Chapter 5
Dealing with Noisy Data

Abstract This chapter focuses on the noise imperfections of the data. The presence of noise in data is a common problem that produces several negative consequences in classification problems. Noise is an unavoidable problem, which affects the data collection and data preparation processes in Data Mining applications, where errors commonly occur. The performance of the models built under such circumstances will heavily depend on the quality of the training data, but also on the robustness against the noise of the model learner itself. Hence, problems containing noise are complex problems and accurate solutions are often difficult to achieve without using specialized techniques—particularly if they are noise-sensitive. Identifying the noise is a complex task that will be developed in Sect. 5.1. Once the noise has been identified, the different kinds of such an imperfection are described in Sect. 5.2. From this point on, the two main approaches carried out in the literature are described. On the first hand, modifying and cleaning the data is studied in Sect. 5.3, whereas designing noise robust Machine Learning algorithms is tackled in Sect. 5.4. An empirical comparison between the latest approaches in the specialized literature is made in Sect. 5.5.

5.1 Identifying Noise

Real-world data is never perfect and often suffers from corruptions that may harm interpretations of the data, models built and decisions made. In classification, noise can negatively affect the system performance in terms of classification accuracy, building time, size and interpretability of the classifier built [99, 100]. The presence of noise in the data may affect the intrinsic characteristics of a classification problem. Noise may create small clusters of instances of a particular class in parts of the instance space corresponding to another class, remove instances located in key areas within a concrete class or disrupt the boundaries of the classes and increase overlapping among them. These alterations corrupt the knowledge that can be extracted from the problem and spoil the classifiers built from that noisy data with respect to the original classifiers built from the clean data that represent the most accurate implicit knowledge of the problem [100].

© Springer International Publishing Switzerland 2015

S. García et al., *Data Preprocessing in Data Mining*,
Intelligent Systems Reference Library 72, DOI 10.1007/978-3-319-10247-4_5

Noise is specially relevant in supervised problems, where it alters the relationship between the informative features and the measure output. For this reason noise has been specially studied in classification and regression where noise hinders the knowledge extraction from the data and spoils the models obtained using that noisy data when they are compared to the models learned from clean data from the same problem, which represent the real implicit knowledge of the problem [100]. In this sense, *robustness* [39] is the capability of an algorithm to build models that are insensitive to data corruptions and suffer less from the impact of noise; that is, the more robust an algorithm is, the more similar the models built from clean and noisy data are. Thus, a classification algorithm is said to be more robust than another if the former builds classifiers which are less influenced by noise than the latter. Robustness is considered more important than performance results when dealing with noisy data, because it allows one to know a priori the expected behavior of a learning method against noise in cases where the characteristics of noise are unknown.

Several approaches have been studied in the literature to deal with noisy data and to obtain higher classification accuracies on test data. Among them, the most important are:

- *Robust learners* [8, 75] These are techniques characterized by being less influenced by noisy data. An example of a robust learner is the C4.5 algorithm [75]. C4.5 uses pruning strategies to reduce the chances reduce the possibility that trees overfit to noise in the training data [74]. However, if the noise level is relatively high, even a robust learner may have a poor performance.
- *Data polishing methods* [84] Their aim is to correct noisy instances prior to training a learner. This option is only viable when data sets are small because it is generally time consuming. Several works [84, 100] claim that complete or partial noise correction in training data, with test data still containing noise, improves test performance results in comparison with no preprocessing.
- *Noise filters* [11, 48, 89] identify noisy instances which can be eliminated from the training data. These are used with many learners that are sensitive to noisy data and require data preprocessing to address the problem.

Noise is not the only problem that supervised ML techniques have to deal with. Complex and nonlinear boundaries between classes are problems that may hinder the performance of classifiers and it often is hard to distinguish between such overlapping and the presence of noisy examples. This topic has attracted recent attention with the appearance of works that have indicated relevant issues related to the degradation of performance:

- *Presence of small disjuncts* [41, 43] (Fig. 5.1a) The minority class can be decomposed into many sub-clusters with very few examples in each one, being surrounded by majority class examples. This is a source of difficulty for most learning algorithms in detecting precisely enough those sub-concepts.
- *Overlapping between classes* [26, 27] (Fig. 5.1b) There are often some examples from different classes with very similar characteristics, in particular if they are located in the regions around decision boundaries between classes. These examples refer to overlapping regions of classes.

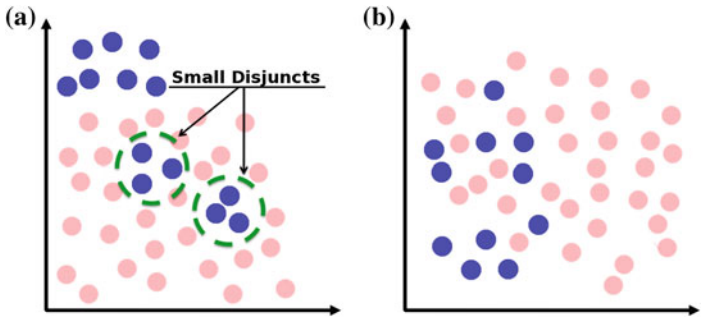

Fig. 5.1 Examples of the interaction between classes: **a** small disjuncts and **b** overlapping between classes

Closely related to the overlapping between classes, in [67] another interesting problem is pointed out: the higher or lower presence of examples located in the area surrounding class boundaries, which are called borderline examples. Researchers have found that misclassification often occurs near class boundaries where overlapping usually occurs as well and it is hard to find a feasible solution [25]. The authors in [67] showed that classifier performance degradation was strongly affected by the quantity of borderline examples and that the presence of other noisy examples located farther outside the overlapping region was also very difficult for re-sampling methods.

- *Safe examples* are placed in relatively homogeneous areas with respect to the class label.
- *Borderline examples* are located in the area surrounding class boundaries, where either the minority and majority classes overlap or these examples are very close to the difficult shape of the boundary—in this case, these examples are also difficult as a small amount of the attribute noise can move them to the wrong side of the decision boundary [52].
- *Noisy examples* are individuals from one class occurring in the safe areas of the other class. According to [52] they could be treated as examples affected by class label noise. Notice that the term *noisy examples* will be further used in this book in the wider sense of [100] where noisy examples are corrupted either in their attribute values or the class label.

The examples belonging to the two last groups often do not contribute to correct class prediction [46]. Therefore, one could ask a question whether removing them (all or the most difficult misclassification part) should improve classification performance. Thus, this book examines the usage of noise filters to achieve this goal, because they are widely used obtaining good results in classification, and in the application of techniques designed to deal with noisy examples.

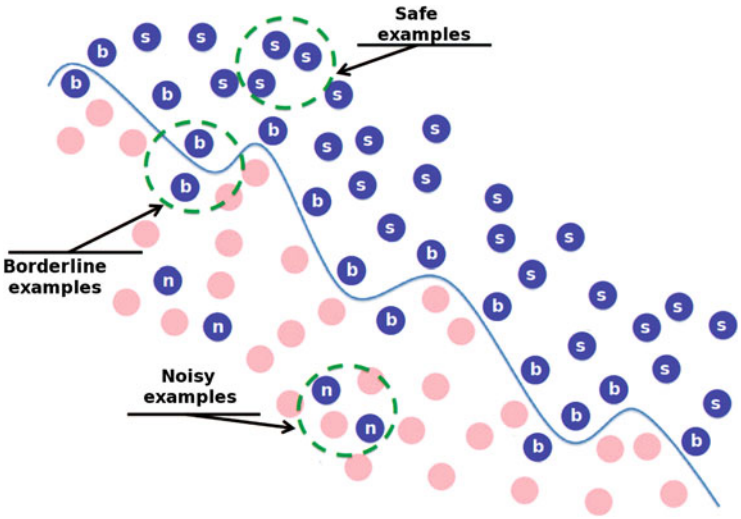

Fig. 5.2 The three types of examples considered in this book: safe examples (labeled as *s*), borderline examples (labeled as *b*) and noisy examples (labeled as *n*). The *continuous line* shows the decision boundary between the two classes

5.2 Types of Noise Data: Class Noise and Attribute Noise

A large number of components determine the quality of a data set [90]. Among them, the class labels and the attribute values directly influence the quality of a classification data set. The quality of the class labels refers to whether the class of each example is correctly assigned; otherwise, the quality of the attributes refers to their capability of properly characterizing the examples for classification purposes—obviously, if noise affects attribute values, this capability of characterization and therefore, the quality of the attributes, is reduced. Based on these two information sources, two types of noise can be distinguished in a given data set [12, 96]:

1. **Class noise** (also referred as *label noise*) It occurs when an example is incorrectly labeled. Class noise can be attributed to several causes, such as subjectivity during the labeling process, data entry errors, or inadequacy of the information used to label each example. Two types of class noise can be distinguished:
   - *Contradictory examples* There are duplicate examples in the data set having different class labels [31].
   - *Misclassifications* Examples are labeled with class labels different from their true label [102].
2. **Attribute noise** It refers to corruptions in the values of one or more attributes. Examples of attribute noise are: erroneous attribute values, missing or unknown attribute values, and incomplete attributes or "do not care" values.

In this book, class noise refers to misclassifications, whereas attribute noise refers to erroneous attribute values, because they are the most common in real-world data [100]. Furthermore, erroneous attribute values, unlike other types of attribute noise, such as MVs (which are easily detectable), have received less attention in the literature.

Treating class and attribute noise as corruptions of the class labels and attribute values, respectively, has been also considered in other works in the literature [69, 100]. For instance, in [100], the authors reached a series of interesting conclusions, showing that attribute noise is more harmful than class noise or that eliminating or correcting examples in data sets with class and attribute noise, respectively, may improve classifier performance. They also showed that attribute noise is more harmful in those attributes highly correlated with the class labels. In [69], the authors checked the robustness of methods from different paradigms, such as probabilistic classifiers, decision trees, instance based learners or SVMs, studying the possible causes of their behavior.

However, most of the works found in the literature are only focused on class noise. In [9], the problem of multi-class classification in the presence of labeling errors was studied. The authors proposed a generative multi-class classifier to learn with labeling errors, which extends the multi-class quadratic normal discriminant analysis by a model of the mislabeling process. They demonstrated the benefits of this approach in terms of parameter recovery as well as improved classification performance. In [32], the problems caused by labeling errors occurring far from the decision boundaries in Multi-class Gaussian Process Classifiers were studied. The authors proposed a Robust Multi-class Gaussian Process Classifier, introducing binary latent variables that indicate when an example is mislabeled. Similarly, the effect of mislabeled samples appearing in gene expression profiles was studied in [98]. A detection method for these samples was proposed, which takes advantage of the measuring effect of data perturbations based on the SVM regression model. They also proposed three algorithms based on this index to detect mislabeled samples. An important common characteristic of these works, also considered in this book, is that the suitability of the proposals was evaluated on both real-world and synthetic or noisy-modified real-world data sets, where the noise could be somehow quantified.

In order to model class and attribute noise, we consider four different synthetic noise schemes found in the literature, so that we can simulate the behavior of the classifiers in the presence of noise as presented in the next section.

5.2.1 Noise Introduction Mechanisms

Traditionally the label noise introduction mechanism has not attracted as much attention in its consequences as it has in the knowledge extracted from it. However, as the noise treatment is being embedded in the classifier design, the nature of noise becomes more and more important. Recently, the authors in Frenay and Verleysen [19] have adopted the statistical analysis for the MVs introduction described

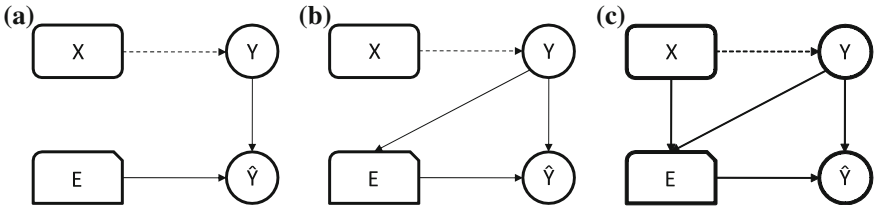

Fig. 5.3 Statistical taxonomy of label noise as described in [19]. **a** Noisy completely at random (NCAR), **b** Noisy at random (NAR), and **c** Noisy not at random (NNAR). X is the array of input attributes, Y is the true class label, $\hat{Y}$ is the actual class label and E indicates whether a labeling error occurred ($Y \neq \hat{Y}$). *Arrows* indicate statistical dependencies

in Sect. 4.2. That is, we will distinguish between three possible statistical models for label noise as depicted in Fig. 5.3. In the three subfigures of Fig. 5.3 the dashed arrow points out a the implicit relation between the input features and the output that is desired to be modeled by the classifier. In the most simplistic case in which the noise procedure is not dependent of either the true value of the class Y or the input attribute values X, the label noise is called noise completely at random or NCAR as shown in Fig. 5.3a. In [7] the observed label is different from the true class with a probability $p_n = P(E = 1)$, that is also called the error rate or noise rate. In binary classification problems, the labeling error in NCAR is applied symmetrically to both class labels and when $p_n = 0.5$ the labels will no longer provide useful information. In multiclass problems when the error caused by noise (i.e. $E = 1$) appears the class label is changed by any other different one available. In the case in which the selection of the erroneous class label is made by a uniform probability distribution, the noise model is known as *uniform label/class noise*.

Things get more complicated in the noise at random (NAR) model. Although the noise is independent of the inputs X, the true value of the class make it more or less prone to be noisy. This asymmetric labeling error can be produced by the different cost of extracting the true class, as for example in medical case-control studies, financial score assets and so on. Since the wrong class label is subject to a particular true class label, the labeling probabilities can be defined as:

$$P(\hat{Y} = \hat{y}|Y = y) = \sum_{e \in 0,1} P(\hat{Y} = \hat{y}|E = e, Y = y)P(E = e|Y = y). \quad (5.1)$$

Of course this probability definition span over all the class labels and the possibly erroneous class that the could take. As shown in [70] this conforms a transition matrix γ where each position γ_{ij} shows the probability of $P(\hat{Y} = c_i|Y = c_j)$ for the possible class labels c_i and c_j. Some examples can be examined with detail in [19]. The NCAR model is a special case of the NAR label noise model in which the probability of each position γ_{ij} denotes the independency between $\hat{Y}$ and Y: $\gamma_{ij} = P(\hat{Y}_i, Y = c_j)$.

Apart from the uniform class noise, the NAR label noise has widely studied in the literature. An example is the pairwise label noise, where two selected class labels are chosen to be labeled with the other with certain probability. In this pairwise label noise (or pairwise class noise) only two positions of the γ matrix are nonzero outside of the diagonal. Another problem derived from the NAR noise model is that it is not trivial to decide whether the class labels are useful or not.

The third and last noise model is the noisy not at random (NNAR), where the input attributes somehow affect the probability of the class label being erroneous as shown in Fig. 5.3c. An example of this illustrated by Klebanov [49] where evidence is given that difficult samples are randomly labeled.It also occurs that those examples similar to existing ones are labeled by experts in a biased way, having more probability of being mislabeled the more similar they are. NNAR model is the more general case of class noise [59] where the error E depends on both X and Y and it is the only model able to characterize mislabelings in the class borders or due to poor sampling density. As shown in [19] the probability of error is much more complex than in the two previous cases as it has to take into account the density function of the input over the input feature space $\mathfrak{X}$ when continuous:

$$p_n = P(E = 1) = \sum_{c_i \in C} \times \int_{x \in \mathfrak{X}} P(X = x|Y = y)P(E = 1|X = x, Y = y)dx. \tag{5.2}$$

As a consequence the perfect identification and estimation of the NNAR noise is almost impossible, relying in approximating it from the expert knowledge of the problem and the domain.

In the case of attribute noise, the modelization described above can be extended and adapted. In this case, we can distinguish three possibilities as well:

- When the noise appearance does not depend either on the rest of the input features' values or the class label the NCAR noise model applies. This type of noise can occur when distortions in the measures appear at random, for example in faulty hand data insertion or network errors that do not depend in the data content itself.
- When the attribute noise depends on the true value x_i but not on the rest of input values $x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n$ or the observed class label y the NAR model is applicable. An illustrative example is when the different temperatures affect their registration in climatic data in a different way depending on the proper temperature value.
- In the last case the noise probability will depend on the value of the feature x_i but also on the rest of the input feature values $x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n$. This is a very complex situation in which the value is altered when the rest of features present a particular combination of values, as in medical diagnosis when some test results are filled by an expert prediction without conducting the test due to high costs.

For the sake of brevity we will not develop the probability error equations here as their expressions would vary depending on the nature of the input feature, being different

from real valued ones with respect to nominal attributes. However we must point out that in attribute noise the probability dependencies are not the only important aspect to be considered. The probability distribution of the noise is also fundamental.

For numerical data, the noisy datum $\hat{x}_i$ may be a slight variation of the true value x_i or a completely random value. The density function of the noise values is very rarely known. Simple examples of the first type of noise would be perturbations caused by a normal distribution with the mean centered in the true value and with a fixed variance. The second type of noise is usually estimated by assigning an uniform probability to all the possible values of the input feature's range. This procedure is also typical with nominal data, where no preference of one value is taken. Again note that the distribution of the noise is not the same as the probability of its appearance discussed above: first the noise must be introduced with a certain probability (following the NCAR, NAR or NNAR models) and then the noise value is stated or analyzed to follow the aforementioned density functions.

5.2.2 Simulating the Noise of Real-World Data Sets

Checking the effect of noisy data on the performance of classifier learning algorithms is necessary to improve their reliability and has motivated the study of how to generate and introduce noise into the data. Noise generation can be characterized by three main characteristics [100]:

1. **The place where the noise is introduced** Noise may affect the input attributes or the output class, impairing the learning process and the resulting model.
2. **The noise distribution** The way in which the noise is present can be, for example, uniform [84, 104] or Gaussian [100, 102].
3. **The magnitude of generated noise values** The extent to which the noise affects the data set can be relative to each data value of each attribute, or relative to the minimum, maximum and standard deviation for each attribute [100, 102, 104].

In contrast to other studies in the literature, this book aims to clearly explain how noise is defined and generated, and also to properly justify the choice of the noise introduction schemes. Furthermore, the noise generation software has been incorporated into the KEEL tool (see Chap. 10) for its free usage. The two types of noise considered in this work, class and attribute noise, have been modeled using four different noise schemes; in such a way that, the presence of these types of noise will allow one to simulate the behavior of the classifiers in these two scenarios:

1. **Class noise** usually occurs on the boundaries of the classes, where the examples may have similar characteristics—although it can occur in any other area of the domain. In this book, class noise is introduced using an *uniform class noise scheme* [84] (randomly corrupting the class labels of the examples) and a *pairwise class noise scheme* [100, 102] (labeling examples of the majority class with the second majority class). Considering these two schemes, noise affecting any class label and noise affecting only the two majority classes is simulated respectively.

Whereas the former can be used to simulate a NCAR noise model, the latter is useful to produce a particular NAR noise model.

2. **Attribute noise** can proceed from several sources, such as transmission constraints, faults in sensor devices, irregularities in sampling and transcription errors [85]. The erroneous attribute values can be totally unpredictable, i.e., random, or imply a low variation with respect to the correct value. We use the *uniform attribute noise scheme* [100, 104] and the *Gaussian attribute noise scheme* in order to simulate each one of the possibilities, respectively. We introduce attribute noise in accordance with the hypothesis that interactions between attributes are weak [100]; as a consequence, the noise introduced into each attribute has a low correlation with the noise introduced into the rest.

Robustness is the capability of an algorithm to build models that are insensitive to data corruptions and suffer less from the impact of noise [39]. Thus, a classification algorithm is said to be more robust than another if the former builds classifiers which are less influenced by noise than the latter, i.e., more robust. In order to analyze the degree of robustness of the classifiers in the presence of noise, we will compare the performance of the classifiers learned with the original (without induced noise) data set with the performance of the classifiers learned using the noisy data set. Therefore, those classifiers learned from noisy data sets being more similar (in terms of results) to the noise free classifiers will be the most robust ones.

5.3 Noise Filtering at Data Level

Noise filters are preprocessing mechanisms to detect and eliminate noisy instances in the training set. The result of noise elimination in preprocessing is a reduced training set which is used as an input to a classification algorithm. The separation of noise detection and learning has the advantage that noisy instances do not influence the classifier building design [24].

Noise filters are generally oriented to detect and eliminate instances with class noise from the training data. Elimination of such instances has been shown to be advantageous [23]. However, the elimination of instances with attribute noise seems counterproductive [74, 100] since instances with attribute noise still contain valuable information in other attributes which can help to build the classifier. It is also hard to distinguish between noisy examples and true exceptions, and henceforth many techniques have been proposed to deal with noisy data sets with different degrees of success.

We will consider three noise filters designed to deal with mislabeled instances as they are the most common and the most recent: the *Ensemble Filter* [11], the *Cross-Validated Committees Filter* [89] and the *Iterative-Partitioning Filter* [48]. It should be noted that these three methods are ensemble-based and vote-based filters. A motivation for using ensembles for filtering is pointed out in [11]: when it is assumed that some instances in the data have been mislabeled and that the label errors

are independent of the particular model being fitted to the data, collecting information from different models will provide a better method for detecting mislabeled instances than collecting information from a single model.

The implementations of these three noise filters can be found in KEEL (see Chap. 10). Their descriptions can be found in the following subsections. In all descriptions we use D_T to refer to the training set, D_N to refer to the noisy data identified in the training set (initially, $D_N = \emptyset$) and Γ is the number of folds in which the training data is partitioned by the noise filter.

The three noise filters presented below use a voting scheme to determine which instances to eliminate from the training set. There are two possible schemes to determine which instances to remove: consensus and majority schemes. The consensus scheme removes an instance if it is misclassified by all the classifiers, while the majority scheme removes an instance if it is misclassified by more than half of the classifiers. Consensus filters are characterized by being conservative in rejecting good data at the expense of retaining bad data. Majority filters are better at detecting bad data at the expense of rejecting good data.

5.3.1 Ensemble Filter

The *Ensemble Filter* (EF) [11] is a well-known filter in the literature. It attempts to achieve an improvement in the quality of the training data as a preprocessing step in classification, by detecting and eliminating mislabeled instances. It uses a set of learning algorithms to create classifiers in several subsets of the training data that serve as noise filters for the training set.

The identification of potentially noisy instances is carried out by performing an Γ-FCV on the training data with μ classification algorithms, called filter algorithms. In the developed experimentation for this book we have utilized the three filter algorithms used by the authors in [11], which are C4.5, 1-NN and LDA [63]. The complete process carried out by EF is described below:

- Split the training data set D_T into Γ equal sized subsets.
- For each one of the μ filter algorithms:
  - For each of these Γ parts, the filter algorithm is trained on the other $\Gamma - 1$ parts. This results in Γ different classifiers.
  - These Γ resulting classifiers are then used to tag each instance in the excluded part as either correct or mislabeled, by comparing the training label with that assigned by the classifier.
- At the end of the above process, each instance in the training data has been tagged by each filter algorithm.
- Add to D_N the noisy instances identified in D_T using a voting scheme, taking into account the correctness of the labels obtained in the previous step by the μ filter algorithms. We use a consensus vote scheme in this case.
- Remove the noisy instances from the training set: $D_T \leftarrow D_T \setminus D_N$.

5.3.2 Cross-Validated Committees Filter

The *Cross-Validated Committees Filter* (CVCF) [89] uses ensemble methods in order to preprocess the training set to identify and remove mislabeled instances in classification data sets. CVCF is mainly based on performing an Γ-FCV to split the full training data and on building classifiers using decision trees in each training subset. The authors of CVCF place special emphasis on using ensembles of decision trees such as C4.5 because they think that this kind of algorithm works well as a filter for noisy data.

The basic steps of CVCF are the following:

- Split the training data set D_T into Γ equal sized subsets.
- For each of these Γ parts, a base learning algorithm is trained on the other $\Gamma - 1$ parts. This results in Γ different classifiers. We use C4.5 as base learning algorithm in our experimentation as recommended by the authors.
- These Γ resulting classifiers are then used to tag each instance in the training set D_T as either correct or mislabeled, by comparing the training label with that assigned by the classifier.
- Add to D_N the noisy instances identified in D_T using a voting scheme (the majority scheme in our experimentation), taking into account the correctness of the labels obtained in the previous step by the Γ classifier built.
- Remove the noisy instances from the training set: $D_T \leftarrow D_T \setminus D_N$.

5.3.3 Iterative-Partitioning Filter

The *Iterative-Partitioning Filter* (IPF) [48] is a preprocessing technique based on the *Partitioning Filter* [102]. It is employed to identify and eliminate mislabeled instances in large data sets. Most noise filters assume that data sets are relatively small and capable of being learned after only one time, but this is not always true and partitioning procedures may be necessary.

IPF removes noisy instances in multiple iterations until a stopping criterion is reached. The iterative process stops if, for a number of consecutive iterations s, the number of identified noisy instances in each of these iterations is less than a percentage p of the size of the original training data set. Initially, we have a set of noisy instances $D_N = \emptyset$ and a set of good data $D_G = \emptyset$. The basic steps of each iteration are:

- Split the training data set D_T into Γ equal sized subsets. Each of these is small enough to be processed by an induction algorithm once.
- For each of these Γ parts, a base learning algorithm is trained on this part. This results in Γ different classifiers. We use C4.5 as the base learning algorithm in our experimentation as recommended by the authors.

- These Γ resulting classifiers, are then used to tag each instance in the training set D_T as either correct or mislabeled, by comparing the training label with that assigned by the classifier.
- Add to D_N the noisy instances identified in D_T using a voting scheme, taking into account the correctness of the labels obtained in the previous step by the Γ classifier built. For the IPF filter we use the majority vote scheme.
- Add to D_G a percentage y of the good data in D_T. This step is useful when we deal with large data sets because it helps to reduce them faster. We do not eliminate good data with the IPF method in our experimentation (we set $y = 0$, so D_G is always empty) and nor do we lose generality.
- Remove the noisy instances and the good data from the training set: $D_T \leftarrow D_T \backslash \{D_N \cup D_G\}$.

At the end of the iterative process, the filtered data is formed by the remaining instances of D_T and the good data of D_G; that is, $D_T \cup D_G$.

A particularity of the voting schemes in IPF is that a noisy instance should also be misclassified by the model which was induced in the subset containing that instance as an additional condition. Moreover, by varying the required number of filtering iterations, the level of conservativeness of the filter can also be varied in both schemes, consensus and majority.

5.3.4 More Filtering Methods

Apart from the three aforementioned filtering methods, we can find many more in the specialized literature. We try to provide a helpful summary of the most recent and well-known ones in the following Table 5.1. For the sake of brevity, we will not carry out a deep description of these methods as done in the previous sections. A recent categorization of the different filtering procedures made by Frenay and Verleysen [19] will be followed as it matches our descriptions well.

5.4 Robust Learners Against Noise

Filtering the data has also one major drawback: some instances will be dropped from the data sets, even if they are valuable. Instead of filtering the data set or modifying the learning algorithm, we can use other approaches to diminish the effect of noise in the learned model. In the case of labeled data, one powerful approach is to train not a single classifier but several ones, taking advantage of their particular strengths. In this section we provide a brief insight into classifiers that are known to be robust to noise to a certain degree, even when the noise is not treated or cleansed. As said in Sect. 5.1 C4.5 has been considered as a paradigmatic robust learner against noise. However, it is also true that classical decision trees have been considered

Table 5.1 Filtering approaches by category as of [19]

Detection based on thresholding of a measure		Partition filtering for large data sets	
Measure: classification confidence	[82]	For large and distributed data sets	[102, 103]
Least complex correct hypothesis	[24]	*Model influence*	
Classifier predictions based		LOOPC	[57]
Cost sensitive learning based	[101]	Single perceptron perturbation	[33]
SVM based	[86]	*Nearest neighbor based*	
ANN based	[42]	CNN	[30]
Multi classifier system	[65]	BBNR	[15]
C4.5	[44]	IB3	[3]
Nearest instances to a candidate	[78, 79]	Tomek links	[88]
Voting filtering		PRISM	[81]
Ensembles	[10, 11]	DROP	[93]
Bagging	[89]	*Graph connections based*	
ORBoost	[45]	Grabiel graphs	[18]
Edge analysis	[92]	Neighborhood graphs	[66]

sensitive to class noise as well [74]. This instability has make them very suitable for ensemble methods. As a countermeasure for this lack of stability some strategies can be used. The first one is to carefully select an appropriate splitting criteria measure. In [2] several measures are compared to minimize the impact of label noise in the constructed trees, empirically showing that the imprecise info-gain measure is able to improve the accuracy and reduce the tree growing size produced by the noise.

Another approach typically described as useful to deal with noise in decision trees is the use of pruning. Pruning tries to stop the overfitting caused by the overspecialization over the isolated (and usually noisy) examples. The work of [1] eventually shows that the usage of pruning helps to reduce the effect and impact of the noise in the modeled trees. C4.5 is the most famous decision tree and it includes this pruning strategy by default, and can be easily adapted to split under the desired criteria.

We have seen that the usage of ensembles is a good strategy to create accurate and robust filters. Whether an ensemble of classifiers is robust or not against noise can be also asked.

Many ensemble approaches exist and their noise robustness has been tested. An ensemble is a system where the base learners are all of the same type built to be as varied as possible. The two most classic approaches bagging and boosting were compared in [16] showing that bagging obtains better performance than boosting when label noise is present. The reason shown in [1] indicates that boosting (or the particular implementation made by AdaBoost) increase the weights of noisy instances too much, making the model construction inefficient and imprecise, whereas mislabeled instances favour the variability of the base classifiers in bagging [19]. As AdaBoost is not the only boosting algorithm, other implementations as LogitBoost and BrownBoost have been checked as more robust to class noise [64]. When the base

classifiers are different we talk of Multiple Classifier Systems (MCSs). They are thus a generalization of the classic ensembles and they should offer better improvements in noisy environments. They are tackled in Sect. 5.4.1.

We can separate the labeled instances in several "bags" or groups, each one containing only those instances belonging to the same class. This type of decomposition is well suited for those classifiers that can only work with binary classification problems, but has also been suggested that this decomposition can help to diminish the effects of noise. This decomposition is expected to decrease the overlapping between the classes and to limit the effect of noisy instances to their respective bags by simplifying the problem and thus alleviating the effect of the noise if the whole data set were considered.

5.4.1 Multiple Classifier Systems for Classification Tasks

Given a set of problems, finding the best overall classification algorithm is sometimes difficult because some classifiers may excel in some cases and perform poorly in others. Moreover, even though the optimal match between a learning method and a problem is usually sought, this match is generally difficult to achieve and perfect solutions are rarely found for complex problems [34, 36]. This is one reason for using Multi-Classifier Systems [34, 36, 72], since it is not necessary to choose a specific learning method. All of them might be used, taking advantage of the strengths of each method, while avoiding its weaknesses. Furthermore, there are other motivations to combine several classifiers [34]:

- To avoid the choice of some arbitrary but important initial conditions, e.g. those involving the parameters of the learning method.
- To introduce some randomness to the training process in order to obtain different alternatives that can be combined to improve the results obtained by the individual classifiers.
- To use complementary classification methods to improve dynamic adaptation and flexibility.

Several works have claimed that simultaneously using classifiers of different types, complementing each other, improves classification performance on difficult problems, such as satellite image classification [60], fingerprint recognition [68] and foreign exchange market prediction [73]. Multiple Classifier Systems [34, 36, 72, 94] are presented as a powerful solution to these difficult classification problems, because they build several classifiers from the same training data and therefore allow the simultaneous usage of several feature descriptors and inference procedures. An important issue when using MCSs is the way of creating *diversity* among the classifiers [54], which is necessary to create discrepancies among their decisions and hence, to take advantage of their combination.

MCSs have been traditionally associated with the capability of working accurately with problems involving noisy data [36]. The main reason supporting this

hypothesis could be the same as one of the main motivations for combining classifiers: the improvement of the generalization capability (due to the complementarity of each classifier), which is a key question in noisy environments, since it might allow one to avoid the overfitting of the new characteristics introduced by the noisy examples [84]. Most of the works studying MCSs and noisy data are focused on techniques like bagging and boosting [16, 47, 56], which introduce diversity considering different samples of the set of training examples and use only one baseline classifier. For example, in [16] the suitability of randomization, bagging and boosting to improve the performance of C4.5 was studied. The authors reached the conclusion that with a low noise level, boosting is usually more accurate than bagging and randomization. However, bagging outperforms the other methods when the noise level increases. Similar conclusions were obtained in the paper of Maclin and Opitz [56]. Other works [47] compare the performance of boosting and bagging techniques dealing with imbalanced and noisy data, reaching also the conclusion that bagging methods generally outperforms boosting ones. Nevertheless, explicit studies about the adequacy of MCSs (different from bagging and boosting, that is, those introducing diversity using different base classifiers) to deal with noisy data have not been carried out yet. Furthermore, most of the existing works are focused on a concrete type of noise and on a concrete combination rule. On the other hand, when data is suffering from noise, a proper study on how the robustness of each single method influences the robustness of the MCS is necessary, but this fact is usually overlooked in the literature.

There are several strategies to use more than one classifier for a single classification task [36]:

- *Dynamic classifier selection* This is based on the fact that one classifier may outperform all others using a global performance measure but it may not be the best in all parts of the domain. Therefore, these types of methods divide the input domain into several parts and aim to select the classifier with the best performance in that part.
- *Multi-stage organization* This builds the classifiers iteratively. At each iteration, a group of classifiers operates in parallel and their decisions are then combined. A dynamic selector decides which classifiers are to be activated at each stage based on the classification performances of each classifier in previous stages.
- *Sequential approach* A classifier is used first and the other ones are used only if the first does not yield a decision with sufficient confidence.
- *Parallel approach* All available classifiers are used for the same input example in parallel. The outputs from each classifier are then combined to obtain the final prediction.

Although the first three approaches have been explored to a certain extent, the majority of classifier combination research focuses on the fourth approach, due to its simplicity and the fact that it enables one to take advantage of the factors presented in the previous section. For these reasons, this book focus on the fourth approach.

5.4.1.1 Decisions Combination in Multiple Classifiers Systems

As has been previously mentioned, parallel approaches need a posterior phase of combination after the evaluation of a given example by all the classifiers. Many decisions combination proposals can be found in the literature, such as the intersection of decision regions [29], voting methods [62], prediction by top choice combinations [91], use of the Dempster–Shafer theory [58, 97] or ranking methods [36]. In concrete, we will study the following four combination methods for the MCSs built with heterogeneous classifiers:

1. **Majority vote (MAJ)** [62] This is a simple but powerful approach, where each classifier gives a vote to the predicted class and the one with the most votes is chosen as the output.
2. **Weighted majority vote (W-MAJ)** [80] Similarly to MAJ, each classifier gives a vote for the predicted class, but in this case, the vote is weighted depending on the competence (accuracy) of the classifier in the training phase.
3. **Naïve Bayes** [87] This method assumes that the base classifiers are mutually independent. Hence, the predicted class is the one that obtains the highest posterior probability. In order to compute these probabilities, the confusion matrix of each classifier is considered.
4. **Behavior-Knowledge Space (BKS)** [38] This is a multinomial method that indexes a cell in a look-up table for each possible combination of classifiers outputs. A cell is labeled with the class to which the majority of the instances in that cell belong to. A new instance is classified by the corresponding cell label; in case the cell is not labeled or there is a tie, the output is given by MAJ.

We always use the same training data set to train all the base classifiers and to compute the parameters of the aggregation methods, as is recommended in [53]. Using a separate set of examples to obtain such parameters can imply some important training data to be ignored and this fact is generally translated into a loss of accuracy of the final MCS built.

In MCSs built with heterogeneous classifiers, all of them may not return a confidence value. Even though each classifier can be individually modified to return a confidence value for its predictions, such confidences will come from different computations depending on the classifier adapted and their combination could become meaningless. Nevertheless, in MCSs built with the same type of classifier, this fact does not occur and it is possible to combine their confidences since these are homogeneous among all the base classifiers [53]. Therefore, in the case of bagging, given that the same classifier is used to train all the base classifiers, the confidence of the prediction can be used to compute a weight and, in turn, these weights can be used in a weighted voting combination scheme.

5.4.2 Addressing Multi-class Classification Problems by Decomposition

Usually, the more classes in a problem, the more complex it is. In multi-class learning, the generated classifier must be able to separate the data into more than a pair of classes, which increases the chances of incorrect classifications (in a two-class balanced problem, the probability of a correct random classification is 1/2, whereas in a multi-class problem it is 1/M). Furthermore, in problems affected by noise, the boundaries, the separability of the classes and therefore, the prediction capabilities of the classifiers may be severely hindered.

When dealing with multi-class problems, several works [6, 50] have demonstrated that decomposing the original problem into several binary subproblems is an easy, yet accurate way to reduce their complexity. These techniques are referred to as binary decomposition strategies [55]. The most studied schemes in the literature are: *One-vs-One* (OVO) [50], which trains a classifier to distinguish between each pair of classes, and *One-vs-All* (OVA) [6], which trains a classifier to distinguish each class from all other classes. Both strategies can be encoded within the Error Correcting Output Codes framework [5, 17]. However, none of these works provide any theoretical nor empirical results supporting the common assumption that assumes a better behavior against noise of decomposition techniques compared to not using decomposition. Neither do they show what type of noise is better handled by decomposition techniques.

Consequently, we can consider the usage of the OVO strategy, which generally out-stands over OVA [21, 37, 76, 83], and check its suitability with noisy training data. It should be mentioned that, in real situations, the existence of noise in the data sets is usually unknown-therefore, neither the type nor the quantity of noise in the data set can be known or supposed *a priori*. Hence, tools which are able to manage the presence of noise in the data sets, despite its type or quantity (or unexistence), are of great interest. If the OVO strategy (which is a simple yet effective methodology when clean data sets are considered) is also able to properly (better than the baseline non-OVO version) handle the noise, its usage could be recommended in spite of the presence of noise and without taking into account its type. Furthermore, this strategy can be used with any of the existing classifiers which are able to deal with two-class problems. Therefore, the problems of algorithm level modifications and preprocessing techniques could be avoided; and if desired, they could also be combined.

5.4.2.1 Decomposition Strategies for Multi-class Problems

Several motivations for the usage of binary decomposition strategies in multi-class classification problems can be found in the literature [20, 21, 37, 76]:

- The separation of the classes becomes easier (less complex), since less classes are considered in each subproblem [20, 61]. For example, in [51], the classes in a

digit recognition problem were shown to be linearly separable when considered in pairs, becoming a simpler alternative than learning a unique non-linear classifier over all classes simultaneously.
- Classification algorithms, whose extension to multi-class problems is not easy, can address multi-class problems using decomposition techniques [20].
- In [71], the advantages of using decomposition were pointed out when the classification errors for different classes have distinct costs. The binarization allows the binary classifiers generated to impose preferences for some of the classes.
- Decomposition allows one to easily parallelize the classifier learning, since the binary subproblems are independent and can be solved with different processors.

Dividing a problem into several new subproblems, which are then independently solved, implies the need of a second phase where the outputs of each problem need to be aggregated. Therefore, decomposition includes two steps:

1. *Problem division*. The problem is decomposed into several binary subproblems which are solved by independent binary classifiers, called *base classifiers* [20]. Different decomposition strategies can be found in the literature [55]. The most common one is OVO [50].
2. *Combination of the outputs*. [21] The different outputs of the binary classifiers must be aggregated in order to output the final class prediction. In [21], an exhaustive study comparing different methods to combine the outputs of the base classifiers in the OVO and OVA strategies is developed. Among these combination methods, the Weighted Voting [40] and the approaches in the framework of probability estimates [95] are highlighted.

This book focuses the OVO decomposition strategy due to the several advantages shown in the literature with respect to OVA [20, 21, 37, 76]:

- OVO creates simpler borders between classes than OVA.
- OVO generally obtains a higher classification accuracy and a shorter training time than OVA because the new subproblems are easier and smaller.
- OVA has more of a tendency to create imbalanced data sets which can be counterproductive [22, 83].
- The application of the OVO strategy is widely extended and most of the software tools considering binarization techniques use it as default [4, 13, 28].

5.4.2.2 One-vs-One Decomposition Scheme

The OVO decomposition strategy consists of dividing a classification problem with M classes into $M(M-1)/2$ binary subproblems. A classifier is trained for each new subproblem only considering the examples from the training data corresponding to the pair of classes (λ_i, λ_j) with $i < j$ considered.

When a new instance is going to be classified, it is presented to all the the binary classifiers. This way, each classifier discriminating between classes λ_i and λ_j provides a confidence degree $r_{ij} \in [0, 1]$ in favor of the former class (and hence, r_{ji} is

computed by $1 - r_{ij}$). These outputs are represented by a score matrix R:

$$R = \begin{pmatrix} - & r_{12} & \cdots & r_{1M} \\ r_{21} & - & \cdots & r_{2M} \\ \vdots & & & \vdots \\ r_{M1} & r_{M2} & \cdots & - \end{pmatrix} \tag{5.3}$$

The final output is derived from the score matrix by different aggregation models. The most commonly used and simplest combination, also considered in the experiments of this book, is the application of a voting strategy:

$$Class = \arg\max_{i=1,\ldots,M} \sum_{1 \le j \ne i \le M} s_{ij} \tag{5.4}$$

where s_{ij} is 1 if $r_{ij} > r_{ji}$ and 0 otherwise. Therefore, the class with the largest number of votes will be predicted. This strategy has proved to be competitive with different classifiers obtaining similar results in comparison with more complex strategies [21].

5.5 Empirical Analysis of Noise Filters and Robust Strategies

In this section we want to illustrate the advantages of the noise approaches described above.

5.5.1 Noise Introduction

In the data sets we are going to use (taken from Chap. 2), as in most of the real-world data sets, the initial amount and type of noise present is unknown. Therefore, no assumptions about the base noise type and level can be made. For this reason, these data sets are considered to be noise free, in the sense that no recognizable noise has been introduced. In order to control the amount of noise in each data set and check how it affects the classifiers, noise is introduced into each data set in a supervised manner. Four different noise schemes proposed in the literature, as explained in Sect. 5.2, are used in order to introduce a noise level x% into each data set:

1. **Introduction of class noise.**
   - **Uniform class noise** [84] x% of the examples are corrupted. The class labels of these examples are randomly replaced by another one from the M classes.
   - **Pairwise class noise** [100, 102] Let X be the majority class and Y the second majority class, an example with the label X has a probability of $x/100$ of being incorrectly labeled as Y.

2. **Introduction of attribute noise**
   - **Uniform attribute noise** [100, 104] x% of the values of each attribute in the data set are corrupted. To corrupt each attribute A_i, x% of the examples in the data set are chosen, and their A_i value is assigned a random value from the domain $\mathbb{D}_i$ of the attribute A_i. An uniform distribution is used either for numerical or nominal attributes.
   - **Gaussian attribute noise** This scheme is similar to the uniform attribute noise, but in this case, the A_i values are corrupted, adding a random value to them following Gaussian distribution of $mean = 0$ and *standard deviation* = (*max-min*)*/5*, being *max* and *min* the limits of the attribute domain ($\mathbb{D}_i$). Nominal attributes are treated as in the case of the uniform attribute noise.

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

1. A level of noise $x\%$, of either class noise (uniform or pairwise) or attribute noise (uniform or Gaussian), is introduced into a copy of the full original data set.
2. Both data sets, the original and the noisy copy, are partitioned into 5 equal 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 data set, that is, the noise free data set.

We introduce noise, either class or attribute noise, only into the training sets since we want to focus on the effects of noise on the training process. This will be carried out observing how the classifiers built from different noisy training data for a particular data set behave, considering the accuracy of those classifiers, with the same clean test data. Thus, the accuracy of the classifier built over the original training set without additional noise acts as a reference value that can be directly compared with the accuracy of each classifier obtained with the different noisy training data. Corrupting the test sets also affects the accuracy obtained by the classifiers and therefore, our conclusions will not only be limited to the effects of noise on the training process.

The accuracy estimation of the classifiers in a data set is obtained by means of 5 runs of a stratified 5-FCV. Hence, a total of 25 runs per data set, noise type and level are averaged. 5 partitions are used because, if each partition has a large number of examples, the noise effects will be more notable, facilitating their analysis.

The robustness of each method is estimated with the *relative loss of accuracy* (RLA) (Eq. 5.5), which is used to measure the percentage of variation of the accuracy of the classifiers at a concrete noise level with respect to the original case with no additional noise:

$$RLA_{x\%} = \frac{Acc_{0\%} - Acc_{x\%}}{Acc_{0\%}}, \tag{5.5}$$

where $RLA_{x\%}$ is the relative loss of accuracy at a noise level $x\%$, $Acc_{0\%}$ is the test accuracy in the original case, that is, with 0% of induced noise, and $Acc_{x\%}$ is the test accuracy with a noise level $x\%$.

5.5.2 Noise Filters for Class Noise

The usage of filtering is claimed to be useful in the presence of noise. This section tries to show whether this claim is true or not and to what extent. As a simple but representative case of study, we show the results of applying noise filters based on detecting and eliminating mislabeled training instances. We want to illustrate how applying filters is a good strategy to obtain better results in the presence of even low amounts of noise. As filters are mainly designed for class noise, we will focus on the two types of class noise described in this chapter: the uniform class noise and the pairwise class noise.

Three popular classifiers will be used to obtain the accuracy values that are C4.5, Ripper and a SVM. Their selection is not made at random: SVMs are known to be very accurate but also sensitive to noise. Ripper is a rule learning algorithm able to perform averagely well, but as we saw in Sect. 5.4 rule learners are also sensitive to noise when they are not designed to cope with it. The third classifier is C4.5 using the pruning strategy, that it is known for diminishing the effects of noise in the final tree. Table 5.2 shows the average results for the three noise filters for each kind of class noise studied. The amount of noise ranges from 5 to 20 %, enough to show the differences between no filtering (labeled as "None") and the noise filters. The results shown are the average over all the data sets considered in order to ease the reading.

The evolution of the results and their tendencies can be better depicted by using a graphical representation. Figure 5.4a shows the performance of SVM from an amount of 0 % of controlled pairwise noise to the final 20 % introduced. The accuracy can be seen to drop from an initial amount of 90–85 % by only corrupting 20 % of the class labels. The degradation is even worse in the case of uniform class noise depicted in Fig. 5.4b, as all the class labels can be affected. The evolution of not using any

Table 5.2 Filtering of class noise over three classic classifiers

		Pairwise class noise					Uniform random class noise				
		0%	5%	10%	15%	20%	0%	5%	10%	15%	20%
	None	90.02	88.51	86.97	86.14	84.86	90.02	87.82	86.43	85.18	83.20
SVM	EF	90.49	89.96	89.07	88.33	87.40	90.49	89.66	88.78	87.78	86.77
	CVCF	90.56	89.86	88.94	88.28	87.76	90.48	89.56	88.72	87.92	86.54
	IPF	90.70	90.13	89.37	88.85	88.27	90.58	89.79	88.97	88.48	87.37
	None	82.46	81.15	80.35	79.39	78.49	82.46	79.81	78.55	76.98	75.68
Ripper	EF	83.36	82.87	82.72	82.43	81.53	83.46	83.03	82.87	82.30	81.66
	CVCF	83.17	82.93	82.64	82.03	81.68	83.17	82.59	82.19	81.69	80.45
	IPF	83.74	83.59	83.33	82.72	82.44	83.74	83.61	82.94	82.94	82.48
	None	83.93	83.66	82.81	82.25	81.41	83.93	82.97	82.38	81.69	80.28
C4.5	EF	84.18	84.07	83.70	83.20	82.36	84.16	83.96	83.53	83.38	82.66
	CVCF	84.15	83.92	83.24	82.54	82.13	84.15	83.61	83.00	82.84	81.61
	IPF	84.44	84.33	83.92	83.38	82.53	84.44	83.89	83.84	83.50	82.72

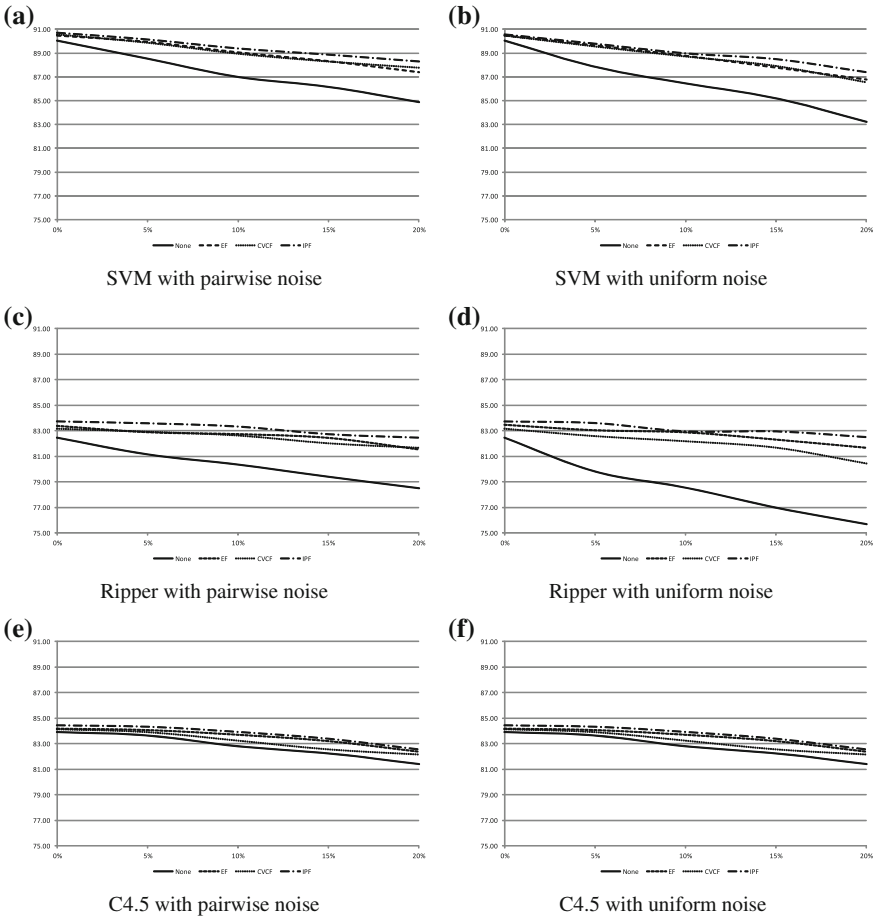

Fig. 5.4 Accuracy over different amounts and types of noise. The different filters used are named by their acronyms. "None" denotes the absence of any filtering. **a** SVM with pairwise noise **b** SVM with uniform noise **c** Ripper with pairwise noise **d** Ripper with uniform noise **e** C4.5 with pairwise noise **f** C4.5 with uniform noise

noise filter denoted by "None" is remarkably different from the lines that illustrate the usage of any noise filter. The IPF filter is slightly better than the other due to its greater sophistication, but in overall the use of filters is highly recommended. Even in the case of 0 % of controlled noise, the noise already present is also cleansed, allowing the filtering to improve even in this base case. Please note that the divergence appears even in the 5 % case, showing that noise filtering is worth trying in low noise frameworks.

Ripper obtains a lower overall accuracy than SVM, but the conclusions are akin: the usage of noise filters is highly recommended as can be seen in Fig. 5.4c, d. It is remarkable that not applying filtering for Ripper causes a fast drop in performance,

indicating that the rule base modeled is being largely affected by the noise. Thanks to the use of the noise filter the inclusion of misleading rules is controlled, resulting in a smoother drop in performance, even slower than that for SVM.

The last case is also very interesting. Being that C4.5 is more robust against noise than SVM and Ripper, the accuracy drop over the increment of noise is lower. However the use of noise filters is still recommended as they improve both the initial case 0% and the rest of levels. The greater differences between not filtering and the use of any filter are found in uniform class noise (Fig. 5.4f). As we indicated when describing the SVM case, uniform class noise is more disruptive but the use of filtering for C4.5 make its performance comparable to the case of pairwise noise (Fig. 5.4e).

Although not depicted here, the size of C4.5 trees, Ripper rule base size and the number of support vectors of SVM is lower with the usage of noise filters when the noise amount increases, resulting in a shorter time when evaluating examples for classification. This is specially critical for SVM, whose evaluation times dramatically increase with the increment of selected support vectors.

5.5.3 *Noise Filtering Efficacy Prediction by Data Complexity Measures*

In the previous Sect. 5.5.2 we have seen that the application of noise filters are beneficial in most cases, especially when higher amounts of noise are present in the data. However, applying a filter is not "free" in terms of computing time and information loss. Indiscriminate application of noise filtering may be interpreted as the outcome of the aforementioned example study, but it would be interesting to study the noise filters' behavior further and to obtain hints about whether filtering is useful or not depending on the data case.

In an ideal case, only the examples that are completely wrong would be erased from the data set. The truth is both correct examples and examples containing valuable information may be removed, as the filters are ML techniques with their inherent limitations. This fact implies that these techniques do not always provide an improvement in performance. The success of these methods depends on several circumstances, such as the kind and nature of the data errors, the quantity of noise removed or the capabilities of the classifier to deal with the loss of useful information related to the filtering. Therefore, the efficacy of noise filters, i.e., whether their use causes an improvement in classifier performance, depends on the noise-robustness and the generalization capabilities of the classifier used, but it also strongly depends on the characteristics of the data.

Describing the characteristics of the data is not an easy task, as specifying what "difficult" means is usually not straightforward or it simply does not depend on a single factor. Data complexity measures are a recent proposal to represent characteristics of the data that are considered difficult in classification tasks, e.g. the overlapping

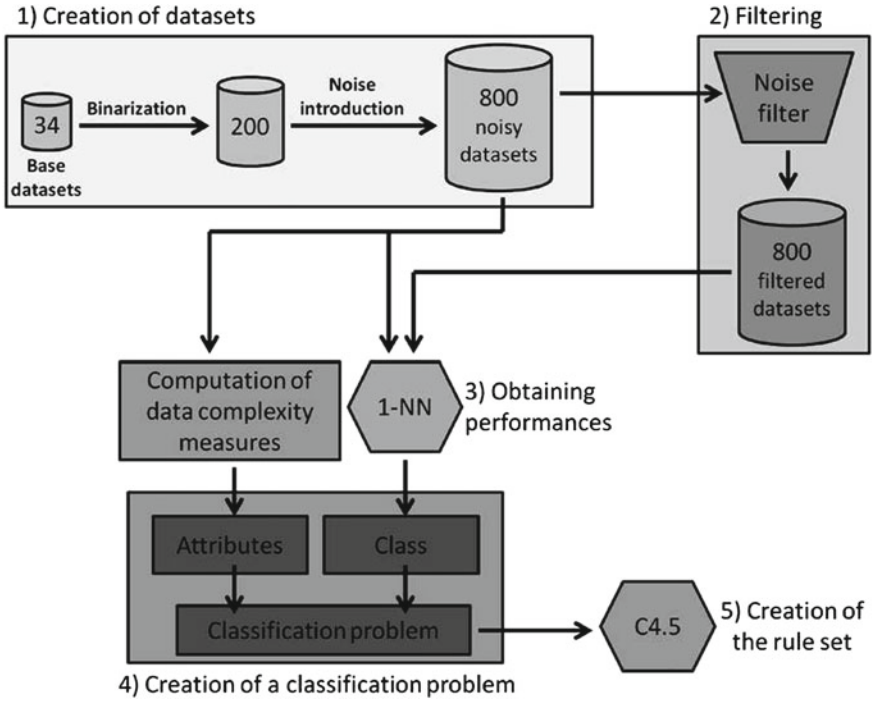

Fig. 5.5 Using C4.5 to build a rule set to predict noise filtering efficacy

among classes, their separability or the linearity of the decision boundaries. The most commonly used data complexity set of measures are those gathered together by Ho and Basu [35]. They consist of 12 metrics designed for binary classification problems that numerically estimate the difficulty of 12 different aspects of the data. For some measures lower/higher values mean a more difficult problem regarding to such a characteristic. Having a numeric description of the difficult aspects of the data opens a new question: can we predict which characteristics are related with noise and will they be successfully corrected by noise filters?

This prediction can help, for example, to determine an appropriate noise filter for a concrete noisy data set such a filter providing a signicant advantage in terms of the results or to design new noise filters which select more or less aggressive filtering strategies considering the characteristics of the data. Choosing a noise-sensitive learner facilitates the checking of when a filter removes the appropriate noisy examples in contrast to a robust learner-the performance of classiers built by the former is more sensitive to noisy examples retained in the data set after the ltering process.

A way to formulate rules that describe when it is appropriate to filter the data follows the scheme depicted in Fig. 5.5. From an initial set of 34 data sets from those described in Chap. 2, a large amount of two-class data sets are obtained by

binarization along their data complexity measures. Thus the filtering efficacy is compared by using 1-NN as a classifier to obtain the accuracy of filtering versus not filtering. This comparison is achieved by using a Wilcoxon Signed Rank test. If the statistical test yields differences favouring the filtering, the two-class data set is labeled as appropriate for filtering, and not favorable in other case. As a result for each binary data set we will have 12 data complexity measures and a label describing whether the data set is eligible for filtering or not. A simple way to summarize this information into a rule set is to use a decision tree (C4.5) using the 12 data complexity values as the input features, and the appropriateness label as the class.

An important appreciation about the scheme presented in Fig. 5.5 is that for every label noise filter we want to consider, we will obtain a different set of rules. For the sake of simplicity we will limit this illustrative study to our selected filters—EF, CVCF and IPF—in Sect. 5.3.

How accurate is the set of rules when predicting the suitability of label noise filters? Using a 10-FCV over the data set obtained in the fourth step in Fig. 5.5, the training and test accuracy of C4.5 for each filter is summarized in Table 5.3.

The test accuracy above 80 % in all cases indicates that the description obtained by C4.5 is precise enough.

Using a decision tree is also interesting not only due to the generated rule set, but also because we can check which data complexity measures (that is, the input attributes) are selected first, and thus are considered as more important and discriminant by C4.5. Averaging the rank of selection of each data complexity measure over the 10 folds, Table 5.4 shows which complexity measures are the most dis-

Table 5.3 C4.5 accuracy in training and test for the ruleset describing the adequacy of label noise filters

Noise filter	% Acc. training	% Acc. Test
EF	0.9948	0.8176
CVCF	0.9966	0.8353
IPF	0.9973	0.8670

Table 5.4 Average rank of each data complexity measure selected by C4.5 (the lower the better)

Metric	EF	CVCF	IPF	Mean
F1	5.90	4.80	4.50	**5.07**
F2	1.00	1.00	1.00	**1.00**
F3	10.10	3.40	3.30	**5.60**
N1	9.10	9.90	7.10	8.70
N2	3.30	2.00	3.00	**2.77**
N3	7.80	8.50	9.50	8.60
N4	9.90	9.70	10.50	10.03
L1	7.90	10.00	6.00	7.97
L2	9.30	6.80	10.00	8.70
L3	4.60	8.70	5.90	6.40
T1	5.20	6.80	11.00	7.67

criminating, and thus more interesting, for C4.5 to discern when a noise filter will behave well or badly. Based on these rankings it is easy to observe that F2, N2, F1 and F3 are the predominant measures in the order of choice. Please remember that behind these acronyms, the data complexity measures aim to describe one particular source of difficulty for any classification problem. Following the order from the most important of these four outstanding measures to the least, the volume of overlap region (F2) is key to describe the effectiveness of a class noise filter. The less any attribute is overlapped, the better the filter is able to decide if the instance is noisy. It is complemented with the ratio of average intra/inter class distance as defined by the nearest neighbor rule. When the examples sharing the same class are closer than the examples of other classes the filtering is effective for 1-NN. This measure is expected to change if another classifier is chosen to build the classification problem. F1 and F3 are also measures of individual attribute overlapping as F2, but they are less important in general.

If the discriminant abilities of these complexity measures are as good as their ranks indicate, using only these few measures we can expect to obtain a better and more concise description of what a easy-to-filter problem is. In order to avoid the study of all the existing combinations of the five metrics, the following experimentation is mainly focused on the measures F2, N2 and F3, the most discriminative ones since the order results can be considered more important than the percentage results. The incorporation of F1 into this set is also studied. The prediction capability of the measure F2 alone, since is the most discriminative one, is also shown. All these results are presented in Table 5.5.

The use of the measure F2 alone to predict the noise filtering efficacy with good performance can be discarded, since its results are not good enough compared with the cases where more than one measure is considered. This fact reflects that the use of single measures does not provide enough information to achieve a good filtering efficacy prediction result. Therefore, it is necessary to combine several measures which examine different aspects of the data. Adding the rest of selected measures provides comparable results to those shown in Table 5.3 yet limits the complexity of the rule set obtained.

The work carried out in this section is studied further in [77], showing how a rule set obtained for one filter can be applied to other filters, how these rule sets are validated with unseen data sets and even increasing the number of filters involved.

Table 5.5 Performance results of C4.5 predicting the noise filtering efficacy (measures used: F2, N2, F3, and F1)

	F2		F2-N2-F3-F1		F2-N2-F3	
Noise Filter	Training	Test	Training	Test	Training	Test
CVCF	1.0000	0.5198	0.9983	0.7943	0.9977	0.8152
EF	1.0000	0.7579	0.9991	0.8101	0.9997	0.8421
IPF	1.0000	0.7393	0.9989	0.8119	0.9985	0.7725
Mean	1.0000	0.6723	0.9988	0.8054	0.9986	0.8099

5.5.4 Multiple Classifier Systems with Noise

We will dispose of three well-known classifiers to build the MCS used in this illustrative section. SVM [14], C4.5 [75] and KNN [63] are chosen based on their good performance in a large number of real-world problems. Moreover, they were selected because these methods have a highly differentiated and well known noise-robustness, which is important in order to properly evaluate the performance of MCSs in the presence of noise. Considering thec previous classifiers (SVM, C4.5 and KNN), a MCS composed by 3 individual classifiers (SVM, C4.5 and 1-NN) is built. Therefore, the MCSs built with heterogeneous classifiers (MCS3-1) will contain a noise-robust algorithm (C4.5), a noise-sensitive method (SVM) and a local distance dependent method with a low tolerance to noise (1-NN).

5.5.4.1 First Scenario: Data Sets with Class Noise

Table 5.6 shows the performance (top part of the table) and robustness (bottom part of table) results of each classification algorithm at each noise level on data sets with class noise. Each one of these parts in the table (performance and robustness parts) is divided into another two parts: one with the results of the uniform class noise

Table 5.6 Performance and robustness results on data sets with class noise

			Results				p-values MCS3-1 vs.		
		x%	SVM	C4.5	1-NN	MCS3-1	SVM	C4.5	1-NN
Performance	Uniform	0%	83.25	82.96	81.42	85.42	5.20E-03	1.80E-03	7.10E-04
		10%	79.58	82.08	76.28	83	1.10E-04	3.90E-01	1.30E-07
		20%	76.55	79.97	71.22	80.09	5.80E-05	9.5E-01*	1.00E-07
		30%	73.82	77.9	65.88	77.1	2.80E-04	3.5E-01*	6.10E-08
		40%	70.69	74.51	61	73.2	7.20E-03	2.0E-01*	7.00E-08
		50%	67.07	69.22	55.55	67.64	5.40E-01	1.4E-01*	1.10E-07
	Pairwise	0%	83.25	82.96	81.42	85.42	5.20E-03	1.80E-03	7.10E-04
		10%	80.74	82.17	77.73	83.95	1.20E-04	5.80E-02	1.20E-07
		20%	79.11	80.87	74.25	82.21	2.00E-04	3.50E-01	8.20E-08
		30%	76.64	78.81	70.46	79.52	2.10E-03	8.20E-01	5.60E-08
		40%	73.13	74.83	66.58	75.25	3.80E-02	8.0E-01*	4.50E-08
		50%	65.92	60.29	63.06	64.46	2.6E-02*	2.60E-05	1.10E-01
RLA	Uniform	10%	4.44	1.1	6.16	2.91	7.20E-03	1.5E-06*	1.00E-07
		20%	8.16	3.78	12.1	6.4	1.50E-02	3.1E-05*	1.20E-06
		30%	11.38	6.36	18.71	9.95	8.80E-02	4.6E-05*	1.80E-07
		40%	15.08	10.54	24.15	14.54	6.60E-01	8.2E-05*	1.50E-06
		50%	19.47	17	30.58	21.08	1.9E-01*	1.3E-04*	1.30E-05
	Pairwise	10%	2.97	1	4.2	1.73	6.60E-03	2.0E-04*	6.70E-06
		20%	4.86	2.66	8.21	3.86	7.20E-02	1.7E-03*	1.00E-05
		30%	7.81	5.33	12.75	7.11	3.80E-01	2.7E-03*	6.30E-06
		40%	12.01	10.19	17.2	12.08	5.50E-01	7.8E-03*	4.40E-05
		50%	20.3	26.7	21.18	24.13	1.4E-04*	2.60E-03	1.1E-01*

and another with the results of the pairwise class noise. A star '$*$' next to a p-value indicates that the corresponding single algorithm obtains more ranks than the MCS in Wilcoxon's test comparing the individual classifier and the MCS. Note that the robustness can only be measured if the noise level is higher than 0 %, so the robustness results are presented from a noise level of 5 % and higher.

From the raw results we can extract some interesting conclusions. If we consider the performance results with uniform class noise we can observe that MCS3-k is statistically better than SVM, but in the case of C4.5 statistical differences are only found at the lowest noise level. For the rest of the noise levels, MCS3-1 is statistically equivalent to C4.5. Statistical differences are found between MCS3-1 and 1-NN for all the noise levels, indicating that MCS are specially suitable when taking noise sensitive classifiers into account.

In the case of pairwise class noise the conclusions are very similar. MCS3-1 statistically outperforms its individual components when the noise level is below 45 %, whereas it only performs statistically worse than SVM when the noise level reaches 50 % (regardless of the value of 1). MCS3-1 obtains more ranks than C4.5 in most of the cases; moreover, it is statistically better than C4.5 when the noise level is below 15 %. Again MCS3-1 statistically outperforms 1-NN regardless of the level of noise.

In uniform class noise MCS3-1 is significantly more robust than SVM up to a noise level of 30 %. Both are equivalent from 35 % onwards—even though MCS3-1 obtains more ranks at 35–40 % and SVM at 45–50 %. The robustness of C4.5 excels with respect to MCS3-1, observing the differences found. MCS3-1 is statistically better than 1-NN. The Robustness results with pairwise class noise present some differences with respect to uniform class noise. MCS3-1 statistically overcomes SVM up to a 20 % noise level, they are equivalent up to 45 % and MCS3-1 is outperformed by SVM at 50 %. C4.5 is statistically more robust than MCS3-1 (except in highly affected data sets, 45–50 %) and The superiority of MCS3-1 against 1-NN is notable, as it is statistically better at all noise levels.

It is remarkable that the uniform scheme is the most disruptive class noise for the majority of the classifiers. The higher disruptiveness of the uniform class noise in MCSs built with heterogeneous classifiers can be attributed to two main reasons: (i) this type of noise affects all the output domain, that is, all the classes, to the same extent, whereas the pairwise scheme only affects the two majority classes; (ii) a noise level x% with the uniform scheme implies that exactly x% of the examples in the data sets contain noise, whereas with the pairwise scheme, the number of noisy examples for the same noise level x% depends on the number of examples of the majority class N_{maj}; as a consequence, the global noise level in the whole data set is usually lower—more specifically, the number of noisy examples can be computed as $(x \cdot N_{maj})/100$.

With the performance results in uniform class noise MCS3-1 generally outperforms its single classifier components. MCS3-1 is better than SVM and 1-NN, whereas it only performs statistically better than C4.5 at the lowest noise levels. In pairwise class noise MCS3-1 improves SVM up to a 40 % noise level, it is better than C4.5 at the lowest noise levels—these noise levels are lower than those of the

uniform class noise—and also outperforms 1-NN. Therefore, the behavior of MCS3-1 with respect to their individual components is better in the uniform scheme than in the pairwise one.

5.5.4.2 Second Scenario: Data Sets with Attribute Noise

Table 5.7 shows the performance and robustness results of each classification algorithm at each noise level on data sets with attribute noise.

At first glance we can appreciate that the results on data sets with uniform attribute noise are much worse than those on data sets with Gaussian noise for all the classifiers, including MCSs. Hence, the most disruptive attribute noise is the uniform scheme. As the uniform attribute noise is the most disruptive noise scheme, MCS3-1 outperforms SVM and 1-NN. However, with respect to C4.5, MCS3-1 is significantly better only at the lowest noise levels (up to 10–15 %), and is equivalent at the rest of the noise levels. With gaussian attribute noise MCS3-1 is only better than 1-NN and SVM, and better than C4.5 at the lowest noise levels (up to 25 %).

The robustness of the MCS3-1 with uniform attribute noise does not outperform that of its individual classifiers, as it is statistically equivalent to SVM and sometimes worse than C4.5. Regarding 1-NN, MCS3-1 performs better than 1-NN. When

Table 5.7 Performance and robustness results on data sets with attribute noise

			Results				p-values MCS3-1 vs.		
		x%	SVM	C4.5	1-NN	MCS3-1	SVM	C4.5	1-NN
Performance	Uniform	0%	83.25	82.96	81.42	85.42	5.20E-03	1.80E-03	7.10E-04
		10%	81.78	81.58	78.52	83.33	3.90E-03	8.30E-02	8.10E-06
		20%	78.75	79.98	75.73	80.64	4.40E-03	0.62	5.20E-06
		30%	76.09	77.64	72.58	77.97	2.10E-03	9.4E-01*	1.00E-05
		40%	72.75	75.19	69.58	74.84	9.90E-03	7.6E-01*	1.30E-06
		50%	69.46	72.12	66.59	71.36	1.20E-02	3.1E-01*	2.70E-05
	Gaussian	0%	83.25	82.96	81.42	85.42	5.20E-03	1.80E-03	7.10E-04
		10%	82.83	82.15	80.08	84.49	3.40E-03	4.80E-03	4.00E-05
		20%	81.62	81.16	78.52	83.33	4.40E-03	2.10E-02	1.00E-05
		30%	80.48	80.25	76.74	81.85	1.10E-02	2.00E-01	1.00E-05
		40%	78.88	78.84	74.82	80.34	1.60E-02	0.4	1.00E-05
		50%	77.26	77.01	73.31	78.49	0.022	3.40E-01	3.00E-05
RLA	Uniform	10%	1.38	1.68	3.63	2.4	6.7E-01*	6.6E-02*	1.30E-02
		20%	5.06	3.6	6.78	5.54	6.10E-01	5.2E-03*	1.30E-02
		30%	8.35	6.62	10.73	8.85	7.80E-01	2.6E-03*	1.20E-02
		40%	12.13	9.6	14.29	12.44	7.8E-01*	1.1E-02*	2.00E-03
		50%	16.28	13.46	17.7	16.68	8.5E-01*	3.4E-03*	2.00E-02
	Gaussian	10%	0.25	0.94	1.6	1.03	1.70E-01	2.4E-01*	2.60E-01
		20%	1.74	2.1	3.36	2.36	3.30E-01	1.2E-01*	1.00E-01
		30%	3.14	3.25	5.64	4.14	7.3E-01*	1.3E-02*	7.20E-02
		40%	5.23	5.02	7.91	5.93	9.40E-01	1.5E-02*	2.60E-03
		50%	7.28	7.29	9.51	8.14	0.98	6.0E-02*	0.051

focusing in gaussian noise the robustness results are better than those of the uniform noise. The main difference in this case is that MCS3-1 and MCS5 are not statistically worse than C4.5.

5.5.4.3 Conclusions

The results obtained have shown that the MCSs studied do not always significantly improve the performance of their single classification algorithms when dealing with noisy data, although they do in the majority of cases (if the individual components are not heavily affected by noise). The improvement depends on many factors, such as the type and level of noise. Moreover, the performance of the MCSs built with heterogeneous classifiers depends on the performance of their single classifiers, so it is recommended that one studies the behavior of each single classifier before building the MCS. Generally, the MCSs studied are more suitable for class noise than for attribute noise. Particularly, they perform better with the most disruptive class noise scheme (the uniform one) and with the least disruptive attribute noise scheme (the gaussian one).

The robustness results show that the studied MCS built with heterogeneous classifiers will not be more robust than the most robust among their single classification algorithms. In fact, the robustness can always be shown as an average of the robustness of the individual methods. The higher the robustness of the individual classifiers are, the higher the robustness of the MCS is.

5.5.5 Analysis of the OVO Decomposition with Noise

In this section, the performance and robustness of the classification algorithms using the OVO decomposition with respect to its baseline results when dealing with data suffering from noise are analyzed. In order to investigate whether the decomposition is able to reduce the effect of noise or not, a large number of data sets are created introducing different levels and types of noise, as suggested in the literature. Several well-known classification algorithms, with or without decomposition, are trained with them in order to check when decomposition is advantageous. The results obtained show that methods using the One-vs-One strategy lead to better performances and more robust classifiers when dealing with noisy data, especially with the most disruptive noise schemes. Section 5.5.5.1 is devoted to the study of the class noise scheme, whereas Sect. 5.5.5.2 analyzes the attribute noise case.

5.5.5.1 First Scenario: Data Sets with Class Noise

Table 5.8 shows the test accuracy and RLA results for each classification algorithm at each noise level along with the associated p-values between the OVO and the

Table 5.8 Test accuracy, RLA results and p-values on data sets with class noise. Cases where the baseline classifiers obtain more ranks than the OVO version in the Wilcoxon's test are indicated with a star (*)

			Uniform random class noise						Pairwise class noise					
			C4.5		Ripper		5-NN		C4.5		Ripper		5-NN	
			Base	OVO	Base	OVO	Base	OVO	Base	OVO	Base	OVO	Base	OVO
Test accuracy	Results	0%	81.66	82.7	77.92	82.15	82.1	83.45	81.66	82.7	77.92	82.15	82.1	83.45
		10%	80.5	81.71	71.3	79.86	81.01	82.56	80.94	81.86	75.94	80.71	81.42	82.82
		20%	78.13	80.27	66.71	77.35	79.55	81.36	79.82	81.03	74.77	79.62	79.41	81.01
		30%	75.22	78.87	62.91	74.98	77.21	79.82	78.49	79.26	73.38	78.05	75.29	76.81
		40%	71.1	76.88	58.32	72.12	73.82	76.83	76.17	76.91	71.6	76.19	69.89	71.65
		50%	64.18	73.71	53.79	67.56	68.04	72.73	63.63	63.52	67.11	65.78	64.02	65.52
	p-values	0%	-		-		-		0.007		0.0002		0.093	
		10%	0.0124		0.0001		0.0036		0.0033		0.0003		0.0137	
		20%	0.0028		0.0001		0.0017		0.0017		0.0002		0.0022	
		30%	0.0002		0.0001		0.0013		0.009		0.0001		0.01	
		40%	0.0002		0.0001		0.0111		0.0276		0.0003		0.004	
		50%	0.0001		0.0001		0.0008		0.5016(*)		0.0930(*)		0.0057	
RLA value	Results	0%	-		-		-		-		-		-	
		10%	1.56	1.28	9.35	2.79	1.46	1.12	0.91	1.01	2.38	1.72	0.89	0.82
		20%	4.63	3.15	15.44	5.86	3.48	2.67	2.39	2.13	3.52	3.03	3.29	2.93
		30%	8.36	4.9	20.74	8.82	6.4	4.53	4.16	4.49	5.13	4.84	8.05	7.81
		40%	13.46	7.41	26.72	12.35	10.3	8.11	7.01	7.38	7.25	7.12	14.37	13.68
		50%	21.87	11.29	32.74	18.1	17.47	13.12	21.03	22.28	12.22	18.75	21.06	20.67
	p-values	0%	-		-		-		-		-		-	
		10%	0.5257		0.0001		0.1354		0.8721(*)		0.3317		0.3507	
		20%	0.0304		0.0001		0.0479		0.0859		0.4781		0.0674	
		30%	0.0006		0.0001		0.0124		0.6813		0.6542		0.3507	
		40%	0.0001		0.0001		0.0333		0.6274		0.6274(*)		0.0793	
		50%	0.0001		0.0001		0.0015		0.0400(*)		0.0001(*)		0.062	

non-OVO version from the Wilcoxon's test. The few exceptions where the baseline classifiers obtain more ranks than the OVO version in the Wilcoxon's test are indicated with a star next to the p-value.

For random class noise the test accuracy of the methods using OVO is higher in all the noise levels. Moreover, the low p-values show that this advantage in favor of OVO is significant. The RLA values of the methods using OVO are lower than those

of the baseline methods at all noise levels. These differences are also statistically significant as reflected by the low p-values. Only at some very low noise levels—5 % and 10 % for C4.5 and 5 % for 5-NN - the results between the OVO and the non-OVO version are statistically equivalent, but notice that the OVO decomposition does not hinder the results, simply the loss is not lower.

These results also show that OVO achieves more accurate predictions when dealing with **pairwise class noise**, however, it is not so advantageous with C4.5 or RIPPER as with 5-NN in terms of robustness when noise only affects one class. For example, the behavior of RIPPER with this noise scheme can be related to the hierarchical way in which the rules are learned: it starts learning rules of the class with the lowest number of examples and continues learning those classes with more examples. When introducing this type of noise, RIPPER might change its training order, but the remaining part of the majority class can still be properly learned, since it now has more priority. Moreover, the original second majority class, now with noisy examples, will probably be the last one to be learned and it would depend on how the rest of the classes have been learned. Decomposing the problem with OVO, a considerable number of classifiers will have a notable quantity of noise—those of the majority and the second majority classes—and hence, the tendency to predict the original majority class decreases—when the noise level is high, it strongly affects the accuracy, since the majority has more influence on it.

In contrast with the rest of noise schemes, with pairwise noise scheme, all the data sets have different real percentages of noisy examples at the same noise level of $x\%$. This is because each data set has a different number of examples of the majority class, and thus a noise level of $x\%$ does not affect all the data sets in the same way. In this case, the percentage of noisy examples with a noise level of $x\%$ is computed as $(x \cdot N_{maj})/100$, where N_{maj} is the percentage of examples of the majority class.

5.5.5.2 Second Scenario: Data Sets with Attribute Noise

In this section, the performance and robustness of the classification algorithms using OVO in comparison to its non-OVO version when dealing with data with attribute noise are analyzed. The test accuracy, RLA results and p-values of each classification algorithm at each noise level are shown in Table 5.9.

In the case of uniform attribute noise it can be pointed out that the test accuracy of the methods using OVO is always statistically better at all the noise levels. The RLA values of the methods using OVO are lower than those of the baseline methods at all noise levels—except in the case of C4.5 with a 5 % of noise level. Regarding the p-values, a clear tendency is observed, the p-value decreases when the noise level increases with all the algorithms. With all methods—C4.5, RIPPER and 5-NN—the p-values of the RLA results at the lowest noise levels (up to 20–25 %) show that the robustness of OVO and non-OVO methods is statistically equivalent. From that point on, the OVO versions statistically outperform the non-OVO ones. Therefore, the usage of OVO is clearly advantageous in terms of accuracy and robustness when

Table 5.9 Test accuracy, RLA results and p-values on data sets with attribute noise. Cases where the baseline classifiers obtain more ranks than the OVO version in the Wilcoxon's test are indicated with a star (*)

			Uniform random attribute noise						Gaussian attribute noise					
			C4.5		Ripper		5-NN		C4.5		Ripper		5-NN	
			Base	OVO	Base	OVO	Base	OVO	Base	OVO	Base	OVO	Base	OVO
Test accuracy	Results	0%	81.66	82.7	77.92	82.15	82.1	83.45	81.66	82.7	77.92	82.15	82.1	83.45
		10%	80.31	81.65	76.08	80.85	79.81	81.34	80.93	81.67	76.53	81.12	80.91	82.52
		20%	78.71	80.27	73.95	79.15	77.63	79.38	79.77	81.11	75.35	80.06	80.16	81.74
		30%	76.01	78.25	71.25	77.06	74.68	76.46	79.03	80.4	74.46	78.93	78.84	80.77
		40%	73.58	76.19	68.66	74.56	71.29	73.65	77.36	79.51	72.94	78.1	77.53	79.11
		50%	70.49	73.51	65.5	71.66	67.72	70.07	75.29	78.03	71.57	76.27	76.02	77.72
	p-values	0%	0.007		0.0002		0.093		0.007		0.0002		0.093	
		10%	0.0169		0.0003		0.091		0.1262		0.0004		0.0064	
		20%	0.0057		0.0003		0.0015		0.0048		0.0002		0.0036	
		30%	0.0043		0.0001		0.0112		0.0051		0.0003		0.0025	
		40%	0.0032		0.0001		0.0006		0.0019		0.0003		0.1262	
		50%	0.0036		0.0007		0.0011		0.0004		0.0008		0.0251	
RLA value	Results	0%	-		-		-		-		-		-	
		10%	1.82	1.32	2.56	1.62	3.03	2.62	0.92	1.27	1.9	1.26	1.68	1.13
		20%	3.88	3.11	5.46	3.77	5.72	5.03	2.4	1.99	3.49	2.59	2.51	2.09
		30%	7.54	5.77	9.2	6.42	9.57	8.76	3.42	2.91	4.66	3.95	4.34	3.32
		40%	10.64	8.25	12.81	9.64	13.73	12.08	5.67	4.03	6.74	4.96	6.03	5.38
		50%	14.74	11.74	17.21	13.33	18.14	16.55	8.37	5.87	8.5	7.35	7.82	7.16
	p-values	0%	-		-		-		-		-		-	
		10%	0.4781		0.5755		1.0000(*)		0.0766(*)		0.8519(*)		0.4115	
		20%	0.2471		0.1454		0.1354		0.8405		0.9108(*)		0.3905	
		30%	0.0304		0.0438		0.1672		0.6542		0.2627(*)		0.2627	
		40%	0.0569		0.0036		0.0111		0.1169		0.3905		0.9405	
		50%	0.0152		0.0064		0.0228		0.009		0.6542(*)		0.218	

noise affects the attributes in a random and uniform way. This behavior is particularly notable with the highest noise levels, where the effects of noise are expected to be more detrimental.

On the other hand, analyzing the gaussian attribute noise results in the test accuracy of the methods using OVO being better at all the noise levels. The low p-values show that this advantage, also in favor of OVO, is statistically significant. With respect to the RLA results the p-values show a clear decreasing tendency when the noise level increases in all the algorithms. In the case of C4.5, OVO is statistically better from a 35 % noise level onwards. RIPPER and 5-NN are statistically equivalent at all noise levels—although 5-NN with OVO obtains higher Wilcoxon's ranks.

Hence, the OVO approach is also suitable considering the accuracy achieved with this type of attribute noise. The robustness results are similar between the OVO and non-OVO versions with RIPPER and 5-NN. However, for C4.5 there are statistical differences in favor of OVO at the highest noise levels. It is important to note that in some cases, particularly in the comparisons involving RIPPER, some RLA results show that OVO is better than the non-OVO version in average but the latter obtains more ranks in the statistical test—even though these differences are not significant. This is due to the extreme results of some individual data sets, such as *led7digit* or *flare*, in which the RLA results of the non-OVO version are much worse than those of the OVO version. Anyway, we should notice that average results themselves are not meaningful and the corresponding non-parametric statistical analysis must be carried out in order to extract meaningful conclusions, which reflects the real differences between algorithms.

5.5.5.3 Conclusions

The results obtained have shown that the OVO decomposition improves the baseline classifiers in terms of accuracy when data is corrupted by noise in all the noise schemes shown in this chapter. The robustness results are particularly notable with the more disruptive noise schemes—the uniform random class noise scheme and the uniform random attribute noise scheme—where a larger amount of noisy examples and with higher corruptions are available, which produce greater differences (with statistical significance).

In conclusion, we must emphasize that one usually does not know the type and level of noise present in the data of the problem that is going to be addressed. Decomposing a problem suffering from noise with OVO has shown a better accuracy, higher robustness and homogeneity in all the classification algorithms tested. For this reason, the use of the OVO decomposition strategy in noisy environments can be recommended as an easy-to-applicate, yet powerful tool to overcome the negative effects of noise in multi-class problems.

References

1. Abellán, J., Masegosa, A.R.: Bagging decision trees on data sets with classification noise. In: Link S., Prade H. (eds.) FoIKS, Lecture Notes in Computer Science, vol. 5956, pp. 248–265. Springer, Heidelberg (2009)

2. Abellán, J., Masegosa, A.R.: Bagging schemes on the presence of class noise in classification. Expert Syst. Appl. **39**(8), 6827–6837 (2012)
3. Aha, D.W., Kibler, D.: Noise-tolerant instance-based learning algorithms. In: Proceedings of the 11th International Joint Conference on Artificial Intelligence, Vol. 1, IJCAI'89, pp. 794–799. Morgan Kaufmann Publishers Inc. (1989)
4. Alcalá-Fdez, J., Sánchez, L., García, S., del Jesus, M., Ventura, S., Garrell, J., Otero, J., Romero, C., Bacardit, J., Rivas, V., Fernández, J., Herrera, F.: KEEL: a software tool to assess evolutionary algorithms for data mining problems. Soft Comput. Fus. Found. Methodol. Appl. **13**, 307–318 (2009)
5. Allwein, E.L., Schapire, R.E., Singer, Y.: Reducing multiclass to binary: a unifying approach for margin classifiers. J. Mach. Learn. Res. **1**, 113–141 (2000)
6. Anand, R., Mehrotra, K., Mohan, C.K., Ranka, S.: Efficient classification for multiclass problems using modular neural networks. IEEE Trans. Neural Netw. **6**(1), 117–124 (1995)
7. Angluin, D., Laird, P.: Learning from noisy examples. Mach. Learn. **2**(4), 343–370 (1988)
8. Bonissone, P., Cadenas, J.M., Carmen Garrido, M., Díaz-Valladares, A.: A fuzzy random forest. Int. J. Approx. Reason. **51**(7), 729–747 (2010)
9. Bootkrajang, J., Kaban, A.: Multi-class classification in the presence of labelling errors. In: ESANN 2011, 19th European Symposium on Artificial Neural Networks, Bruges, Belgium, 27–29 April 2011, Proceedings, ESANN (2011)
10. Brodley, C.E., Friedl, M.A.: Identifying and eliminating mislabeled training instances. In: Clancey W.J., Weld D.S. (eds.) AAAI/IAAI, Vol. 1, pp. 799–805 (1996)
11. Brodley, C.E., Friedl, M.A.: Identifying mislabeled training data. J. Artif. Intell. Res. **11**, 131–167 (1999)
12. Catal, C., Alan, O., Balkan, K.: Class noise detection based on software metrics and ROC curves. Inf. Sci. **181**(21), 4867–4877 (2011)
13. Chang, C.C., Lin, C.J.: A library for support vector machines. ACM Trans. Intell. Syst. Technol. **2**(3), 1–27 (2011)
14. Cortes, C., Vapnik, V.: Support vector networks. Mach. Learn. **20**, 273–297 (1995)
15. Delany, S.J., Cunningham, P.: An analysis of case-base editing in a spam filtering system. In: Funk P., González-Calero P.A. (eds.) ECCBR, pp. 128–141 (2004)
16. 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)
17. Dietterich, T.G., Bakiri, G.: Solving multiclass learning problems via error-correcting output codes. J. Artif. Intell. Res. **2**(1), 263–286 (1995)
18. Du, W., Urahama, K.: Error-correcting semi-supervised pattern recognition with mode filter on graphs. J. Adv. Comput. Intell. Intell. Inform. **15**(9), 1262–1268 (2011)
19. Frenay, B., Verleysen, M.: Classification in the presence of label noise: a survey. Neural Netw. Learn. Syst. IEEE Trans. **25**(5), 845–869 (2014)
20. Fürnkranz, J.: Round robin classification. J. Mach. Learn. Res. **2**, 721–747 (2002)
21. Galar, M., Fernández, A., Barrenechea, E., Bustince, H., Herrera, F.: An overview of ensemble methods for binary classifiers in multi-class problems: experimental study on one-vs-one and one-vs-all schemes. Pattern Recognit. **44**(8), 1761–1776 (2011)
22. Galar, M., Fernández, A., Tartas, E.B., Sola, H.B., Herrera, F.: A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches. IEEE Trans. Syst. Man Cybern. Part C **42**(4), 463–484 (2012)
23. Gamberger, D., Boskovic, R., Lavrac, N., Groselj, C.: Experiments with noise filtering in a medical domain. In: Proceedings of the Sixteenth International conference on machine learning, pp. 143–151. Morgan Kaufmann Publishers (1999)
24. Gamberger, D., Lavrac, N., Dzeroski, S.: Noise detection and elimination in data preprocessing: experiments in medical domains. Appl. Artif. Intell. **14**, 205–223 (2000)
25. García, V., Alejo, R., Sánchez, J., Sotoca, J., Mollineda, R.: Combined effects of class imbalance and class overlap on instance-based classification. In: Corchado, E., Yin, H., Botti, V., Fyfe, C. (eds.) Intelligent Data Engineering and Automated Learning IDEAL 2006. Lecture Notes in Computer Science, vol. 4224, pp. 371–378. Springer, Berlin (2006)

26. García, V., Mollineda, R., Sánchez, J.: On the k-NN performance in a challenging scenario of imbalance and overlapping. Pattern Anal. Appl. **11**(3–4), 269–280 (2008)
27. García, V., Sánchez, J., Mollineda, R.: An empirical study of the behavior of classifiers on imbalanced and overlapped data sets. In: Rueda, L., Mery, D., Kittler, J. (eds.) CIARP 2007. LNCS, vol. 4756, pp. 397–406. Springer, Heidelberg (2007)
28. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA data mining software: an update. SIGKDD Explor. Newsl. **11**(1), 10–18 (2009)
29. Haralick, R.M.: The table look-up rule. Commun. Stat. Theory Methods A **5**(12), 1163–1191 (1976)
30. Hart, P.E.: The condensed nearest neighbor rule. IEEE Trans. Inf. Theory **14**, 515–516 (1968)
31. Hernández, M.A., Stolfo, S.J.: Real-world data is dirty: data cleansing and the merge/purge problem. Data Min. Knowl. Discov. **2**, 9–37 (1998)
32. Hernández-Lobato, D., Hernández-Lobato, J.M., Dupont, P.: Robust multi-class gaussian process classification. In: Shawe-Taylor J., Zemel R.S., Bartlett P.L., Pereira F.C.N., Weinberger K.Q. (eds.) Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12–14 December 2011, Granada, Spain, NIPS, pp. 280–288 (2011)
33. Heskes, T.: The use of being stubborn and introspective. In: Ritter, H., Cruse, H., Dean, J. (eds.) Prerational Intelligence: Adaptive Behavior and Intelligent Systems Without Symbols and Logic, pp. 725–741. Kluwer, Dordrecht (2001)
34. Ho, T.K.: Multiple classifier combination: lessons and next steps. In: Kandel, Bunke E. (eds.) Hybrid Methods in Pattern Recognition, pp. 171-198. World Scientific, New York (2002)
35. Ho, T.K., Basu, M.: Complexity measures of supervised classification problems. IEEE Trans. Pattern Anal. Mach. Intell. **24**(3), 289–300 (2002)
36. Ho, T.K., Hull, J.J., Srihari, S.N.: Decision combination in multiple classifier systems. IEEE Trans. Pattern Anal. Mach. Intell. **16**(1), 66–75 (1994)
37. Hsu, C.W., Lin, C.J.: A comparison of methods for multiclass support vector machines. IEEE Trans. Neural Netw. **13**(2), 415–425 (2002)
38. Huang, Y.S., Suen, C.Y.: A method of combining multiple experts for the recognition of unconstrained handwritten numerals. IEEE Trans. Pattern Anal. Mach. Intell. **17**, 90–93 (1995)
39. Huber, P.J.: Robust Statistics. Wiley, New York (1981)
40. Hüllermeier, E., Vanderlooy, S.: Combining predictions in pairwise classification: an optimal adaptive voting strategy and its relation to weighted voting. Pattern Recognit. **43**(1), 128–142 (2010)
41. Japkowicz, N.: Class imbalance: are we focusing on the right issue? In: II Workshop on learning from imbalanced data sets, ICML, pp. 17–23 (2003)
42. Jeatrakul, P., Wong, K., Fung, C.: Data cleaning for classification using misclassification analysis. J. Adv. Comput. Intell. Intell. Inform. **14**(3), 297–302 (2010)
43. Jo, T., Japkowicz, N.: Class Imbalances versus small disjuncts. SIGKDD Explor. **6**(1), 40–49 (2004)
44. John, G.H.: Robust decision trees: removing outliers from databases. In: Fayyad, U.M., Uthurusamy, R. (eds.) Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD-95), pp. 174–179. Montreal, Canada, August (1995)
45. Karmaker, A., Kwek, S.: A boosting approach to remove class label noise. Int. J. Hybrid Intell. Syst. **3**(3), 169–177 (2006)
46. Kermanidis, K.L.: The effect of borderline examples on language learning. J. Exp. Theor. Artif. Intell. **21**, 19–42 (2009)
47. Khoshgoftaar, T., Van Hulse, J., Napolitano, A.: Comparing boosting and bagging techniques with noisy and imbalanced data. IEEE Trans. Syst. Man Cybern. Part A Syst. Hum. **41**(3), 552–568 (2011)
48. Khoshgoftaar, T.M., Rebours, P.: Improving software quality prediction by noise filtering techniques. J. Comput. Sci. Technol. **22**, 387–396 (2007)
49. Klebanov, B.B., Beigman, E.: Some empirical evidence for annotation noise in a benchmarked dataset. In: Human Language Technologies: The 2010 Annual Conference of the

North American Chapter of the Association for Computational Linguistics, HLT '10, pp. 438–446. Association for Computational Linguistics (2010)
50. Knerr, S., Personnaz, L., Dreyfus, G.: Single-layer learning revisited: a stepwise procedure for building and training a neural network. In: Fogelman Soulié F., Hérault J. (eds.) Neurocomputing: Algorithms, Architectures and Applications, pp. 41–50. Springer, Heidelberg (1990)
51. Knerr, S., Personnaz, L., Dreyfus, G., Member, S.: Handwritten digit recognition by neural networks with single-layer training. IEEE Trans. Neural Netw. **3**, 962–968 (1992)
52. Kubat, M., Matwin, S.: Addresing the curse of imbalanced training sets: one-side selection. In: Proceedings of the 14th International Conference on Machine Learning, pp. 179–186 (1997)
53. Kuncheva, L.: Combining Pattern Classifiers: Methods and Algorithms. Wiley, Chichester (2004)
54. Kuncheva, L.I.: Diversity in multiple classifier systems. Inform. Fus. **6**, 3–4 (2005)
55. Lorena, A., de Carvalho, A., Gama, J.: A review on the combination of binary classifiers in multiclass problems. Artif. Intell. Rev. **30**, 19–37 (2008)
56. Maclin, R., Opitz, D.: An empirical evaluation of bagging and boosting. In: Proceedings of the fourteenth national conference on artificial intelligence and ninth conference on Innovative applications of artificial intelligence, pp. 546–551 (1997)
57. Malossini, A., Blanzieri, E., Ng, R.T.: Detecting potential labeling errors in microarrays by data perturbation. Bioinformatics **22**(17), 2114–2121 (2006)
58. Mandler, E., Schuermann, J.: Combining the classification results of independent classifiers based on the Dempster/Shafer theory of evidence. In: Gelsema E.S., Kanal L.N. (eds.) Pattern Recognition and Artificial Intelligence, pp. 381–393. Amsterdam: North-Holland (1988)
59. Manwani, N., Sastry, P.S.: Noise tolerance under risk minimization. IEEE Trans. Cybern. **43**(3), 1146–1151 (2013)
60. Maulik, U., Chakraborty, D.: A robust multiple classifier system for pixel classification of remote sensing images. Fundamenta Informaticae **101**(4), 286–304 (2010)
61. Mayoraz, E., Moreira, M.: On the decomposition of polychotomies into dichotomies (1996)
62. Mazurov, V.D., Krivonogov, A.I., Kazantsev, V.S.: Solving of optimization and identification problems by the committee methods. Pattern Recognit. **20**, 371–378 (1987)
63. Mclachlan, G.J.: Discriminant Analysis and Statistical Pattern Recognition (Wiley Series in Probability and Statistics). Wiley-Interscience, New York (2004)
64. Melville, P., Shah, N., Mihalkova, L., Mooney, R.J.: Experiments on ensembles with missing and noisy data. In: Roli F., Kittler J., Windeatt T. (eds.) Multiple Classifier Systems, Lecture Notes in Computer Science, vol. 3077, pp. 293–302. Springer, Heidelberg (2004)
65. 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 l., Baruque B. (eds.) HAIS, Lecture Notes in Computer Science, vol. 5572, pp. 417–424. Springer, Heidelberg (2009)
66. Muhlenbach, F., Lallich, S., Zighed, D.A.: Identifying and handling mislabelled instances. J. Intell. Inf. Syst. **22**(1), 89–109 (2004)
67. Napierala, K., Stefanowski, J., Wilk, S.: Learning from imbalanced data in presence of noisy and borderline examples. Rough Sets and Current Trends in Computing. LNCS, vol. 6086, pp. 158–167. Springer, Berlin (2010)
68. Nath, R.K.: Fingerprint recognition using multiple classifier system. Fractals **15**(3), 273–278 (2007)
69. Nettleton, D., Orriols-Puig, A., Fornells, A.: A Study of the Effect of Different Types of Noise on the Precision of Supervised Learning Techniques. Artif. Intell. Rev. **33**, 275–306 (2010)
70. Pérez Carlos Javier, G.F.J.M.J.R.M.R.C.: Misclassified multinomial data: a Bayesian approach. RACSAM **101**(1), 71–80 (2007)
71. Pimenta, E., Gama, J.: A study on error correcting output codes. In: Portuguese Conference on Artificial Intelligence EPIA 2005, 218–223 (2005)
72. Polikar, R.: Ensemble based systems in decision making. IEEE Circ. Syst. Mag. **6**(3), 21–45 (2006)

73. Qian, B., Rasheed, K.: Foreign exchange market prediction with multiple classifiers. J. Forecast. **29**(3), 271–284 (2010)
74. Quinlan, J.R.: Induction of decision trees. Mach. Learn. **1**(1), 81–106 (1986)
75. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, San Francisco (1993)
76. Rifkin, R., Klautau, A.: In defense of one-vs-all classification. J. Mach. Learn. Res. **5**, 101–141 (2004)
77. Sáez, J.A., Luengo, J., Herrera, F.: Predicting noise filtering efficacy with data complexity measures for nearest neighbor classification. Pattern Recognit. **46**(1), 355–364 (2013)
78. Sánchez, J.S., Barandela, R., Marqués, A.I., Alejo, R., Badenas, J.: Analysis of new techniques to obtain quality training sets. Pattern Recognit. Lett. **24**(7), 1015–1022 (2003)
79. Segata, N., Blanzieri, E., Delany, S.J., Cunningham, P.: Noise reduction for instance-based learning with a local maximal margin approach. J. Intell. Inf. Syst. **35**(2), 301–331 (2010)
80. Shapley, L., Grofman, B.: Optimizing group judgmental accuracy in the presence of interdependencies. Pub. Choice **43**, 329–343 (1984)
81. Smith, M.R., Martinez, T.R.: Improving classification accuracy by identifying and removing instances that should be misclassified. In: IJCNN, pp. 2690–2697 (2011)
82. Sun, J., ying Zhao, F., Wang, C.J., Chen, S.: Identifying and correcting mislabeled training instances. In: FGCN (1), pp. 244–250. IEEE (2007)
83. Sun, Y., Wong, A.K.C., Kamel, M.S.: Classification of Imbalanced Data: a Review. Int. J. Pattern Recognit. Artif. Intell. **23**(4), 687–719 (2009)
84. Teng, C.M.: Correcting Noisy Data. In: Proceedings of the Sixteenth International Conference on Machine Learning, pp. 239–248. Morgan Kaufmann Publishers, San Francisco, USA (1999)
85. Teng, C.M.: Polishing blemishes: Issues in data correction. IEEE Intell. Syst. **19**(2), 34–39 (2004)
86. Thongkam, J., Xu, G., Zhang, Y., Huang, F.: Support vector machine for outlier detection in breast cancer survivability prediction. In: Ishikawa Y., He J., Xu G., Shi Y., Huang G., Pang C., Zhang Q., Wang G. (eds.) APWeb Workshops, Lecture Notes in Computer Science, vol. 4977, pp. 99–109. Springer (2008)
87. Titterington, D.M., Murray, G.D., Murray, L.S., Spiegelhalter, D.J., Skene, A.M., Habbema, J.D.F., Gelpke, G.J.: Comparison of discriminant techniques applied to a complex data set of head injured patients. J. R. Stat. Soc. Series A (General) **144**, 145–175 (1981)
88. Tomek, I.: Two Modifications of CNN. IEEE Tran. Syst. Man Cybern. **7**(2), 679–772 (1976)
89. Verbaeten, S., Assche, A.V.: Ensemble methods for noise elimination in classification problems. In: Fourth International Workshop on Multiple Classifier Systems, pp. 317–325. Springer, Heidelberg (2003)
90. Wang, R.Y., Storey, V.C., Firth, C.P.: A framework for analysis of data quality research. IEEE Trans. Knowl. Data Eng. **7**(4), 623–640 (1995)
91. Wemecke, K.D.: A coupling procedure for the discrimination of mixed data. Biometrics **48**, 497–506 (1992)
92. Wheway, V.: Using boosting to detect noisy data. In: Revised Papers from the PRICAI 2000 Workshop Reader, Four Workshops Held at PRICAI 2000 on Advances in Artificial Intelligence, pp. 123–132. Springer (2001)
93. Wilson, D.R., Martinez, T.R.: Instance pruning techniques. In: Proceedings of the Fourteenth International Conference on Machine Learning, ICML '97, pp. 403–411. Morgan Kaufmann Publishers Inc. (1997)
94. Woźniak, M., Graña, M., Corchado, E.: A survey of multiple classifier systems as hybrid systems. Inform. Fus. **16**, 3–17 (2013)
95. Wu, T.F., Lin, C.J., Weng, R.C.: Probability estimates for multi-class classification by pairwise coupling. J. Mach. Learn. Res. **5**, 975–1005 (2004)
96. Wu, X.: Knowledge Acquisition From Databases. Ablex Publishing Corp, Norwood (1996)
97. Xu, L., Krzyzak, A., Suen, C.Y.: Methods of combining multiple classifiers and their applications to handwriting recognition. IEEE Trans. Syst. Man Cybern. **22**(3), 418–435 (1992)

98. Zhang, C., Wu, C., Blanzieri, E., Zhou, Y., Wang, Y., Du, W., Liang, Y.: Methods for labeling error detection in microarrays based on the effect of data perturbation on the regression model. Bioinformatics **25**(20), 2708–2714 (2009)
99. Zhong, S., Khoshgoftaar, T.M., Seliya, N.: Analyzing software measurement data with clustering techniques. IEEE Intell. Syst. **19**(2), 20–27 (2004)
100. Zhu, X., Wu, X.: Class noise vs. attribute noise: a quantitative study. Artif. Intell. Rev. **22**, 177–210 (2004)
101. Zhu, X., Wu, X.: Class noise handling for effective cost-sensitive learning by cost-guided iterative classification filtering. IEEE Trans. Knowl. Data Eng. **18**(10), 1435–1440 (2006)
102. Zhu, X., Wu, X., Chen, Q.: Eliminating class noise in large datasets. In: Proceeding of the Twentieth International Conference on Machine Learning, pp. 920–927 (2003)
103. Zhu, X., Wu, X., Chen, Q.: Bridging local and global data cleansing: Identifying class noise in large, distributed data datasets. Data Min. Knowl. Discov. **12**(2–3), 275–308 (2006)
104. Zhu, X., Wu, X., Yang, Y.: Error detection and impact-sensitive instance ranking in noisy datasets. In: Proceedings of the Nineteenth National Conference on Artificial Intelligence, pp. 378–383. AAAI Press (2004)