Received: 18 December 2022 | Revised: 12 February 2023 | Accepted: 2 March 2023

DOI: 10.1002/cem.3472

RESEARCH ARTICLE

Journal of CHEMOMETRICS WILEY

Noise simulation in classification with the noisemodel R package: Applications analyzing the impact of errors with chemical data

José A. Sáez

Department of Statistics and Operations Research, University of Granada, Fuente Nueva s/n, Granada 18071, Spain

Correspondence
José A. Sáez, Department of Statistics and Operations Research, University of Granada, Fuente Nueva s/n, 18071 Granada, Spain.
Email: joseasaezm@ugr.es

Funding information
Funding for open access charge: University of Granada / CBUA.

Abstract

Classification datasets created from chemical processes can be affected by errors, which impair the accuracy of the models built. This fact highlights the importance of analyzing the robustness of classifiers against different types and levels of noise to know their behavior against potential errors. In this context, noise models have been proposed to study noise-related phenomenology in a controlled environment, allowing errors to be introduced into the data in a supervised manner. This paper introduces the *noisemodel* R package, which contains the first extensive implementation of noise models for classification datasets, proposing it as support tool to analyze the impact of errors related to chemical data. It provides 72 noise models found in the specialized literature that allow errors to be introduced in different ways in classes and attributes. Each of them is properly documented and referenced, unifying their results through a specific S3 class, which benefits from customized print, summary and plot methods. The usage of the package is illustrated through four application examples considering real-world chemical datasets, where errors are prone to occur. The software presented will help to deepen the understanding of the problem of noisy chemical data, as well as to develop new robust algorithms and noise preprocessing methods properly adapted to different types of errors in this scenario.

KEYWORDS

attribute noise, chemical data, classification, label noise, noise models

1 | INTRODUCTION

Real-world data are often subject to errors or noise in a variety of fields.[1,2] Among them, datasets coming from chemical analyses and procedures in diverse applications, such as agri-food or medicine, may be particularly prone to imperfections.[3,4] These errors proceed from different sources associated with chemical practice, including human action and the instruments used, among others.[5,6] Thus, noisy data can come from human errors when performing experiments, such as inaccuracies when diluting ingredients or small spills when handling chemicals during transfer.[7] Another sources of

error occur when estimating a measurement. For example, a mark may be exceeded when a beaker is filled to a specific volume or several experts may estimate different times in reactions involving a color change.[8] On the other hand, incorrect calibration of certain chemical instruments or equipment, which is not performed with proper regularity, can lead to inaccuracies in experiments and data collection.[5] Laboratory equipment limitations when making measurements, such as the precision usually associated with measuring flasks, are also a potential source of error.[5,8]

In this context, applying the results of chemical procedures to create classification datasets may imply the introduction of noise in the form of corruptions in both output class labels and input attributes.[9,10] Learning from noisy data usually involves building less accurate and more complex models as a result of overfitting errors.[11] These facts have led to the development of a multitude of proposals to overcome the inconveniences of noise, such as robust algorithms or noise preprocessing techniques.[12,13] Robust algorithms[12] are methods characterized by being less influenced by noisy data. An example of a robust method is C4.5,[14] which includes pruning mechanisms to reduce the influence of errors in model building. Other well-known examples are *Repeated Incremental Pruning to Produce Error Reduction* (RIPPER)[15] and *Random Forest* (RF),[16] which are widely used in the specialized literature. On the other hand, within noise preprocessing techniques, the main representatives are known as noise filters.[13] They identify and remove noisy samples from the data, which has proven to be advantageous in many domains.[17] Furthermore, the separation of noise detection and learning avoids the usage of noisy samples during the classifier building process.

In R,[18] there is a variety of software packages implementing proposals belonging to the above approaches.[19,20] For example, *robustDA*[21] implements *Robust Mixture Discriminant Analysis*, which allows building robust classifiers when data suffer from noise in class labels. Similarly, the *AdaSampling* package[22] codes an adaptive sampling procedure, which is also applicable for learning with mislabeled data. The C4.5 and RIPPER robust methods can be found in the *RWeka* package,[23] whereas RF is implemented in *randomForest*.[24] Many of the aforementioned robust algorithms can also be found in the well-known *caret* package.[19] Related to these approaches, it is worth noting another type of robust techniques known as high-breakdown methods.[4,25,26] Some of them are implemented in the *rrcov*[26] and *rrcovHD*[27] packages. On the other hand, the *cellWise* package[28] contains procedures to detect cellwise outliers and robust methods to analyze contaminated data,[29,30] as well as functions to generate multivariate normal datasets with various types of outliers.[31,32] Among the software devoted to noisy data preprocessing, the *fmf* package[33] proposes a fast class noise detector, which provides a noise score for each of the samples in the dataset, whereas the *UBL* package[20] implements various filtering methods with a special focus on imbalanced learning.

The above algorithms, either robust or noise preprocessing methods, are designed to deal with data in which noise may have particular characteristics.[11,34] However, noise affecting real-world data and, particularly, chemical datasets, is usually not quantifiable, and its characteristics are unknown. In order to test the effectiveness of these methods in a controlled environment, noise models[35] have been proposed to introduce errors into datasets in a supervised way. The utility of noise models in the field of noisy data research is clearly reflected by their wide use in the specialized literature, as they are frequently employed in dozens of papers published each year in this area.[17,36,37] Because they facilitate controlling the type of noise injected into the data, as well as its amount and characteristics, these models allow the design of experimental frameworks appropriate to the objective of the study and draw significant conclusions from them.[9,36] Their usage allows, for example, studying the circumstances (types and levels of noise) in which classifiers are more affected or evaluating the effectiveness of noise preprocessing techniques to detect and treat the errors introduced.[12,17]

However, the usefulness of noise models is not limited solely to the field of research. They also have a practical utility, because they can be used by any practitioner with the aim of analyzing the impact of noise on the particular dataset addressed. Even though the robustness of well-known algorithms has already been studied in previous works, it is important to note that their behavior is not always uniform and may depend on the specific data processed.[1,12,38] Thus, as a consequence of an exhaustive experimental study, Nettleton et al[12] concluded that the behavior of each technique depends on the type and percentage of noise, the class imbalance, and the characteristics of the datasets themselves. This fact implies that an algorithm known to be robust does not always have to offer the best result in all the datasets affected by noise. It should also be noted that variations in the parameters of algorithms can affect their robustness and that there are classification algorithms whose robustness under different noise circumstances has not been sufficiently studied.[1,39] Something similar occurs with noise preprocessing algorithms, whose efficiency depends on the characteristics of the processed data.[40] For the above reasons, in practice, when it is desired to study the impact of noise on new classification problems, it would be advisable to previously simulate such errors using noise models and thus analyze their impact in a controlled manner. Despite the importance of these models both in research and in practice for noisy data analysis, to date, most of them are not yet implemented in any software package.

This paper presents the *noisemodel* R package, which provides the implementation of a total of 72 noise models for classification data found in the specialized literature.[35] It includes methods to introduce label noise, attribute noise, and both in combination.[10,41] The software is built following an S3 formulation of methods (with default and formula class alternatives), allowing the models to be applied in a unified and user-friendly way. The package is available from the *Comprehensive R Archive Network* (CRAN) at https://CRAN.R-project.org/package=noisemodel. It has complete documentation* available to the user, providing details and examples of use of each of the functions implemented. In addition to these usage examples, this paper shows the functionality of the package through the analysis of the impact of errors in different applications that make use of real-world chemical data. Thus, the noise models included in this software will provide the field of chemometrics with a solid support for the research and development of multiple applications related to noisy data, including:

- deepening the study of the impacts of noisy chemical data in the field of classification;
- providing tools proposed in the specialized literature to introduce errors simulating those in real-world data;
- accelerating the development of new algorithms dealing with erroneous chemical data;
- verifying how existing classifiers behave when some adverse effect causes inaccuracies in the data;
- creation of data with controlled errors to test the effectiveness of preprocessing methods for noise treatment; and
- easing the experimentation and reproducibility of research results related to noisy chemical data.

The rest of this paper is organized as follows. Section 2 presents an introduction to noise models in classification. Section 3 details the usage of the *noisemodel* package, from its installation to the application of each of its functionalities. Then, Section 4 shows practical examples to illustrate the use of the software on real-world chemical datasets. Finally, Section 5 concludes this work offering some final remarks.

2 | NOISE MODELS IN CLASSIFICATION

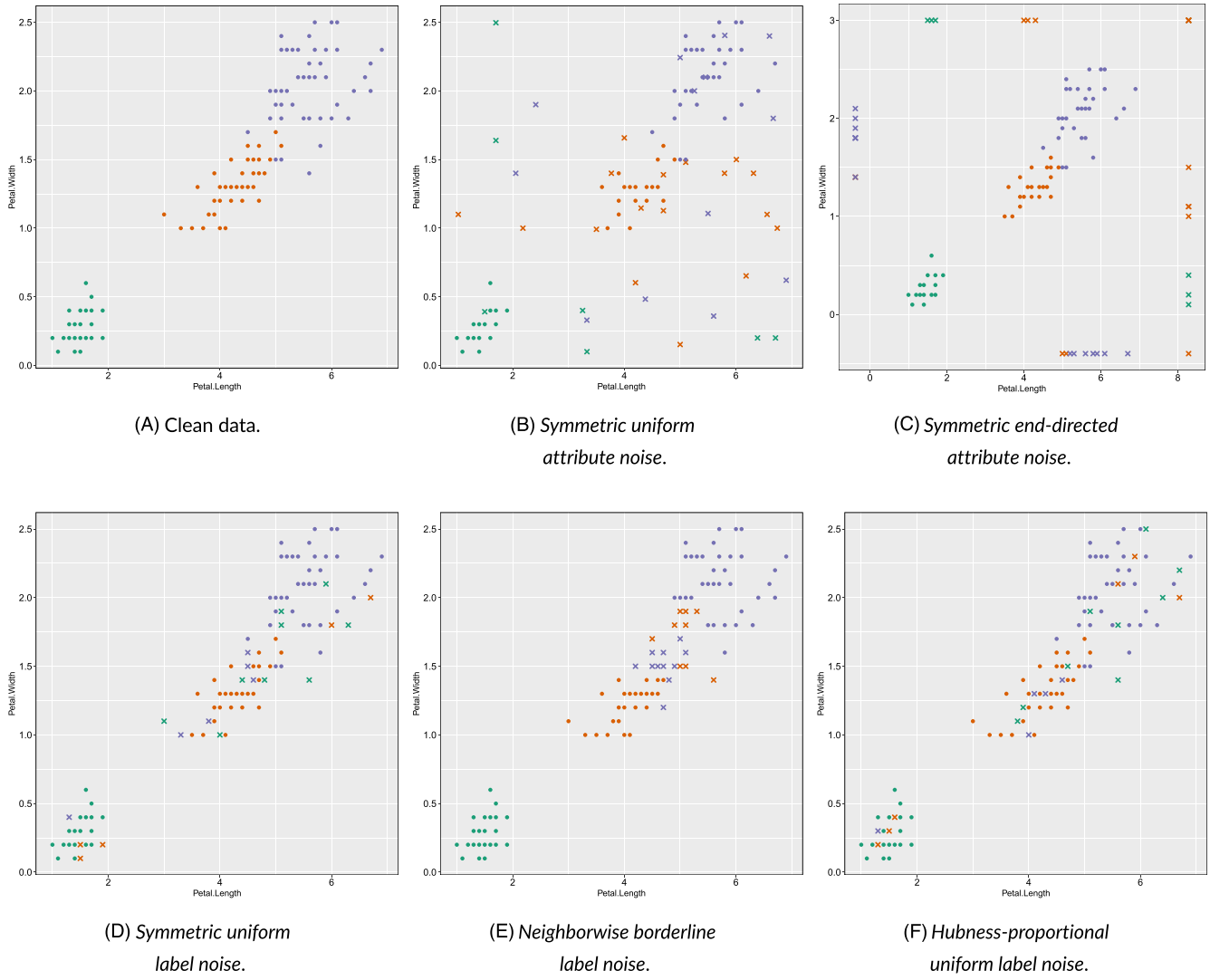
Consider a classification dataset composed by n samples x_i ($i \in \{1,...,n\}$), m input attributes v_j ($j \in \{1,...,m\}$) and one output class label v_0 taking one value in $\mathcal{L} = \{1,...,c\}$. The value of the variable v_j in the sample x_i is denoted as $x_{i,j}$. If $x_{i,j}$ is corrupted by the noise model, it is represented as $\overline{x}_{i,j}$. In classification, noise models aim at simulating errors in class labels and attribute values occurring in real-world datasets.[10,41] The main sources of label noise are often related to human error, primarily due to inaccurate information or subjectivity during the labeling process.[9] On the other hand, attribute noise is usually related to instrumental measurement errors.[11] As an example, Figure 1 shows a two-dimensional version of the well-known *iris* dataset before (Figure 1A) and after (Figure 1B–F) the application of several noise models implemented in the *noisemodel* package considering a noise level $\rho = 0.2$, that is, 20% of the values in the dataset are noisy.

Label noise models[17,34] are designed to corrupt the class label $x_{i,0}$ of the samples x_i in a dataset. One of the most widely used models in this group is *Symmetric uniform label noise*[41] (Figure 1D), which introduces errors affecting any part of the domain. All samples x_i have the same probability ρ that their class label $x_{i,0}$ is corrupted. Once a sample x_i is chosen to be mislabeled in a dataset with c classes, its noisy label $\overline{x}_{i,0}$ is determined by the probability mass function:

$$f(\overline{x}_{i,0}) = \begin{cases} \dfrac{1}{c-1} & \text{if } \overline{x}_{i,0} \neq x_{i,0}, \\ 0 & \text{if } \overline{x}_{i,0} = x_{i,0}. \end{cases} \tag{1}$$

Instead of giving equal noise chance to all samples, other models assign a higher probability to those near decision limits, where they share similar characteristics.[17] These models use the distance d_i of each sample x_i to the decision boundaries to estimate its noise probability based on different probability density functions (pdf).[9] For example, the pdf of the exponential (Equation 2, left) or the gamma distribution (Equation 2, right) have been used[9]:

*https://cran.r-project.org/web/packages/noisemodel/noisemodel.pdf.



(A) Clean data. (B) *Symmetric uniform attribute noise.* (C) *Symmetric end-directed attribute noise.*

(D) *Symmetric uniform label noise.* (E) *Neighborwise borderline label noise.* (F) *Hubness-proportional uniform label noise.*

FIGURE 1 Application of label and attribute noise models on the *iris* dataset with a noise level $\rho = 0.2$. Each color represents a class. Circles (•) symbolize clean samples, whereas crosses ($\times$) are noisy samples.

$$f(d_i;\lambda) = \lambda e^{-\lambda d_i} \qquad f(d_i;\alpha,\beta) = \frac{d_i^{\alpha-1} e^{-\beta d_i} \beta^{\alpha}}{\Gamma(\alpha)}. \tag{2}$$

A different approach for borderline label noise is followed by *Neighborwise borderline label noise*[17] (Figure 1E), which computes a noise score $N(x_i) = d(x_i,x_j)/d(x_i,x_k)$ for each sample x_i based on the Euclidean distances d to its closest samples of the same (x_j) and different class (x_k). The $n \cdot \rho$ samples with highest scores are then chosen to be mislabeled. Models have also been designed to introduce label noise in other parts of the dataset. For example, *Hubness-proportional uniform label noise*[34] (Figure 1F) assigns a higher probability $P(x_i) = N_k(x_i)/(n \cdot k)$ of being corrupted to those samples x_i closer to hubs in the dataset, with $N_k(x_i)$ being the number of times that x_i is among the k nearest neighbors of any other sample.

Finally, there are models that follow other different strategies to introduce label noise. For example, in *Symmetric center-based label noise*,[42] the probability $\rho_{i,j}$ of confusing two classes $i,j \in \mathcal{L}$ is greater the closer they are

$$\rho_{i,j} = \frac{\sqrt{1/d_{i,j}}}{\sum_{k \neq i} \sqrt{1/d_{i,k}}} \cdot \rho, \tag{3}$$

with $d_{i,k}$ the distance between the centers of classes i and k. Other approaches consider the usage of default values to mislabel samples,[43] the existence of a hierarchy among classes,[44] or the peculiarities of label noise in binary classification.[36]

On the other hand, attribute noise models[45,46] are designed to simulate inaccuracies in attribute values $x_{i,j}$ $(j \geq 1)$ in the dataset. *Symmetric uniform attribute noise*[39] (Figure 1B) allows introducing errors affecting any part of the domain. It introduces $n \cdot \rho$ errors in each attribute v_j by randomly selecting a noisy value $\overline{x}_{i,j}$ following a uniform distribution within the attribute domain:

$$\overline{x}_{i,j} \sim \mathcal{U}[\min(v_j), \max(v_j)] | \overline{x}_{i,j} \neq x_{i,j}. \tag{4}$$

Other models introduce attribute noise in different areas in the dataset.[10,47] For example, *Symmetric end-directed attribute noise*[10] (Figure 1C) introduces errors in the limits of the problem domain. *Boundary/dependent Gaussian attribute noise*[47] alters samples close to decision limits by means of zero-mean additive Gaussian noise. A Gaussian error distribution is also used by *Symmetric Gaussian attribute noise*,[38] in which each noisy value $\overline{x}_{i,j}$ is obtained as

$$\overline{x}_{i,j} = x_{i,j} + \varepsilon | \varepsilon \sim \mathcal{N}(0, \sigma^2), \tag{5}$$

with $\sigma^2 = k^2(\max(v_j) - \min(v_j))^2$ and $k \in (0,1]$. Note that, even though most attribute noise models alter different samples in each variable, some of them corrupt all attributes in each sample simultaneously.[46,47] An example of this type of models is *Symmetric/dependent uniform attribute noise*,[46] which chooses noisy values uniformly in the domain.

An important question regarding noise simulation can arise when choosing which model may be the most suitable for the available dataset. Although the characteristics of noise occurring in real-world applications are difficult to know exactly, the different noise schemes have been proposed in specific contexts, which makes them more oriented to certain situations.[10,17] The most direct choice of model can be made depending on which type of variables (class or attributes) are most likely to be affected by errors in the dataset and, therefore, which type of noise is more interesting to study (label noise or attribute noise). However, as reflected above, each type of noise in turn allows for multiple ways of simulating its errors, for example, focusing on noise in decision boundaries, domain extrema, binary datasets, and so on. For a practical recommendation on the potential noise model to be used based on the knowledge of the problem treated, the characteristics of the available data, and the noise to be simulated, the reader may consult Sáez.[35]

3 | USAGE OF THE NOISEMODEL PACKAGE

3.1 | Installation

The *noisemodel* package is available from the CRAN repository at https://CRAN.R-project.org/package=noisemodel. It can be installed from the R command line by typing:

```
R> install.packages("noisemodel")
```

This command also installs all the dependencies of the software, which can be checked in the *Imports* section of the CRAN website. In order to access all the package's functions, it must be attached in the usual way:

```
R> library(noisemodel)
```

3.2 | Package content

The software offers the following functionality to the user:

- a total of 72 functions implementing noise models for classification data used in the specialized literature;
- the print, summary, and plot methods, which allow displaying the information on the noise introduction process concisely and visually; and

- a two-dimensional version of the well-known *iris* dataset, as well as a variant with discretized attributes, which are used for illustrative purposes in the code examples of each of the models.

The software offers a wide range of alternatives for noise introduction, with 54 models to introduce label noise, 16 for attribute noise, and two combine both types of noise simultaneously. All the implemented noise models can be consulted through a complete *Reference manual*[†] associated with the package. The print and summary methods allow displaying an outline of the information associated with the noise introduction process, whereas the plot method graphically represents the resulting noisy dataset (these are discussed in Section 3.6). Finally, the 2D version of the *iris* dataset (see Figure 1A) maintains the numeric input attributes *petal-length* and *petal-width*, as well as its factor output label. In order to simplify these data in the application examples, duplicate (equal samples) and contradictory samples (those with the same input attributes and different class labels) are removed. The command data(iris2D) allow loading this dataset, creating the iris2D data.frame. The discretized version can be loaded using data(diris2D).

3.3 | Documentation

Each one of the functions in the *noisemodel* package has been properly documented using *roxygen2*.[48] The documentation allows consulting specific information of each available function, including a brief description of the purpose of the function (*Description*), the modes to call the function (*Usage*), an explanation of the input parameters of the function and types of variables (*Arguments*), a description of the function referring to its input arguments (*Details*), the output variables of the function (*Value*), a relevant contribution in which the implemented functionality was used (*References*), and a complete R code block with an example of how the function can be used (*Examples*).

The above information can be accessed from the *Reference manual* in the CRAN webpage. Furthermore, the documentation pages can be examined with the ? and help orders. For example, in order to access the documentation for the aforementioned *Symmetric uniform label noise* (sym_uni_ln),[41] one can type:

```
R> ?sym_uni_ln
R> help(sym_uni_ln)
```

3.4 | Application of noise models

The *noisemodel* package provides two standard ways to call the noise introduction models:

- a *default* method, which receives the input attributes in the x argument (a data.frame) and the class labels of each sample in the y argument (a factor vector), and
- a *formula* class method, which receives the formula argument (an expression relating the input attributes and the output class) and the data argument (a data.frame containing the variables in the formula).

In addition to these parameters, each noise model function has as arguments a series of specific parameters, as well as a sortid parameter indicating whether the indices of clean and noisy samples should be sorted in the output (sortid = TRUE by default, but can be set to FALSE to save time applying the model).

The name of the function associated with each noise model is divided into three parts, which are separated by the symbol _ and are composed of three to four letters referring to each identifier of the tripartite nomenclature of noise models.[35] For example, the function name for *Symmetric uniform label noise*[41] is sym_uni_ln, for *Hubness-proportional uniform label noise*[34] is hubp_uni_ln, and for *Gaussian-level uniform label noise*[49] is glev_uni_ln. Below are some examples of the usage of the default function for these noise models along with their arguments in different colors (red for dataset arguments, blue for specific arguments, and brown for the index sorting argument):

[†]https://cran.r-project.org/web/packages/noisemodel/noisemodel.pdf

```
sym_uni_ln(x, y, level, sortid = TRUE, ...)

hubp_uni_ln(x, y, level, k = 3, sortid = TRUE, ...)

glev_uni_ln(x, y, level, sd = 0.01, sortid = TRUE, ...)
```

Table 1 shows the rest of function names for each noise model, along with a reference on which the implementation is based and the parameters for the default function. A description of each implemented noise model can be found in the *Reference Manual* associated with the package, as well as using the aforementioned ? and help orders, which also detail each of the arguments. Note that, in order to standardize the functions and make them applicable to all types of data, some may differ slightly from the exact approach used in the associated work. For example, the models used in Petety et al,[46] which are employed on data with binary attributes, have been adapted to deal with higher cardinality attributes.

3.5 | The ndmodel class

All functions implementing a noise model return an object of ndmodel class. This S3 class, which is designed to unify the return value of all noise schemes inside the *noisemodel* package, is a list containing 10 elements with the most relevant information of the process:

- xnoise: a data.frame with the noisy input attributes.
- ynoise: a factor vector with the noisy output class.
- numnoise: an integer vector with the amount of noisy samples per variable.
- idnoise: an integer vector list with the indices of noisy samples per variable.
- numclean: an integer vector with the amount of clean samples per variable.
- idclean: an integer vector list with the indices of clean samples per variable.
- distr: an integer vector with the samples per class in the original data.
- model: the full name of the noise model used.
- param: a list of the argument values.
- call: the function call.

In order to verify the structure of a ndmodel object in a practical example, the resulting value returned by the sym_uni_ln method can be inspected:

```
R> output <- sym_uni_ln(x = iris2D[,-ncol(iris2D)], y = iris2D[,ncol(iris2D)], level = 0.1)
```

The output variable is a list with the 10 aforementioned elements. They can be easily obtained using the $ operator. The elements xnoise and ynoise are a data.frame and a factor vector of equal size as the original x and y arguments, respectively, but they contain the version with noisy values. Note that for label noise models, xnoise is equal to x and, for attribute noise models, ynoise is equal to y.

The numnoise and numclean elements show the number of noisy and clean samples in the dataset, respectively. In label noise models, they indicate these amounts per class, whereas in attribute noise models, they indicate amounts per attribute. On the other hand, the idnoise and idclean elements refer, for each corrupted variable, to indices of the noisy and clean samples, respectively. Therefore, the number of noisy samples and their indices in the above code can be queried using:

```
R> output$numnoise
setosa versicolor virginica
3         2         5
R> output$idnoise
[[1]]
[1] 10 14 22 27 35 58 61 65 68 71
```

TABLE 1 Default functions implementing each noise model in the *noisemodel* package, along with their main parameters.

Noise type	Function	Ref.	Tuning parameters	Function	Ref.	Tuning parameters
Label	asy_def_ln	[37]	x, y, level, def, order, sortid	pmd_con_ln	[50]	x, y, level, sortid
	asy_spa_ln	[51]	x, y, levelO, levelE, order, sortid	qua_uni_ln	[52]	x, y, level, att1, att2, sortid
	asy_uni_ln	[53]	x, y, level, order, sortid	sco_con_ln	[54]	x, y, level, sortid
	attm_uni_ln	[55]	x, y, level, sortid	sigb_uni_ln	[56]	x, y, level, order, sortid
	clu_vot_ln	[57]	x, y, k, sortid	smam_bor_ln	[58]	x, y, level, k, sortid
	exp_bor_ln	[9]	x, y, level, rate, k, sortid	smu_cuni_ln	[59]	x, y, level, sortid
	exps_cuni_ln	[60]	x, y, level, sortid	sym_adj_ln	[61]	x, y, level, order, sortid
	fra_bdir_ln	[62]	x, y, level, sortid	sym_cen_ln	[42]	x, y, level, sortid
	gam_bor_ln	[63]	x, y, level, shape, rate, k, sortid	sym_con_ln	[64]	x, y, level, sortid
	gau_bor_ln	[65]	x, y, level, mean, sd, k, sortid	sym_cuni_ln	[66]	x, y, level, sortid
	gaum_bor_ln	[65]	x, y, level, mean, sd, w, k, sortid	sym_ddef_ln	[67]	x, y, level, def1, def2, order, sortid
	glev_uni_ln	[49]	x, y, level, sd, sortid	sym_def_ln	[43]	x, y, level, def, order, sortid
	hubp_uni_ln	[34]	x, y, level, k, sortid	sym_dia_ln	[37]	x, y, level, order, sortid
	irs_bdir_ln	[36]	x, y, level, sortid	sym_dran_ln	[68]	x, y, level, sortid
	lap_bor_ln	[69]	x, y, level, mu, b, k, sortid	sym_exc_ln	[70]	x, y, level, sortid
	larm_uni_ln	[58]	x, y, level, sortid	sym_hie_ln	[71]	x, y, level, group, order, sortid
	maj_udir_ln	[72]	x, y, level, sortid	sym_hienc_ln	[44]	x, y, level, group, order, sortid
	mind_bdir_ln	[73]	x, y, level, pos, sortid	sym_natd_ln	[37]	x, y, level, sortid
	minp_uni_ln	[74]	x, y, level, sortid	sym_nean_ln	[75]	x, y, level, sortid
	mis_pre_ln	[57]	x, y, sortid	sym_nexc_ln	[76]	x, y, level, order, sortid
	mulc_udir_ln	[57]	x, y, level, goal, order, sortid	sym_nuni_ln	[77]	x, y, level, tramat, sortid
	nei_bor_ln	[17]	x, y, level, k, sortid	sym_opt_ln	[37]	x, y, level, levelH, order, sortid
	nlin_bor_ln	[17]	x, y, level, k, sortid	sym_pes_ln	[37]	x, y, level, levelL, order, sortid
	oned_uni_ln	[78]	x, y, level, att, lower, upper, order, sortid	sym_uni_ln	[41]	x, y, level, sortid
	opes_idnn_ln	[75]	x, y, level, openset, order, sortid	sym_usim_ln	[79]	x, y, level, sortid
	opes_idu_ln	[75]	x, y, level, openset, order, sortid	ugau_bor_ln	[69]	x, y, level, mean, sd, k, order, sortid
	pai_bdir_ln	[80]	x, y, level, pairs, order, sortid	ulap_bor_ln	[69]	x, y, level, mu, b, k, order, sortid
Attribute	asy_int_an	[45]	x, y, level, nbins, sortid	sym_sgau_an	[2]	x, y, level, k, sortid
	asy_uni_an	[46]	x, y, level, sortid	sym_uni_an	[39]	x, y, level, sortid
	boud_gau_an	[47]	x, y, level, k, sortid	symd_gau_an	[81]	x, y, level, k, sortid
	imp_int_an	[45]	x, y, level, nbins, ascending, sortid	symd_gimg_an	[82]	x, y, level, sortid
	sym_cuni_an	[83]	x, y, level, sortid	symd_rpix_an	[82]	x, y, level, sortid
	sym_end_an	[10]	x, y, level, scale, sortid	symd_uni_an	[46]	x, y, level, sortid
	sym_gau_an	[38]	x, y, level, k, sortid	unc_fixw_an	[84]	x, y, level, k, sortid
	sym_int_an	[45]	x, y, level, nbins, sortid	unc_vgau_an	[81]	x, y, level, sortid
Combined	sym_cuni_cn	[83]	x, y, level, sortid	uncs_guni_cn	[85]	x, y, level, k, sortid

Note: Note that functions of the formula class exchange the x and y parameters for the formula and data arguments.

In this example, numnoise shows that 3, 2, and 5 samples of the *setosa*, *versicolor*, and *virginica* classes have been, respectively, corrupted. Finally, the rest of the elements (distr, model, param, and call) provide additional information about the noise introduction process, which is especially useful for the print, summary, and plot methods described in the next section.

3.6 | print, summary, and plot methods

In order to display the result of the noise introduction process contained in the ndmodel object after applying a noise model, specific print, summary, and plot methods have been implemented. The print function receives as argument the object of ndmodel class. After its usage, brief information about the noise injection process is displayed in the R console including the name of the noise introduction model, the parameters associated with the noise model and the number of noisy and clean values in the dataset. Thus, to use this function on the output object obtained above, one can simply type:

```
R> print(output)
```

On the other hand, the summary method receives as arguments both an object of ndmodel class and the showid parameter indicating whether the indices of noisy samples should be displayed. The information provided by summary is similar to that of print, but it adds some extra elements, such as the function call, the number of noisy samples per class/attribute, the number of clean samples per class/attribute, and the indices of the noisy samples (if showid = TRUE). The function call has the following form:

```
R> summary(output, showid = TRUE)
```

Finally, the ndmodel class also allows displaying a graphical representation of the resulting noisy dataset using the plot method, which has the following structure:

```
R> plot(output, noise = NA, xvar = 1, yvar = 2, pca = FALSE)
```

where

- noise is a logical indicating which samples should be displayed on the graph, including the noisy samples (noise = TRUE), the clean samples (noise = FALSE) or all of them (noise = NA).
- pca is a logical indicating if a PCA approach using singular value decomposition must be used before displaying the graph. This technique can be useful for visualizing data with more than two dimensions by plotting, for example, the principal components with the highest percentages of explained variance.
- xvar and yvar are two integers with the indices of the attributes (if pca = FALSE) or the principal components (if pca = TRUE) to represent in the x and y axes of the resulting plot, respectively.

4 | USAGE OF THE NOISEMODEL PACKAGE ANALYZING THE IMPACT OF ERRORS IN CHEMICAL DATA

This section presents four application examples in which the *noisemodel* package is used with several chemical datasets. Section 4.1 focuses on studying the prediction of different types of wine samples (clean and noisy) based on their constituents. Then, Section 4.2 aims to analyze the noise robustness of different classification algorithms in distinguishing real celadons from imitations, whereas Section 4.3 studies the impact of noise on different components by classifying various types of small pieces of glass. Finally, Section 4.4 discusses the importance of accounting for randomness in experiments with noise models and provides some general recommendations in this regard.

4.1 | Performance predicting clean and noisy samples

In order to exemplify the first application, data resulting from a chemical analysis of wines from three different cultivars in a region of Italy are used.[86] The dataset consists of 178 wine samples, in which the amounts of 13 wine constituents are determined, including alcohol, malic acid, magnesium, phenols, and flavonoids, among others. The training set will consist of 70% of the data, which are used to create a classifier using the C4.5 algorithm with the aim of predicting the cultivar from the chemical characteristics of the wine. Thus, 70% of the original dataset is selected and the model is created using the createDataPartition and train functions from the *caret* package, respectively:

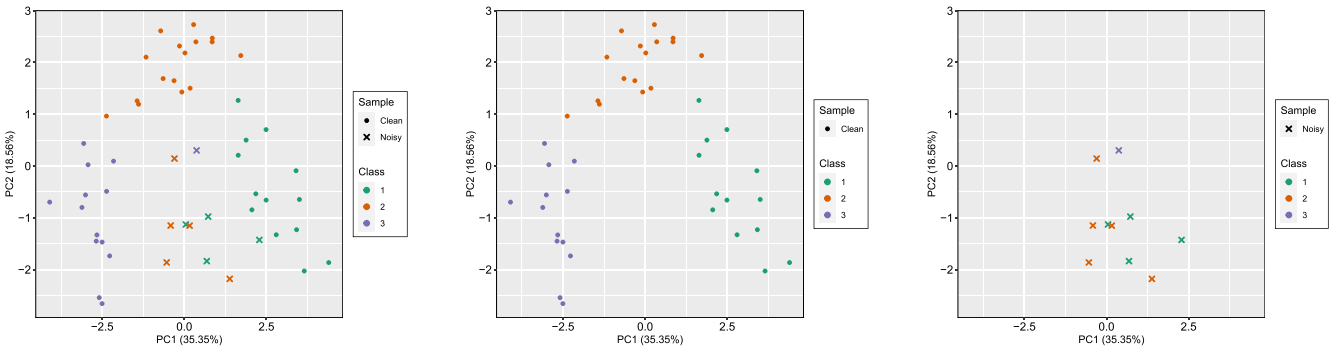
```
R> tra_idx <- createDataPartition(y = data[,ncol(data)], p = 0.7, list = FALSE, times = 1)
R> tra <- data[tra_idx,]
R> model <- train(x = tra[,-ncol(tra)], y = tra[,ncol(tra)], method = "J48", trControl =
     trainControl(method = "none"))
```

In the code above, data is a data.frame with the original dataset with the class label in its last column, tra is the training set, and model contains the C4.5 classifier (note that C4.5 is called J48 in the train function of the *caret* package). The model created is used to predict the cultivar in samples that will be received in the future from different laboratories devoted to chemical analysis (represented by the test set with 30% samples remaining). Imagine that, due to instrumental and transcription errors in the data, around 20% of the samples could contain random errors in their attributes. In order to simulate this situation and analyze the behavior of the classifier built in this scenario, noise is introduced into the test set (tst) using a noise model included in the *noisemodel* package. Note that this noise scheme (clean training/noisy test) could also occur in other applications and has been previously considered in relevant papers on noisy data.[1,12] The *Symmetric/dependent uniform attribute noise* model[46] (symd_uni_an) will be employed to simulate errors in this case, as it allows the attribute values to be randomly altered for a proportion level of the samples:

```
R> tst <- data[-tra_idx,]
R> tst_noise <- symd_uni_an(x = tst[,-ncol(tst)], y = tst[,ncol(tst)], level = 0.2)
```

Information about the noise introduction process can be obtained by using the commands print(tst_noise) or summary(tst_noise) described in Section 3.6. The graphical representation of the different types of samples in this noisy test data can also be obtained using the plot function (see Figure 2):

```
R> plot(tst_noise, pca = TRUE, noise = NA)
R> plot(tst_noise, pca = TRUE, noise = FALSE)
R> plot(tst_noise, pca = TRUE, noise = TRUE)
```



(A) Test set: all samples (`pca = TRUE, noise = NA`).

(B) Test set: clean samples (`pca = TRUE, noise = FALSE`).

(C) Test set: noisy samples (`pca = TRUE, noise = TRUE`).

FIGURE 2 Results of applying the plot function on the test set in the first application example, with pca = TRUE and varying the noise argument. Its remaining parameters, xvar and yvar, are left with the default values 1 and 2, respectively.

These graphs show three different groups for clean samples in the test set (Figure 2B), whereas noisy samples are far from them (Figure 2C). Finally, in order to study how the classifier created will behave in this scenario, the labels of the test data created with the *noisemodel* package are predicted and the classification accuracy is computed. For this, the xnoise and ynoise elements of the ndmodel class must be accessed:

```
R> pred <- predict(model, tst_noise$xnoise)
R> sum(pred == tst_noise$ynoise)/length(tst_noise$ynoise)
[1] 0.8269231
```

Similarly, because the indices of the clean and noisy samples in the test set are known through the idclean and idnoise elements of the ndmodel class, the accuracy classifying these types of samples can also be computed:

```
R> idclean <- tst_noise$idclean[[1]]
R> sum(pred[idclean] == tst_noise$ynoise[idclean])/length(idclean)
[1] 0.9761905
```

```
R> idnoise <- tst_noise$idnoise[[1]]
R> sum(pred[idnoise] == tst_noise$ynoise[idnoise])/length(idnoise)
[1] 0.2
```

The above results show that the classifier built has, in general, a test accuracy of 82.69% predicting the whole test data containing both clean and noisy samples. However, because noisy samples have been introduced in a controlled way, this result can be broken down according to the type of samples (clean or noisy) classified. Thus, whereas 97.62% of the clean test samples are correctly classified, only 20% of the noisy test samples are predicted without error. This analysis shows the need to reinforce the attention on this type of noisy samples in the future and apply mechanisms to detect errors during their classification due to their impact on system performance.

4.2 | Analysis of robustness to noise

Robustness refers to the ability of algorithms to build models that are less affected by noisy data.[87] In the second application example, the robustness of different classification algorithms when training with noisy chemical data is analyzed. In order to exemplify this scenario, a dataset with samples of Longquan celadon is used,[88] in which an analysis has been carried out to determine their chemical components, such as sodium oxide, magnesium oxide, silicon dioxide, or aluminum oxide. The classification task will consist of distinguishing real from imitations samples of Longquan celadon based on their chemical composition. First, similarly to Section 4.1, 70% of the data is considered for the training set and the remaining 30% for the test set:

```
R> tra_idx <- createDataPartition(y = data[,ncol(data)], p = 0.7, list = FALSE, times = 1)
R> tra <- data[tra_idx,]
R> tst <- data[-tra_idx,]
```

Because labeling is performed manually, let us assume that the distinction between types of samples can generate label noise. Because of this, in order to create a robust model from the data, practitioners decide to choose an alternative between two algorithms considered to be noise tolerant: RIPPER[15] or RF[16] (which are called JRip and rf in *caret*, respectively). In order to determine which of these algorithms is the most robust against noise in this domain, an experiment introducing noise into the class labels using the well-known *Symmetric uniform label noise*[41] (sym_uni_ln) is performed. Different noise levels, from 0 to 0.3 (by increments of 0.1), are introduced into the training set, keeping the test set unchanged. Thus, the impact of noise on the training process and the models created can be studied from variations in performance on the same test set.[39] To compute the test accuracy of each algorithm, RIPPER and RF, by corrupting the training set with each noise level $\rho \in \{0,0.1,0.2,0.3\}$, the following block of code can be used:

```
R> nl <- seq(0, 0.3, 0.1)
R> method <- c("JRip", "rf")
R> acc <- matrix(data = -1, nrow = length(method), ncol = length(nl), dimnames = list(method, nl))
R> for(p in 1:length(nl)){
   tra_noise <- sym_uni_ln(x = tra[,-ncol(tra)], y = tra[,ncol(tra)], level = nl[p])
   for(m in 1:length(method)){
   model <- train(x = tra_noise$xnoise, y = tra_noise$ynoise, method = method[m], trControl =
   trainControl(method = "none"))
   pred <- predict(model, tst[,-ncol(tst)])
   acc[m,p] <- sum(pred == tst[,ncol(tst)])/nrow(tst)
  }
}
```

Once the accuracy (acc) for each algorithm and noise level is obtained, an analysis of the robustness to noise of RIPPER and RF in this noisy chemical dataset is performed. For this purpose, the *Equalized Loss of Accuracy* (ELA)[87] metric can be employed, which allows analyzing the robustness of classification algorithms against noise. It measures the loss of accuracy due to noise from a perfect score weighted by the accuracy with no noise:

$$ELA_\rho = \frac{1 - A_\rho}{A_0} \tag{6}$$

with ELA_ρ being the value of the ELA metric for a noise level ρ, A_0 is the accuracy of a classifier without noise, and A_ρ is the accuracy with a noise level ρ. Lower ELA values indicate a higher robustness of the classification algorithm. Thus, the ELA robustness measure is studied for each method and noise level, obtaining the following results:

```
R> ela <- matrix(data = -1, nrow = length(method), ncol = length(nl), dimnames = list(method, nl))
R> for(p in 1:length(nl)){
   for(m in 1:length(method)){
      ela[m,p] <- (1-acc[m,p])/acc[m,1]
   }
}
R> ela
```

	0	0.1	0.2	0.3
JRip	0.04545455	0.09090909	0.18181820	0.63636360
rf	0.00000000	0.04347826	0.13043480	0.21739130

These results show that the robustness of both classification algorithms is affected as the noise level increases, because ELA worsens as the number of errors increases. In addition, RF is more robust than RIPPER in the studied example, because it obtains lower ELA values in all the noise levels. Thus, in case of building classifiers in this domain with noisy data, it is advisable to use RF instead of RIPPER because the models created will be less affected by errors in the data.

4.3 | Impact of noise on different attributes

The third application example considers a dataset focused on distinguishing 6 types of glass defined in terms of 9 variables, including their oxide content (such as sodium, iron, potassium, and others) and refractive index.[89] This classification problem is motivated by criminological research, because the correct identification of small pieces of glass can be used as evidence. In this case, the impact of noise on different attributes will be studied. After the training (tra) and test

(tst) sets are created, noise is introduced into only one of their attributes using an asymmetric attribute noise model (asy_uni_an[46]), which allows a different noise level to be injected into each attribute. In the experiment below, a noise level of 20% is considered and the classifiers are created with C4.5:

```
R> acc <- rep(-1, 9)
R> names(acc) <- names(data)[-ncol(data)]
R> for(i in 1:9){
   nl <- rep(0, 9)
   nl[i] <- 0.2
   tra_noise <- asy_uni_an(x = tra[,-ncol(tra)], y = tra[,ncol(tra)], level = nl)
   tst_noise <- asy_uni_an(x = tst[,-ncol(tst)], y = tst[,ncol(tst)], level = nl)
   model <- train(x = tra_noise$xnoise, y = tra_noise$ynoise, method = "J48", trControl =
      trainControl(method = "none"))
   pred <- predict(model, tst_noise$xnoise)
   acc[i] <- sum(pred == tst_noise$ynoise)/length(tst_noise$ynoise)
}
R> acc
```

RI	Na	Mg	Al	Si
0.6885246	0.6557377	0.6721311	0.6885246	0.5409836
K	Ca	Ba	Fe	
0.6557377	0.5573770	0.6557377	0.6557377	

These results show that the noise that most deteriorates the classification performance is the one affecting Si and Ca, which obtain the lowest accuracies. On the other hand, noise in Al and the refractive index RI affect performance less. Thus, special attention should be paid to errors in those attributes that have a greater impact on performance in this domain to ensure that the classifier built is not greatly affected.

4.4 | On the randomness of experiments with noisy data

In the previous sections, both data partitioning and noise introduction are affected by randomness, which may affect the results and conclusions obtained. This section focuses on this issue and offers some recommendations for experimentation. In order to illustrate this aspect, consider the dataset with samples of celadon used above. After the training and test sets are created, 20% of noise will be introduced into the training set using the default method of the sym_uni_ln noise model. The result of the noise introduction process can be easily inspected using the print function:

```
R> outdef <- sym_uni_ln(x = tra[,-ncol(tra)], y = tra[,ncol(tra)], level = 0.2)
R> print(outdef)

## Noise model:
Symmetric uniform label noise

## Parameters:
- level = 0.2
- sortid = TRUE

## Number of noisy and clean samples:
- Noisy samples: 11/57 (19.30%)
- Clean samples: 46/57 (80.70%)
```

Similarly, the method of formula class can be used with the same noise model and level:

```
R> outfrm <- sym_uni_ln(formula = class ~ ., data = tra, level = 0.2)
```

In the code above, the formula argument allows indicating how attributes and class are related. The point . in the formula class∼. can be employed to indicate the use of all input attributes to predict the class label. By inspecting this output with the print(outfrm) command, one can check that it is identical to what was previously obtained with the default method. However, it should be noted that most of noise models follow random processes. Thus, although the percentages of clean and noisy samples are the same in both calls (which is the result of using the same noise level), the corrupted samples may be different. This fact can be verified by setting the argument showid = TRUE in the summary function, which shows the indices of the noisy samples. For example, when inspecting the output of the default method using summary, the following information is displayed:

```
R> summary(outdef, showid = TRUE)

#########################################################
Noise introduction process: Summary
#########################################################

## Original call:
sym_uni_ln(x = tra[, -ncol(tra)], y = tra[, ncol(tra)], level = 0.2)

## Noise model:
Symmetric uniform label noise

## Parameters:
- level = 0.2
- sortid = TRUE

## Number of noisy and clean samples:
- Noisy samples: 11/57 (19.30%)
- Clean samples: 46/57 (80.70%)

## Number of noisy samples per class label:
- Class FLQ-M: 2/19 (10.53%)
- Class other: 9/38 (23.68%)

## Number of clean samples per class label:
- Class FLQ-M: 17/19 (89.47%)
- Class other: 29/38 (76.32%)

## Indices of noisy samples:
- Output class: 19, 20, 21, 24, 29, 37, 40, 46, 47, 48, 49
```

Thus, both partitioning and noise introduction are usually random in experiments related to noisy data. In order to obtain the most robust results possible, it is recommended to carry out several iterations in experiments involving noise models and analyze the results obtained from all of them. A possible alternative to do this is to define a function experiment, which receives the original dataset and other parameters of interest as inputs, to partition the data, introduce noise and, finally, return the result of the experiment (for example, the classification accuracy). Then, it would be enough to call this function the desired number of iterations (niter) and study the distribution of the results in all the executions using descriptive metrics, such as mean and standard deviation. The repetition of an experiment with noisy data allows to increase the robustness of the results obtained and the significance of the conclusions reached. As an example, in an experiment with noisy data to analyze accuracy, code similar to the following could be used:

TABLE 2 Results after repeating 50 times the experiments in Sections 4.1–4.3.

Accuracy predicting clean and noisy samples				
All	0.8135 ± 0.0424		Impact of noise on different attributes	
Clean	0.9271 ± 0.0406		RI	0.6479 ± 0.0471
Noisy	0.3360 ± 0.1290		Na	0.6570 ± 0.0459
			Mg	0.6695 ± 0.0472
Analysis of robustness to noise			Al	0.6639 ± 0.0486
Level	RIPPER	RF	Si	0.6685 ± 0.0590
0	0.0649 ± 0.0584	0.0242 ± 0.0332	K	0.6666 ± 0.0555
0.1	0.0967 ± 0.0785	0.0477 ± 0.0424	Ca	0.6610 ± 0.0538
0.2	0.1554 ± 0.1213	0.1036 ± 0.0675	Ba	0.6616 ± 0.0544
0.3	0.2571 ± 0.1565	0.2167 ± 0.1029	Fe	0.6656 ± 0.0503

Note: For the prediction of clean and noisy samples (Section 4.1) and the impact of noise on different attributes (Section 4.3), the accuracy is shown. For the robustness analysis (Section 4.2), the ELA metric is studied.

```
R> res <- rep(-1, niter)
R> set.seed(20)
R> for(i in 1:niter){
   res[i] <- experiment(data)
}
```

In the code above, note the utility of setting the random seed with the set.seed() function before applying noise models to facilitate reproducibility of experiments. Following this approach, the analyses of the previous sections are repeated considering 50 iterations to generate different datasets in the noise introduction processes. Table 2 shows the results obtained in each application example of Sections 4.1–4.3 once all the iterations have been carried out.

The conclusions obtained in Sections 4.1 (prediction of clean and noisy samples) and 4.2 (robustness of classification algorithms) are maintained considering multiple iterations, although the results in absolute value may vary to some extent. The most notable contrast is the second application example in the comparison of RIPPER and RF with 30% noise, because the difference between the two is not as pronounced as in the individual experiment. Another important aspect is reflected by the standard deviations in Table 2: The first experiment shows that the greatest dispersion occurs when predicting noisy samples, whereas the second experiment shows that increasing the noise level also increases the dispersion of the robustness. These facts may be associated with the greater complexity in these scenarios.

The most disparate conclusions are obtained in the experiment analyzing the impact of noise on different attributes (Section 4.3). Even though the individual experiment showed that noise in the Si and Ca variables was the most detrimental to performance, the multiple iterations in Table 2 show a fairly homogeneous impact of noise on all attributes. In fact, the most damaging noise in this case is the one affecting the refractive index RI, which was one of the least affected variables in the individual experiment. These facts show the need to repeat several times experiments involving noise models in order to obtain meaningful conclusions based on the most robust results possible.

5 | CONCLUDING REMARKS

This paper has presented the *noisemodel* R package, proposing it as support tool in the analysis of impacts of errors related to chemical data. This contains the first extensive implementation of noise models for classification problems. In order to introduce the context and motivation for this work, the problem of noise with chemical data, their potential solutions found in related software, and the need to simulate it using noise models have been discussed. The models for introducing label and attribute noise have been briefly presented, offering an overview of possible alternatives in each case. Then, a detailed description of the *noisemodel* software has been provided, showing the installation process, contents of the package, and how to access the documentation of each function. Furthermore, the usage of the implemented functions has been illustrated, as well as their output, which is an object of ndmodel class. This class benefits from specific print, summary, and plot methods, which allow its content to be displayed.

The usage of the software has been shown through four illustrative application examples considering real-world chemical datasets. In the first application, where data from the chemical analysis of wines have been used, the *noisemodel* package has been employed as part of the analysis to estimate the accuracy on different types of samples (clean and noisy). Then, the second application has focused on the analysis of the robustness of various classification algorithms in the domain of the chemical composition of ceramics, whereas the third application has studied the impact of noise on different variables that determine the composition of several types of glass. Finally, the fourth application example has discussed the importance of accounting for randomness in experiments involving noise models.

Note that the possibility of having and being able to easily use a large number of noise models proposed in the specialized literature in a single software package is an advance in this field, because many of these models do not even have a public code that allows their usage both by the scientific community and practitioners in any field. Noise models are commonly used in the field of noisy data research, because they facilitate the control of the amount and characteristics of noise to obtain meaningful conclusions from the experiments.[17,37] On the other hand, the use of noise models in practical applications to analyze the impact of errors in a specific dataset can also be recommended, because there are works showing that the general behavior of classifiers and noise preprocessing methods can vary depending on the characteristics of the processed data.[12,40] In the context of noisy chemical data, due to the frequency of errors in real-world applications and its consequences when obtaining knowledge from them, the *noisemodel* package represents a useful tool in noisy data analysis and validation of classifiers and preprocessing techniques. The software presented will help both to delve into the consequences of errors in chemical data and to facilitate the design of new noise treatment techniques to mitigate their negative effects.

DATA AVAILABILITY STATEMENT

The software associated with this paper is openly available from the Comprehensive R Archive Network (CRAN) at https://CRAN.R-project.org/package=noisemodel.

ORCID

José A. Sáez https://orcid.org/0000-0002-4592-1538

REFERENCES

1. Zhu X, Wu X. Class noise vs. attribute noise: a quantitative study. *Artif Intell Rev*. 2004;22:177-210.
2. Koziarski M, Krawczyk B, Wozniak M. Radial-based oversampling for noisy imbalanced data classification. *Neurocomputing*. 2019;343:19-33.
3. Cappozzo A, Duponchel L, Greselin F, Murphy TB. Robust variable selection in the framework of classification with label noise and outliers: applications to spectroscopic data in agri-food. *Anal Chim Acta*. 2021;1153:338245.
4. Vranckx I, Raymaekers J, De Ketelaere B, Rousseeuw PJ, Hubert M. Real-time discriminant analysis in the presence of label and measurement noise. *Chemom Intell Lab Syst*. 2021;208:104197.
5. Analytical Methods Committee (AMCTB No 56). What causes most errors in chemical analysis? *Anal Methods*. 2013;5(12):2914-2915.
6. Bai Y, Wu Y. Analysis of influencing factors of data error in chemical analysis of iron and steel materials. *J Phys Conf Ser*. 2021;2133(1):12017.
7. Kuselman I, Pennecchi F, Epstein M, Fajgelj A, Ellison SLR. Monte Carlo simulation of expert judgments on human errors in chemical analysis—a case study of ICP-MS. *Talanta*. 2014;130:462-469.
8. Harvey D. *Modern Analytical Chemistry*. McGraw-Hill; 2000.
9. Bootkrajang J. A generalised label noise model for classification in the presence of annotation errors. *Neurocomputing*. 2016;192:61-71.
10. Khoshgoftaar TM, Hulse JV. Empirical case studies in attribute noise detection. *IEEE Trans Syst Man Cybern - Part C: Appl Rev*. 2009;39(4):379-388.
11. Sáez JA, Corchado E. ANCES: a novel method to repair attribute noise in classification problems. *Pattern Recognit*. 2022;121:108198.
12. 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*. 2010;33:275-306.
13. Sáez JA, Krawczyk B, Woźniak M. On the influence of class noise in medical data classification: treatment using noise filtering methods. *Appl Artif Intell*. 2016;30(6):590-609.
14. Han L, Li W, Su Z. An assertive reasoning method for emergency response management based on knowledge elements C4.5 decision tree. *Expert Syst Appl*. 2019;122:65-74.
15. Cohen WW. Fast effective rule induction. In: Proceedings of the 12th International Conference on Machine Learning. Morgan Kaufmann Publishers; 1995:115-123.
16. Breiman L. Random forests. *Mach Learn*. 2001;45(1):5-32.
17. Garcia LPF, Lehmann J, de Carvalho ACPLF, Lorena AC. New label noise injection methods for the evaluation of noise filters. *Knowl-Based Syst*. 2019;163:693-704.

18. Crawley MJ. *The R Book*. Wiley; 2012.
19. Kuhn M. Building predictive models in R using the caret package. *J Stat Softw*. 2008;28(5):1-26. https://CRAN.R-project.org/package=caret
20. Branco P, Ribeiro RP, Torgo L. UBL: an implementation of re-sampling approaches to utility-based learning for both classification and regression tasks. https://CRAN.R-project.org/package=UBL, R package version 0.0.7; 2021.
21. Bouveyron C, Girard S. robustDA: robust mixture discriminant analysis. https://CRAN.R-project.org/package=robustDA, R package version 1.2; 2020.
22. Yang P. AdaSampling: adaptive sampling for positive unlabeled and label noise learning. https://CRAN.R-project.org/package=AdaSampling, R package version 1.3; 2019.
23. Hornik K, Buchta C, Zeileis A. Open-source machine learning: R meets Weka. *Comput Stat*. 2009;24(2):225-232. https://CRAN.R-project.org/package=RWeka
24. Liaw A, Wiener M. Classification and regression by randomForest. *R News*. 2002;2(3):18-22. https://CRAN.R-project.org/package=randomForest
25. Hubert M, Rousseeuw PJ, Aelst SV. High-breakdown robust multivariate methods. *Stat Sci*. 2008;23(1):92-119.
26. Todorov V, Filzmoser P. An object-oriented framework for robust multivariate analysis. *J Stat Softw*. 2009;32(3):1-47. https://CRAN.R-project.org/package=rrcov
27. Todorov V. rrcovHD: robust multivariate methods for high dimensional data. https://CRAN.R-project.org/package=rrcovHD, R package version 0.2-7; 2021.
28. Raymaekers J, Rousseeuw P. cellWise: analyzing data with cellwise outliers. https://CRAN.R-project.org/package=cellWise, R package version 2.5.0; 2022.
29. Rousseeuw PJ, Bossche WVD. Detecting deviating data cells. *Technometrics*. 2018;60(2):135-145.
30. Hubert M, Rousseeuw PJ, den Bossche WV. MacroPCA: an all-in-one pca method allowing for missing values as well as cellwise and rowwise outliers. *Technometrics*. 2019;61(4):459-473.
31. Agostinelli C, Leung A, Yohai VJ, Zamar RH. Robust estimation of multivariate location and scatter in the presence of cellwise and casewise contamination. *Test*. 2015;24:441-461.
32. Raymaekers J, Rousseeuw P. Handling cellwise outliers by sparse regression and robust covariance. *J Data Sci Stat Visualisation*. 2021;1(3).
33. Zheng W, Jin M. fmf: fast class noise detector with multi-factor-based learning. https://CRAN.R-project.org/package=fmf, R package version 1.1.1; 2020.
34. Tomasev N, Buza K. Hubness-aware kNN classification of high-dimensional data in presence of label noise. *Neurocomputing*. 2015;160:157-172.
35. Sáez JA. Noise models in classification: unified nomenclature, extended taxonomy and pragmatic categorization. *Mathematics*. 2022;10(20):3736.
36. Chen B, Xia S, Chen Z, Wang B, Wang G. RSMOTE: a self-adaptive robust SMOTE for imbalanced problems with label noise. *Inf Sci*. 2021;553:397-428.
37. Prati RC, Luengo J, Herrera F. Emerging topics and challenges of learning from noisy data in nonstandard classification: a survey beyond binary class noise. *Knowl Inf Syst*. 2019;60(1):63-97.
38. Sáez JA, Galar M, Luengo J, Herrera F. Analyzing the presence of noise in multi-class problems: alleviating its influence with the one-vs-one decomposition. *Knowl Inf Syst*. 2014;38(1):179-206.
39. Sáez JA, Galar M, Luengo J, Herrera F. Tackling the problem of classification with noisy data using multiple classifier systems: analysis of the performance and robustness. *Inf Sci*. 2013;247:1-20.
40. Sáez JA, Luengo J, Herrera F. Predicting noise filtering efficacy with data complexity measures for nearest neighbor classification. *Pattern Recognit*. 2013;46(1):355-364.
41. Wei Y, Gong C, Chen S, Liu T, Yang J, Tao D. Harnessing side information for classification under label noise. *IEEE Trans Neural Netw Learn Syst*. 2020;31(9):3178-3192.
42. Pu X, Li C. Probabilistic information-theoretic discriminant analysis for industrial label-noise fault diagnosis. *IEEE Trans Ind Inform*. 2021;17(4):2664-2674.
43. Ren M, Zeng W, Yang B, Urtasun R. Learning to reweight examples for robust deep learning. In: Proceedings of the 35th International Conference on Machine Learning, Vol 80. PMLR; 2018:4331-4340.
44. Kaneko T, Ushiku Y, Harada T. Label-noise robust generative adversarial networks. In: Proceedings of the 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition. IEEE; 2019:2462-2471.
45. Mannino MV, Yang Y, Ryu Y. Classification algorithm sensitivity to training data with non representative attribute noise. *Decis Support Syst*. 2009;46(3):743-751.
46. Petety A, Tripathi S, Hemachandra N. Attribute noise robust binary classification. In: Proceedings of the 34th AAAI Conference on Artificial Intelligence. Association for the Advancement of Artificial Intelligence; 2020:13897-13898.
47. Bi J, Zhang T. Support vector classification with input data uncertainty. *Advances in Neural Information Processing Systems*, Vol 17. MIT Press;2004:161-168.
48. Wickham H, Danenberg P, Csárdi G, Eugster M. roxygen2: in-line documentation for r. https://CRAN.R-project.org/package=roxygen2, R package version 7.2.1; 2022.

49. Liu D, Yang G, Wu J, Zhao J, Lv F. Robust binary loss for multi-category classification with label noise. In: Proceedings of the 2021 IEEE International Conference on Acoustics, Speech and Signal Processing. IEEE; 2021:1700-1704.
50. Zhang Y, Zheng S, Wu P, Goswami M, Chen C. Learning with feature-dependent label noise: a progressive approach. In: Proceedings of the 9th International Conference on Learning Representations; 2021:1-13. OpenReview.net
51. Wei J, Liu Y. When optimizing f-divergence is robust with label noise. In: Proceedings of the 9th International Conference on Learning Representations; 2021:1-11. OpenReview.net
52. Ghosh A, Manwani N, Sastry PS. Making risk minimization tolerant to label noise. *Neurocomputing*. 2015;160:93-107.
53. Zhao Z, Chu L, Tao D, Pei J. Classification with label noise: a Markov chain sampling framework. *Data Min Knowl Disc*. 2019;33(5):1468-1504.
54. Chen P, Ye J, Chen G, Zhao J, Heng P-A. Beyond class-conditional assumption: a primary attempt to combat instance-dependent label noise. In: Proceedings of the 35th AAAI Conference on Artificial Intelligence. Association for the Advancement of Artificial Intelligence; 2021:11442-11450.
55. Nicholson B, Sheng VS, Zhang J. Label noise correction and application in crowdsourcing. *Expert Syst Appl*. 2016;66:149-162.
56. Cheng J, Liu T, Ramamohanarao K, Tao D. Learning with bounded instance and label-dependent label noise. In: Proceedings of the 37th International Conference on Machine Learning, Vol 119. PMLR; 2020:1789-1799.
57. Wang Q, Han B, Liu T, Niu G, Yang J, Gong C. Tackling instance-dependent label noise via a universal probabilistic model. In: Proceedings of the 35th AAAI Conference on Artificial Intelligence. Association for the Advancement of Artificial Intelligence; 2021:10183-10191.
58. Amid E, Warmuth MK, Srinivasan S. Two-temperature logistic regression based on the Tsallis divergence. In: Proceedings of 22nd International Conference on Artificial Intelligence and Statistics. Vol 89. PMLR; 2019:2388-2396.
59. Thulasidasan S, Bhattacharya T, Bilmes JA, Chennupati G, Mohd-Yusof J. Combating label noise in deep learning using abstention. In: Proceedings of the 36th International Conference on Machine Learning.Vol 97. PMLR; 2019:6234-6243.
60. Denham B, Pears R, Naeem MA. Null-labelling: a generic approach for learning in the presence of class noise. In: Proceedings of the 20th IEEE International Conference on Data Mining. IEEE; 2020:990-995.
61. Cano JR, Luengo J, García S. Label noise filtering techniques to improve monotonic classification. *Neurocomputing*. 2019;353:83-95.
62. Salekshahrezaee Z, Leevy JL, Khoshgoftaar TM. A reconstruction error-based framework for label noise detection. *J Big Data*. 2021;8(1):1-16.
63. Bootkrajang J. A generalised label noise model for classification. In: Proceedings of the 23rd European Symposium on Artificial Neural Networks; 2015:349-354. www.i6doc.com
64. Ortego D, Arazo E, Albert P, O'Connor NE, McGuinness K. Towards robust learning with different label noise distributions. In: Proceedings of the 25th International Conference on Pattern Recognition. IEEE;2020:7020-7027.
65. Bootkrajang J, Chaijaruwanich J. Towards instance-dependent label noise-tolerant classification: a probabilistic approach. *Pattern Anal Applic*. 2020;23(1):95-111.
66. Ghosh A, Lan AS. Contrastive learning improves model robustness under label noise. In: Proceedings of the 2021 IEEE Conference on Computer Vision and Pattern Recognition Workshops. IEEE; 2021:2703-2708.
67. Han B, Yao J, Niu G, et al. Masking: a new perspective of noisy supervision. *Advances in Neural Information Processing Systems*, Vol 31. Curran Associates; 2018:5841-5851.
68. Ghosh A, Lan AS. Do we really need gold samples for sample weighting under label noise? In: *Proceedings of the 2021 IEEE Winter Conference on Applications of Computer Vision*. IEEE; 2021:3921-3930.
69. Du J, Cai Z. Modelling class noise with symmetric and asymmetric distributions. In: Proceedings of the 29th AAAI Conference on Artificial Intelligence. Association for the Advancement of Artificial Intelligence; 2015:2589-2595.
70. Schneider J, Handali JP, vom Brocke J. Increasing trust in (big) data analytics. In: Proceedings of the 2018 Advanced Information Systems Engineering Workshops, LNBIP, Vol 316. Springer; 2018:70-84.
71. Hendrycks D, Mazeika M, Wilson D, Gimpel K. Using trusted data to train deep networks on labels corrupted by severe noise. *Advances in Neural Information Processing Systems*, Vol 31. Curran Associates; 2018:10477-10486.
72. Li J, Zhu Q, Wu Q, Zhang Z, Gong Y, He Z, Zhu F. SMOTE-NaN-DE: addressing the noisy and borderline examples problem in imbalanced classification by natural neighbors and differential evolution. *Knowl-Based Syst*. 2021;223:107056.
73. Folleco A, Khoshgoftaar TM, Hulse JV, Bullard LA. Software quality modeling: the impact of class noise on the random forest classifier. In: Proceedings of the 2008 IEEE Congress on Evolutionary Computation. IEEE; 2008:3853-3859.
74. Zhu X, Wu X. Cost-guided class noise handling for effective cost-sensitive learning. In: Proceedings of the 4th IEEE International Conference on Data Mining. IEEE; 2004:297-304.
75. Seo PH, Kim G, Han B. Combinatorial inference against label noise. *Advances in Neural Information Processing Systems*. Vol. 32. Curran Associates; 2019:1171-1181.
76. Gehlot S, Gupta A, Gupta R. A CNN-based unified framework utilizing projection loss in unison with label noise handling for multiple myeloma cancer diagnosis. *Med Image Anal*. 2021;72:102099.
77. Kang J, Fernández-Beltran R, Duan P, Kang X, Plaza AJ. Robust normalized softmax loss for deep metric learning-based characterization of remote sensing images with label noise. *IEEE Trans Geosci Remote Sens*. 2021;59(10):8798-8811.
78. Görnitz N, Porbadnigk A, Binder A, Sannelli C, Braun ML, Müller K-R, Kloft M. Learning and evaluation in presence of non-i.i.d. label noise. In: Proceedings of the 17th International Conference on Artificial Intelligence and Statistics, Vol 33. PMLR; 2014:293-302.

79. Jindal I, Pressel D, Lester B, Nokleby MS. An effective label noise model for DNN text classification. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics; 2019:3246-3256.
80. Fefilatyev S, Shreve M, Kramer K, et al. Label-noise reduction with support vector machines. In: Proceedings of the 21st International Conference on Pattern Recognition. IEEE; 2012:3504-3508.
81. Huang X, Shi L, Suykens JAK. Support vector machine classifier with pinball loss. *IEEE Trans Pattern Anal Mach Intell*. 2014;36(5):984-997.
82. Huang L, Zhang C, Zhang H. Self-adaptive training: beyond empirical risk minimization. *Advances in Neural Information Processing Systems*. Vol 33. Curran Associates; 2020:19365-19376.
83. Teng C-M. Polishing blemishes: issues in data correction. *IEEE Intell Syst*. 2004;19(2):34-39.
84. Ramdas A, Póczos B, Singh A, Wasserman LA. An analysis of active learning with uniform feature noise. In: Proceedings of the 17th International Conference on Artificial Intelligence and Statistics, Vol 33. PMLR; 2014:805-813.
85. Kazmierczak S, Mandziuk J. A committee of convolutional neural networks for image classification in the concurrent presence of feature and label noise. In: Proceedings of the 16th International Conference on Parallel Problem Solving from Nature, LNCS. Vol 12269. Springer; 2020:498-511.
86. Raymer M, Doom T, Kuhn L, Punch W. Knowledge discovery in medical and biological datasets using a hybrid Bayes classifier/evolutionary algorithm. *IEEE Trans Syst Man Cybern - Part B: Cybern*. 2003;33:802-813.
87. Sáez JA, Luengo J, Herrera F. Evaluating the classifier behavior with noisy data considering performance and robustness: the equalized loss of accuracy measure. *Neurocomputing*. 2016;176:26-35.
88. He Z, Zhang M, Zhang H. Data-driven research on chemical features of Jingdezhen and Longquan celadon by energy dispersive X-ray fluorescence. *Ceram Int*. 2016;42(4):5123-5129.
89. Denoeux T. A neural network classifier based on Dempster-Shafer theory. *IEEE Trans Syst Man Cybern - Part A: Syst Humans*. 2000;30(2):131-150.

How to cite this article: Sáez JA. Noise simulation in classification with the noisemodel R package: Applications analyzing the impact of errors with chemical data. *Journal of Chemometrics*. 2023;37(5):e3472. doi:10.1002/cem.3472