A First Study on the Use of Boosting for Class Noise Reparation

Pablo Morales Álvarez(✉), Julián Luengo, and Francisco Herrera

Department of Computer Science and Artificial Intelligence, CITIC-UGR, University of Granada, 18071 Granada, Spain
pablomorales@correo.ugr.es, {julianlm,herrera}@decsai.ugr.es

Abstract. Class noise refers to the incorrect labeling of examples in classification, and is known to negatively affect the performance of classifiers. In this contribution, we propose a boosting-based hybrid algorithm that combines data removal and data reparation to deal with noisy instances. A experimental procedure to compare its performance against no-preprocessing is developed and analyzed, laying the foundations for future works.

Keywords: Data reparation · Data filtering · Class noise · Boosting · Classification

1 Introduction

The main goal of Data Mining (DM) consists of extracting useful and valuable knowledge out of large amounts of raw data [6,15]. Evidently, the relevance and interest of this knowledge is strongly influenced by the quality of the used data, and this turns preprocessing into one of the most crucial and time-consuming steps in the DM process [14,22]. In this stage, a very common problem is the presence of *noise* in the dataset, i.e. corrupted data items not following the general distribution of the dataset, which can lead to excessively complex models with deteriorated performance [31]. We focus our interest on classification task, where two types of noise are distinguished: *class noise*, when it affects to the class label of the instances, and *attribute noise*, when it affects to the rest of attributes. The former is known to be the most disruptive one [24,30]. Consequently, many recent works, including this contribution, are devoted to overcome it or at least to minimize its effects (see [10] for a comprehensive and updated survey).

Among the approaches followed to address this problem, data reparation is one of most delicate, since incorrect relabeling may dramatically harm the borders between classes, leading to deteriorated accuracy for later classifiers [8,20]. This supposes a significant lack of literature and methods on the subject, which usually prevents researchers and practitioners from taking advantage of reparation benefits.

In this contribution, we propose using the well-known boosting mechanism [11,12] to introduce a hybrid algorithm that combines data removal and data

F. Martínez-Álvarez et al. (Eds.): HAIS 2016, LNAI 9648, pp. 549–559, 2016.
DOI: 10.1007/978-3-319-32034-2_46

reparation. The reason for the choice of boosting is twofold: its sensitivity to noise allows for detecting noisy instances through the values of their weights, and the reliability of the ensemble of classifiers built along the different iterations allows for evaluating the most likely class for the noisy instances and somehow quantify its confidence.

We will develop a experimental study to compare our proposal against "no preprocessing", considering 13 datasets and noise levels from 0 % to 35 % by increments of 5 %. To leave out exernal effects on the noise treatment, we will avoid using robust classifiers in the evaluation, taking 1-NN instead. Finally, we will include a Wilcoxon signed-rank test to assess the statistical evidence of enhancement.

The layout of the contribution is as follows: in Sect. 2 we set the context for our proposal, including previous related works (Sect. 2.1), and some relevant notes on boosting (Sect. 2.2). In Sect. 3, our novel hybrid algorithm is detailed and analyzed. Section 4 addresses the empirical evaluation, setting the experimental framework in Sect. 4.1 and explaining the results in Sect. 4.2. Some conclusions, comments, and future directions are included in Sect. 5.

2 Background and Related Work

In this section we set the context for our proposal. Specifically, Sect. 2.1 briefly reviews the main techniques for class noise preprocessing, focusing on reparation. Section 2.2 includes some appropriate comments on the use of boosting.

2.1 Tackling Class Noise Preprocessing

In the literature, there are two main ways to deal with class noise [30]: *algorithm level approaches* [7,23], which attempt to create robust classification algorithms that are little influenced by the presence of noise, and *data level approaches* [5,13], which try to develop strategies to cleanse the dataset as a previous step to the fit of the classifier. We will follow the second approach, since it allows to carry out the preprocessing just once and apply any classifier therafter, whereas the first treatment is specific for each classification algorithm.

In the chosen approach, the most classical procedure consists of detecting polluted instances and *removing* them from the dataset. Such algorithms are commonly called *filters*, and the detection step can be accomplished by means of different paradigms, such as ensemble-based techniques (e.g. Ensemble Filter (EF), see [5]), iterative procedures (e.g. Iterative Partitioning Filter (IPF), see [17]), or metric-based mechanisms (e.g. Saturation Filter (SF), see [13]).

However, there exist also other interesting ideas, like additionally allowing for data *reparation* (or *relabeling*) rather than only *removal* [3,18,21,25,33]. In fact, data is usually difficult to obtain, and getting rid of instances inexorably leads to loss of information, so it could be avoided when the real class can be guessed with

a considerable level of confidence. This damage for the learning process is even worse in specific frameworks such as imbalanced classification [28].

The discussion between data filtering, data reparation, and their possible synergy is a dynamic and challenging research area: several studies warn about the dangers of incorrectly relabeling instances [8,20], whereas many others claim the problems derived from removing too many instances from the dataset [18,27]. Hence, most proposals in the subject try to develop an appropriate trade-off to address the problem.

Among these algorithms that integrate data reparation, we can find different approaches: techniques based on votes from an ensemble of classifiers [3], methods based on noise measurements and thresholds [25], nearest neighbors-related techniques [2,18], approaches based on neighborhood graphs [19,21], and other particular mechanisms like in [32,33], where neural networks and decision trees are used respectively under a similar idea. As we will comment in Sect. 3, our algorithm somehow combines the two first approaches by means of *boosting*.

2.2 Elements of Boosting

Boosting [12] is a well-known ensemble mechanism to enhance the performance of a single classifier (called the *weak classifier*, although it is not necessarily a simple one) along several iterations. Briefly, it trains that same classifier in different rounds, focusing each time in those instances misclassified in the previous step. Once the ensemble is completed, it combines the predictions of all those weak classifiers, weighted by their training accuracy, to classify previously unobserved instances.

One of the greatest exponent of this technique is AdaBoost (Adaptive Boosting) algorithm [11], which distinguishes between binary classification task (AdaBoost.M1 algorithm) and general classification (AdaBoost.M2 algorithm, which includes the notion of "pseudo-loss"). Since the former is the most popular and easy-to-handle version, in this first study we will focus on binary classification, postponing the general framework with AdaBoost.M2 for future studies (see Sect. 5).

The described procedure of boosting makes it prone to overfit noisy instances [9], with weak classifiers persistently misclassifying them and thus getting high values for their weights. Interestingly, this can be used as an indicator to detect corrupted items. For instance, in [16] the authors present a classifier called ORBoost (Outlier Removal Boosting), a noise-robust adaptation of AdaBoost which removes in each iteration those instances whose weight exceeds an appropriate threshold. Likewise, in [29] the values of certain evaluation metrics associated to the boosting process, called *edge* and *margin*, are used to detect noisy instances.

Our proposal elaborates over these ideas, and integrates the possibility of reparation based on the predictions from the underlying ensemble of classifiers.

3 Noise Reparation Using AdaBoost

Before going into specific details, let us summarize the key ideas which define our cleansing algorithm:

1. We apply the scheme of AdaBoost.M1, with its particular weights update [11,12]. Recall we are dealing with binary classification in this first approach.
2. In each iteration, we detect as "potentially noisy" those instances whose weight is over a certain threshold.
3. Those instances are either removed, relabeled or kept unchanged depending on the predictions from the underlying ensemble of weak classifiers built until that round.

Notice that points 3 and 2 above correspond to the two first approaches explained in last paragraph of Sect. 2.1 as typical ways to address the problem of noise data reparation: on the one hand, techniques based on ensembles, and on the other hand those relying on noise measurements and thresholds.

Now, let us enumerate the specific parameters and elements that allow for tuning our proposal. Their default values for the experimentation will be provided in Sect. 4.1:

– *T:* number of iterations for the AdaBoost mechanism.
– *weakCL:* the weak classifier used in each round.
– *NoiseTH:* threshold for considering an instance as potentially noisy (recall point 2 above).
– *RepTH, KeepTH:* respectively, thresholds for repairing or keeping those potentially noisy items (recall point 3 above).

The process is sketched in Algorithm 1. It starts with an uniform distribution of weights, i.e. $\omega_1(i) = 1/m$ for all $i = 1, \ldots, m$, where m denotes the initial number of instances in the dataset. For each iteration $t = 1, \ldots, T$, the well-known process of AdaBoost.M1 is applied, obtaining a weak classifier h_t with an associated confidence $\beta_t \in (0, +\infty)$, and updating the weights to $\{\omega_{t+1}(i)\}_i$ (the range of i will progressively decrease as removal occurs). Then, we select those instances $i_1, \ldots, i_k$ whose weight exceeds the threshold *noiseTH*. In order to analyze them, we consider the corresponding predictions of the weak classifiers $h_1, \ldots, h_t$ built until that iteration, weighted by their confidence values $\beta_1, \ldots, \beta_t$. Normalizing, we have k pairs of values $(c_{11}, c_{12}), \ldots, (c_{k1}, c_{k2})$ in the interval $[0, 1]$, describing the confidence of each "suspicious" instance belonging to each one of the two classes. Now, for each $j = 1, \ldots, k$, we take the class c with highest confidence between c_{j1} and c_{j2}:

– If c is the actual class of the instance in that step, it is kept unchanged if the confidence is greater than *KeepTH*, and removed otherwise.
– If c is not the actual class of the instance in that step, it is relabeled if the confidence is greater than *RepTH*, and removed otherwise.

Before going into next iteration, the weights of relabeled instances are initialized to $1/m_t$, with m_t the number of instances in the dataset after iteration t, the weights of those analyzed but kept instances are given back its previous value (before exceeding the threshold), and the rest of weights are not modified. After proper normalization, the process gets into next round.

Algorithm 1. AdaBoost-based approach for filtering and repairing.

Input: Dataset D with m initial instances, and tuning parameters T, *weakCL*, *NoiseTH*, *RepTH*, *KeepTH*.
Initialize: Weights $\omega_1(i) = 1/m$ for all $i = 1, \ldots, m$.
for $t = 1, \ldots, T$ **do**
  Apply AdaBoost.M1 to obtain weak classifier h_t with confidence β_t.
  Update weights to ω_{t+1} following AdaBoost.M1 scheme.
  Select "potentially noisy instances": $i_1, \ldots, i_k$ such that $\omega_{t+1}(i_j) > NoiseTH$.
  for $j = 1, \ldots, k$ **do**
    Compute c_{j1} and c_{j2} confidences for i_j belonging to class 1 and 2 respectively, based on predictions from $h_1, \ldots, h_t$ weighted by $\beta_1, \ldots, \beta_t$.
    Put $c = \max(c_{j1}, c_{j2})$ and $\gamma \in \{1, 2\}$ the correspondent class.
    if class$(i_j) = \gamma$ **then**
      if $c > KeepTH$ **then**
        Keep instance i_j in D unchanged.
      else
        Remove instance i_j from D.
      end if
    end if
    if class$(i_j) \neq \gamma$ **then**
      if $c > RepTH$ **then**
        Relabel instance i_j in D with class γ.
      else
        Remove instance i_j from D.
      end if
    end if
  end for
  Weights for relabeled items are initialized to $1/m_t$, with m_t current size of D.
  Weights for analyzed but kept items are set back to ω_t.
  Weights are re-normalized.
end for
return D

4 Experiments

In this section we develop a experimental study to compare our preprocessing proposal against no preprocessing. Specific framework and parameters are fixed in Sect. 4.1, whereas results are shown and analyzed in Sect. 4.2.

4.1 Experimental Framework

In order to assess the performance of our proposal, we consider 13 datasets for binary classification from KEEL dataset repository[1] [1]. Table 1 summarizes the main features of these datasets.

Table 1. Datasets used in the experimental study. #EX denotes the number of examples or instances, and #AT the number of attributes (apart from the class).

Dataset	banana	german	heart	ionosphere	magic	monk	phoneme
#EX	5300	1000	270	351	19020	432	5404
#AT	2	20	13	33	10	6	5
Dataset	pima	ring	sonar	spambase	twonorm	wdbc	
#EX	768	7400	208	4597	7400	569	
#AT	8	20	60	57	20	30	

A stratified 5-folds cross validation scheme is used to test the accuracy of 1-NN classifier without preprocessing and after applying our preprocessing algorithm. Noise levels from 0 % to 35 %, by increments of 5 %, are introduced in the training sets following a *uniform class noise scheme* [26]: the corresponding percentage of examples are corrupted randomly replacing their class labels by other ones from the set of classes. Thus, using this scheme, both classes may be affected by the noise.

In total, two runs of the same scheme (for preprocessing and no preprocessing), with eight noise levels, five train-test pairs, and thirteen datasets, adds up to $2 \times 8 \times 5 \times 13 = 1040$ experiments.

Finally, before going into the results, let us specify the default values for our algorithm's tuning parameters:

1. $T = 50$ iterations.
2. CART decision tree [4] is taken as weak classifier, which is a classical choice for boosting.
3. The noise threshold $NoiseTH$ is set to depend on the size of the dataset, concretely $NoiseTH = 10/m$, where m is the initial number of instances.
4. Both $RepTH$ and $KeepTH$ are chosen so that "repairing" or "keeping" holds when the highest confidence is double the next one. In two-class classification, this implies $RepTH = KeepTH = 2/3$.

4.2 Analysis of Results

Complete results of the experiments are shown in Table 2. Since there are five train-test pairs for each dataset and noise level, the precision displayed is the mean value of test accuracy.

[1] http://keel.es/datasets.php.

Table 2. Classification accuracy for 1-NN with no preprocessing (NP) and with preprocessing (P). Results are given for each dataset and noise level considered, highlighting in each case in bold the best option between NP and P (in case of ties, both options are bolded). Since we are just displaying three decimal places, there are pairs of precision values which seem to be equal but actually differ (see for instance the dataset "sonar" at 20 % noise level).

	0 %		5 %		10 %		15 %	
	NP	P	NP	P	NP	P	NP	P
banana	0.872	**0.900**	0.855	**0.894**	0.833	**0.869**	**0.817**	**0.817**
german	0.683	**0.735**	0.668	**0.715**	0.659	**0.690**	0.654	**0.679**
heart	0.744	**0.807**	0.730	**0.804**	0.722	**0.796**	0.711	**0.770**
ionosphere	0.855	**0.858**	**0.843**	0.840	0.803	**0.846**	0.786	**0.832**
magic	0.816	**0.842**	0.800	**0.803**	0.784	**0.785**	**0.767**	**0.767**
monk	0.734	**0.752**	0.724	**0.738**	0.724	**0.761**	0.704	**0.771**
phoneme	**0.899**	0.863	0.880	**0.886**	0.863	**0.867**	**0.842**	**0.842**
pima	0.692	**0.763**	0.671	**0.745**	0.672	**0.751**	0.656	**0.691**
ring	0.748	**0.760**	0.733	**0.748**	0.729	**0.749**	0.710	**0.752**
sonar	**0.846**	0.836	0.832	**0.856**	0.827	**0.841**	0.822	**0.827**
spambase	**0.909**	0.906	0.889	**0.908**	0.871	**0.908**	0.849	**0.904**
twonorm	0.949	**0.966**	0.928	**0.965**	0.909	**0.967**	0.879	**0.965**
wdbc	0.946	**0.949**	0.919	**0.951**	0.896	**0.956**	0.888	**0.958**
	20 %		25 %		30 %		35 %	
	NP	P	NP	P	NP	P	NP	P
banana	**0.798**	**0.798**	**0.765**	**0.765**	**0.763**	**0.763**	**0.743**	**0.743**
german	0.657	**0.670**	0.626	**0.634**	0.616	**0.618**	0.630	**0.638**
heart	0.641	**0.722**	0.674	**0.733**	0.681	**0.726**	0.689	**0.719**
ionosphere	0.775	**0.838**	0.761	**0.852**	0.761	**0.843**	0.758	**0.852**
magic	**0.751**	**0.751**	**0.741**	**0.741**	**0.721**	**0.721**	**0.704**	**0.704**
monk	0.699	**0.769**	0.697	**0.722**	0.657	**0.674**	0.641	**0.669**
phoneme	**0.817**	**0.817**	**0.804**	**0.804**	**0.773**	**0.773**	**0.761**	**0.761**
pima	0.643	**0.664**	0.664	**0.681**	0.650	**0.663**	**0.609**	0.606
ring	0.698	**0.733**	0.687	**0.687**	**0.672**	**0.672**	**0.670**	**0.670**
sonar	0.769	**0.769**	0.779	**0.803**	0.692	**0.712**	0.803	**0.812**
spambase	0.831	**0.833**	**0.811**	**0.811**	**0.788**	**0.788**	**0.782**	**0.782**
twonorm	0.859	**0.965**	0.831	**0.835**	0.817	**0.817**	**0.787**	**0.787**
wdbc	0.828	**0.953**	0.822	**0.946**	0.840	**0.944**	0.812	**0.954**

Table 3. Explicit comparison between preprocessing and no preprocessing for classification accuracy with 1-NN. A symbol '+' denotes the proposed algorithm outperforms the no preprocessing. Symbols '=' and '−' indicate equality or deterioration respectively.

	0	5	10	15	20	25	30	35
banana	+	+	+	=	=	=	=	=
german	+	+	+	+	+	+	+	+
heart	+	+	+	+	+	+	+	+
ionosphere	+	−	+	+	+	+	+	+
magic	+	+	+	=	=	=	=	=
monk	+	+	+	+	+	+	+	+
phoneme	−	+	+	=	=	=	=	=
pima	+	+	+	+	+	+	+	−
ring	+	+	+	+	+	+	=	=
sonar	−	+	+	+	+	+	+	+
spambase	−	+	+	+	+	=	=	=
twonorm	+	+	+	+	+	+	+	=
wdbc	+	+	+	+	+	+	+	+
+	10	12	13	10	10	9	8	6
= +	10	12	13	13	13	13	13	12

Table 3 summarizes the direct comparison between preprocessing and no-preprocessing in a more illustrative and clear fashion. Briefly, we can conclude that our method allows for accurate preprocessing while noise does not remove too much information and patterns from the dataset, when it passes to exhibit an appropriately conservative behavior. Indeed, penultimate row reveals that our algorithm (strictly) improves accuracy at low noise levels (even at 0 % level, when no artificial noise is introduced), and that this enhancement progressively decreases as noise grows. Moreover, last row shows that our algorithm avoids incorrect reparation or unsure filtering for those higher noise levels, where precision does not deteriorate with respect to no-preprocessing.

In fact, this behavior is a natural consequence of the internal mechanism of our algorithm: as noise level grows, the precision of the weak classifier used in the boosting scheme decreases, leading to high error rates ϵ_t at each iteration. When these values get close to random guessing (i.e. $\epsilon_t \approx 0.5$), the AdaBoost weights updating scheme becomes trivial, leading to a very similar weight distribution for next round. Hence, the boosting process gets stalled, and the proposed preprocessing has no effect. This will be further discussed in Sect. 5.

To check the statistical evidence of enhancement, Table 4 shows the results of a Wilcoxon signed-rank test for each noise level, with the alternative hypothesis that our method improves the precision accuracy. Low p-values let us refuse

Table 4. Results from Wilcoxon signed-rank test applied to compare our preprocessing proposal with no preprocessing, obtained for each noise level. R^+ and R^- denote, respectively, the sum of ranks for preprocessing and no preprocessing.

	0 %	5 %	10 %	15 %	20 %	25 %	30 %	35 %
R^+	75	90	91	55	55	45	36	27
R^-	16	1	0	0	0	0	0	1
p-value	0.0199	0.0002	0.0001	0.003	0.003	0.0046	0.0071	0.0173

the null hypothesis at all noise levels, even at very low significance levels for moderate amounts of noise.

5 Conclusions and Future Work

In this first study on the use of boosting for class noise reparation, we motivate and suggest a novel cleansing algorithm, which is then shown to outperform no preprocessing in a experimental study. Results displayed in Tables 2, 3 and 4 are promising and lay the foundations for deeper works. These extensions may address several interesting aspects:

- General classification problem, with more than two classes. In principle, the same algorithm based on AdaBoost.M1 can be applied with minor modifications in the tuning parameters. However, AdaBoost.M2 is a specific adaptation developed by the same authors for general classification [11,12], and could lead to better performance of the preprocessing mechanism too.
- Dealing with high noise levels. We explained in last Sect. 4.2 how our algorithm gets stalled in presence of considerable noise levels, producing no effect. This can be regarded as a safe mechanism to avoid filtering or repairing when there is not much certainty about it. However, a bit more flexible behaviors could be explored so that, at these noise levels, preprocessing also improves, at least slightly, the lack of it.
- Experimental studies can be extended at least in two directions: a wider range of classifiers (including robust ones) could be used to assess the performance of our preprocessing approach, and direct comparison against most popular cleansing algorithms might be tackled.

All these extensions can motivate, or even require, a deep and challenging comparison between the two approaches we are dealing with: data filtering and data reparation.

Acknowledgments. This work was supported by the National Research Project TIN2014-57251-P and Andalusian Research Plans P10-TIC-6858 and P11-TIC-7765. P.M. Álvarez also holds ICARO contract 135849 from University of Granada.

References

1. Alcalá, J., Fernández, A., Luengo, J., Derrac, J., García, S., Sánchez, L., Herrera, F.: Keel data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework. J. Multiple-Valued Logic Soft Comput. **17**(2–3), 255–287 (2010)
2. Barandela, R., Gasca, E.: Decontamination of training samples for supervised pattern recognition methods. In: Amin, A., Pudil, P., Ferri, F., Iñesta, J.M. (eds.) SPR 2000 and SSPR 2000. LNCS, vol. 1876, pp. 621–630. Springer, Heidelberg (2000)
3. Barandela, R., Valdovinos, R.M., Sánchez, J.S.: New applications of ensembles of classifiers. Pattern Anal. Appl. **6**(3), 245–256 (2003)
4. Breiman, L., Friedman, J., Stone, C.J., Olshen, R.A.: Classification and Regression Trees. CRC Press, Boca Raton (1984)
5. Brodley, C.E., Friedl, M.A.: Identifying mislabeled training data. J. Artif. Intell. Res. **11**, 131–167 (1999)
6. Cherkassky, V., Mulier, F.M.: Learning from Data: Concepts, Theory, and Methods. John Wiley & Sons, New York (2007)
7. Cohen, W.W.: Fast effective rule induction. In: Proceedings of the Twelfth International Conference on Machine Learning, pp. 115–123, July 1995
8. Cuendet, S., Hakkani-Tür, D., Shriberg, E.: Automatic labeling inconsistencies detection and correction for sentence unit segmentation in conversational speech. In: Popescu-Belis, A., Renals, S., Bourlard, H. (eds.) MLMI 2007. LNCS, vol. 4892, pp. 144–155. Springer, Heidelberg (2008)
9. Dietterich, T.G.: An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. Mach. learn. **40**(2), 139–157 (2000)
10. Frénay, B., Verleysen, M.: Classification in the presence of label noise: A survey. IEEE Trans. Neural Netw. Learn. Syst. **25**(5), 845–869 (2014)
11. Freund, Y., Schapire, R.E.: A decision-theoretic generalization of on-line learning and an application to boosting. J. Comput. Syst. Sci. **55**(1), 119–139 (1997)
12. Freund, Y., Schapire, R.E.: Boosting: Foundations and algorithms. MIT press, Cambridge (2012)
13. Gamberger, D., Lavrac, N., Groselj, C.: Experiments with noise filtering in a medical domain. In: ICML, pp. 143–151, June 1999
14. García, S., Luengo, J., Herrera, F.: Data Preprocessing in Data Mining. Springer, New York (2015)
15. Hastie, T., Tibshirani, R., Friedman, J., Franklin, J.: The elements of statistical learning: Data mining, inference and prediction. Math. Intel. **27**(2), 83–85 (2005)
16. Karmaker, A., Kwek, S.: A boosting approach to remove class label noise. In: Fifth International Conference on Hybrid Intelligent Systems, 2005, HIS 2005. p. 6. IEEE, November 2005
17. Khoshgoftaar, T.M., Rebours, P.: Improving software quality prediction by noise filtering techniques. J. Comput. Sci. Technol. **22**(3), 387–396 (2007)
18. Koplowitz, J., Brown, T.A.: On the relation of performance to editing in nearest neighbor rules. Pattern Recogn. **13**(3), 251–255 (1981)
19. Lallich, S., Muhlenbach, F., Zighed, D.A.: Improving classification by removing or relabeling mislabeled instances. In: Hacid, M.-S., Raś, Z.W., Zighed, D.A., Kodratoff, Y. (eds.) ISMIS 2002. LNCS (LNAI), vol. 2366, pp. 5–15. Springer, Heidelberg (2002)

20. Miranda, A.L.B., Garcia, L.P.F., Carvalho, A.C.P.L.F., Lorena, A.C.: Use of classification algorithms in noise detection and elimination. In: Corchado, E., Wu, X., Oja, E., Herrero, A., Baruque, B. (eds.) HAIS 2009. LNCS, vol. 5572, pp. 417–424. Springer, Heidelberg (2009)
21. Muhlenbach, F., Lallich, S., Zighed, D.A.: Identifying and handling mislabelled instances. J. Intell. Inf. Syst. **22**(1), 89–109 (2004)
22. Pyle, D.: Data Preparation for Data Mining, vol. 1. Morgan Kaufmann, San Francisco (1999)
23. Quinlan, J.R.: C4.5: Programs for Machine Learning. Elsevier, Amsterdam (2014)
24. Sáez, J.A., Galar, M., Luengo, J., Herrera, F.: Analyzing the presence of noise in multi-class problems: Alleviating its influence with the one-vs-one decomposition. Knowl. Inf. Syst. **38**(1), 179–206 (2014)
25. Sun, J.W., Zhao, F.Y., Wang, C.J., Chen, S.F.: Identifying and correcting mislabeled training instances. In: Future Generation Communication and Networking (FGCN 2007), vol. 1, pp. 244–250. IEEE, December 2007
26. Teng, C.M.: Correcting noisy data. In: ICML, pp. 239–248, June 1999
27. Teng, C.M.: Dealing with data corruption in remote sensing. In: Famili, A.F., Kok, J.N., Peña, J.M., Siebes, A., Feelders, A. (eds.) IDA 2005. LNCS, vol. 3646, pp. 452–463. Springer, Heidelberg (2005)
28. Van Hulse, J., Khoshgoftaar, T.: Knowledge discovery from imbalanced and noisy data. Data Knowl. Eng. **68**(12), 1513–1542 (2009)
29. Wheway, V.: Using boosting to detect noisy data. In: Kowalczyk, R., Loke, S.W., Reed, N.E., Graham, G. (eds.) PRICAI-WS 2000. LNCS (LNAI), vol. 2112, pp. 123–130. Springer, Heidelberg (2001)
30. Wu, X., Zhu, X.: Class noise vs. attribute noise: A quantitative study. Artif. Intell. Rev. **22**(3), 177–210 (2004)
31. Wu, X., Zhu, X.: Mining with noise knowledge: Error-aware data mining. IEEE Trans. Syst. Man Cybern. Part A: Syst. Hum. **38**(4), 917–932 (2008)
32. Zeng, X., Martinez, T.R.: An algorithm for correcting mislabeled data. Intel. Data Anal. **5**(6), 491–502 (2001)
33. Zeng, X., Martinez, T.R.: Using decision trees and soft labeling to filter mislabeled data. J. Intell. Syst. **17**(4), 331–354 (2008)