Information Fusion 27 (2016) 19–32

Contents lists available at ScienceDirect

Information Fusion

journal homepage: www.elsevier.com/locate/inffus

INFFC: An iterative class noise filter based on the fusion of classifiers with noise sensitivity control

José A. Sáez [a,*], Mikel Galar [c], Julián Luengo [d], Francisco Herrera [b]

[a] ENGINE Centre, Wrocław University of Technology, Wybrzeże Wyspiańskiego 27, 50-370 Wrocław, Poland
[b] Department of Computer Science and Artificial Intelligence, University of Granada, CITIC-UGR, Granada 18071, Spain
[c] Department of Automática y Computación, Universidad Pública de Navarra, Pamplona 31006, Spain
[d] Department of Civil Engineering, LSI, University of Burgos, Burgos 09006, Spain

A R T I C L E I N F O

Article history:
Received 10 September 2014
Received in revised form 18 February 2015
Accepted 28 April 2015
Available online 12 May 2015

Keywords:
Fusion of classifiers
Noisy data
Class noise
Noise filters
Classification

A B S T R A C T

In classification, noise may deteriorate the system performance and increase the complexity of the models built. In order to mitigate its consequences, several approaches have been proposed in the literature. Among them, noise filtering, which removes noisy examples from the training data, is one of the most used techniques. This paper proposes a new noise filtering method that combines several filtering strategies in order to increase the accuracy of the classification algorithms used after the filtering process. The filtering is based on the fusion of the predictions of several classifiers used to detect the presence of noise. We translate the idea behind multiple classifier systems, where the information gathered from different models is combined, to noise filtering. In this way, we consider the combination of classifiers instead of using only one to detect noise. Additionally, the proposed method follows an iterative noise filtering scheme that allows us to avoid the usage of detected noisy examples in each new iteration of the filtering process. Finally, we introduce a noisy score to control the filtering sensitivity, in such a way that the amount of noisy examples removed in each iteration can be adapted to the necessities of the practitioner. The first two strategies (use of multiple classifiers and iterative filtering) are used to improve the filtering accuracy, whereas the last one (the noisy score) controls the level of conservation of the filter removing potentially noisy examples. The validity of the proposed method is studied in an exhaustive experimental study. We compare the new filtering method against several state-of-the-art methods to deal with datasets with class noise and study their efficacy in three classifiers with different sensitivity to noise.

1. Introduction

Data collection and preparation processes are commonly subjected to errors in Data Mining applications [1,2]. For this reason, real-world datasets usually contain imperfections or *noise* [3–5]. In classification, a model is built from labeled examples, which should be capable of reliably predicting the class for new previously unobserved examples. Obviously, if the data used to train this model (formally known as a classifier) are corrupted, both the learning phase and the model obtained will be negatively affected. The former will require more time to find a solution but also more examples in order to be able to obtain an accurate classifier. As a consequence, the final model will probably be less accurate due to the presence of noise, and it will be more complex, since non-real patterns may be modeled.

Two different types of noise can be found in classification datasets: attribute and class noise [3]. Class noise is the most disruptive type of noise since incorrectly labeled examples have a high impact when building classifiers, whose performance is often reduced [3,6]. On this account, many works in the literature, including this paper, focus on its treatment [7–10]. Among these works, two types of approaches have been proposed to deal with class noise [3]:

1. *Algorithm level approaches* [11,12]. The methods in this category comprise the adaptations of existing algorithms to properly handle the noise or being less influenced by its presence.
2. *Data level approaches* [7,8]. These methods consist of preprocessing the datasets aiming at getting rid of the noisy examples as a previous step.

* Corresponding author.
E-mail addresses: jose.saezmunoz@pwr.edu.pl (J.A. Sáez), mikel.galar@unavarra.es (M. Galar), jluengo@ubu.es (J. Luengo), herrera@decsai.ugr.es (F. Herrera).

http://dx.doi.org/10.1016/j.inffus.2015.04.002
1566-2535/© 2015 Elsevier B.V. All rights reserved.

Algorithm level approaches are not often an available choice since they depend on the particular adaptation of each classification algorithm, and therefore they are not directly extensible to other learning algorithms. Otherwise, data level approaches are independent of the classifier used and allow one to preprocess the datasets beforehand in order to use them to train different classifiers (hence, the computation time needed to prepare the data is only required once). Thus, the latter type of techniques is usually the most popular choice.

Among data level approaches, noise filters, which remove noisy examples from the training data, are widely used due to their benefits in the learning in terms of classification accuracy and complexity reduction of the models built [7,13]. The study of the noise filtering schemes that are proposed in the literature [7,14–16] focuses our attention on three main paradigms:

- *Ensemble-based filtering* [7]. There are studies where some authors proposed the usage of ensembles for filtering. The main advantage of these approaches is based on the hypothesis that collecting predictions from different classifiers could provide a better class noise detection than collecting information from a single classifier.
- *Iterative filtering* [17]. The strength of these types of filters is the usage of an iterative elimination of noisy examples under the idea that the examples removed in one iteration do not influence the noise detection in subsequent ones.
- *Metric-based filtering* [14,9]. These noise filters are based on the computation of measures over the training data and usually allow the practitioner to control the level of conservation of the filtering in such a way that only examples whose estimated noise level exceed a prefixed threshold are removed.

On this account, this paper proposes a novel noise filtering technique combining these three noise filtering paradigms: the usage of ensembles for filtering, the iterative filtering and the computation of noise measures. Therefore, the proposal of this paper removes noisy examples in multiple iterations considering filtering techniques that employ systems based on the *Fusion of Classifiers* (FC), also known as *Multiple Classifier Systems* (MCSs) [18–21]. This type of systems have already shown a good behavior with noisy data in the field of classification [22,6]. Ensemble-based filters have previously considered the usage of FC to remove noisy instances [7,23]. For example, Barandela et al. [23] proposed a filtering method that removes or re-labels mislabeled examples depending on the degree of agreement among the predictions of several classifiers built over the training data. A different way to design an ensemble-based filter is that proposed by Sánchez and Kuncheva [24]. They applied a well-known filter over several subsets of the training data, and the resulting filtered data from each subset were combined into only one set using different techniques. The main differences between our proposal and previous ensemble-based filters are: (a) the iterative elimination of noisy examples, and (b) our filtering proposal also uses a noise sensitivity control in order to determine which potentially noisy examples are finally eliminated in each iteration. In this way, we take advantage of the three different aforementioned paradigms. The proposed method is called *Iterative Noise Filter based on the Fusion of Classifiers* (INFFC).

A thorough empirical study will be developed comparing several representative noise filters with our proposal. All of them will be used to preprocess 25 real-world datasets in which different class noise levels will be introduced (from 5% to 30%, by increments of 5%). The filtered datasets will be then used to create classifiers with three learning methods of a different and well-known behavior against noise: a learner considered robust to noise as C4.5 [11] is, a *Support Vector Machine* (SVM) [25], considered accurate but being noise sensitive and the Nearest Neighbor (NN) rule [26] which is also considered very noise sensitive. Their test accuracy over the datasets preprocessed with our proposal and the other existing filters will be compared using the appropriate statistical tests [27] in order to check the significance of the differences found. Full results and details of the experimentations are available in the webpage associated to this paper at http://sci2s.ugr.es/INFFC.

The rest of this paper is organized as follows. Section 2 presents an introduction to classification with noisy data. In Section 3, we introduce the details of the noise filter proposed. In Section 4, we describe the experimental framework, whereas in Section 5 we analyze the results obtained. Finally, in Section 6 we enumerate some concluding remarks.

2. Classification with noisy data

This section first introduces the problem of noisy data in the classification framework in Section 2.1. Next, previous works on noise filters are briefly reviewed in Section 2.2, paying special attention to those filters on which our proposal is based.

2.1. Noise in classification problems

Noise may be present as errors in the source and input of the data affecting the quality of the dataset [28]. In classification, this quality is mainly influenced by two information sources, that is, the class labeling and the attributes' value sampling. Based on these two information sources, two types of noise are traditionally distinguished in the literature [3]: attribute noise and class noise.

Attribute noise affects the values of one or more attributes of examples in a dataset. It can proceed from several sources such as transmission constraints, faults in sensor devices and transcription errors [29]. *Class noise* (or labeling errors) is produced when the examples are labeled with the wrong classes. Note that this definition of noise differs from that of outlier. An outlier is an example of a concrete class L_A which appears to be inconsistent with respect to other examples of the same class, since it is situated within a different class [30]. However, different from noisy examples, this one is not corrupted by errors in its class label or attribute values, being L_A its correct label.

Class noise is known to be more harmful than attribute noise to classifier performance mainly due to the fact that whereas the importance of each feature for learning may be different, labels always have a large impact on it [3,22,6]. For this reason, this paper focuses on class noise, aiming at removing those wrongly labeled examples from the datasets.

Class noise can be attributed to several causes [31]. One of them is the inadequacy of the information used to label each example, for example, when a amnesic patient imprecisely answers the questions of the doctor [32]. Data entry errors and the subjectivity during the labeling process also can produce class noise; for example, in medical applications a variability in the labeling by several experts may exist [33].

Several works of the literature are focused on analyzing the impact of class noise in the learning of classifiers from data with class noise [34,7,6]. For example, in [34], the authors investigated the behavior of the nearest neighbor classifier when mislabeled data are considered. Among the effects of class noise in the system performance, the most frequently reported consequence is the decrement of classification accuracy [6]. Class noise can also affect the complexity of the classifier built in terms of size and interpretability (for example, in [7], the increase of the size of decision trees when learning from data affected by class noise was shown).

Errors in real-world datasets are therefore common and techniques that eliminate noise or reduce its impact are need [3]. Two main alternatives have been proposed in the literature to deal with noisy data:

- *Algorithm level approaches*. Also known as *robust learners*, these are techniques characterized by being less influenced by noisy data. Examples of a robust learner are C4.5 [11] or RIPPER [12]. These classifiers have been adapted to properly handle the noise. Thus, for example, C4.5 uses pruning strategies to reduce the chances that the trees are overfitting due to noise in the training data [35].
- *Data level approaches*. The most well-known type of methods within this group is that of *noise filters* [7,17]. They identify noisy instances which can be eliminated from the training data. These methods are used with many learners that are sensitive to noisy data and require data preprocessing to address the problem, even though robust learners also benefit from their usage. The separation of noise detection and learning phase has the advantage of avoiding the usage of noisy instances in the classifier building process [14]. Our proposal is included into this group of methods.

2.2. Noise filters

Preprocessing the dataset aiming to clean the noisy examples is one of the most common approaches when training data are affected by noise [7]. Noise filters are designed to eliminate noisy examples in the training set, which is then used as an input to classifiers [7,17,5]. They are particularly oriented to remove class noise, since the elimination of such examples has shown to be advantageous [8]. However, the elimination of instances with attribute noise seems to be counterproductive [35,3], since they still contain valuable information in other attributes which can help to build a more accurate classifier.

Even though several noise filtering schemes are proposed in the literature [7,14–16], the following three groups are based on some interesting properties as we have already commented in Section 1: *ensemble-based filtering* [7], *iterative filtering* [17] and *metric-based filtering* [14]. Even though metric-based filtering methods has the advantage of controlling the level of conservation of the filtering, they are generally simple approaches and do not perform as well as other more advanced types of noise filters in many cases. For this reason, many other approaches have been proposed in the literature [9,10,8,7].

There are filtering methods that are based on the fact that the k-NN classifier [26] is sensitive to noisy data [9,10], particularly when k is low [36]. Other types of noise filters use classifiers to detect noisy examples, which are those that are misclassified. The *Classification Filter* (CF) [8] performs a partitioning of the training set into n subsets, then a set of classifiers is trained from the union of any $n - 1$ subsets; those classifiers are afterwards used to classify the examples in the excluded subset, eliminating the incorrectly classified examples. This filter has the risk of removing too many instances due to the usage of a single classifier. In order to solve this problem, ensembles of classifiers are used to identify mislabeled instances; the proposals of [7,17] are two of the most representative and well known methods within this field (described hereafter):

- The *Ensemble Filter* (EF) [7] uses a set of three learning algorithms (C4.5 [11], 1-NN [26] and LDA [37]) to remove potentially noisy instances. The training data is classified using a n-fold cross-validation with each classification algorithm and the noisy examples are identified using a voting scheme. Two voting schemes are proposed: consensus (which removes an example if it is misclassified by all the classifiers) and majority (which removes an example if it is misclassified by more than a half of the classifiers).
- The *Iterative-Partitioning Filter* (IPF) [17] proposes a similar technique, but removes the noisy data iteratively using several classifiers built with the same learning algorithm. IPF removes noisy examples in multiple iterations until the quantity of examples eliminated is under a threshold. In each iteration, the current training dataset is split into n equal sized subsets and the C4.5 classifier is built over each of these n subsets to evaluate the whole training set. Then, the incorrectly labeled examples are removed from it (according to one of the two aforementioned voting schemes) and a new iteration is started.

Both methods, EF and IPF, claimed two important postulates in the field of the filtering techniques. On the one hand, Brodley and Friedl (EF) [7] stated that, since noisy data are independent of the particular classifier built, using the predictions of different classifiers could imply an improvement in noise detection against considering an specific classifier. On the other hand, Khoshgoftaar and Rebours (IPF) [17] claimed that an iterative elimination of noise usually results in a more accurate noise filtering.

However, the methods belonging to these approaches also have some drawbacks. Since EF, which is an ensemble-based filter, does not follow an iterative elimination of noisy examples, the classifiers are built from wrong data, and therefore their noise detection may be biased. Otherwise, the IPF iterative noise filter only considers one classification algorithm to build the filter, and thus it does not benefit from collecting information from different models built with different classification algorithms.

In this work, we aim to follow these postulates at the same time, combining both types of mechanisms to develop a new filtering method. In addition, we also take into account different measures of noise in each potentially noise example in order to decide whether it should be removed or not. In this way, we take advantage of the strengths of each type of filtering method to create a new noise filter. The complete proposal is explained in the next section.

3. INFFC: Iterative Noise Filter based on the Fusion of Classifiers

Inspired by the ideas that motivated the design of the EF and IPF filters, the proposal of this paper removes noisy examples in multiple iterations considering filtering techniques based on the usage of multiple classifiers [18,22,38]. For this reason, we have called our proposal *Iterative Noise Filter based on the Fusion of Classifiers*. Fig. 1 shows an scheme of the filtering method proposed. Three steps are carried out in each iteration. First, a preliminary filtering is performed with a FC-based filter. Then, another FC-based filter is built from the examples that are not identified as noisy in the preliminary filtering to detect the noisy examples in the full set of instances in the current iteration. Finally, in order to control the noise sensitivity of the filter, noisy examples are only removed if they exceed a noise score metric.

The building of the FC-based filter is described in Section 3.1. Then, the three main steps carried out at each iteration are described in separate sections:

1. **Preliminary filtering** (Section 3.2). This first step removes a part of the existing noise in the current iteration to reduce its influence in posterior steps. More specifically, noise examples identified with high confidence are expected to be removed in this step.

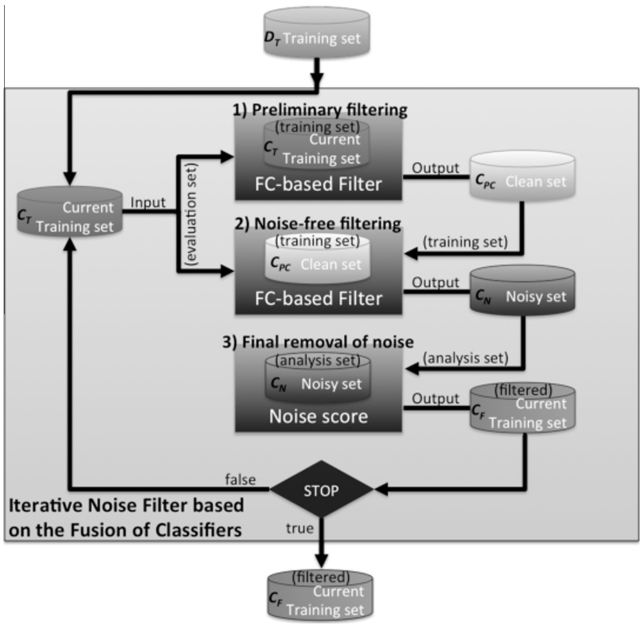


Fig. 1. Iterative noise filter based on the fusion of classifiers.

2. **Noise-free filtering** (Section 3.3). A new filtering, which is built from the partially clean data from the previous step, is applied over all the training examples in the current iteration resulting into two sets of examples: a clean and a noisy set. This filtering is expected to be more accurate than the previous one since the noise filters are built from cleaner data.
3. **Final removal of noise** (Section 3.4). A noise score is computed over each potentially noisy example from the noisy set obtained in the previous step in order to determine which ones are finally eliminated.

Finally, Section 3.5 is devoted to the comparison of the time complexity of INFFC with respect to those of other ensemble-based filters, such as EF or IPF.

We introduce each one of the three aforementioned steps (Sections 3.2,3.3,3.4) in each iteration aiming at removing noisy examples with maximum reliability, that is, those which are noisy examples with great confidence. In this way, examples that may be noisy or not, are left in the training set for posterior processing. Ensuring that only the examples that are most likely to be noise are removed implies that it will be less probable to delete noise-free examples, which would harm the learning process.

The iterative process stops when, for a number of consecutive iterations g, the number of identified noisy examples in each of these iterations g is less than a percentage p of the size of the original training dataset (in this paper, these parameters are set to $g = 3$ iterations and $p = 1\%$).

Table 1
Sets of examples used in the proposed noise filter.

Set	Description
D_T	Initial training set
C_T	Training set at the start of the iteration
C_{PC} and C_{PN}	Sets of clean and noisy examples from C_T provided by the preliminary filtering
C_C and C_N	Sets of clean and noisy examples from C_T provided by the noise-free filtering
C_F	Filtered C_T

Table 1 describes the notation used in the following sections to describe the different sets of examples considered in our filter.

3.1. Noise filter based on the fusion of classifiers

The filtering technique used in the steps 2 and 3 in each iteration of the method proposed in this paper is based on the combination of different classifiers. This filtering strategy was previously used in the EF noise filter [7]. However, there are two main differences of our filtering technique with respect to EF:

1. *Noise evaluation strategy*. The authors of EF proposed the usage of a k-fold cross-validation to label each example as correct or noisy by each classifier, whereas in our proposal all the training examples are considered to create only one model with each classifier to label the examples. Thus, the time complexity of the noise evaluation strategy of INFFC is reduced with respect to that of EF.
2. *Classifiers used to build the filter*. The authors of EF propose to use C4.5 [11], 1-NN [26] and LDA [37], whereas we propose to change 1-NN by 3-NN and LDA by Logistic regression (LOG) [37], respectively. Therefore, the classifiers used by our FC-based filter are C4.5, 3-NN and LOG. The aforementioned changes are mainly motivated by the noise evaluation strategy used. Since our FC-based filter only builds one model to label each example as clean or noisy, it needs to include classifiers that behave better with noisy data than the very noise-sensitive 1-NN and LDA methods used in EF. Because of this, we increment the k value of the k-NN method, increasing its robustness against noise [36]. On the other hand, LOG is an statistical classifier, such as LDA, but is recognized to behave better in several domains. In this way, we improve the global behavior of the filter when detecting noisy examples since 3-NN and LOG could be considered better than LDA and 1-NN when dealing with noisy data.

After obtaining the prediction of each one of the aforementioned classifiers over the set of examples to evaluate, those examples incorrectly classified by the majority of the classifiers, that is, 2 of the 3 classifiers, are labeled as noisy. It is important to note that we can also use the consensus scheme to determine which examples are noisy, but in this paper we will use the majority scheme in all the ensemble-based filters, making the results comparable. Furthermore, remember that in our proposal we use the same data to train each classifier and evaluate them to determine the examples to be tagged as noisy.

3.2. Preliminary filtering

Data that we want to filter C_T (note that $C_T = D_T$ at the first iteration) are, logically, likely to contain noisy examples. Therefore, filtering based on these noisy data may be misleading since the filtering models built are affected by the noisy examples. Thus, these data are not reliable enough to decide the final elimination of the noisy examples. On this account, we first perform a preliminary filtering of the dataset C_T at each iteration in order to remove most of the potentially noisy examples. Afterwards, we consider a second filtering step, the *noise-free filtering*, in which the filter is trained only with the examples considered as noise-free (C_{PC}) at this stage, and thus its noise identification is expected to be more reliable.

The filtering consists of the creation of a system based on FC considering the three aforementioned classifiers (C4.5, 3-NN and LOG) from C_T (the training set at the start of the iteration). This FC-based filter is used to evaluate the examples of the same set

C_T. The noisy examples C_{PN} identified by the filter are removed from C_T, resulting in the training data C_{PC}.

Please, note that the noise identification of this first filtering is based on data which may contain noise. Thus, the noise identified in this step may be erroneous and noisy data will be probably considered as clean data and vice versa; this is the reason why we call this step *preliminary filtering*.

3.3. Noise-free filtering

The filtered dataset provided by the preliminary filtering (C_{PC}) is a cleaner version than the training set at the start of the iteration C_T. Therefore, the FC-based filter built from these cleaner data C_{PC} is expected to perform a more accurate identification of the noise in C_T, since the models built for filtering are not affected by the most disruptive noisy examples previously detected and removed from C_T.

Therefore, in the second step of each iteration, a new FC-based filter is created considering C_{PC}, that is, C_T after the preliminary filtering (without the noisy examples identified in the first step). This filter is evaluated on the examples of the whole set C_T (all the training examples in the current iteration). This results in two different sets of examples: $C_C \subseteq C_T$, which consists of those examples considered as clean by the filter and $C_N \subseteq C_T$, which is the set of examples considered as potentially noisy (note that $C_C \cap C_N = \emptyset$ and $C_C \cup C_N = C_T$).

Note that the noise identification carried out by the FC-based filter in this step is based on the filtered C_T, so it is expected to be more accurate in identifying noise that the filter in the previous step; this is the reason why we call this step *noise-free filtering*.

3.4. Final removal of noise: the noise score

This last step controls the noise sensitivity of the filter moderating the amount of noisy examples removed. By means of this last step, we try to ensure that only true noisy examples are removed. Hence, questionable examples are analyzed in posterior iterations, since the elimination of non-noisy examples may carry a significant decrease in accuracy and therefore it is important to be sure that they are truly noise.

The noisy examples identified in the second step of the iteration C_N are those considered to be analyzed with the noise score. They are ordered according to this noise score, from those which are more probably noise to those which are less probably noise or that may be indeed clean examples wrongly identified as noisy by the filter. Finally, the examples that exceed a threshold set by the user are eliminated (the effects of using different values of the threshold on the filtering will be studied in Section 5.6). In order to define the noise score, we have made the following observations:

1. *The class label of some training examples may be erroneous.*
   Any dataset is susceptible of containing noise [3]. Since our proposal is particularly designed to deal with datasets with class noise, one cannot blindly trust on the class of all the examples.
2. *Noisy examples detected by any filter may be incorrect.*
   From the above premise, decisions obtained from noisy data may be also incorrect. In our proposal, this observation is related to the decisions made by the noise filter in which the examples that are noisy are presented. Therefore, the set of noisy examples detected in the second step at each iteration, that is, the set of noisy examples to analyze with the noise score, need not be correct. Thus, examples labeled as noise might be clean and vice versa.
3. *Examples in noisy clusters are less reliable.*
   The information obtained from a cluster of noisy examples, that is, an agglomeration of noisy examples, is less reliable. Our proposal will be tested with several percentages of examples with class noise in the training set, more specifically, up to 30% (see Section 4.1). Therefore, it is very likely that clusters of noisy examples will be created, particularly for the higher noise levels, in which a higher quantity of examples are corrupted. In this scenario, an example that was labeled as noise by the filter may be clean (clean examples within the cluster may be labeled as noise), and vice versa. The same occurs with the class labels: it is clear that in a cluster of noisy examples, most of them would have the class label incorrectly assigned. Therefore, information coming from these clusters should be taken cautiously (with less confidence).
4. *The presence of examples with different class labels in the neighborhood of an example may indicate that it is a noisy example.*
   The more examples in the neighborhood (k nearest neighbors) of an example e have their class label different to that of the example e, the more likely for this example e to be noisy is. It will be even more likely for e to be noisy if, in addition, its k nearest neighbors have been labeled as clean examples by the noise filter.
5. *The presence of examples with the same class label in the neighborhood of an example may indicate that it is a clean example.*

The more examples in the neighborhood (k nearest neighbors) of an example e have the same class label to that of the example e, the more likely for the example e to be clean is. It will be even more likely for e to be clean if, in addition, its nearest neighbors have been labeled as clean examples by the noise filter.

On account of these observations, we will consider the following information (provided by the examples from C_T) in order to set the confidence of an example $e \in C_N$ labeled as noisy to be noisy:

1. **Detection results of the noise-free FC-based filter** (related to the observations 1 and 2): each example is labeled as *clean* or *noisy* by the FC-based filter constructed in the second step of the method.
2. **Information of each training example as belonging to the neighborhood of other noisy examples** (related to the observation 3): the times that one example is among the k nearest neighbors of other examples labeled as noisy in C_N (denoted as $t(e)$). This value provides an idea of how involved is an example in noisy areas (clusters with noise). If the value is high, it means that this example is among the nearest neighbors of many other examples that may have noise.
3. **Information of the neighborhood of each example** e (related to the observations 4 and 5): the classes of e and the examples close to e, that is, its k nearest neighbors ($k = 5$ is considered in this paper).

Based on the observation 3, we have defined the function *confidence*(e) (see Eq. (1)), which checks whether the example e is close to other noisy examples. This function returns values in the interval $(0, 1]$. The value of *confidence*(e) is higher if e is not in the neighborhood of other noisy examples, whereas it is lower if e is in the neighborhood of several noisy examples. Hence, if *confidence*(e) $= 1$ (when e is not among the nearest neighbors of any noisy example), the information that this example provides (such as its class label and its label *clean* or *noise* given by the FC-based filter) is very reliable. However, if *confidence*(e) ≈ 0 (when e is among the nearest neighbors of many noisy examples), the information that it provides must not be taken into account.

$$confidence(e) = \frac{1}{\sqrt{1 + t(e)^2}} \tag{1}$$

Similarly, based on the observations 3–5, we have defined the function *neighborhood*(e) (Eq. (2)). This function aims to analyze the neighborhood of a given example e (its k nearest neighbors) to determine the degree of e being in a noisy cluster.

$$neighborhood(e) = \frac{\sum_{i=1}^{k} clean(e_i) \cdot confidence(e_i) \cdot differentClasses(e, e_i)}{k} \tag{2}$$

This function computes an average value of the k nearest neighbors considering their classes (function *differentClasses*(e, e_i)), the degree of the cleanness of each neighbor of e (function *clean*(e_i)) and the reliability of each neighbor (function *confidence*(e_i)).

The function *differentClasses*(e_1, e_2), defined in Eq. (3), takes into account the observations 4 (different classes increase the noise score) and 5 (coincident classes reduce the noise score). In such away, if the example e and its neighbor e_i have different class labels the value of *neighborhood*(e) is increased, whereas if they have the same class label the value of *neighborhood*(e) is reduced.

$$differentClasses(e_1, e_2) = \begin{cases} 1, & \text{if } class(e_1) \neq class(e_2) \\ -1, & \text{if } class(e_1) = class(e_2) \end{cases} \tag{3}$$

Furthermore, the observations 4 and 5 state that *clean* examples must have a higher weight than examples with *noise* in the computation of the noise score. Thus, the function *clean*(e_i) is defined based on the consideration that the number of noisy examples surrounding a given example e_i (that is, $n(e_i)$) is an indicator of the cleanness of that example – see Table 2 in order to check the difference between the functions $t(e)$ and $n(e)$. Thus, clean examples surrounded by many clean examples have a higher degree of cleanness than clean examples surrounded by noisy ones, since one must not trust the information provided by areas with many noisy examples (observation 3). The same occurs with the noisy examples: if a noisy example is surrounded by many other noisy examples, one can say that this example has a lower degree of noise than other noisy example surrounded by clean examples, which is placed in a more reliable area. For these reasons, the function *clean*(e_i) is defined as follows:

$$clean(e_i) = \frac{k + isnoise(e_i) \cdot (n(e_i) - k)}{2k},$$
$$isnoise(e_i) = \begin{cases} 1, & \text{if } e_i \text{ is noise} \\ -1, & \text{if } e_i \text{ is clean} \end{cases} \tag{4}$$

The function *isnoise*(e_i), used by *clean*(e_i), simply returns 1 if e_i is a *noisy* example and -1 if it is *clean*. Recall that the belonging of each example to the *clean* and *noise* sets is determined by the FC-based filter used in the second step of the filtering.

Thus, the function *clean*(e) (see Fig. 2) provides a value of the cleanness of the example e (not only if it is *clean* or *noise* as it is performed by the FC-based filter). The motivation for its usage is that some examples are expected to have a higher degree of confidence to be *clean* than other ones, and the same occurs with the noisy examples. It considers the fact that a *clean* example is more reliable than an example with *noise* (observations 4 and 5). As

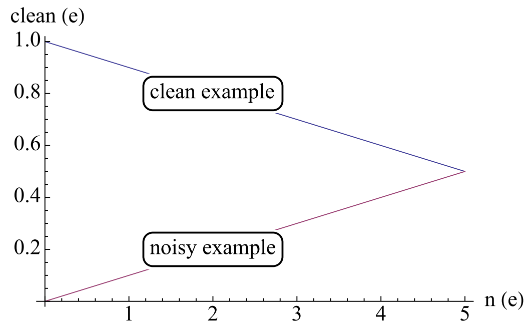


Fig. 2. Graphical representation of the function *clean*(e). The value of *clean*(e) depends on the label of e (*clean* or *noisy*) provided by the noise-free filtering (the second step in each iteration).

Fig. 2 shows, this function returns values in the interval $[0, 1]$, being 0 the value corresponding to an example with a maximum degree of being noisy and 1 the value corresponding to an example having a maximum degree of being clean. Concretely, the interval $[0, 0.5]$ is that of the values of noisy examples (0 for a noisy example in a cluster of clean examples and 0.5 for a noisy example in a cluster of noisy examples), whereas the interval $[0.5, 1]$ is that of the values of clean examples (0.5 for a clean example in a cluster of noisy ones and 1 for a clean example in a clean cluster). Thus, a higher value of cleanness is assigned to clean examples (considered by the FC-based filter) than to noisy examples.

Finally, the computation of the noise score NS for an example $e \in C_N$ is mainly based on the analysis of its neighborhood, represented by *neighborhood*(e), and this value is weighted by the reliability of the own example e, represented by *confidence*(e). Thus, both functions are combined to define the noise score $NS(e)$ as follows:

$$NS(e) = confidence(e) \cdot neighborhood(e) \tag{5}$$

As we have previously mentioned, the function *confidence*(e) is defined in the interval $(0, 1]$, whereas the function *neighborhood*(e) is defined in $[-1, 1]$. Therefore, NS is defined in the interval $[-1, 1]$, being higher if the example e is more likely to have noise. The sign of the result provided by the function *neighborhood*(e) defines if the example e is in fact *clean* (negative values) or *noisy* (positive values), whereas its absolute value defines the degree of confidence in this choice (being -1 the value corresponding if the example e is totally clean and 1 if the example e is probably noise). A value $NS(e) = 0$ implies that there is not reliable information about the example e to be labeled as clean or noisy. On the other hand, the function *confidence*(e) is another factor that establishes how representative the result provided by *neighborhood*(e) is, based on the degree of membership of e to noisy clusters.

After calculating the noise score for each potential noise example in C_N, those examples with a noise score higher than a threshold set by the user are removed. In the experimentation carried out in this paper, this threshold is by default fixed to 0. Thus, those examples in which there is any indication that they are noisy because their $NS(e) > 0$ are deleted. However, we also study the behavior of proposed filter with other different values of the threshold in Section 5, showing that the proposed method is robust with respect to this value.

Table 2
Difference between the functions $t(e)$ and $n(e)$.

Function	Description
$t(e)$	Number of times that e is among the k nearest neighbors of other noisy examples in C_N
$n(e)$	Number of noisy examples in C_N among the k nearest neighbors of the example e

3.5. On the time complexity of INFFC

There are two main aspects to consider when comparing the time required for filtering of INFFC to that of the other ensemble-based filters considered in this paper (EF and IPF):

- On the one hand, the noise evaluation strategy used by INFFC is faster than those of EF or IPF, since INFFC does not perform any internal partitioning of the training set to identify the noisy examples. Thus, INFFC always build 3 classifiers over the training set, whereas the time complexity of the noise evaluation strategy of EF and IPF depends on the number of partitions.
- On the other hand, the increase in the overall time complexity of INFFC with respect to EF and IPF is due to two main causes: its iterative elimination of noisy examples (which is not used by EF, but it is by IPF) and the additional step of each iteration where the noise score is computed (which is not used by neither EF or IPF).

Therefore, the most significant difference in time complexity of INFFC with respect to both EF and IPF is the computation of the noise score, which is mainly based on the usage of the k-NN classifier. The complexity of k-NN is of $O(n_{at} \cdot n_{ex})$, where n_{at} is the number of attributes and n_{ex} is the number of examples in the training set. However, one must note that the noise score is computed only over the set of noisy examples C_N. If we assume that the majority of the training data are clean (which is the most common scenario), the cardinality of C_N is usually not too large. Thus, we will compute the nearest neighbors of the examples belonging to the set C_N and maybe the nearest neighbors of some clean examples surrounding these noisy examples in C_N (which is required for computing the function $clean(e)$ of all the nearest neighbors of each noisy example). Besides, the size of C_N is usually reduced at each iteration of INFFC (since a fewer number of noisy examples are expected to be found at each successive iteration), so it is expected that the time required for the computation of the noise score is lower at each iteration.

Thus, even though INFFC might be slower than other ensemble-based filters, such as EF and IPF (mainly due to the computation of the noise score), the time is not the most important aspect to be considered since it is only required once – note that all the noise filters employed in this paper are offline preprocessing methods that are not continuously receiving new data. In any case, the overall time cost of INFFC can be reduced if its internal FC-based filters and the aforementioned nearest neighbors that are needed to compute the noise score are implemented in parallel, which should be the most appropriate choice to take advantage of the possibilities of the design of the method.

4. Experimental framework

This section presents the details of the experimental study carried out in order to check the validity of the proposed noise filter. First, Section 4.1 describes the datasets used. Then, Section 4.2 shows the parameter setup for the classification algorithms used in the FC-based filter. Section 4.3 presents the noise filters compared with our proposal. Finally, Section 4.4 describes the methodology followed to analyze the results.

4.1. Datasets

The experimentation is based on the 25 datasets from the KEEL-Dataset and UCI repositories [39,40] shown in Table 3, where #EX refers to the number of examples, #AT to the number of attributes and #CL to the number of classes. Some of the largest datasets (*penbased*, *satimage*, *shuttle* and *splice*) were stratified at 10% in order to reduce the computational time required for training, given the large amount of executions carried out. Examples containing missing values are removed from the datasets before their usage.

In order to control the amount of noise in each dataset, different noise levels $x\%$ are introduced into each training dataset in a supervised manner following an *uniform class noise scheme* [41]: $x\%$ of the examples are corrupted randomly replacing the class labels of these examples by other ones from the set of classes. Thus, using this scheme any of the classes of the dataset may be affected by the noise. This results in a more general noise introduction scheme than considering the other well-known class noise introduction scheme, that is, the pairwise class noise scheme [3], which only affects to the majority class and has a lower impact on classifier performance [22]. We will consider the noise levels ranging from $x = 0\%$ (base datasets) to $x = 30\%$, by increments of 5%. As a consequence, 150 noisy datasets with class noise are created from the aforementioned 25 base datasets (a total of 175 datasets). All these datasets are available on the webpage associated with this paper.

Table 3
Base datasets used in the experimentation.

Dataset	#EX	#AT	#CL	Dataset	#EX	#AT	#CL
automobile	159	25(15/10)	6	monk	432	6(6/0)	2
balance	625	4(4/0)	3	new-thyroid	215	5(5/0)	3
banana	5300	2(2/0)	2	penbased	1099	16(16/0)	10
car	1728	6(0/6)	4	pima	768	8(8/0)	2
cleveland	297	13(13/0)	5	satimage	643	36(36/0)	7
contraceptive	1473	9(9/0)	3	shuttle	2175	9(9/0)	7
dermatology	358	33(1/32)	6	splice	319	60(0/60)	3
ecoli	336	7(7/0)	8	twonorm	7400	20(20/0)	2
flare	1066	11(0/11)	6	vehicle	846	18(18/0)	4
german	1000	20(13/7)	2	wdbc	569	30(30/0)	2
glass	214	9(9/0)	7	yeast	1484	8(8/0)	10
ionosphere	351	33(33/0)	2	zoo	101	16(0/16)	7
iris	150	4(4/0)	3				

In order to create a noisy dataset from the original one, the noise is introduced into the training partitions as follows:

1. A level of noise $x\%$ of class noise is introduced into a copy of the full original dataset.
2. Both datasets, the original one and the noisy copy, are partitioned into 5 equivalent folds, that is, with the same examples in each one.
3. The training partitions are built from the noisy copy, whereas the test partitions are formed from examples from the base dataset, that is, the noise free dataset.

The accuracy estimation of the classifiers in a dataset is obtained by means of 5 runs of a stratified 5-fold stratified cross-validation (SCV). Hence, a total of 25 runs per dataset and noise level are averaged. The aforementioned 175 datasets will be preprocessed with our approach an other 7 noise filters resulting in 1400 new preprocessed datasets. The preprocessing with all the 8 noise filters of the 5×5 folds for each one of the 175 unprocessed datasets implies a total of 35,000 executions, and the running of each one of the three classification algorithms (C4.5, SVM and k-NN) of the 5×5-fold SCV of the 1575 datasets (175 unprocessed and 1400 processed) results in 118,125 additional executions, from which the results obtained are analyzed in this paper. Furthermore, we have performed additional experiments with several thresholds for our proposal, which increases the number of experiments carried out.

4.2. Parameter setup of the FC-based filter

The parameter setup for the three classification algorithms used by our FC-based filter is presented in Table 4:

4.3. Noise filtering methods

The noise filters used for the comparison have been chosen due they apply different filtering strategies and are well-known representatives of the field. They are briefly described in the following:

1. *Edited Nearest Neighbor* (ENN) [9]. This algorithm removes those examples which class does not agree with that of the majority of its k nearest neighbors.
2. *All k-Nearest Neighbors* (AllKNN) [16]. This applies the k-NN rule k times varying the number of neighbors considered between 1 to k. Those examples misclassified by k-NN are removed from the training set when all the values of k have been considered.
3. *Classification Filter* (CF) [8]. CF splits the training set into n subsets. A set of classifiers is trained from the union of any $n - 1$ subsets. The examples misclassified in the excluded subset are then eliminated from the training set.
4. *Multiedit* (ME) [15]. This splits the training data into n folds. k-NN classifies the examples from the part x considering the part $(x + 1)$ mod n as training set and the misclassified examples are removed. This process is repeated until no examples are eliminated.
5. *Nearest Centroid Neighbor Edition* (NCNE) [10]. This is a modification of ENN, which consists of discarding from the training set every example misclassified by the k nearest centroid neighbors (k-NCN) rule.
6. *Ensemble Filter* (EF) [7]. EF classifies the training data using an n-fold cross-validation with several classification algorithms. Then, the noisy examples are identified using a voting scheme (consensus or majority) and removed from the training data.
7. *Iterative-Partitioning Filter* (IPF) [17]. IPF removes noisy examples in multiple iterations. In each iteration, the training data is split into n folds and C4.5 is built over each of these subsets to evaluate all the examples. Then, the examples misclassified are removed (using the consensus or majority scheme) and a new iteration is started.

The parameter setup for all the noise filters is shown in Table 5. Even though almost all of the parameters are the default ones recommended by the authors of such filters, we have set the majority scheme and $n = 3$ partitions for ensemble-based filters in order to establish a fair comparison among these types of filters.

Note that EF and IPF are filtering approaches based on the use of ensembles (*ensemble-based filters*), whereas the rest of the methods are based on single classifiers and/or measures (*non-ensemble filters*). Therefore, we will consider these two groups in the comparisons carried out in the following section.

4.4. Methodology of analysis

The effect of the aforementioned filters along with our proposal will be analyzed comparing the performance obtained for each dataset with three different classifiers: C4.5 [11], SVM [25] and k-NN [26] with $k = 1$. As we have already mentioned, the performance estimation is obtained by means of 5 runs of a 5-fold SCV, averaging the test accuracy results. Given the large amount of results obtained, for the sake of brevity only averaged results are shown in the paper (the detailed results can be found on the web page associated with this paper), but it must be taken into account that our conclusions are based on the proper statistical analysis, which considers all the results (not averaged). In addition to the accuracy results, the number of datasets preprocessed with each filter in which each classifier obtains the best result is shown.

Table 4
Parameter specification for the classification algorithms used in the FC-based filter.

Classifier	Ref.	Parameters
C4.5	[11]	Confidence: 0.25, minimal instances per leaf: 2, prune after the tree building
k-NN	[26]	$k = 3$, Euclidean distance
LOG	[37]	Ridge value in the log-likelihood: 10^{-8}

Table 5
Parameter specification for the noise filters.

Filter	Ref.	Abbreviation	Parameters
AllKNN	[16]	AllKNN	k value: 3, Distance: Euclidean
Classification filter	[8]	CF	Classifier: C4.5, n: 3
Edited nearest neighbor	[9]	ENN	k value: 3, Distance: Euclidean
Multiedit	[15]	ME	k value: 1, Subblocks: 3, Distance: Euclidean
Nearest centroid neighborhood edition	[10]	NCNE	k value: 3
Ensemble filter	[7]	EF	Voting scheme: majority, n: 3
Iterative-partitioning filter	[17]	IPF	Voting scheme: majority, n: 3
Iterative noise filter based on the fusion of classifiers	–	INFFC	Voting scheme: majority, n: 3

The performance of our approach is studied with each classifier (C4.5, SVM and 1-NN) using statistical comparisons in three different scenarios:

1. Comparison between INFFC and not applying preprocessing (Section 5.2). By means of this comparison we try to check if the application of noise filtering techniques implies an advantage with respect to no preprocessing (*None*).
2. Comparison among INFFC and the *non-ensemble* filters (Section 5.3). Its motivation is to check the behavior of our proposal against classic state-of-the-art filters belonging to other different paradigms from ensemble-based methods. These include the noise filters AllKNN, CF, ENN, ME and NCNE.
3. Comparison among INFFC and the *ensemble-based* filters (Section 5.4). We also compare our proposal, which is based on ensembles of classifiers, with other related noise filters that also use multiple classifiers (EF and IPF).

We have separately studied the differences between our proposal and the *non-ensemble* and *ensemble* methods for two main reasons. First, the separation is motivated by the different nature of the methods of both groups. Second, performing a multiple statistical comparison usually requires a larger number of datasets to detect significant differences when the number of comparison methods increases. This is due to the fact that multiple statistical comparisons are limited by the number of datasets and an unified comparison can only be performed if a lager number of datasets than the one considered in this paper is available for study.

Wilcoxon's test [42] will be applied to study the differences between the proposal of this paper and no using preprocessing. The p-values associated with the comparison of the results of the two methods involved over all the datasets will be obtained. The p-value represents the lowest level of significance of a hypothesis that results in a rejection and it allows one to know whether two algorithms are significantly different and the degree of their difference. We will consider a difference to be significant if the p-value obtained is lower than 0.1 – even though p-values slightly higher than 0.1 might be showing important differences.

Regarding the comparison between our approach and the other noise filters (either *non-ensemble* or *ensemble-based* methods), the Aligned Friedman test [27,43] will be used. We will use this test to compute the set of ranks that represent the effectiveness associated with each algorithm and the p-value related to the significance of the differences found by this test. In addition, the adjusted p-value with Holm test [44] will be computed. More information about these tests and other statistical procedures can be found at http://sci2s.ugr.es/sicidm/.

In addition, it is important to carry out an analysis of which examples are removed from the data set belong to those corrupted by the noise introduction scheme and which to the non-corrupted set of examples. Consider that $D_T = D_N \sqcup D_O$, being D_N the set of examples whose class labels have been corrupted by the noise scheme and D_O the set of original non-corrupted examples. On the other hand, each noise filter removes a set of examples $D_R \subseteq D_T$. Based on these sets of examples, we have defined two different metrics in order to check the elimination capabilities of each noise filter (see Table 6).

Finally, we will also study the effect of the noise threshold in the performance of our approach, by comparing the results of INFFC considering several thresholds by means of the usage of the Aligned Friedman procedure and the Holm test.

5. Analysis of results

This section presents the analysis of the results obtained. First, performance results are presented and analyzed in Section 5.1. In order to analyze the results obtained, several statistical comparisons are performed, studying the differences among the proposal of this paper and not preprocessing (Section 5.2), its comparison with the other *non-ensemble methods* (Section 5.3) and and its comparison with the other *ensemble-based methods* (Section 5.4). The analysis of the examples removed by each noise filter is presented in Section 5.5, whereas the study on the behavior of INFFC with different thresholds for the noise score is shown in Section 5.6.

5.1. Accuracy results and number of datasets with best result

Table 7 shows the test accuracy results obtained by each classifier when using each one of the 8 filters considered and without preprocessing (*None* column). This table also shows the number of datasets on which each filter provides the best result with each one of the three classifiers and noise level. The best results at each noise level are highlighted in boldface. From this table, several remarks can be made:

1. **Test accuracy results:**
   - For all the classifiers (C4.5, SVM and 1-NN), INFFC is the best method at all the noise levels, and also without additional noise.
   - Considering C4.5, the results of IPF and EF are also remarkable (even though they are lower than those of INFFC). In general, the worst filters are ME and AllKNN. The behavior of *None* should be mentioned: at the lowest noise levels (up to 10%), it obtains good test accuracy results; at intermediate noise levels, from 15% to 20%, it obtains medium results compared with the rest of the noise filters; finally, when the noise level is higher than 25% it obtains the worst results, as expected.
   - The results of SVM are similar to those of C4.5. The two best filters after INFFC are IPF and EF. The worst filters are again ME and AllKNN (up to 5% of noise) and from 10% onwards, *None* is clearly the worst method. However, *None* obtains very good results without noise.
   - Regarding to 1-NN, INFFC is usually followed by IPF, even though CF (up to 20%) and EF (from 20% onwards) are obtain remarkable results. The worst results are usually obtained by ME and *None*.
   - Even though filters may be counterproductive without excessive noise, since they are more likely to remove clean examples, one must realize that they only imply a low loss of accuracy without noise (0% of noise level). In any case, we acknowledge that this observation is based on average results. In order to reach meaningful conclusions we should use statistical tests as it is recommended in the specialized literature [27], which are performed in the following sections.
2. **Number of datasets with best result:**
   - INFFC generally obtains the best results in more datasets than the rest of the methods independently of the classifier considered. There are two exceptions: for the noise sensitive classifiers, SVM and 1-NN, *None* excels over the rest of filtering techniques. These exceptions can be attributed to the fact that, without any noise, SVM and 1-NN are able to build accurate classifiers. Thus, in absence of noise, the more information (number of examples) they have, the more exact the models constructed may be (so the application of filtering techniques is not needed in this case). However, their behavior obviously worsen when the noise level increases, since they are very noise-sensitive techniques.
   - For C4.5, IPF and EF (and *None* at the lowest noise levels, up to 10%) also obtains good results.
   - For SVM, IPF only obtains good results at the highest noise levels, from 25% onwards. The results of EF, AllKNN (up to 10%) or *None* must be also considered, since they are also good compared with the rest of noise filters.
   - For 1-NN, IPF and EF also highlight at many noise levels, whereas AllKNN and NCNE only obtains good results at the lowest noise levels (up to 10%).

Table 6
Measures computed.

Metric description	Expression
Eliminations among the corrupted examples	$100\frac{\lvert D_N \cap D_R\rvert}{\lvert D_N\rvert}$
Eliminations among the non-corrupted examples	$100\frac{\lvert D_O \cap D_R\rvert}{\lvert D_O\rvert}$

5.2. Comparison between INFFC and not preprocessing

The results of INFFC with each one of the classifiers (C4.5, SVM and 1-NN) are compared with those of the datasets without preprocessing (*None*). In order to study whether there are statistical differences among them, Wilcoxon's test has been performed – see Table 8. In this table, INFFC and *None* using different classifiers (C4.5, 1-NN and SVM) are compared using the Wilcoxon's test and the associated *p*-values are shown. From the very low *p*-values obtained in these comparisons, one can conclude that there exist statistical differences for all the classifiers at all the noise levels between the usage of our filter and *None* (considering a significance level of $\alpha = 0.1$). For C4.5 and SVM without noise, even though these differences are not significant, very low *p*-values are obtained. This fact shows the great advantage of applying filtering techniques with respect to no preprocessing, particularly when the noise level increases.

5.3. Comparison among INFFC and the non-ensemble filters

Table 9 presents the statistical comparison performed among INFFC and the *non-ensemble* filters using each one of the three classifiers considered (C4.5, SVM and 1-NN). It shows for each filter results in the form *ranks/p-value*, where *ranks* represent the ranks obtained by the Aligned Friedman procedure and *p-value* is the adjusted *p*-value computed by the Holm test. This table also shows the *p*-value obtained by the Aligned Friedman test. Looking at Table 9, we can observe that:

Table 7
Test accuracy of each classifier (C4.5, SVM and 1-NN) and number of datasets on which each filter provides the best result (best results are remarked in bold).

Method	Test accuracy							Best (out of 25)						
	0%	5%	10%	15%	20%	25%	30%	0%	5%	10%	15%	20%	25%	30%
C4.5														
AllKNN	79.20	78.87	78.48	78.00	77.39	77.36	76.35	1	0	0	1	1	1	1
CF	80.43	80.21	79.83	79.37	78.87	78.63	78.01	0	0	0	0	1	0	0
ENN	80.09	79.87	79.75	79.06	78.76	78.33	77.65	0	2	2	0	3	5	1
EF	80.41	80.22	79.83	79.56	79.33	79.01	78.46	2	5	6	5	5	3	3
IPF	81.18	80.79	80.56	79.92	79.32	79.27	79.03	6	3	2	3	1	5	1
ME	77.88	77.60	76.79	76.60	75.52	75.32	74.45	3	1	0	0	0	2	0
NCNE	80.58	80.23	79.98	79.18	78.65	78.43	77.41	1	1	2	3	1	1	0
INFFC	**81.77**	**81.57**	**81.21**	**80.97**	**80.44**	**80.07**	**79.99**	**8**	**9**	**9**	**12**	**12**	**8**	**17**
None	81.32	80.89	80.35	79.46	78.24	77.21	76.16	7	5	5	2	2	0	2
SVM														
AllKNN	77.89	77.50	77.48	76.76	75.86	75.14	74.20	3	3	4	2	1	1	1
CF	79.53	79.00	78.79	78.28	77.75	77.40	76.75	0	1	0	2	0	1	1
ENN	78.60	78.48	78.24	77.91	77.25	76.87	75.83	1	1	1	1	1	1	0
EF	79.72	79.33	79.02	78.73	78.20	78.24	77.56	3	4	3	4	4	2	4
IPF	79.76	79.42	79.28	78.88	78.47	78.29	77.84	2	1	1	0	1	4	4
ME	77.25	76.87	76.21	75.49	74.90	74.25	73.25	0	0	0	1	0	2	0
NCNE	78.92	78.56	78.01	77.33	76.15	75.21	74.11	3	3	1	3	1	0	1
INFFC	**80.71**	**80.43**	**80.17**	**79.89**	**79.67**	**79.41**	**78.97**	6	**9**	**12**	**10**	**13**	**10**	**11**
None	79.98	77.80	76.06	74.71	72.87	71.05	69.77	**8**	4	3	4	4	4	3
1-NN														
AllKNN	79.09	78.80	78.38	77.75	77.31	76.65	76.24	6	5	5	2	1	0	1
CF	79.99	79.76	79.56	79.07	79.01	78.18	77.73	1	0	2	1	2	2	0
ENN	79.58	79.32	78.98	78.56	78.18	77.43	76.92	0	1	0	1	1	0	0
EF	79.95	79.61	79.32	79.05	78.84	78.43	78.27	2	4	2	5	6	3	6
IPF	80.29	79.95	79.70	79.40	79.22	78.74	78.52	2	4	4	4	3	4	4
ME	77.18	76.88	76.17	75.85	75.37	74.81	74.01	0	0	0	1	1	4	1
NCNE	80.01	79.70	79.22	78.46	77.87	77.29	76.22	4	3	5	2	2	1	0
INFFC	**80.75**	**80.37**	**80.10**	**79.89**	**79.65**	**79.25**	**79.17**	4	**8**	**8**	**10**	**8**	**12**	**13**
None	79.15	76.43	74.01	71.67	69.49	67.04	64.07	**7**	2	0	0	1	0	0

Table 8
p-values of the Wilcoxon's test after comparing the proposed filter versus no preprocessing using each one of the three classifiers: C4.5, SVM and 1-NN.

Method	0%	5%	10%	15%	20%	25%	30%
C4.5	0.14854	0.02831	0.02735	0.00172	0.00010	0.00004	0.00006
SVM	0.17024	0.00189	0.00081	0.00019	0.00017	0.00017	0.00010
1-NN	0.01702	0.00002	0.00001	0.00001	0.00001	0.00001	0.00001

1. *Results of C4.5.* INFFC is significantly better than all the other filters at all the noise levels considering a significance level of $\alpha = 0.1$, although there is a exception: the comparison versus NCNE at the two lowest noise levels (0% and 5%), even thought the *p*-values are relatively low (close to 0.2 and 0.1, respectively).
2. *Results of SVM.* INFFC is statistically better than the rest of the filters, even though with NCNE and CF without noise these *p*-values are slightly higher than 0.1.
3. *Results of 1-NN.* INFFC is statistically better than AllKNN and ME at all the noise levels. It also is statistically better than ENN and NCNE from 15% onwards (and it obtains a very low *p*-value close to 0.1 versus ENN with 10% of noise level). Finally, INFFC is statistically better than CF at the highest noise levels, from 25% onwards, although relatively low *p*-values are obtained with 15% and 20% (close to 0.17 and 0.14, respectively). Thus, in the worst cases, at the lowest noise levels, INFFC shows a better behavior, but without statistical differences.

5.4. Comparison among INFFC and the ensemble-based filters

Table 10 presents the statistical comparison performed among INFFC and the *ensemble-based* filters using each one of the three classifiers considered (C4.5, SVM and 1-NN). Looking at Table 10, we can observe that, for C4.5 and SVM, INFFC is clearly better than EF and IPF at all the noise levels. With 1-NN, our proposal is statistically better than the other two *ensemble-based* filters from 20% onwards, considering a significance level $\alpha = 0.1$ (at the other noise levels, very low *p*-values are obtained, all close to 0.1 except for 10% noise level in which the *p*-value is higher).

5.5. Number of examples removed by each filter

Table 11 shows the results obtained for the two metrics described in Table 6, referring to percentages of examples removed by each noise filter at each noise level in the sets of corrupted examples and non-corrupted examples. For the sake of brevity, only averaged results for each metric at each noise level are shown. The complete results for each data set at each noise level can be found in the webpage associated with this paper. The analysis of this table of results leads to following observations:

1. **Corrupted examples removed.** Generally, the number of examples removed among the corrupted examples (those with induced noise) is maintained for all the filters at the different noise levels. However, the capability of noise detection is reduced in most of the noise filters when the noise level increases, except for INFFC, IPF and ME, which are less affected by the increase of the noise level.
   Even though INFFC is the filter that detects a lower number of noisy examples. This fact is more clearly observed at the lowest

Table 9
Results for each *non-ensemble* filter and noise level in the form *ranks/p-value*, where *ranks* represent the ranks obtained by the Aligned Friedman test and *p-value* is the adjusted *p*-value computed by the Holm test. Those cases where the null hypothesis is not rejected ($\alpha = 0.1$) are indicated with a star (*).

Method	0%	5%	10%	15%	20%	25%	30%
C4.5							
AllKNN	98.46/0.00007	109.56/0.00000	109.54/0.00000	109.37/0.00000	108.15/0.00000	102.79/0.00000	109.31/0.00000
CF	80.50/0.01292	72.42/0.04831	77.42/0.00481	81.94/0.00046	77.13/0.00166	70.42/0.03724	72.27/0.00101
ENN	71.25/0.06861	71.29/0.04831	66.23/0.04736	62.10/0.02753	62.73/0.02124	66.96/0.03724	68.85/0.00135
ME	114.88/0.00000	114.06/0.00000	117.08/0.00000	113.90/0.00000	120.25/0.00000	117.85/0.00000	117.35/0.00000
NCNE	61.17/0.18944*	61.40/0.12673*	62.85/0.04736	69.21/0.01115	68.87/0.01043	72.15/0.03724	74.56/0.00075
INFFC	44.73	42.27	37.88	34.48	33.87	40.83	28.67
SVM							
AllKNN	95.54/0.00020	97.92/0.00008	89.94/0.00289	91.65/0.00033	98.73/0.00000	98.31/0.00000	90.46/0.00000
CF	67.13/0.12529*	72.88/0.04587	69.31/0.08286	72.02/0.03531	66.69/0.01048	67.69/0.00590	66.38/0.00276
ENN	85.00/0.00393	78.15/0.02111	74.00/0.06993	69.77/0.03531	68.17/0.01048	68.13/0.00590	73.27/0.00079
ME	110.54/0.00000	107.62/0.00000	107.98/0.00001	111.69/0.00000	106.98/0.00000	105.50/0.00000	105.73/0.00000
NCNE	68.06/0.12529*	70.04/0.04587	82.19/0.01720	83.58/0.00295	97.23/0.00000	100.48/0.00000	106.27/0.00000
INFFC	44.73	44.38	47.58	42.29	33.19	30.88	28.88
1-NN							
AllKNN	91.08/0.01439	94.65/0.00418	90.85/0.00738	97.77/0.00012	103.52/0.00000	98.23/0.00000	90.31/0.00001
CF	69.06/0.49687*	64.71/0.74839*	62.56/0.39177*	62.38/0.17781*	56.13/0.14441*	59.19/0.05091	60.25/0.02636
ENN	75.54/0.28394*	75.98/0.22132*	77.65/0.11785*	73.10/0.05527	74.31/0.00723	85.27/0.00012	83.44/0.00009
ME	118.90/0.00000	119.02/0.00000	118.69/0.00000	116.12/0.00000	114.19/0.00000	107.38/0.00000	104.46/0.00000
NCNE	61.83/0.56389*	63.06/0.74839*	69.42/0.32044*	76.13/0.04347	85.00/0.00050	86.19/0.00012	100.12/0.00000
INFFC	54.6	53.58	51.83	45.5	37.85	34.73	32.42

Table 10
Results for each *ensemble-based* filter and noise level in the form *ranks/p-value*, where *ranks* represent the ranks obtained by the Aligned Friedman test and *p-value* is the adjusted *p*-value computed by the Holm test. Those cases where the null hypothesis is not rejected ($\alpha = 0.1$) are indicated with a star (*).

Method	0%	5%	10%	15%	20%	25%	30%
C4.5							
EF	48.78/0.00032	45.60/0.00106	46.28/0.00185	44.26/0.00078	40.92/0.00702	42.58/0.01139	46.16/0.00017
IPF	39.70/0.02143	44.16/0.00123	41.86/0.00944	46.20/0.00047	48.78/0.00014	44.44/0.00924	45.88/0.00017
INFFC	25.52	24.24	25.86	23.54	24.3	26.98	21.96
SVM							
EF	44.04/0.00330	43.80/0.00153	44.16/0.00737	44.22/0.00753	47.76/0.00007	46.02/0.00060	46.24/0.00105
IPF	44.68/0.00330	45.94/0.00087	43.58/0.00737	43.42/0.00753	44.04/0.00040	44.24/0.00088	42.90/0.00343
INFFC	25.28	24.26	26.26	26.36	22.2	23.74	24.86
1-NN							
EF	42.90/0.10551*	41.54/0.13166*	39.22/0.33988*	42.12/0.11285*	40.64/0.09284	44.60/0.00761	39.96/0.11043*
IPF	40.14/0.13644*	41.90/0.13166*	41.62/0.33988*	41.52/0.11285*	43.08/0.07571	42.64/0.00999	43.92/0.05036
INFFC	30.96	30.56	33.16	30.36	30.28	26.76	30.12

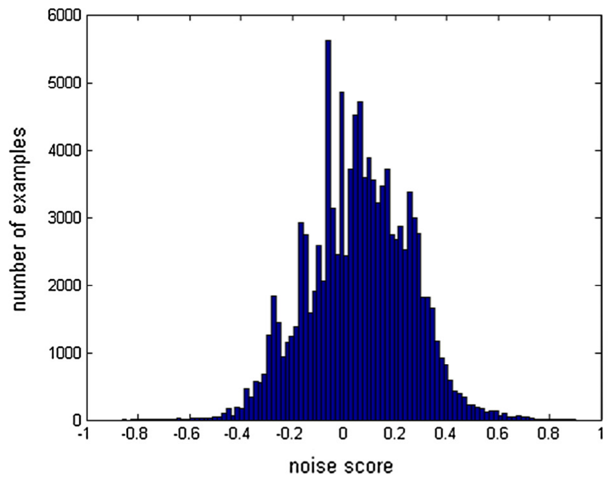
Table 11
Percentages of examples removed by each noise filter at each noise level in the sets of corrupted examples and non-corrupted examples (the best results are remarked in bold).

Set	Corrupted examples						Non-corrupted examples					
Noise level	5%	10%	15%	20%	25%	30%	5%	10%	15%	20%	25%	30%
AllKNN	**95.67**	**94.77**	**94.47**	**94.57**	**93.85**	**93.58**	31.81	34.86	37.87	40.73	43.65	46.82
CF	90.98	90.63	90.26	90.12	89.51	88.90	20.38	21.02	22.31	23.33	24.40	26.07
ENN	92.36	90.81	90.23	90.00	88.89	88.28	20.39	21.42	22.60	23.92	25.65	27.48
EF	93.18	92.65	92.32	92.08	91.65	91.63	19.21	19.92	21.02	21.95	22.77	24.46
IPF	90.51	89.72	89.68	89.55	89.09	89.16	15.42	15.56	16.17	16.62	16.87	17.31
ME	92.98	92.73	92.58	92.54	92.39	92.70	30.54	32.51	34.69	36.56	38.64	40.97
NCNE	91.84	90.17	89.52	88.12	87.78	86.24	20.71	22.58	24.45	26.66	28.77	31.19
INFFC	87.86	86.75	86.86	86.73	87.17	87.03	**11.58**	**11.84**	**12.00**	**12.34**	**12.78**	**12.92**

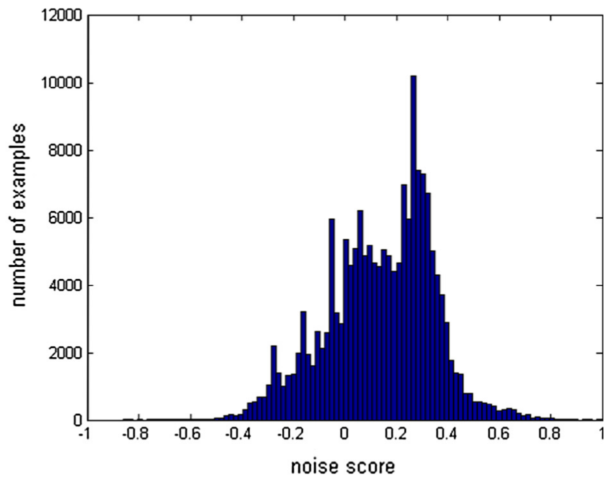
noise levels (for example, between AllKNN and INFFC a high difference can be observed). However, this difference is usually decreased when noise level increases. This may be due to the fact that, with low noise levels, the amount of noisy examples is small and hence, the detection or un-detection of some of these examples may notably alter this percentage, whereas more examples identified as noisy are need to alter this percentage at higher noise levels.

Therefore, even though INFFC detects less noisy examples, it obtains good noise elimination results (around 87% at all the noise levels). It can be considered to be comparable to the rest of the noise filters, particularly when the noise level increases (for example, with CF, ENN, IPF or NCNE).

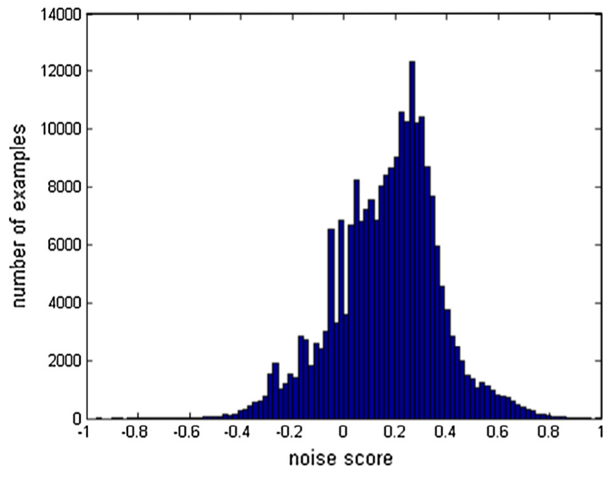
2. **Non-corrupted examples removed.** The number of examples removed among non-corrupted examples (those in which noise has not been introduced) has a greater variation in the majority of the noise filters when the noise level increases. Thus, for example, AllKNN drastically removes more examples when the noise level increases (from a 31.82% at 5% of noise level to 46.82% at 30% of noise level). Something similar occurs with



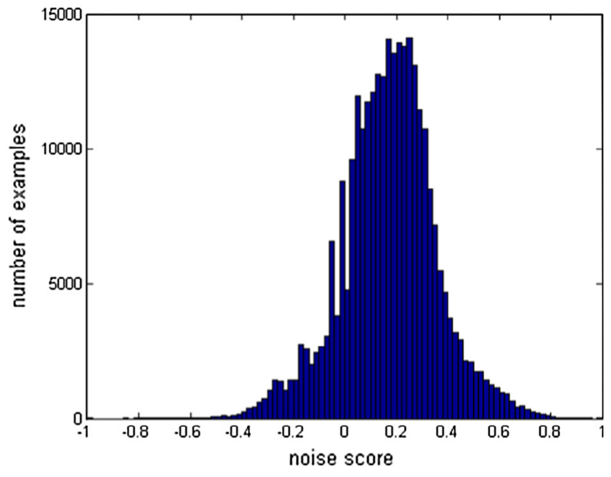
(a) 0% of class noise.



(b) 10% of class noise.



(c) 20% of class noise.



(d) 30% of class noise.

Fig. 3. Distribution of the noise score for different noise levels. Each figure represents the number of examples corresponding to each value of the noise score computed by INFFC (considering all the datasets used in this paper).

Table 12
Results for comparison among INFFC considering different thresholds in the form *ranks/p-value*, where *ranks* represent the ranks obtained by the Aligned Friedman test and *p-value* is the adjusted *p*-value computed by the Holm test. Those cases where the null hypothesis is not rejected ($\alpha = 0.1$) are indicated with a star (*).

Threshold	0%	5%	10%	15%	20%	25%	30%
−0.5	160.90/1.00000*	169.80/0.41766*	155.68/1.00000*	144.08/0.99949*	134.06/1.00000*	119.68/1.00000*	110.46/1.00000*
−0.4	159.38/1.00000*	167.74/0.43093*	156.20/1.00000*	143.88/0.99949*	132.62/1.00000*	118.50/1.00000*	110.80/1.00000*
−0.3	158.92/1.00000*	163.52/0.43225*	158.00/1.00000*	141.60/0.99949*	124.26/1.00000*	115.98/1.00000*	108.54/1.00000*
−0.2	167.56/1.00000*	155.38/0.60225*	158.36/1.00000*	136.18/0.99949*	120.24/1.00000*	118.64/1.00000*	109.16/1.00000*
−0.1	172.50/1.00000*	147.38/0.74059*	150.86/1.00000*	127.82/1.00000*	116.84/1.00000*	112.54/1.00000*	112.76/1.00000*
0	159.08/1.00000*	137.98/0.81359*	141.06/1.00000*	120.82/1.00000*	112.76/1.00000*	115.18/1.00000*	104.36
0.1	156.10/1.00000*	114.24	141.66/1.00000*	104.58/1.00000*	108.48	103.18	110.22/1.00000*
0.2	146.1	117.40/0.91208*	124.88	102.14	115.66/1.00000*	113.00/1.00000*	129.10/1.00000*
0.3	159.70/1.00000*	165.72/0.43225*	160.02/1.00000*	192.36/0.01295	171.92/0.21311*	188.38/0.02328	195.10/0.01217
0.4	182.54/1.00000*	204.50/0.01450	200.64/0.07303	230.98/0.00006	224.54/0.00045	244.80/0.00001	253.30/0.00000
0.5	204.02/0.42982*	226.40/0.00089	219.26/0.00974	239.64/0.00002	263.28/0.00000	264.26/0.00000	270.58/0.00000
0.6	209.42/0.31528*	229.60/0.00070	228.14/0.00339	256.98/0.00000	274.98/0.00000	279.40/0.00000	278.90/0.00000
0.7	210.20/0.31528*	229.84/0.00070	231.14/0.00266	258.86/0.00000	278.82/0.00000	281.88/0.00000	282.24/0.00000
0.8	210.58/0.31528*	227.50/0.00083	231.10/0.00266	257.08/0.00000	278.54/0.00000	281.58/0.00000	281.48/0.00000

the rest of the filters, with the exception of INFFC and IPF, which are able to better maintain the number of examples removed from this set. INFFC is the method that removes less examples among the non-corrupted ones, with great differences with respect to the rest of the noise filters considered. For example, it obtains 12.92% versus 17.31% of IPF (the second method that removes less examples) and versus 46.82% of AllKNN (the method that removes the largest number of examples).

As a conclusion, it can be said that our proposal is able to maintain the level of elimination in both the corrupted and non-corrupted sets of examples regardless of the noise level. Even though it removes less examples among the corrupted ones, it maintains a similar elimination level to other noise filters, particularly when the noise level increases. However, almost all the filters behaves worse considering the non-corrupted set of examples. They eliminate large amounts of possible clean examples compared with INFFC. Thus our method is able to maintain a good balance between the examples that one must delete and those that must not. Moreover, observing the results obtained along this study, the importance of maintaining as many non-corrupted examples as possible can be highlighted, since statistically better results are obtained even though less noisy examples are removed. Hence, a good balance between the elimination of noisy examples and the correct detection of noise-free ones should be maintained.

5.6. Influence of the threshold in INFFC

In addition to the comparison with respect to the state-of-the-art filtering methods, we have studied the effect of the value of the threshold in our proposal. In order to perform this analysis, we have studied the number of examples corresponding to each value of the noise score computed by INFFC (considering all the datasets used in this paper). These results are represented in Fig. 3 for four noise levels, that are 0%, 10%, 20% and 30% (the graphics for rest of noise levels can be found in the webpage associated to this paper).

Attending to Fig. 3, two points must be remarked:

- The distribution of the noise score displaces to the right (to higher values) when the noise level increases. This is due to the fact that, increasing the number of noisy examples in a dataset produces a higher appearance of examples that are more easily identified as noisy by the noise metric (even though they are not necessarily noisy since they may be in noisy clusters), and therefore their noise score is higher.
- Most of the values of the noise score are in the interval (-0.5, 0.8). On this account, we have chosen this interval of values to perform an study of the behavior of the proposed filter considering several values of the threshold, in order to check how it affects to the results obtained.

Table 12 shows for each of the thresholds considered in INFFC (from 0.5 to 0.8, by increments of 0.1) the results in the form *ranks*/*p-value*, where *ranks* represent the ranks obtained by the Aligned Friedman procedure and *p-value* is the adjusted *p*-value computed by the Holm test. In order to perform this comparison we have considered the results of the C4.5 classifier, although the results with SVM and 1-NN (found in the webpage of the paper) provide similar results to those shown here. The results of Table 12 show that the thresholds with the best Aligned Friedman rankings are 0, 0.1 and 0.2 at most of the noise levels. In general, no statistical differences are found between these thresholds and the usage of lower threshold values, but they are generally statistically better than higher values of the threshold. This fact shows that removing all the potentially noisy examples in each iteration is not always the best option. Thus, in many cases, it is better to adopt more conservative strategies to detect and remove the noisy examples (even though no statistical differences may be found). However, removing few noisy examples in each iteration seems to be a worse alternative, since statistical differences are found comparing the highest thresholds with the lower ones set as control methods by the Aligned Friedman test.

As final remark, one must realize that all the reported results and the statistical comparisons performed show the suitability of our filtering approach dealing with class noise datasets. The combination of the iterative ensemble-based filter and the noise score to control the noise sensitivity has shown to be a good alternative to other existing noise filters found in the literature. The threshold of the proposal must be carefully fixed in order to obtain the best possible results. Very high values of the threshold seems to be inferior to choosing lower values – we recommended that this threshold is fixed close to 0, according to our experimentation.

6. Concluding remarks

This paper proposes a new noise filtering method combining three different noise filtering paradigms: the usage of ensembles for filtering, the iterative filtering and the computation of noise measures. Thus, we propose an iterative noise filtering method based on the fusion of the predictions of several classifiers. We also introduce a noisy score to control the filtering sensitivity to remove more or less noisy examples according to the practitioner's necessities. We have compared our proposal against other well-known filters found in the literature over a large collection of real-world datasets with different levels of class noise.

The most remarkable fact behind the analysis of results is that almost all the filters studied in this paper eliminate higher amounts of clean examples compared to INFFC. However, INFFC maintains a similar elimination level of noisy examples than that of other noise filters, particularly when the noise level increases. Thus our method is able to maintain a good balance between the examples that one must delete (noisy examples) and those that must not (clean examples). Furthermore, from the experimental results we can conclude that our noise filter enhance the performance of the rest of the noise filters and no preprocessing. The statistical analysis performed supports our conclusions.

The analysis on the impact of the threshold fixed in the noise score shows that removing all the potentially noisy examples in each iteration is not always the best option, but neither removing very low quantities of noisy examples. Hence, a balance must be found in order to obtain the desired results. For this reason, we recommend a threshold close to 0, whose optimal value can be adapted depending on the problem.

Acknowledgment

Supported by the Projects TIN2011-28488, TIN2013-40765-P, P10-TIC-06858 and P11-TIC-7765. J.A. Sáez was supported by EC under FP7, Coordination and Support Action, Grant Agreement Number 316097, ENGINE European Research Centre of Network Intelligence for Innovation Enhancement (http://engine.pwr.wroc.pl/).

References

[1] X. Wu, X. Zhu, Mining with noise knowledge: error-aware data mining, IEEE Trans. Syst. Man Cybernet. – Part A: Syst. Humans 38 (4) (2008) 917–932.
[2] S. García, J. Luengo, F. Herrera, Data preprocessing in data mining, Intelligent Systems Reference Library, vol. 72, Springer, 2015.
[3] X. Zhu, X. Wu, Class noise vs. attribute noise: a quantitative study, Artif. Intell. Rev. 22 (2004) 177–210.
[4] B. Frenay, M. Verleysen, Classification in the presence of label noise: a survey, IEEE Trans. Neural Netw. Learning Syst. 25 (5) (2014) 845–869.
[5] J.A. Sáez, J. Luengo, F. Herrera, Predicting noise filtering efficacy with data complexity measures for nearest neighbor classification, Pattern Recogn. 46 (1) (2013) 355–364.
[6] J.A. Sáez, M. Galar, J. Luengo, F. Herrera, Analyzing the presence of noise in multi-class problems: alleviating its influence with the one-vs-one decomposition, Knowl. Inform. Syst. 38 (1) (2014) 179–206.
[7] C.E. Brodley, M.A. Friedl, Identifying mislabeled training data, J. Artif. Intell. Res. 11 (1999) 131–167.
[8] D. Gamberger, R. Boskovic, N. Lavrac, C. Groselj, Experiments with noise filtering in a medical domain, in: Proceedings of the Sixteenth International Conference on Machine Learning, Morgan Kaufman Publishers, 1999, pp. 143–151.
[9] D. Wilson, Asymptotic properties of nearest neighbor rules using edited data, IEEE Trans. Syst. Man Cybernet. 2 (3) (1972) 408–421.
[10] J. Sánchez, R. Barandela, A. Márques, R. Alejo, J. Badenas, Analysis of new techniques to obtain quality training sets, Pattern Recogn. Lett. 24 (2003) 1015–1022.
[11] J.R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufman Publishers, San Francisco, CA, USA, 1993.
[12] W.W. Cohen, Fast effective rule induction, in: Proceedings of the Twelfth International Conference on Machine Learning, Morgan Kaufman Publishers, 1995, pp. 115–123.

[13] D. Gamberger, N. Lavrac, S. Dzeroski, Noise elimination in inductive concept learning: a case study in medical diagnosis, in: Proc. of the 7th International Workshop on Algorithmic Learning Theory, Springer, 1996, pp. 199–212.

[14] D. Gamberger, N. Lavrac, S. Dzeroski, Noise detection and elimination in data preprocessing: experiments in medical domains, Appl. Artif. Intell. 14 (2000) 205–223.

[15] P. Devijver, On the editing rate of the MULTIEDIT algorithm, Pattern Recogn. Lett. 4 (1) (1986) 9–12.

[16] I. Tomek, An experiment with the edited nearest-neighbor rule, IEEE Trans. Syst. Man Cybernet. 6 (6) (1976) 448–452.

[17] T.M. Khoshgoftaar, P. Rebours, Improving software quality prediction by noise filtering techniques, J. Comput. Sci. Technol. 22 (2007) 387–396.

[18] T.K. Ho, J.J. Hull, S.N. Srihari, Decision combination in multiple classifier systems, IEEE Trans. Pattern Anal. Machine Intell. 16 (1) (1994) 66–75.

[19] M. Woźniak, M. Graña, E. Corchado, A survey of multiple classifier systems as hybrid systems, Inform. Fusion 16 (2014) 3–17.

[20] L.I. Kuncheva, J.J. Rodríguez, A weighted voting framework for classifiers ensembles, Knowl. Inform. Syst. 38 (2) (2014) 259–275.

[21] S. Sun, Local within-class accuracies for weighting individual outputs in multiple classifier systems, Pattern Recogn. Lett. 31 (2) (2010) 119–124.

[22] J.A. Sáez, M. Galar, J. Luengo, F. Herrera, Tackling the problem of classification with noisy data using multiple classifier systems: analysis of the performance and robustness, Inform. Sci. 247 (2013) 1–20.

[23] R. Barandela, R.M. Valdovinos, J.S. Sánchez, New applications of ensembles of classifiers, Pattern Anal. Appl. 6 (3) (2003) 245–256.

[24] J.S. Sánchez, L.I. Kuncheva, Data reduction using classifier ensembles, in: ESANN, 2007, pp. 379–384.

[25] V. Vapnik, Statistical Learning Theory, Wiley, New York, USA, 1998.

[26] G.J. Mclachlan, Discriminant Analysis and Statistical Pattern Recognition (Wiley Series in Probability and Statistics), Wiley Interscience, 2004.

[27] S. García, A. Fernández, J. Luengo, F. Herrera, Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: experimental analysis of power, Inform. Sci. 180 (2010) 2044–2064.

[28] R.Y. Wang, V.C. Storey, C.P. Firth, A framework for analysis of data quality research, IEEE Trans. Knowl. Data Eng. 7 (4) (1995) 623–640.

[29] C.M. Teng, Polishing blemishes: issues in data correction, IEEE Intell. Syst. 19 (2004) 34–39.

[30] G.H. John, Robust decision trees: removing outliers from databases, in: Proceedings of the First International Conference on Knowledge Discovery and Data Mining, AAAI Press, 1995, pp. 174–179.

[31] R.J. Hickey, Noise modelling and evaluating learning from examples, Artif. Intell. 82 (1–2) (1996) 157–179.

[32] A.P. Dawid, A.M. Skene, Maximum likelihood estimation of observer error-rates using the EM algorithm, Appl. Stat. 28 (1) (1979) 20–28.

[33] A. Malossini, E. Blanzieri, R.T. Ng, Detecting potential labeling errors in microarrays by data perturbation, Bioinformatics 22 (17) (2006) 2114–2121.

[34] G. Lugosi, Learning with an unreliable teacher, Pattern Recogn. 25 (1) (1992) 79–87.

[35] J.R. Quinlan, Induction of decision trees, Machine Learn. 1 (1) (1986) 81–106.

[36] I. Kononenko, M. Kukar, Machine Learning and Data Mining: Introduction to Principles and Algorithms, Horwood Publishing Limited, 2007.

[37] S. le Cessie, J. van Houwelingen, Ridge estimators in logistic regression, Appl. Stat. 41 (1) (1992) 191–201.

[38] J. Maudes, J.J. Rodríguez, C. García-Osorio, N. García-Pedrajas, Random feature weights for decision tree ensemble construction, Inform. Fusion 13 (1) (2012) 20–30.

[39] J. Alcalá-Fdez, A. Fernández, J. Luengo, J. Derrac, S. García, L. Sánchez, F. Herrera, Keel data-mining software tool: data set repository, integration of algorithms and experimental analysis framework, J. Multiple-Valu. Logic Soft Comput. 17 (2–3) (2011) 255–287.

[40] M. Lichman, UCI machine learning repository, 2013. <http://archive.ics.uci.edu/ml>.

[41] C.-M. Teng, Correcting noisy data, in: Proceedings of the Sixteenth International Conference on Machine Learning, Morgan Kaufman Publishers, San Francisco, CA, USA, 1999, pp. 239–248.

[42] J. Demšar, Statistical comparisons of classifiers over multiple data sets, J. Machine Learn. Res. 7 (2006) 1–30.

[43] S. García, F. Herrera, An extension on ‘‘Statistical comparisons of classifiers over multiple data sets for all pairwise comparisons’’, J. Machine Learn. Res. 9 (2008) 2677–2694.

[44] S. Holm, A simple sequentially rejective multiple test procedure, Scand. J. Stat. 6 (1979) 65–70.