C. Crick, J. L. Elman, and G. E. Hinton, "Parallel Distributed Processing: Explorations in the Microstructure of Cognition", *Computational Models of Cognition and Perception*, pp. 318–362, 1986.

[16] C. L. Castro and A. D. P. Braga, "artificial neural networks learning in roc space", In: *Proc. of the International Joint Conference on Computational Intelligence*, pp. 484–489, 2009.

[17] S. H. Oh, "Error back-propagation algorithm for classification of imbalanced data", *Neurocomputing*, Vol. 74, No. 6, pp. 1058–1061, 2011.

[18] C. L. Castro and A. P. Braga, "Novel cost-sensitive approach to improve the multilayer perceptron performance on imbalanced data", *IEEE Transactions on Neural Networks and Learning Systems*, Vol. 24, No. 6, pp. 888–899, 2013.

[19] Z. A. Huang, Y. Sang, Y. Sun, and J. Lv, "Neural network with absent minority class samples and boundary shifting for imbalanced data classification", *Neural Computing and Applications*, pp. 1–17, 2023.

[20] M. Lázaro and A. F. Vidal, "Neural network for ordinal classification of imbalanced data by minimizing a Bayesian cost", *Pattern Recognition*, vol. 137, pp. 109303, 2023.

[21] S. H. Khan, M. Hayat, M. Bennamoun, F. Sohel, R. Togneri, and C. V Mar, "Cost-Sensitive Learning of Deep Feature Representations from Imbalanced Data", *IEEE Transactions on Neural Networks and Learning Systems*, Vol. 29, No. 8, pp. 3573–3587, 2018.

[22] C. Huang, Y. Li, C. C. Loy, and X. Tang, "Learning Deep Representation for Imbalanced Classification", In: *Proc. of 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5375–5384, 2016.

[23] S. Yue, "Imbalanced Malware Images Classification: a CNN based Approach", *arXiv Prepr. arXiv1708.08042*, 2017.

[24] S. Wang, W. Liu, J. Wu, L. Cao, Q. Meng, and P. J. Kennedy, "Training Deep Neural Networks on Imbalanced Data Sets", In: *Proc. of Neural Networks (IJCNN), 2016 International Joint Conference on*, pp. 4368–4374, 2016.

[25] R. Harliman and K. Uchida, "Data- and algorithm-hybrid approach for imbalanced data problems in deep neural network", *International Journal of Machine Learning and Computing*, Vol. 8, No. 3, pp. 208–213, 2018.

[26] G. Douzas and F. Bacao, "Effective data generation for imbalanced learning using

conditional generative adversarial networks", *Expert Syst. Appl.*, Vol. 91, pp. 464–471, 2018,
[27] G. Mariani, F. Scheidegger, R. Istrate, C. Bekas, and C. Malossi, "BAGAN: Data Augmentation with Balancing GAN", *arXiv.org*, 2018.
[28] S. S. Mullick, S. Datta, and S. Das, "Generative adversarial minority oversampling", In: *Proc. of the IEEE International Conference on Computer Vision*, Vol. 2019-Octob, pp. 1695–1704, 2019.
[29] W. Jo and D. Kim, "OBGAN: Minority oversampling near borderline with generative adversarial networks", *Expert Systems With Applications*, Vol. 197, p. 116694, Jul. 2022,
[30] D. Yun, H. Lee, and S. H. Choi, "A deep learning-based approach to non-intrusive objective speech intelligibility estimation", *IEICE Transactions on Information and Systems*, Vol. E101D, No. 4, pp. 1207–1208, 2018.
[31] M. Kubat, R. C. Holte, and S. Matwin, "Machine Learning for the Detection of Oil Spills in Satellite Radar Images", *Machine Learning*, Vol. 30, pp. 195–215, 1998.
[32] Y. Sun, Yanmin and Kamel, S. Mohamed and Wang, "Boosting for Learning Multiple Classes with Imbalanced Class Distribution", In: *Proc. of the Sixth International Conference on Data Mining*, pp. 592-602, 2006.
[33] G. Lemaître, F. Nogueira, and C. K. Aridas, "Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning", *Journal of Machine Learning Research*, Vol. 18, pp. 1–5, 2017.

Appendix

Table 1. List of notations used in this paper

Symbol	Description
NN	Neural Network
MLP	Multi-Layer Perceptrons
CNN	Convolutional Neural Networks
$\boldsymbol{\theta}$	Parameters of the neural network
$\boldsymbol{E}(\cdot)$	Objective function
$\boldsymbol{i}$	Instance of a batch taken from a given dataset
$\boldsymbol{y^{(i)}}$	the true label (as one-hot encoding) of instance i
$\boldsymbol{\hat{y}^{(i)}}$	predicted output vector of the neural network of instance i
$\boldsymbol{N}$	number of instances per batch
$\boldsymbol{l}(\cdot)$	Loss function
$\boldsymbol{\lambda(m)}$	weighting parameter corresponding to the class m
$\boldsymbol{\emptyset(m)}$	Relevance of the class m
$\boldsymbol{x_p^{(i)}}$	Value of the input node p at instance i
$\boldsymbol{L_2}$	Squared loss function
$\boldsymbol{Mshinge}$	Multiclass structured hinge loss (Crammer-Singer loss)
$\boldsymbol{Mshinge_2}$	Squared Multiclass structured hinge loss
$\boldsymbol{Mshinge_3}$	Cubed Multiclass structured hinge loss
$\boldsymbol{\sigma}(\cdot)$	A propability estimate function (such as the softmax and sigmoid function)
$\boldsymbol{log \circ \sigma}$	Cross entropy loss
ReLU	Rectified Linear Unit
G-Mean	Geometric Mean
SGD	Stochastic Gradient Descent