# The **NoiseFiltersR** Package: Label Noise Preprocessing in R

Pablo Morales[1], Julián Luengo[1], Luís P.F. Garcia[2], Ana C. Lorena[3], André C.P.L.F. de Carvalho[2], and Francisco Herrera[1]

[1]Department of Computer Science and Artificial Intelligence, University of Granada, Granada, 18071, Spain
[2]Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, Trabalhador São-carlense Av. 400, São Carlos, São Paulo 13560-970, Brazil
[3]Instituto de Ciência e Tecnologia, Universidade Federal de São Paulo, Talim St. 330, São José dos Campos, São Paulo 12231-280, Brazil

**Abstract**

In Data Mining, the value of extracted knowledge is directly related to the quality of used data, which turns data preprocessing into one of the most important steps of the whole learning process. In classification problems, label noise refers to the incorrect labelling of training instances, and is known to be a very disruptive feature of data. In this work we present the **NoiseFiltersR** package. It contains the first extensive R implementation of classical and state-of-the-art label noise filters, which are the most common technique for preprocessing label noise. All these algorithms are appropriately documented and referenced, they can be called in a R-user-friendly manner, and their results are unified by means of the `"filter"` class, which also benefits from adapted `print` and `summary` methods.

## 1 Introduction

In last years, Data Mining has been faced with increasingly challenging problems in terms of the nature of available data. Not only its size, but also its imperfections and varied shapes, are providing the researchers with plenty of different scenarios to be addressed. Consequently, Data Preprocessing [4] has become an important part of the *KDD (Knowledge Discovery from Databases)* process, and related-software development is also essential to provide practitioners with the adequate tools.

Data Preprocessing intends to appropriately process the collected data so that subsequent learning algorithms can extract maximum knowledge out of it. It is known to be one of the most time-consuming steps in the whole KDD process. There exist several aspects involved in data preprocessing, like *feature selection* or dealing with *missing values* and *noisy data*. Feature selection aims at extracting the most relevant attributes for the learning step, thus reducing the complexity of models and the computing time. The treatment of missing values is also essential to keep as much information as possible in data Finally, noisy data refers to values that are either incorrect or clearly far from the general underlying distribution.

All these tasks have associated software available. For instance, the KEEL tool [1] contains a broad collection of data preprocessing algorithms, which covers all the aforementioned topics . There are other popular options for these tasks, like SPSS or SAS for missing values, and WEKA [18] or RapidMiner for feature selection. Apart from these, there exist many other general-purpose Data Mining software suites like R, KNIME or Python.

Regarding the R statistical software, there are plenty of packages available in the *Comprehensive R Archive Network (CRAN)* repository to address preprocessing tasks. For example, **MICE** [16] and **Amelia** [6] are very popular packages for handling missing values, whereas **caret** [9] or **FSelector** [13] provide a wide range of techniques for feature selection. There are also general-purpose packages for decting outliers and anomalies, like **mvoutlier** [2].

However, to the best of our knowledge, CRAN lacks an extensive collection of classification-oriented label noise preprocessing algorithms, some of which are among the most influential preprocessing techniques [5]. This is the gap we intend to fill with the release of the **NoiseFiltersR** package, whose taxonomy is inspired on the recent survey on label noise by B. Frénay and M. Verleysen [3].

Yet, it should be noted that there are other packages that include some isolated implementations of label noise filters, since they are sometimes needed as auxiliary functions. It is the case of the **unbalanced** [11] package, which deals with imbalanced classification. It contains basic versions of classical filters such as *Tomek-Links* [15] or *ENN* [17], which are tipically applied after oversampling an imbalanced dataset (which is the main purpose of the **unbalanced** package).

In the following Section 2 we briefly introduce the problem of classification with label noise, as well as the most popular techniques to overcome it. In Section 3 we show how to use the **NoiseFiltersR** package to apply these techniques in a unified and R-user-friendly manner. Finally, Section 4 presents a general overview of this work. Once the package is loaded, the source and R code for this vignette is directly available from R with the command `browseVignettes`.

# 2 Label noise preprocessing

Data collection and preparation processes are usually subject to errors in Data Mining applications [19]. Consequently, real-world datasets are commonly affected by imperfections or *noise*. In classification problems, this noise negatively affects the learning process of classifiers, leading to less accurate predictions, excessively complex models, and longer computation time.

Two different types of noise are usually distinguished in the specialized literature for classification: *attribute* noise and *label* noise (which is also called *class* noise) [20]. The former refers to imperfections in the attributes of the training dataset, whereas the latter relates to errors in the labels used for classification. The **NoiseFiltersR** package (and the rest of this work) focuses on label noise, which is known to be the most disruptive one, since label quality is essential for the classifier training [20].

In order to address the problem of label noise, there exist two main approaches in the literature, and both are surveyed in the recent work [3]. On the one hand, *algorithm level* approaches [12] attempt to create robust classification algorithms that are little influenced by the presence of noise. On the other hand, *data level* approaches [8] (also called *filters*) try to develop strategies to cleanse the dataset as a previous step to the fit of the classifier. The **NoiseFiltersR** package follows the second approach, since this allows to carry out the data preprocessing just once and apply any classifier thereafter, whereas the first option is specific for each classification algorithm[1].

Regarding data-level handling of label noise, we take the aforementioned survey by Frénay et al. [3] as the basis for our **NoiseFiltersR** package. That work provides an overview and references for the most popular classical and state-of-the-art filters, which are organized and classified taking into account several aspects:

- Considering how to identify noisy instances, *ensemble* based, *similarity* based and *data complexity* based algorithms are distinguished. The first type makes use of predictions from classifiers ensembles built over different partitions or resamples of training data, the second is based on label distribution in the nearest neighbors of each instance, and the third attempts to reduce complexity metrics which are related to the presence of noise. As we will explain in Section 3 (see Figure 1), the **NoiseFiltersR** package contains implementations of all these types of algorithms, and the explicit distinction is indicated in the documentation page of each function.
- Regarding how to deal with the identified noise, *noise removal* and *noise reparation* strategies are considered. The first option removes the noisy instances, whereas the second one relabels them with the most likely label on the basis of the information available. There also exist *hybrid* approaches, which only carry out relabelling when they have enough confidence on the new label, and otherwise remove. The discussion between noise removal, noise reparation and their possible sinergies is an active and open field of research [3, Section VI.H]: most works agree on the potential damages of incorrect relabelling [10], although other studies also point out the dangers of removing too many instances and advocate hybrid approaches [14]. As we will see in Section 3, the **NoiseFiltersR** package includes filters which implement all these possibilities, and the specific behaviour is explicitly indicated in the documentation page of the corresponding function.

# 3 The NoiseFiltersR package

The released package implements, documents, explains and provides references for a broad collection of label noise filters surveyed in [3]. To the best of our knowledge, it is the first comprehensive review and implementation of this topic for R, which has become an essential tool in Data Mining in the last years.

[1] Of course, in R there exist implementations of very popular label noise robust classifiers (the aforementioned algorithm-level approach), such as *C4.5* and *RIPPER*, which are called *J48* and *JRip* respectively in **RWeka** package [7] (which is a R interface to WEKA software [18]).

Namely, the **NoiseFiltersR** package includes a total of 30 filters which were published along 24 research papers (each one of these papers is referenced in the corresponding filter documentation page, see Section 3.2). Regarding the noise detection strategy, 13 of them are ensemble based filters, 14 can be cataloged as similarity based, and the other 3 are based on data complexity measures. Taking into account the noise handling approach, 4 of them integrate the possibility of relabelling, whereas the other 26 only allow for removing (which clearly evidences a general preference for data removal in the literature). The full list of implemented filters and its distribution according to the two aforementioned criterions is displayed in Figure 1, which provides a general overview of the package.

| | | **Noise identification** | | |
|---|---|---|---|---|
| | | *Ensemble* | *Similarity* | *Data Complexity* |
| **Noise handling** | *Remove* | C45robustFilter<br>C45votingFilter<br>C45iteratedVotingFilter<br>CVCF<br>dynamicCF<br>edgeBoostFilter<br>EF<br>HARF<br>INFFC<br>IPF<br>ORBoostFilter<br>PF | AENN<br>BBNR<br>CNN<br>DROP1<br>DROP2<br>DROP3<br>ENG<br>ENN<br>PRISM<br>RNN<br>TomekLinks | saturationFilter<br>consensusSF<br>classifSF |
| | *Repair/ Hybrid* | hybridRepairFilter | EWF<br>GE<br>ModeFilter | |

Figure 1: Names and taxonomy of available filters in the **NoiseFiltersR** package.

The rest of section is organized as follows. Section 3.1 is devoted to the installation process. In Section 3.2 we present the documentation, where further details of each filter can be looked up. Section 3.3 focuses on the two implemented methods to call the filters. Finally, Section 3.4 presents the `filter` class, which unifies the return value of the filters in **NoiseFiltersR** package.

## 3.1 Installation

The **NoiseFiltersR** package is available at CRAN servers, so it can be downloaded and installed directly from the R command line by typing:

```
install.packages("NoiseFiltersR")
```

This command will also install the eleven dependencies of the package, which mainly provide the classification algorithms needed for the implemented filters, and which can be looked up in the "Imports" section of the CRAN website for the package `https://cran.r-project.org/web/packages/NoiseFiltersR/index.html`.

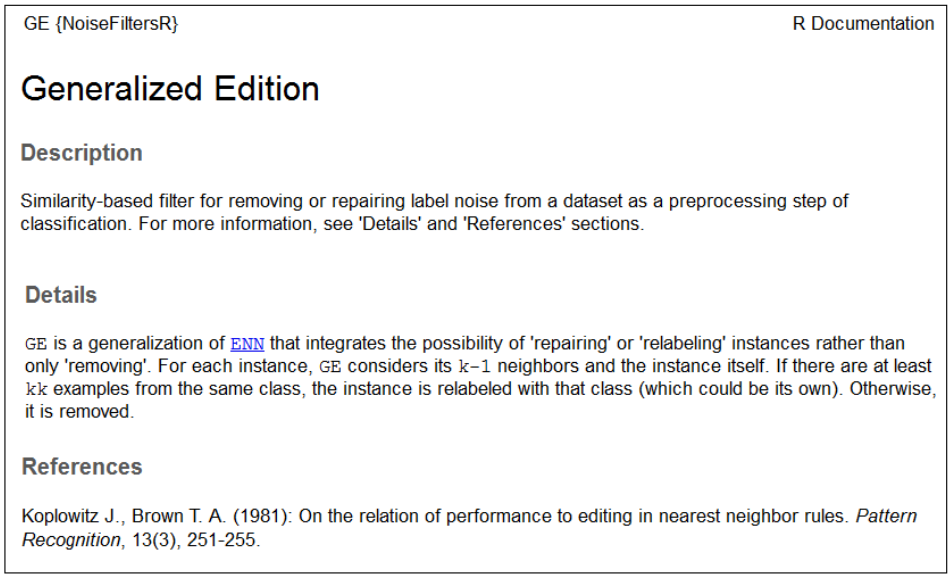
In order to easily access all the package's functions, it must be attached in the usual way:

```
library(NoiseFiltersR)
```

## 3.2 Documentation

Whereas this vignette provides the user with an overview of the **NoiseFiltersR** package, it is also important to have access to specific information for each available filter. This information can be looked up in the corresponding documentation page, that in all cases includes the following essential items (see Figure 2 for an example):

- A *description* section, which indicates the type of filter according to the taxonomy explained at the end of Section 2 and summarized in Figure 1.
- A *details* section, which provides the user with a general explanation of the filter's behaviour and any other usage particularity or warning.
- A *references* section that points to the original contribution where the filter was proposed, where further details, motivations or contextualization can be found.


GE {NoiseFiltersR} R Documentation

Generalized Edition

Description

Similarity-based filter for removing or repairing label noise from a dataset as a preprocessing step of classification. For more information, see 'Details' and 'References' sections.

Details

GE is a generalization of ENN that integrates the possibility of 'repairing' or 'relabeling' instances rather than only 'removing'. For each instance, GE considers its k−1 neighbors and the instance itself. If there are at least kk examples from the same class, the instance is relabeled with that class (which could be its own). Otherwise, it is removed.

References

Koplowitz J., Brown T. A. (1981): On the relation of performance to editing in nearest neighbor rules. *Pattern Recognition*, 13(3), 251-255.

Figure 2: Extract from GE filter's documentation page, showing the highlighted above aspects.

As usually in R, the function documentation pages can be either checked in the CRAN website for the package or loaded from the command line with the orders `?` or `help`:

```
?GE
help(GE)
```

## 3.3 Calling the filters

When it comes to apply a label-noise filter in Data Mining applications, all we need to know is the dataset to be filtered and its *class* variable (i.e. the one that contains the label for each available instance). The **NoiseFiltersR** package provides two standard ways for tagging the class variable when calling the implemented filters (see also Figure 3 and the example below):

- The *default* method receives the dataset to be filtered in the `x` argument, and the number for the class column through the `classColumn` argument. If the latter is not provided, the last column of the dataset is assumed to contain the labels.
- The *formula* method is intended for regular R users, who are used to this approach when fitting regression or classification models. It allows for indicating the class variable (along with the attributes to be used) by means of an expression like `Class~Attr1+...+AttrN` (recall that `Class~.` makes use of all attributes).

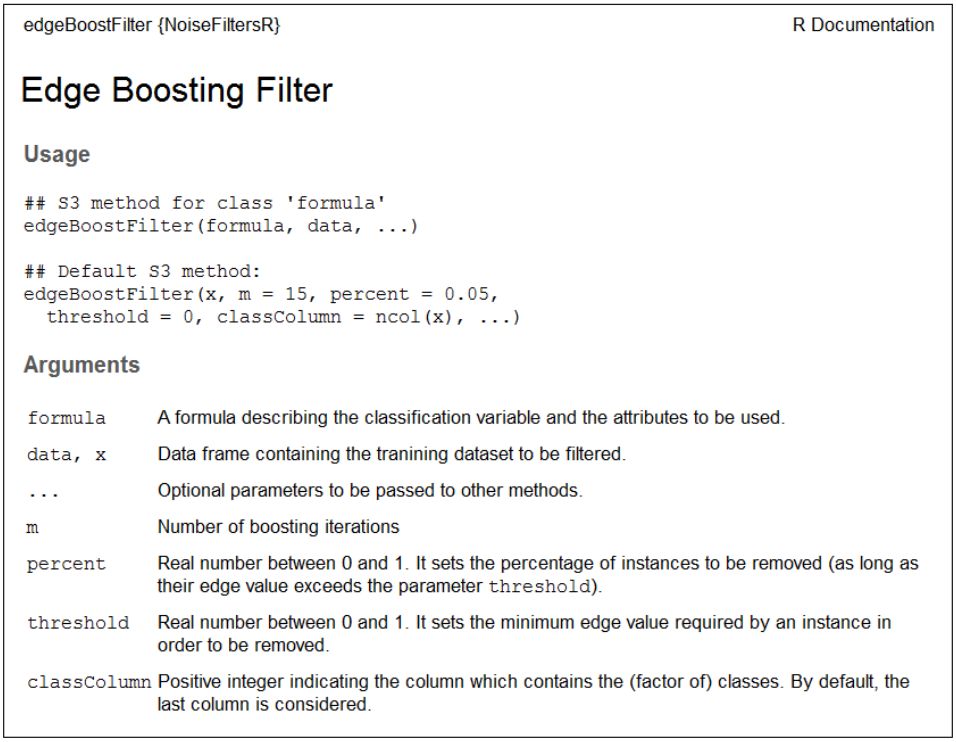
Next, we provide an example on how to use these two methods for filtering out the `iris` dataset with `edgeBoostFilter` (we do not change the default parameters of the filter):

```
data(iris)
str(iris)
```

```
## 'data.frame': 150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```r
# Using the default method:
out_Def <- edgeBoostFilter(iris, classColumn = 5)
# Using the formula method:
out_For <- edgeBoostFilter(Species~., iris)
# Checking that the filtered datasets are identical:
identical(out_Def$cleanData, out_For$cleanData)
```

```
## [1] TRUE
```



edgeBoostFilter {NoiseFiltersR} R Documentation

## Edge Boosting Filter

### Usage

```
## S3 method for class 'formula'
edgeBoostFilter(formula, data, ...)

## Default S3 method:
edgeBoostFilter(x, m = 15, percent = 0.05,
  threshold = 0, classColumn = ncol(x), ...)
```

### Arguments

| | |
|---|---|
| formula | A formula describing the classification variable and the attributes to be used. |
| data, x | Data frame containing the tranining dataset to be filtered. |
| ... | Optional parameters to be passed to other methods. |
| m | Number of boosting iterations |
| percent | Real number between 0 and 1. It sets the percentage of instances to be removed (as long as their edge value exceeds the parameter threshold). |
| threshold | Real number between 0 and 1. It sets the minimum edge value required by an instance in order to be removed. |
| classColumn | Positive integer indicating the column which contains the (factor of) classes. By default, the last column is considered. |

Figure 3: Extract from `edgeBoostFilter`'s documentation page, which shows the two methods for calling filters in **NoiseFiltersR** package. In both cases, tuning parameters of the filter are provided through additional arguments.

Notice that, in the last command of the example, we used the `$` operator to access the objects returned from the filter. In next section we explore the structure and contents of these objects.

### 3.4 The `filter` class

The S3 class `filter` is designed to unify the return value of the filters inside the **NoiseFiltersR** package. It is a list that encapsulates seven elements with the most relevant information of the process:

- `cleanData` is a data.frame containing the filtered dataset.
- `remIdx` is a vector of integers indicating the indexes of removed instances (i.e. their row number with respect to the original data.frame).
- `repIdx` is a vector of integers indicating the indexes of repaired/relabelled instances (i.e. their row number with respect to the original data.frame).
- `repLab` is a factor containing the new labels for repaired instances.
- `parameters` is a list that includes the tuning parameters used for the filter.
- `call` is an expression that contains the original call to the filter.
- `extraInf` is a character vector including additional information not covered by previous items.

As an example, we can check the structure of the above `out_For` object, which was the return value of `egdeBoostFilter` function:

```
str(out_For)
```

```
## List of 7
##  $ cleanData :'data.frame': 142 obs. of  5 variables:
##   ..$ Sepal.Length: num [1:142] 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##   ..$ Sepal.Width : num [1:142] 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##   ..$ Petal.Length: num [1:142] 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##   ..$ Petal.Width : num [1:142] 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##   ..$ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ remIdx    : int [1:8] 58 78 84 107 120 130 134 139
##  $ repIdx    : NULL
##  $ repLab    : NULL
##  $ parameters:List of 3
##   ..$ m        : num 15
##   ..$ percent  : num 0.05
##   ..$ threshold: num 0
##  $ call      : language edgeBoostFilter(formula = Species ~ ., data = iris)
##  $ extraInf  : chr "Highest edge value kept: 0.0669358381115568"
##  - attr(*, "class")= chr "filter"
```

In order to cleanly display this `filter` class in the R console, two specific `print` and `summary` methods were implemented. The appearance of the first one is as follows

```
print(out_For)
```

```
## Call:
## edgeBoostFilter(formula = Species ~ ., data = iris)
##
## Parameters:
## m: 15
## percent: 0.05
## threshold: 0
##
## Results:
## Number of removed instances: 8 (5.333333 %)
## Number of repaired instances: 0 (0 %)
```

and contains three main blocks:

- The original *call* to the filter.
- The tuning *parameters* used for the filter.
- An overview of the *results*, with the number (and percentage of the total) of removed and repaired instances.

The `summary` method displays some extra blocks:

- It always adds a title that summarizes the filter and dataset used.
- If there exists additional information in the `extraInf` component of the object, it is displayed under a homonymous block.
- If the argument `explicit` is set to `TRUE` (it defaults to `FALSE`), the explicit results (i.e. the indexes for removed and repaired instances and the new labels for the latters) are displayed.

In the case of the `out_For` object, the `summary` gets the following:

```
summary(out_For, explicit = TRUE)
```

```
## Filter edgeBoostFilter applied to dataset iris
##
## Call:
## edgeBoostFilter(formula = Species ~ ., data = iris)
##
## Parameters:
## m: 15
## percent: 0.05
## threshold: 0
##
## Results:
```

```
## Number of removed instances: 8 (5.333333 %)
## Number of repaired instances: 0 (0 %)
##
## Additional information:
## Highest edge value kept: 0.0669358381115568
##
## Explicit indexes for removed instances:
## 58 78 84 107 120 130 134 139
```

# 4 Summary

In this vignette we introduced the **NoiseFiltersR** package, which is the first R extensive implementation of classification-oriented label-noise filters. To set a context and motivation for this work, we presented the problem of label noise inside data preprocessing and the related software. As we explained, the released package unifies the return value of the filters by means of a S3 class, which benefits from specific `print` and `summary` methods. Moreover, it provides a R-user-friendly way to call the implemented filters, whose documentation is worth reading and points to the original reference where they were first published.

Regarding the potential extensions of this package, there exist several aspects which can be adressed in future releases. For instance, there exist some other label-noise filters reviewed in the main reference [3] whose noise identification strategy does not belong to the ones covered here: ensemble based, similarity based and data complexity based (as shown in Figure 1). A relevant extension would be the inclusion of some datasets with different levels of artificially introduced label noise, in order to ease the experimentation workflow[2].

# References


[1] J Alcalá, A Fernández, J Luengo, J Derrac, S García, L Sánchez, and F Herrera. Keel data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework. *Journal of Multiple-Valued Logic and Soft Computing*, 17(2-3):255–287, 2010.

[2] Peter Filzmoser and Moritz Gschwandtner. *mvoutlier: Multivariate outlier detection based on robust methods*, 2015. R package version 2.0.6.

[3] Benoît Frénay and Michel Verleysen. Classification in the presence of label noise: a survey. *IEEE transactions on neural networks and learning systems*, 25(5):845–869, 2014.

[4] Salvador García, Julián Luengo, and Francisco Herrera. *Data preprocessing in data mining*. Springer, 2015.

[5] Salvador García, Julián Luengo, and Francisco Herrera. Tutorial on practical tips of the most influential data preprocessing algorithms in data mining. *Knowledge-Based Systems*, 98:1–29, 2016.

[6] James Honaker, Gary King, and Matthew Blackwell. Amelia II: A program for missing data. *Journal of Statistical Software*, 45(7):1–47, 2011.

[7] Kurt Hornik, Christian Buchta, and Achim Zeileis. Open-source machine learning: R meets weka. *Computational Statistics*, 24(2):225–232, 2009.

[8] Taghi M Khoshgoftaar and Pierre Rebours. Improving software quality prediction by noise filtering techniques. *Journal of Computer Science and Technology*, 22(3):387–396, 2007.

[9] Max Kuhn. Caret package. *Journal of Statistical Software*, 28(5), 2008.

[10] André LB Miranda, Luís Paulo F Garcia, André CPLF Carvalho, and Ana C Lorena. Use of classification algorithms in noise detection and elimination. In *International Conference on Hybrid Artificial Intelligence Systems*, pages 417–424. Springer, 2009.

[11] Andrea Dal Pozzolo, Olivier Caelen, and Gianluca Bontempi. *unbalanced: Racing for Unbalanced Methods Selection*, 2015. R package version 2.0.


[2] A wide variety of such datasets can be downloaded from the KEEL dataset repository in the website `http://www.keel.es/`, and then loaded from R.

[12] J Ross Quinlan. *C4.5: programs for machine learning*. Elsevier, 2014.

[13] Piotr Romanski and Lars Kotthoff. *FSelector: Selecting attributes*, 2014. R package version 0.20.

[14] Choh Man Teng. Dealing with data corruption in remote sensing. In *International Symposium on Intelligent Data Analysis*, pages 452–463. Springer, 2005.

[15] Ivan Tomek. Two modifications of cnn. *IEEE Trans. Systems, Man and Cybernetics*, 6:769–772, 1976.

[16] Stef van Buuren and Karin Groothuis-Oudshoorn. mice: Multivariate imputation by chained equations in r. *Journal of Statistical Software*, 45(3):1–67, 2011.

[17] Dennis L Wilson. Asymptotic properties of nearest neighbor rules using edited data. *IEEE Transactions on Systems, Man, and Cybernetics*, (3):408–421, 1972.

[18] Ian H Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.

[19] Xindong Wu and Xingquan Zhu. Mining with noise knowledge: error-aware data mining. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 38(4):917–932, 2008.

[20] Xingquan Zhu and Xindong Wu. Class noise vs. attribute noise: A quantitative study. *Artificial Intelligence Review*, 22(3):177–210, 2004.