Improving MLP Neural Network Performance by Noise Reduction

Mirosław Kordos[1] and Andrzej Rusiecki[2]

[1] University of Bielsko-Biala, Department of Mathematics and Computer Science
Bielsko-Biała, Willowa 2, Poland
mkordos@ath.bielsko.pl
[2] Wroclaw University of Technology
Institute of Computer Engineering, Control and Robotics
Wrocław, Wybrzeze Wyspianskiego 27, Poland
andrzej.rusiecki@pwr.wroc.pl

Abstract. In this paper we examine several methods for improving the performance of MLP neural networks by eliminating the influence of outliers and compare them experimentally on several classification and regression tasks. The examined method include: pre-training outlier elimination, use of different error measures during network training, replacing the weighted input sum with weighted median in the neuron input functions and various combinations of them. We show how these methods influence the network prediction. Based on the experimental results, we also present a novel hybrid approach improving the network performance.

1 Introduction

An outlier is an example that is numerically distant from the rest of the surrounding data. That can be either a point that is close to its neighbors in the input space, but distant from the output space (different class or much different value in the case of regression) or that is far from any points as well in the input as in the output space. Outliers often indicate either of measurement error or some data points that are so rare that should not be taken into account while building the data model. Thus we want either to discard them or use approaches that are robust to outliers. Another problem is that sometimes it cannot be clearly stated if a given point is an outlier or not and rather some degree of being an outlier that a crisp decision is preferred. In that case the model does not entirely disregard such a point, but decreases its influence on the model parameters.

Multilayer perceptron neural networks (MLP) are one of the models that can be used to represent the data. They are trained by minimizing an error function on the training set, to make the network map the input data distribution to the output space variable, which can be either discrete in the case of classification or continuous in the case of regression, or can represent some structured data. The performance of the trained network obviously depends not only on the network architecture and learning algorithms, but on the quality of the training data

as well. A noisy dataset with many outliers does not describe well the desired mapping from the input to output space. In this case also the neural network trained on that data will not implement the proper mapping.

The methods designed to make the network training robust to gross errors and outlying data points are usually tested on artificially generated datasets with variable amount of large outliers generated with different models [1–3, 24]. In this article we intend to investigate the effectiveness of such algorithms on real data without any additional contamination. This is due to the fact that in real problems we do not know whether the data is contaminated, or reliable, when the MLP model is built.

In this work we consider two groups of approaches to deal with this problem. The first group is based on the modification of the neural network parameters, as the error function or neuron input function. The network training is modified and the training data is left in its original state. We discuss these approaches in section 2. The second group of approaches is based on outlier reduction methods. In this case the data is modified and the network is trained using the standard mean square error function. This is discussed in section 3. We also present an algorithm based on joining the aforementioned approaches. Section 4 presents the experimental comparison of the discussed methods on several classification and regression tasks and finally the last section concludes this work.

2 Modification of the Network Parameters

2.1 Outlier Dependent Error

The simplest approach to make the MLP training process more robust to outliers is to replace common MSE (mean squared error) criterion with a function based on the idea of robust statistical methods. The MSE function is typically used for supervised neural networks training methods because it is simple and easy to optimize error measure. However, similarly to the least mean squares method, it is optimal only for data sets contaminated at most by Gaussian white noise [9, 11, 19]. This is due to the fact that MSE is strongly influenced by large errors. In the case of network training, this influence, measured by a derivative with respect to residuals, can be described by a linear function [9, 17]. To overcome the problem of unpredictable MLP model for training data containing outliers, several robust learning algorithms have been proposed. Such algorithms very often make use of modified error function, derived from robust statistical estimators. The training data are not filtered, so the robustness to outliers is based only on reducing the impact of large training residuals, potentially caused by outlying data points.

The error training function can be modified in many ways: in [17] Liano proposed a new LMLS (Least Mean Log Squares) error function based on so-called M-estimators, which should be optimal for the Cauchy distribution but performs well also for other long-tailed error distributions. Chen and Jain [1] decided to use the Hampel's hyperbolic tangent with additional scale estimator β. The scale estimator helped in determining the range of residuals believed to

be outliers. A similar error performance function combined with the annealing scheme to decrease β with the training progress was proposed by Chuang and Su [2]. A more sophisticated approach, using tau-estimators was described in [20]. Also quartile-based estimators were applied as the error function in the LTS (Least Trimmed Squares) algorithm [23] and in [3], where El-Melegy *et al.* presented the Simulated Annealing for Least Median of Squares (SA-LMedS) algorithm. Similar median error function was described in [24]. The error measure based on robust estimators was also combined with approaches known from image processing, as random sample consensus algorithm [4–6].

All the aforementioned methods focus mainly on modifying the error function, in order to decrease the influence that outliers may have on the network training. In this article we decided to test the most popular robust error measure, namely LMLS. This modification of the training algorithm is a highly cited technique [1–3, 6] and this is why we chose to use it. The LMLS error is then defined as:

$$E_{LMLS}(\boldsymbol{w}) = \sum_{k=1}^{n}\sum_{i=1}^{m} \log(1 + \frac{1}{2} {r_{ki}}^2(\boldsymbol{w})), \tag{1}$$

where $r_{ki} = (y_{ki}(\boldsymbol{w}) - t_{ki})$ is the error of i-th output for the k-th training set element, n is the size of the training set and m is the number of network outputs (see fig. 1-left).

Our experiments were performed also with the mean absolute error (MAE) function. This well-known error formula can be also derived from robust M-estimators. As it was demonstrated in [3], the MAE criterion is probably the most effective of all constant error functions, when applied to training data with artificially introduced outliers. We define mean absolute error as:

$$E_{MAE}(\boldsymbol{w}) = \sum_{k=1}^{n}\sum_{i=1}^{m} |r_{ki}(\boldsymbol{w})|. \tag{2}$$

2.2 Median Input Function

An approach to MLP network training using the median neuron input function (MIF) was proposed in [22]. In such networks summation of weighted input signals is replaced with their median. When the summation is replaced by more robust operation, such as median, the neuron output becomes less sensitive to the changes in the input (neuron input, or input weights). Hence, the MIF is not a direct method to make the training process more robust to outliers but it enables the MLP to build a more general model.

Then we can define the MIF neuron output as:

$$y_{out} = f(\text{med}\{w_i x_i\}_{i=1}^{N}), \tag{3}$$

where $f(\cdot)$ denotes neuron transfer function (e.g. sigmoid or linear), x_j are neuron inputs, w_i is the i-th input weight and N denotes input size. However, there exist several problems concerning practical use of MIF networks. It is clearly evident

that calculating MIF output is computationally more expensive than in the case of simple sum. Moreover, the input function given by (3) makes the network error function non-differentiable, so it cannot be simply trained with gradient-based methods. So for that purpose in [22] an approximated algorithm, based on the gradient for a simple sum, was described. This approach can be applied also for non-differentiable error performance based on the median of residuals [24].

Another problem with a MIF is that when the network is trained on regression tasks, the model built by the network may not react to single or small changes in the dataset. For that reason we combine the median input (which provides the outlier-robust part) with the sum (which provides high sensibility), defining a new input function as:

$$y_{out} = f(\delta \text{med}\{w_i x_i\}_{i=1}^{N} + (1-\delta)\sum_{i=1}^{N} w_i x_i), \tag{4}$$

where $0 < \delta < 1$ determines the median influence on neuron input function. We experimentally determined that the optimal range of δ for most datasets is between 0.6 and 0.9. The learning process was not sensible to little changes of δ within this range, so in the experiments we used $\delta = 0.75$.

3 Outlier Reduction

3.1 Instance Selection

The reasons for reducing the number of instances in the training set include: noise reduction by elimination outliers, reducing the data set size and sometimes improving generalization by eliminating instances that are too similar to each other, faster training of the model an a smaller dataset and faster prediction in of the model, especially in the case of lazy-learning algorithms, as k-NN. Thus the aim of the instance selection algorithms can be either noise reduction, as in the case of the Edited Nearest Neighbor algorithm (ENN) [27] or data compression, as in the case of the Condensed Nearest Neighbor rule (CNN) [10] or both. In this work we focus on the first area: noise reduction. Although in practical applications preliminary attribute selection is also beneficial [7] (irrelevant attributes may produce false positive signals in outlier detection algorithms), we do not discuss the aspect here for the sake of focusing on the main topic this work.

A large survey including almost 70 different algorithms of instance selection for classification tasks can be found in [25]. The instance selection issue for regression tasks is much more complex. In classification tasks only the boundaries between classes must be determined, while in regression tasks the output value must be assessed at each point of the input space. Moreover, in classification tasks there are at most several different classes, while in regression tasks, the output of the system is continuous, so there are an unlimited number of possible values to be predicted by the system. The decision about rejection of a given vector in classification tasks can be made based on a right or wrong classification of the vector. In regression problems, rather a threshold defining the difference between

the predicted and the actual value should be set. Determining the threshold (which is rather a function than a constant value) is an important point. Another issue is the measure of the quality of the model, which in classification tasks is very straightforward (classification accuracy), while in regression tasks, it can be defined in several ways. In practical solutions not always the simple error definitions as the MSE (mean square error) work best [14], because also the cost of the error must be taken into account. Depending on a given application, the cost can be higher in different areas of the output (in a similar way as cost matrix in classification tasks) and can depend on the error value. Because of the challenges, there were very few approaches in the literature to instance selection for regression problems. Moreover, the approaches were verified only on artificial datasets generated especially for the purpose of testing the algorithms. Zhang [28] presented a method to select the input vectors while calculating the output with k-NN. Tolvi [26] presented a genetic algorithm to perform feature and instance selection for linear regression models. In their works Guillen et al. [8] discussed the concept of mutual information used for selection of prototypes in regression problems.

Algorithm 1 regENN algorithm

Require: T
  $m \leftarrow sizeof(\mathbf{T})$;
  for $i = 1 \ldots m$ **do**
    $\bar{Y}(\mathbf{x}_i) =$NN$((\mathbf{T} \setminus \mathbf{x}_i), \mathbf{x}_i)$;
    $S \leftarrow$ Model$(\mathbf{T}, \mathbf{x}_i)$
    $\theta = \alpha \cdot std\left(Y\left(\mathbf{X}_S\right)\right)$
    if $\left|Y(\mathbf{x}_i) - \bar{Y}(\mathbf{x}_i)\right| > \theta$ **then**
      $\mathbf{T} \leftarrow \mathbf{T} \setminus \mathbf{x}_i$
    end if
  end for
  $\mathbf{P} \leftarrow \mathbf{T}$
  return P

For the purpose of noise reduction we will use the ENN (Edited Nearest Neighbor) algorithm [27] and its version for the regression tasks - regENN. The main idea of the ENN algorithm is to remove a given instance if its class is different than the majority class of its neighbors. ENN starts with the whole original training set T. Each instance, which is wrongly classified by its k nearest neighbors is removed from the dataset, as it is supposed to be an outlier. In repeated ENN, the process of ENN is iteratively repeated as long as there are any instances wrongly classified. In all k-NN algorithm, the ENN is repeated for all k from $k = 1$ to k_max. In [13] we proposed an extension of several instance selection algorithms for regression tasks. We shortly describe below the ENN algorithm for regression tasks - regENN.

To adjust the ENN algorithms to regression tasks the wrong/correct classification decision is replaced with a distance measure and a similarity threshold, to decide if a given vector can be considered as similar to its neighbors. We use a weighted k-NN with $k = 9$, where the weight w_i exponentially decreases with the distance d_i between the given vector and its i-th neighbor x_i. The predicted output y is given by eq. 1.

$$y = \frac{\sum_{i=1}^{k} w_i y_i}{\sum_{i=1}^{k} w_i} \tag{5}$$

where $w_i = 2^{-0.2d_i}$. We use Euclidean distance measure and a threshold Θ, which expresses the maximum difference between the output values of two vectors to consider them similar. Using Θ proportional to the standard deviation of k nearest neighbors of the vector x_i reflects the speed of changes of the output around x_i and allows adjusting the threshold to that local landscape, what, as the experiments showed, allows for obtaining higher compression of the dataset. As the regression model to predict the output Y(xi) we use k-NN with $k = 9$ as the Model(T,x_i) ($k = 9$ usually produced good results). In case of regression we experimentally evaluated the optimal Θ and we used Θ equal to 5 standard deviations of the 9 nearest neighbors for RegENN.

3.2 Anomaly Detection

We used the ENN and regENN algorithms to reject the outliers prior to the network training. Here we describe an algorithm based on a k-NN Global Anomaly Score algorithm (k-NN GAS), which we use to assess the degree to which a given instance is an outlier prior to the network training and then remain all the instances in the training set, but differentiate the way they are included in the training. The k-NN Global Anomaly Score algorithm calculates the anomaly score based on the k nearest neighbors implementation. The outlier score of an instance is the average distance between the instance and its k nearest neighbors. In the experiments we use k = 9 and Euclidean distance measure. The higher the outlier score the more anomalous the instance is. However, for the purpose of labeled data, we had to extend this score, including for the calculation the distance in the input space d_x and the distance in the output space d_y. In the case of classification we add one to d_y for each neighbor of a different class and zero for each neighbor of the same class. We define the modified anomaly score A_{sc} as:

$$A_{sc} = d_y/d_x \tag{6}$$

Then we assume that the higher the anomaly score is, the more likely the instance is to be an outlier and the less influence it should have on the network training. We obtain this by dividing the error the network makes during the

training on each instance by a greater value if the instance anomaly score for the instance is higher (see fig. 1-right):

$$Error = \begin{cases} Error/A_{sc}^2 & \text{if } A_{sc} > \alpha median(A_{sc}) \\ Error/(\alpha median(A_{sc})) & \text{otherwise} \end{cases} \quad (7)$$

Where α is a parameter. In the experiments we used $\alpha = 1$ This modification (considering also distance in the output space) of the k-NN global anomaly score incorporates the idea of the local density of Local Outlier Probability detection algorithm (LOOP). LOOP, contrary to k-NN global anomaly score, includes only the local density of the points in the hyperspace. If a density measured by the number of points in a hypersphere of a certain radius, where the given point is in the center of the hypersphere, is much smaller then inside hypersheres centered upon k-neighbors of that point, then the point is considered an outlier. The resulting values are scaled to a value range of (0;1). The higher the value the more anomalous the instance is. A survey of outlier detection methods can be found in [12].

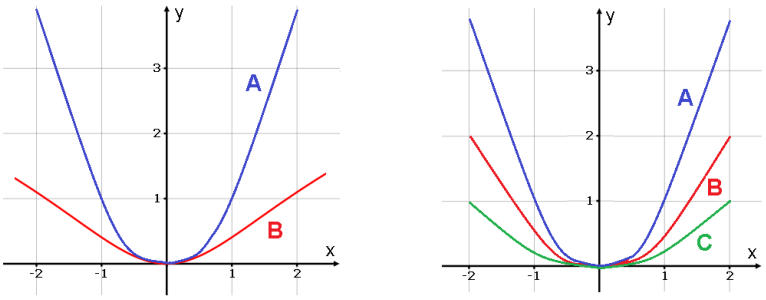

Fig. 1. Letf: The square (A) and the LMLS (B) error function. Right: The square error function used for non-outliers (A), for a weak outlier (B) and for a strong outlier (C).

4 Experimental Comparison

4.1 Datasets

It is worth noticing that all the considered datasets were not artificially contaminated. We did not introduce artificial outliers, testing the algorithms on the real data. We performed the experiments on eight datasets. First all the datasets were standardized so that the mean value of each attribute is zero and the standard deviation is one to make comparison of the results easier. We used four classification and four regression datasets. Six datasets come from the UCI Machine Learning Repository [18]: Iris (3 classes, 4 attributes, 150 instances), Diabetes (2 classes, attr., inst.), Glass (5 classes, attr., inst.), Ionosphere (2

classes, attr., inst.), Concrete Compression Strength (regression, 7 attributes, 1030 instances), Crime and Communities (regression, 7 attr., 320 inst.). Two datasets comes from a metallurgical industry. The purpose of the SteelC dataset (regression, 14 attr., 2384 inst.) is to predict the amount of carbon that must be added in the steel-making process, given various chemical and physical properties of the liquid steel in the furnace. The purpose of the SteelT dataset (regression, 11 attr., 7401 inst.) is to predict the actual temperature of the liquid steel given a set of physical values, as temperature in various points on the surface of the furnace, the energy delivered to the process and others (directly measuring the temperature requires some disruptions of the steel-making process making it longer and more expensive).

4.2 Experimental Setup

We implemented the algorithms in C# and Matlab. The source code and datasets used in the experiments can be downloaded from [16]. The whole process in different configurations was run in 10-fold crossvalidation loops. To be able to compare the results, we always measure and raport in the table the MSE error on the test sets, no matter which error function was used for the network training. Also the MLP architecture was constant (the same for each training method) for a given dataset. We used 5 hidden neurons for iris, diabetes, glass, steelC, and crime, and 6 hidden neurons for the ionosphere, steelT and concrete.

4.3 Modification of the Network Parameters with Gradient-based learning

In our experiments we decided to apply two general training frameworks: gradient-based learning for modified network parameters (different error and neuron input functions), and non-gradient VSS method for the rest of the algorithms. Tested error functions included LMLS and MAE error measures (equations 1 and 2), used as MLP training criteria. We used also MIF (3), and MedSum (4) neuron inputs. For the case of MedSum, we assumed equal participation of both input functions, setting $\delta = 0.5$. The results of such modified networks were compared to traditional approach: the MSE error function and simple weighted sum as neuron input.

For the modified error functions we could not use one of the most popular methods, namely, Levenberg-Marquardt algorithm, which is dedicated to the MSE. Moreover, as it was mentioned in Section 2.2, the MIF nets cannot be trained by regular gradient methods, so following [24, 22] we decided to apply resilient backpropagation (Rprop) algorithm [21].

Analyzing the results obtained for these methods, we cannot definitively decide that there is a single method, which outperforms other approaches. The standard MSE network is the best between gradient-based methods only for one dataset (steelC), similarly MIF (ionosphere), whereas MAE, LMLS, and MedSum are the winners for two datasets each. (As mentioned earlier we compared the MSE on the test sets, no matter which error function was used for network

training.) The performances of the tested approaches differ, depending on the training data sets. Only pure MIF strategy, because of its discontinuity, often worsen the results. In general, these algorithms seem to be more efficient, than outlier reduction methods, for classification tasks, and less accurate for regression problems.

4.4 Outlier Reduction with Nongradient-based learning

The training algorithm we used to test the outlier reduction-based method was a non-gradient based VSS (Variable Step Search Algorithm). The idea of the VSS algorithm is to make advantage of the properties of MLP network error surface in the weight space, on which the training trajectory is situated, that it is more likely that each training epoch the trajectory direction will be only slightly adjusted that totally changed and therefore the VSS algorithm makes guesses about the optimal modification of each neuron weights in each epoch and then the guesses are adjusted as needed. Details of the algorithm can be found in [15]. The VSS algorithm was also used to test the coexistence of outlier reduction methods and the MedSum function. We used the same network architectures as for the Rprop training and about 12 training epochs for each dataset. We are going to join two our programs in one and use only one training method in the future experiments. However, to obtain a clear comparison between the methods tested with Rprop and with VSS, the exact number of epochs was adjusted so that the crossvalidation accuracy on the standard network trained with Rprop and VSS would be the same.

Looking at the results obtained for these methods, we can observe some regularity. First, the ENN usually did not improve the results with the classification problems, even more the results were worse. This can be explained by the fact, that the instances, which were rejected by ENN were in most cases situated close to the decision boundaries and thus removing them made determining the class boundaries less precise. It would probably work better with very noisy data, where it would reject more outliers that vectors situated close to class boundaries. In case of regression the regENN algorithm can be adjusted with the Θ parameter to reject the optimal number of outliers (if the error increases we increase Θ, till the error stop increasing), so it does not cause error increase. However it decreased the error only in two out of four cases. The modified k-NN GAS algorithm adjusts the error measure individually to each training vector. Because it can be adjusted with the α parameter it also did not worsen the results, but it improved them in all but two cases. Joining the modified k-NN GAS algorithm with the MedSum neuron input function proved to work best. The results were further improved in six out of eight cases. This can be explained by the fact that when to the summation in the neuron function a more stable operation, such as median is added, the neuron output becomes less sensitive to perturbances in the data.

Table 1. Experimental results for classification problems - classification accuracy in 10-fold crossvalidation (higher is better)

algorithm	iris	diabetes	glass	ionosphere
MSE	0.963±0.038	0.765±0.046	0.670±0.094	0.911±0.051
LMLS	0.971±0.055	0.773±0.044	0.691±0.089	0.899±0.083
MAE	0.951±0.063	**0.784**±0.052	0.592±0.082	0.828±0.061
MIF	0.940±0.095	0.695±0.061	0.605±0.102	**0.950**±0.048
MedSum	0.956±0.058	0.760±0.037	**0.696**±0.104	0.922±0.060
ENN	0.960±0.040	0.765±0.043	0.614±0.066	0.910±0.059
k-NN GAS	**0.974**±0.045	0.775±0.031	0.674±0.074	0.911±0.029
MedSum + k-NN GAS	0.967±0.032	0.764±0.047	0.684±0.046	0.917±0.053

Table 2. Experimental results for regression problems - MSE in 10-fold crossvalidation (lower is better)

algorithm	steelC	steelT	concrete	crime
MSE	0.031±0.013	0.576±0.111	0.798±0.250	0.343±0.062
LMLS	0.033±0.019	0.551±0.072	0.857±0.296	0.345±0.101
MAE	0.038±0.013	0.560±0.082	0.969±0.358	**0.318**±0.109
MIF	0.070±0.018	0.844±0.070	0.869±0.267	0.341±0.060
MedSum	0.035±0.015	0.574±0.113	0.794±0.321	0.338±0.080
regENN	0.032±0.010	0.576±0.111	0.778±0.256	0.341±0.067
k-NN GAS	0.031±0.018	0.541±0.097	0.778±0.251	0.341±0.060
MedSum + k-NN GAS	**0.028**±0.013	**0.526**±0.092	**0.764**±0.211	0.336±0.073

5 Conclusions

We examined several methods of improving the performance of MLP neural networks by outlier elimination and compare them experimentally on several classification and regression tasks. The examined method included: pre-training outlier elimination, use of different error measures during network training, replacing the weighed input sum with weighed median in the neuron input functions and various combinations of them.

The obtained results are very interesting. There is an additional cost of calculating the median equal to the quicksort algorithm of sorting the arrays of the dimension of the number of instances in the training set. Especially joining the two groups of methods in one network training, allowed for the best error reduction, especially in regression problems. In most cases, modified MLP training methods outperformed traditional MSE approach, however the choice of the best algorithm can be merely indicative. Especially for the regression problems most of the best results were obtained for our novel hybrid algorithm, joining k-NN GAS with MedSum neuron input. This approach can be considered as effective tool to improve the MLP performance on real-life problems.

References

1. Chen, D., Jain, R.: A robust backpropagation learning algorithm for function approximation. Neural Networks, IEEE Transactions on 5(3), 467–479 (1994)
2. Chuang, C.C., Su, S.F., Hsiao, C.C.: The annealing robust backpropagation (arbp) learning algorithm. Neural Networks, IEEE Transactions on 11(5), 1067–1077 (2000)
3. El-Melegy, M., Essai, M., Ali, A.: Robust training of artificial feedforward neural networks. In: Hassanien, A.E., Abraham, A., Vasilakos, A., Pedrycz, W. (eds.) Foundations of Computational, Intelligence Volume 1, Studies in Computational Intelligence, vol. 201, pp. 217–242. Springer Berlin Heidelberg (2009), http://dx.doi.org/10.1007/978-3-642-01082-8_9
4. El-Melegy, M.: Random sampler m-estimator algorithm for robust function approximation via feed-forward neural networks. In: Neural Networks (IJCNN), The 2011 International Joint Conference on. pp. 3134–3140 (2011)
5. El-Melegy, M.: Ransac algorithm with sequential probability ratio test for robust training of feed-forward neural networks. In: Neural Networks (IJCNN), The 2011 International Joint Conference on. pp. 3256–3263 (2011)
6. El-Melegy, M.: Random sampler m-estimator algorithm with sequential probability ratio test for robust function approximation via feed-forward neural networks. Neural Networks and Learning Systems, IEEE Transactions on 24(7), 1074–1085 (2013)
7. Golak, S., Burchart-Korol, D., Czaplicka-Kolarz, K., Wieczorek, T.: Application of neural network for the prediction of eco-efficiency. In: Liu, D., Zhang, H., Polycarpou, M., Alippi, C., He, H. (eds.) Advances in Neural Networks – ISNN 2011, Lecture Notes in Computer Science, vol. 6677, pp. 380–387. Springer Berlin Heidelberg (2011), http://dx.doi.org/10.1007/978-3-642-21111-9_43
8. Guillen, A.: Applying mutual information for prototype or instance selection in regression problems. In: ESANN 2009 (2009)
9. Hampel, F.R., Ronchetti, E.M., Rousseeuw, P.J., Stahel, W.A.: Robust Statistics: The Approach Based on Influence Functions (Wiley Series in Probability and Statistics). Wiley-Interscience, New York, revised edn. (Apr 2005), http://www.worldcat.org/isbn/0471735779
10. Hart, P.: The condensed nearest neighbor rule (corresp.). Information Theory, IEEE Transactions on 14(3), 515–516 (1968)
11. Huber, P.J.: Robust Statistics. Wiley Series in Probability and Statistics, Wiley-Interscience (1981), http://www.worldcat.org/isbn/0471418056
12. I., B.G.: Outlier detection. Kluwer Academic Publishers (2005)
13. Kordos, M., Białka, S., Blachnik, M.: Instance selection in logical rule extraction for regression problems. In: Rutkowski, L., Korytkowski, M., Scherer, R., Tadeusiewicz, R., Zadeh, L., Zurada, J. (eds.) Artificial Intelligence and Soft Computing, Lecture Notes in Computer Science, vol. 7895, pp. 167–175. Springer Berlin Heidelberg (2013), http://dx.doi.org/10.1007/978-3-642-38610-7_16
14. Kordos, M., Blachnik, M., Wieczorek, T.: Temperature prediction in electric arc furnace with neural network tree. In: Honkela, T., Duch, W., Girolami, M., Kaski, S. (eds.) Artificial Neural Networks and Machine Learning – ICANN 2011, Lecture Notes in Computer Science, vol. 6792, pp. 71–78. Springer Berlin Heidelberg (2011), http://dx.doi.org/10.1007/978-3-642-21738-8_10
15. Kordos, M., Duch, W.: Variable step search algorithm for feedforward networks. Neurocomputing 71(13–15), 2470 – 2480 (2008), http://www.sciencedirect.com/science/article/pii/S0925231208002099

16. Kordos, M.: source code and datasets used in the experiments (2013), http://www.ath.bielsko.pl/~mkordos/tpnc2013.html
17. Liano, K.: Robust error measure for supervised neural network learning with outliers. Neural Networks, IEEE Transactions on 7(1), 246–250 (1996)
18. Merz, C., Murphy, P.: Uci repository of machine learning databases (2013), http://www.ics.uci.edu/mlearn/MLRepository.html
19. Olive, D.J., Hawkins, D.M.: Robustifying robust estimators (2007)
20. Pernďž˝a-Espinoza, A.V., Ordieres-Merďž˝, J.B., de Pisďž˝n, F.J.M., Gonzďž˝lez-Marcos, A.: Tao-robust backpropagation learning algorithm. Neural Networks 18(2), 191 – 204 (2005), http://www.sciencedirect.com/science/article/pii/S0893608004002345
21. Riedmiller, M., Braun, H.: A direct adaptive method for faster backpropagation learning: the rprop algorithm. In: Neural Networks, 1993., IEEE International Conference on. pp. 586–591 vol.1 (1993)
22. Rusiecki, A.: Fault tolerant feedforward neural network with median neuron input function. Electronics Letters 41(10), 603–605 (2005)
23. Rusiecki, A.: Robust lts backpropagation learning algorithm. In: Sandoval, F., Prieto, A., Cabestany, J., Grana, M. (eds.) Computational and Ambient Intelligence, Lecture Notes in Computer Science, vol. 4507, pp. 102–109. Springer Berlin Heidelberg (2007), http://dx.doi.org/10.1007/978-3-540-73007-1_13
24. Rusiecki, A.: Robust learning algorithm based on iterative least median of squares. Neural Processing Letters 36(2), 145–160 (2012), http://dx.doi.org/10.1007/s11063-012-9227-z
25. Salvador, G., Derrac, J., Ramon, C.: Prototype selection for nearest neighbor classification: Taxonomy and empirical study. IEEE Transactions on Pattern Analysis and Machine Intelligence 34, 417–435 (2012)
26. Tolvi, J.: Genetic algorithms for outlier detection and variable selection in linear regression models. Soft Computing 8, 527–533 (2004)
27. Wilson, D.L.: Asymptotic properties of nearest neighbor rules using edited data. Systems, Man and Cybernetics, IEEE Transactions on SMC-2(3), 408–421 (1972)
28. Zhang, J.: Intelligent selection of instances for prediction functions in lazy learning algorithms. Artifcial Intelligence Review 11, 175–191 (1997)