Neurocomputing 360 (2019) 1–13

Contents lists available at ScienceDirect

Neurocomputing

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

Original software publication

Smartdata: Data preprocessing to achieve smart data in R

Ignacio Cordón[a], Julián Luengo[a,*], Salvador García[a], Francisco Herrera[a], Francisco Charte[b]

[a] DaSCI Andalusian Institute of Data Science and Computational Intelligence, University of Granada, Granada 18071, Spain
[b] DaSCI Andalusian Institute of Data Science and Computational Intelligence, University of Jaén, Jaén 23071, Spain

ARTICLE INFO

Article history:
Received 28 January 2019
Revised 9 April 2019
Accepted 2 June 2019
Available online 6 June 2019

Communicated by Prof. Zidong Wang

MSC:
68T05

Keywords:
Smart data
Data preprocessing
Machine learning
Preprocessing

ABSTRACT

As the amount of data available exponentially grows, data scientists are aware that finding the value in the data is key to a successful data exploiting. However, the data rarely presents itself in a ordered, clean way. In opposition to dealing with raw data, the term *smart data* is becoming more and more visible both in the specialized literature and companies. While software packages publicly exist to deal with raw data, there is no unified framework that encompasses all the required fields to transform such raw data to smart data. In this paper, the novel `smartdata` package is introduced. Written in R and available at *CRAN* repository, it includes the most recent and relevant algorithms to treat raw data from multiple perspectives, now unified under a simple yet powerful API, which enables the data scientist to easily pipeline their application. The main features of the package, as well as some illustrative examples of its use are detailed throughout this manuscript.

1. Introduction

The term Smart Data [1] refers to the challenge of transforming raw data into quality data that can be appropriately exploited to obtain valuable insights [2]. The origins of Smart Data are strongly related to conceiving Big Data opposed to relational databases (thought as structured), where the former are mostly unstructured [3]. Thus, Smart Data was born as the attempt of transforming Big Data to a new form of structured data where traditional business intelligence and data warehousing procedures [4] can be applied, usually by means of ontologies that would aid the integration process [5]. This semantic approach has been adopted by environments where unstructured data predominates, as in sensor nets [6] and, recently, Internet of Things applications [7]. However, Data Science comprises a wider range of procedures, where data mining outstands, and the requirements of actionable data are not only limited to structuring the data. Therefore, Smart Data discovery is tasked to extract useful information from data, in the form of a subset (big or not), which poses enough quality for a successful data mining process. The impact of Smart Data discovery in industry and academia is two-fold: higher quality data mining and reduction of data storage costs.

Data preprocessing [8] clearly resembles the concept of Smart Data as one of the most important stages of a data mining process. Its goal is to clean and correct input data, so that a machine learning process may be later applied faster and with a greater accuracy. With this definition, data preprocessing techniques should enable data mining algorithms to cope with data science problems more easily [9]. Recently, these methods are also heavily affected by the increase in size and complexity of datasets and they may be unable to provide a preprocessed/smart dataset in a timely manner, and therefore, are being redesigned with Big Data technologies. As such, data preprocessing is the collection of tools that enable the data scientist to achieve smart data from raw, unstructured data.

Nowadays, there exists a trend to release the software created by the authors or even to gather several methods from different authors performed by specialists in the topic with the focus on making the techniques accessible to the general public [10,11]. Plenty of excellent preprocessing algorithms arise every day in the scientific literature, but the software is rarely released. To the best of our knowledge, there are just few open source libraries and packages that constitute a whole approach of preprocessing methods for data science.

First, regarding Java, lots of modules for data preprocessing may be found in the KEEL software suite [12]. It comprises a very complete collection of external and internal approaches, as well as a large number of ensemble methods that work at both levels. For Python, simple preprocessing tasks can be carried out by using `scikit-learn` [13], as well as dimensionality reduction duties.

* Corresponding author.
E-mail addresses: nachocordon@correo.ugr.es (I. Cordón), julianlm@decsai.ugr.es (J. Luengo), salvagl@decsai.ugr.es (S. García), herrera@decsai.ugr.es (F. Herrera), fcharte@ujaen.es (F. Charte).

https://doi.org/10.1016/j.neucom.2019.06.006
0925-2312/

Discretization in Python can be done with `pandas` [14] and `orange` [15] packages.

Finally, several R packages can be found at *CRAN* (The Comprehensive R Archive Network) and *Bioconductor*, the R official repositories containing a wide variety of libraries that target machine learning topics. For instance, `caret` [16] for general preprocessing approaches, `NoiseFiltersR` [17] for noise cleaning, `imbalance` [18] for data resampling in general datasets, `COSNet` [19] for imbalance problem in biological contexts and the packages `MICE` [20] and `AMELIA` [21] for missing values treatment, could be cited, to name a few.

However, there are two main issues associated with the aforementioned software solutions. On the one hand, each package has its own Application Programming Interface (API), which makes the combined usage of several of them quite time-consuming. On the other hand, and possibly the most significant point, it has been observed that few of them contain the latest approaches proposed in the specialized literature.

In this paper, a novel, robust and up-to-date R package including preprocessing techniques for achieving smart data, is presented. Named as `smartdata`[1], it serves as a convenient wrapper for other machine-learning packages and it provides several advantages and commodities to data scientists: a common interface for those libraries, parameter naming unification and the possibility of pipelining the preprocessing in an single sentence, all of that designed with the capability of extensibility.

To present this novel package, the rest of the manuscript is organized as follows. First, Section 2 presents the software framework and enumerates the implemented algorithms. Then, Section 3 shows some illustrative examples. Finally, Section 4 formulates the conclusions.

2. Software

Together with Python, R is the most widely used language today for machine learning tasks. In contrast to tools based on a graphical user interface, such as KNIME or WEKA, R offers a working procedure that facilitates the reproducibility of experiments. Furthermore, it is a tool that does not demand great programming knowledge, as shown by its popularity among statisticians and similar user profiles. These are the main reasons why thousands of researchers develop their work in R.

R programming language has a great variety of packages that target machine learning topics. However, each package has its own API, which makes the task of using several of them quite time-consuming. The main purpose of `smartdata` is to provide a common interface for a collection of well-used machine learning packages, so that using methods from different libraries gets easier. Like most R packages, `smartdata` is designed to be used in an interactive way, from the R command line, or inside R scripts. It does not provide any user graphical interface. Note that for its installation one should do:

```
# This sets both CRAN and Bioconductor as repositories to resolve dependencies
setRepositories(ind = 1:2)
install.packages("smartdata")
```

Henceforth, some of its main strengths will be briefly described as follows. Section 2.1 is devoted to describe the whole package and compare it with other popular tools. Section 2.2 describes how the parameters for each included method can be consulted. Section 2.3 provides hints to easily utilize the proposed package and combine different preprocessing algorithms included. Section 2.4 describes how to contribute with new algorithms to the package. Finally, Section 2.5 introduces the most common problems that may appear during the usage of `smartdata` and hints on how to tackle them.

2.1. A complete data preprocessing suite

`Smartdata` includes preprocessing algorithms for missing values treatment, noise filtering, feature selection, outliers detection, space transformations, oversampling, discretization, normalization and instance selection. Aiming to be as exhaustive as possible, more than a hundred methods have been incorporated into the package including state-of-the art and classical ones.

Each of the aforementioned topics has its corresponding wrapper. Their name and a brief description will follow, given the next assumptions. Let $S = \{(x_i, y_i)\}_{i=1}^{m}$ be the data set. It contains $x_i \in \mathcal{X} \subset \mathbb{R}^n$ and $y_i \in \mathcal{Y}$ where $\mathcal{Y}$ is a finite set of labels. (x_i, y_i) is called an instance.

- `Impute_missing` is the wrapper devoted to treat missing values. Missing values are missing attribute values in a tuple, so a preliminary imputation could have a lot of benefits with respect to a posterior machine learning technique, such as classification.
- `Clean_noise` wrapper contains the techniques used to deal with noisy data. Noisy data are instances which include additional meaningless information apart from the true information they encode. Removing or repairing those instances prior to a classification can have a potential benefit on it.
- `Feature_selection` is dedicated to feature selection algorithms, which consists in given a subset of features $T \subseteq \{1, \ldots, n\}$, projecting the tuples of $S = \{(x_i, y_i)\}_{i=1}^{m}$ to the set of features given by T. Let

  $$p_T((x_{i1}, \ldots, x_{in})) = (x_{ij})_{j \in T}$$

  as the projection to the features in T, then doing a feature selection would result in a set $S' = \{(p_T(x_i), y_i)\}_{i=1}^{m}$, if the picked features were T.
- `Clean_outliers` contains the techniques used to deal with outliers. Outliers are instances that lie far from the others (that is, they seem quite distant from the others, using some kind of predefined distance or measure).
- `Space_transformation` is devoted to space transformation algorithms. Space transformation changes a training set $S = \{(x_i^{(1)}, \ldots, x_i^{(n)})\}_{i=1}^{m}$ where $x_i^{(j)} \in \mathbb{R}$ into another set $S' = \{(\tilde{x}_i^{(1)}, \ldots, \tilde{x}_i^{(k)})\}_{i=1}^{m}$ with better properties for machine learning methods.
- `Oversample` is the wrapper that contains oversampling algorithms, which given $\mathcal{Y}$ with $|\mathcal{Y}| = 2$, if there exist more instances labeled with one class than with the other, oversampling will generate synthetic instances labeled with the minority class, namely E, so that the resulting dataset $S \cup E$ has better characteristics for classification.
- `Discretize` is the wrapper for discretizing techniques. Discretizing a dataset consists in turning continuous variables into discrete attributes, that is, picking an attribute which can take real values and making a binning so that there exists a finite set of values for the variable. This procedure can have a great importance in tasks such as classification.

[1] https://cran.r-project.org/web/packages/smartdata.

Table 1
Algorithms covered by the `smartdata` package.

Group	Package	Methods
Discretization		
	discretization	chi2, chi_merge, extended_chi2, mod_chi2, CAIM, CACC, ameva, mdlp
	infotheo	equalfreq, equalwidth, globalequalwidth
Feature selection		
	Boruta	Boruta
	FSelector	chi_squared, information_gain, gain_ratio, sym_uncertainty, oneR, RF_importance, best_first_search, forward_search, backward_search, hill_climbing, cfs, consistency
Instance selection		
	unbalanced	CNN, ENN
	class	multiedit
	RoughSets	FRIS
Missing values		
	mice	gibbs_sampling
	Amelia	expect_maximization
	DMwR	central_imputation, knn_imputation
	missForest	rf_imputation
	missMDA	PCA_imputation, MCA_imputation, FAMD_imputation
	VIM	hotdeck, iterative_robust, regression_imputation
	denoiseR	ATN
Noise		
	NoiseFiltersR	AENN, ENN, BBNR, DROP1, DROP2, DROP3, EF, ENG, HARF, GE, INFFC, IPF, Mode, PF, PRISM, RNN, ORBoos, edgeBoost, edgeWeight, TomekLinks, dynamic, hybrid, saturation, consensusF, classificationSF, C45robust, C45voting, C45iteratedVoting, CVCF
Normalization		
	clusterSim	z_score, unitization, pos_unitization, min_max, rnorm, rpnorm, sd_quotient, mad_quotient, mad_quotient, range_quotient, max_quotient, mean_quotient, median_quotient, sum_quotient, ssq_quotient, norm, znorm
Outliers		
	MVN	multivariate
	outliers	univariate
Oversampling		
	imbalance	RACOG, wRACOG, PDFOS, RWO, ADASYN, ANSMOTE, SMOTE, MWMOTE, BLSMOTE, DBSMOTE, SLMOTE, RSLMOTE
Space transformation		
	lle	lle_knn, lle_epsilon
	adaptiveGPCA	adaptive_gpca

- `Normalize` implies converting the sample data S into another dataset S' where each tuple is treated so that standard deviation of the data ends up being zero (feature-wise or considered as elements of $\mathbb{R}^n$), or all the data is mapped to [0,1] interval (e.g. dividing by the maximum feature-wise), etc.
- `Instance_selection` consists in picking a subset $S' \subseteq S$, so that certain characteristics are preserved with respect to the original sample (such as distribution of classes) or at least most of representative instances are kept out.

To check the methods a wrapper can be called with, the help function included in `smartdata` could be invoked: `which_options` (e.g. `which_options (''instance_selection'')` would output `Possible methods are: 'CNN', 'ENN', 'multiedit', 'FRIS'`). Analogously, to check generic parameters for a wrapper (which should always be included in the call to the method), it suffices to do `?name of the wrapper`. For example: `?instance_selection` would output:

```
Usage

instance_selection(dataset, method, class_attr = "Class", ...)

Arguments

dataset     we want to perform an instance selection on
method      selected method of instance selection
class_attr  character. Indicates the class attribute from dataset. Must exist in it
...         Further arguments for method
```

Due to the large amount of methods covered by the package, more than a hundred, and that each method also can take a considerable amount of parameters, Table 1 summarizes only the methods included in each wrapper. The user can query the arguments for each one of these methods as shown in the example below.

To consult which further arguments are allowed by a certain method, one should execute `which_options(wrapper, method)`. e.g. `which_options(''instance_selection'', ''CNN'')` would output:

```
For more information do: ?unbalanced::ubCNN
Parameters for CNN are:
  * k: Number of nearest neighbors to perform CNN or ENN
       Default value: 1
```

When a specific parameter has a default value, it is not necessary to include it in the method call. Therefore, examples of valid calls to the `CNN` method would be:

```
instance_selection(dataset, "CNN", class_attr = "Class")
instance_selection(dataset, "CNN", class_attr = "Class", k = 2)
```

In Table 2, a comparison between the proposed `smartdata` package and other well-known data science software tools is presented. Such a comparison is divided by the typology of algorithms indicated in Table 1. It can be appreciated that older, well established tools as KEEL or WEKA contains a large number of preprocessing techniques, specially discretization and instance and feature selection methods. However, `smartdata` is comparable in numbers and even presents a large selection of noise treatment algorithms, outliers detection and normalization techniques. Other tools, as *scikit-learn*, which is the most known data science tool in python, show a lower number of techniques and even requires external packages to cope with some categories. Even WEKA relies on externally developed packages to provide an extension to the originally included preprocessing algorithms (which we do not count here for the sake of simplicity). Comparing with

Table 2
Number of preprocessing algorithms of `smartdata` and other well-known software packages. An '*' indicates that an external package must be used to provide such number of preprocessing techniques.

Software	Discretization	Feature selection	Instance selection	Missing values	Noise	Normalization	Outliers	Sampling	Transformation
smartdata	11	13	4	8	29	17	2	12	3
KEEL	30	25	16	15	7	4	0	20	4
WEKA	2	10	14	1	0	10	0	4	12
Scikit-learn	4	15	0	4	0	5	3	3*	6
caret (preProcess)	0	2	0	3	0	3	0	0	5

another R package, `caret`, it can be appreciated that the number of methods included is scarce when compared to `smartdata`. In summary, `smartdata` presents a large selection of methods, comparable in number with well established software tools in data science, with several advantages due to the unification it proposes.

2.2. Standardization of parameters

The proposed package standardizes names of methods and arguments, so that if a method had a parameter `num_iterations` and other had a parameter `iterations` with similar meaning, then both methods would take the same parameter name in `smartdata`. To illustrate this last assertion, let us have a `dataset` whose class attribute (namely `Class`) is the 5th one, where the noise filters `ModeFilter` and `INFFC`, from the `NoiseFiltersR` package, are to be applied. Below, the original way to call the methods is shown, in contrast with a more homogeneous one, achieved with `smartdata`:

```
NoiseFiltersR::ModeFilter(dataset, classColumn = 5, maxIter = 200)
NoiseFiltersR::INFFC(dataset, classColumn = 5, s = 3)

dataset %>% clean_noise("Mode", class_attr = "Class", num_iterations = 200)
dataset %>% clean_noise("INFFC", class_attr = "Class", num_iterations = 3)
```

2.3. Ease of use

`magrittr` has been used to provide a pipeline through its operator `%>%`, so that a typical preprocessing workflow can be expressed as:

```
result <- dataset %>%
          impute_missing(options) %>%
          clean_noise(options) %>%
          oversample(options) %>%
          feature_selection(options)
```

This pipeline allows the user to combine this workflow with the ones included in the packages from `tidyverse` [22], such as `dplyr`, intended to easily do carpentry work with the datasets (filtering, joining, aggregation,...) or `ggplot2`, which is an excellent library to generate appealing plots. On top of that, `smartdata` API tries to comply with the `tidyverse` style guide[2], ensuring good integration with those packages.

Another remarkable contribution of its API is the `exclude` generic parameter (it is not included in all the wrappers). With that parameter, a vector with the names of the columns to be ignored inside the algorithm, whenever possible, can be provided. Those columns will be appended at the end of the processing, preserving the order of the attributes in the original dataset, if possible. For example, running a min-max `normalization` on all the columns of a `dataset`, but ignoring its `Class` column (it does not make sense to normalize a categorical attribute) would be done in the following way:

```
normalize(dataset, method = "min_max", exclude = "Class", by = "column")
```

2.4. Extensibility

`smartdata` has been designed with extensibility in mind. The package's source code is structured into several files, one per wrapper. So, the wrapper for instance selection methods is in the `instance_selection.R` module, the one for oversampling algorithms is in `oversampling.R`, and so on. To extend the package, adding new methods to an existing wrapper, a set of steps must be fulfilled. Firstly, the new methods have to be mapped to their package. This is done in the header of the corresponding wrapper source code. As an example, here is the mapping for `instance_selection.R` methods (note that if the name of the method in the wrapper coincides with the name it has in the original package, then the `map` argument can be omitted):

```
instSelectionPackages <- list(
  "CNN" = list(
    pkg = "unbalanced",
    map = "ubCNN"
  ),
  "ENN" = list(
    pkg = "unbalanced",
    map = "ubENN"
  ),
  "multiedit" = list(
    pkg       = "class"
  ),
  "FRIS" = list(
    pkg  = "RoughSets",
    map  = "IS.FRIS.FRST"
  )
)
```

Secondly, the documentation for the added methods (e.g. `multiedit`) has to be provided, including a `check` method which tests some condition for the argument (e.g. that it is an integer belonging to the interval $[1, +\infty[$), outputting `TRUE` if the argument is correct and `FALSE` otherwise); an `info` string providing information about the argument; a `default` value for the argument (it will be used in case no value is passed for the argument calling the wrapper with this method); and a `map` field, which is the real name the argument has in the origin package (e.g. the argument `num_folds` is named `V` in the original `multiedit` method). A complete example can be found below:

[2] https://style.tidyverse.org.

```
args.multiedit <- list(
 k = list(
   check = function(arg) { qexpect (arg, rules = "X1[1,Inf)", label = "k") },
   info = "Number of neighbors used in KNN",
   default = 1
 ),
 num_folds = list(
   check = function(arg) { qexpect(arg, rules = "X1[1,Inf)", label = "num_folds") },
   info = "Number of partitions the train set is split in",
   default = 3,
   map = "V"
 ),
 null_passes = list(
   check = function(arg) { qexpect(arg, rules = "X1[1,Inf)", label = "null_passes") },
   info = "Number of null passes to use in the algorithm",
   default = 5,
   map = "I"
 )
)
```

Lastly, a resolver for the package has to be written (a method which decides how to address a machine learning task associated to a method coming from that origin library and outputs the modified dataset). This architecture was used because, oftentimes, methods coming from the same package need similar treatment (removing non-numerical attributes, for example) or have similar name arguments. For example, for `unbalanced` package:

```
doInstSelection.unbalanced <- function(task){
   ...
   # result is the transformation of dataset applying some method
   dataset <- task$dataset
   result
}
```

The `task` that it receives is a list with the following slots:

```
task$dataset: the dataset
task$classAttr: the class attribute passed to the wrapper, if it proceeds
task$classIndex: the index of the class attribute in the dataset, if it proceeds
task$method: an string indicating the method to apply
task$args: a list with further arguments passed to the wrapper
```

This information, along with a whole implementation of the `doInstSelection` method can be consulted in the *Contributing to smartdata* vignette[3].

2.5. Common problems and mistakes

In this section we want to provide the reader with hints on the most common problems that may arise during the package utilization. The most common mistake when using a method included in `smartdata` is to provide a wrong set of parameters for a given method. Any practitioner should consult the parameter list for the desired technique by using the `which_options` method, as have been indicated in Section 2.1. Such method will provide all the information needed to create a correct method call.

Even if the user is aware of the number of arguments needed to invoke any method included in `smartdata`, the correct values for the parameters should be consulted as well. Luckily, `which_options` will provide the user not only with the parameter purpose, but also with the default value as a reference and the restrictions in their values. Whether an attribute can be negative or not, the complete list of accepted values for the argument, etc. will be provided by `which_options` invocation. The reader should take into account that some parameters may accept all possible numeric values, but the correct choice will be dependent on the dataset. In these cases, expertise and trial and error will be needed to adjust the correct parameter value.

The user should also be aware that some preprocessing methods included in `smartdata` are supervised, that is, they need to know which attribute is the label. In some cases, the whole wrapper could be supervised, as in `instance_selection`. This can be checked by using the `?` operator as shown in Section 2.1. Otherwise, the particular method may require the `class_attr` argument, but such requirement can be checked by using `which_options` as indicated in the previous paragraph.

Sometimes the requirements are more subtle. For instance, many preprocessing methods assume that the dataset is complete, that is, no missing values are present. Other implicit requirements are that the attributes cannot be nominal or numeric, some data transformation is needed within the data, etc. These requirements are usually described in the original package as a vignette in CRAN associated to the actual package or in the scientific paper from which the implementation is based on. The user should previously check the data requirements of any method he or she wants to use before applying it to a preprocessing chain.

The requirements associated to the data as mentioned above should be taken into account when combining the preprocessing techniques into a preprocessing chain. There exists specialized literature that pays attention to the effects on how the combination of preprocessing methods can be done and its effects in the performance of learning techniques [23]. Therefore, the user should be aware not only of the requisite of the selected method, but also of the changes in data properties that applying such technique will impose. Otherwise, the combination of techniques will result in explicit errors indicated by the software or deteriorated datasets with poor learning abilities.

3. Examples of use

In this section, an insight of the intuitive usage of `smartdata` is provided. Section 3.1 demonstrates how the package can deliver data cleaning or reduction without altering its dimensions. Section 3.2 is devoted to show how easily the software can be used to apply dimensionality reductions to our data.

3.1. Data preparation

The `banana` dataset is a 2D dataset often used to depict the results of different preprocessing techniques [23]. The following code loads the `smartdata` package, the `banana` dataset (original version and imbalanced one) and it creates a synthetic `banana_mv` dataset (with missing values in its two columns) from the original one. It also shows how to display the methods available in the `instance_selection` wrapper and the documentation for arguments of `multiedit`, one of the instance selection methods:

```
# Load the previously installed smartdata package
library("smartdata")
# First let us load banana (both original and imbalanced)
data(list = c("banana_orig", "banana"),  package = "imbalance")

# Sets the random seed to get reproducible results
set.seed(123456)

# Generate a banana_mv dataset with at most 40%
# missing values in its first and second columns
banana_mv <- banana_orig
num_rows <- nrow(banana_orig)
nas_first_attr <- sample(num_rows, num_rows * 0.4)
nas_second_attr <- setdiff(sample(num_rows, num_rows * 0.4), nas_first_attr)
banana_mv[nas_first_attr, "At1"] <- NA
banana_mv[nas_second_attr, "At2"] <- NA

# To get the possible methods available for a certain wrapper, one can do:
which_options("instance_selection")

# To get information about the parameters available for a method:
which_options("instance_selection", "multiedit")
```

The original banana dataset is included for reference purposes in Fig. 1. The following subsections demonstrate how to apply several preprocessings to it.

[3] https://ncordon.github.io/smartdata/articles/smartdata_dev.html.

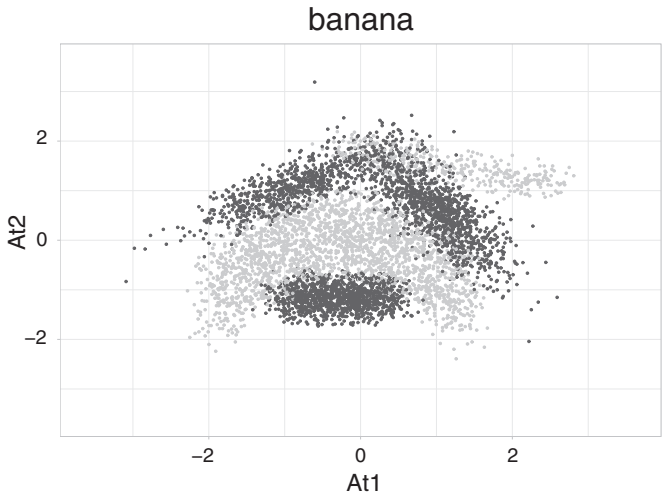


Fig. 1. Banana original dataset.

Fig. 2. Banana after applying `multiedit` instance reduction.

3.1.1. Instance selection

Let us start by performing instance selection using some of the available methods:

```
banana_orig %>% instance_selection("multiedit", class_attr = "Class")
banana_orig %>% instance_selection("ENN", class_attr = "Class", k = 3)
banana_orig %>% instance_selection("CNN", class_attr = "Class")
```

Fig. 2 shows how `multiedit` is able to select only the representative instances, removing redundant information from the dataset. Fig. 3 shows how CNN produces a visible smaller reduction than `multiedit` method, and Fig. 4 is the application of ENN method, which shows no appreciable difference with the original one. Those differences could be due to `multiedit` removing borderline instances, whereas CNN and ENN do not delete them.

3.1.2. Noise cleaning

The next example shows how to remove noisy examples from the data. Figs. 5 and 6 depict the application of EF and IPF noise filters, respectively, showing cleaner borders and less overlapping between the classes.

```
banana_orig %>% clean_noise("EF", class_attr = "Class", consensus = FALSE)
banana_orig %>% clean_noise("IPF", num_folds = 7, consensus = TRUE)
```

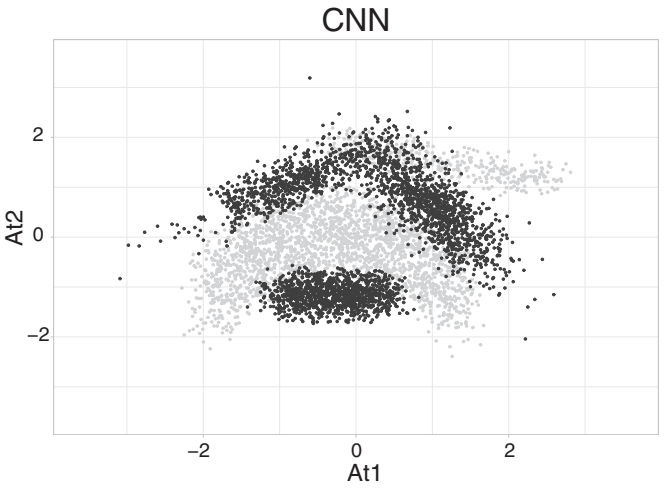


Fig. 3. Banana after applying CNN instance reduction.

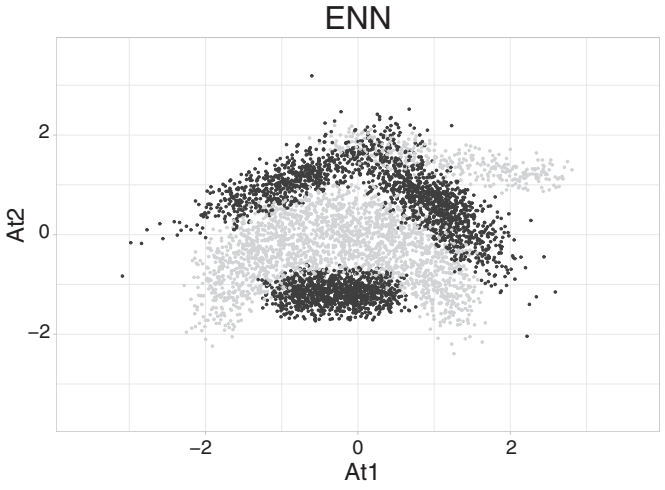


Fig. 4. Banana after applying ENN instance reduction.

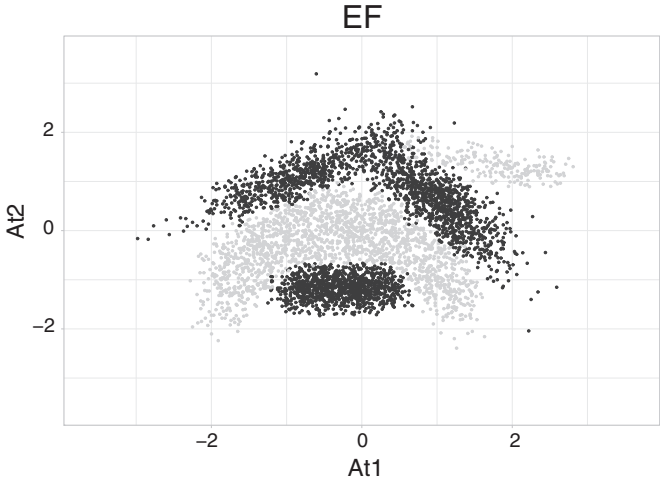


Fig. 5. Filtered banana dataset with EF.

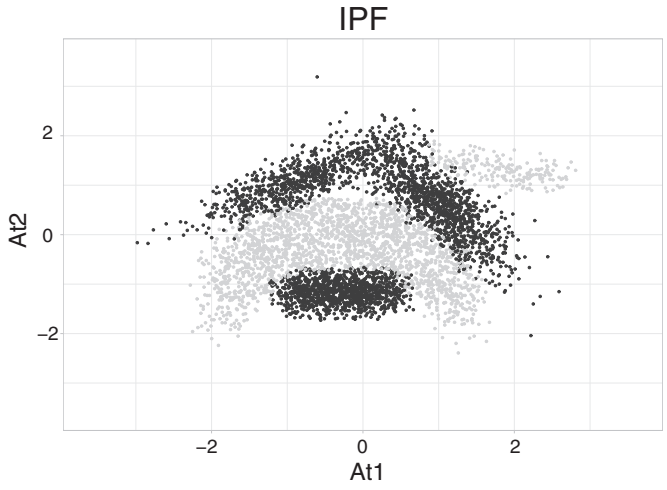


Fig. 6. Filtered banana dataset with `IPF`.

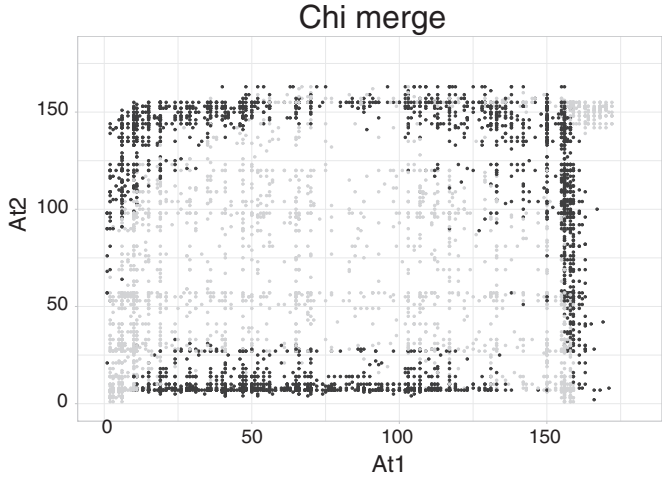


Fig. 7. Discretized banana dataset using `Chi Merge`.

3.1.3. Discretization

By discretizing the attributes their values are converted from real to categorical. The following snippet accomplish this task:

```
banana_orig %>% discretize("chi_merge", class_attr = "Class")
banana_orig %>% discretize("equalfreq", exclude = "Class", num_bins = 50)
```

Fig. 7 is the result of using a discretizer, which transforms real attributes into nominal ones: in this case, the data seems arranged into a grid (created by the discretized values). Fig. 8 shows a binning of the dataset into 50 categories per attribute.

3.1.4. Missing values imputation

Fig. 9 represents imputation for missing values where up to 40% of each attribute values were previously removed: the distribution of points belonging to each class tries to mimic the original form of banana dataset with difficulty as missing values constitute a great information loss. Fig. 10 shows the application of `hotdeck` algorithm, which also presents some difficulties reproducing the original distribution of the data.

```
banana_mv %>% impute_missing("knn_imputation")
banana_mv %>% impute_missing("hotdeck")
```

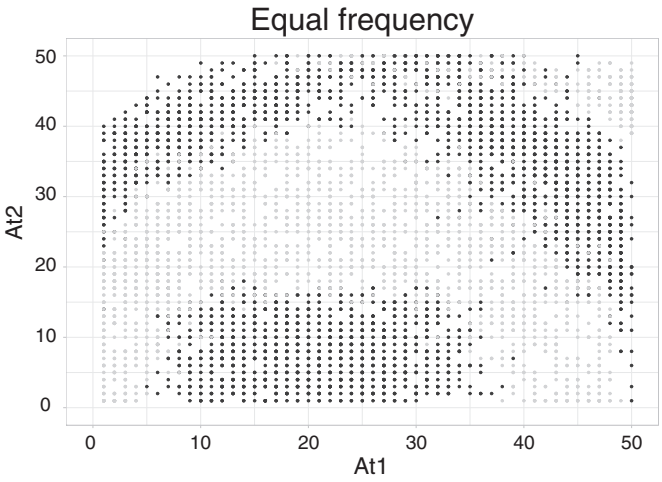


Fig. 8. Discretized banana dataset using `Equal frequency` discretization.

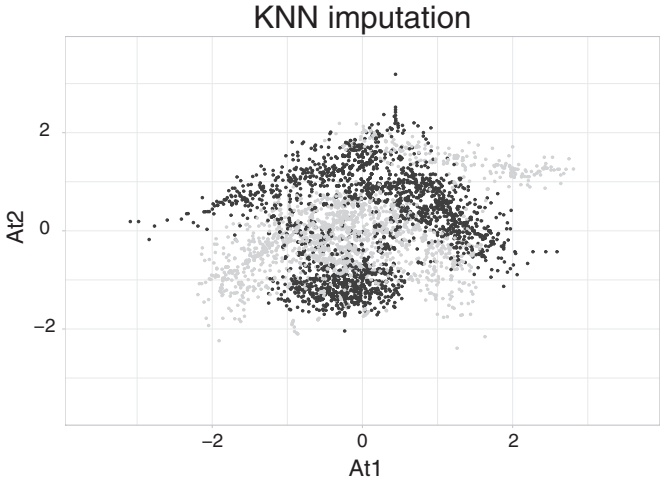


Fig. 9. Imputed values for the missing values created in banana using `KNN imputation`.

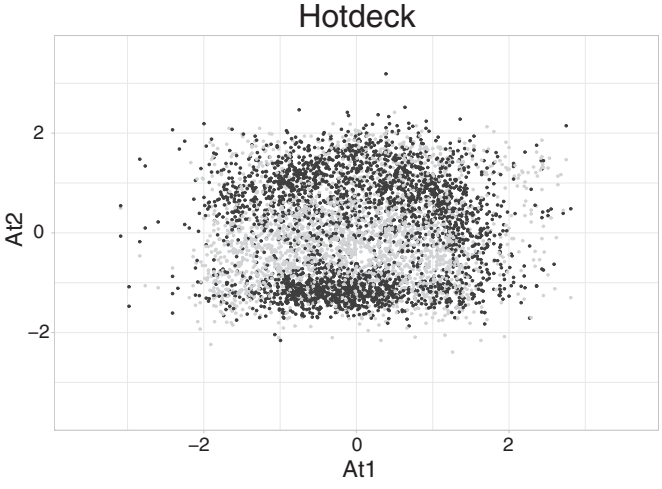


Fig. 10. Imputed values for the missing values created in banana using `equal frequency`.

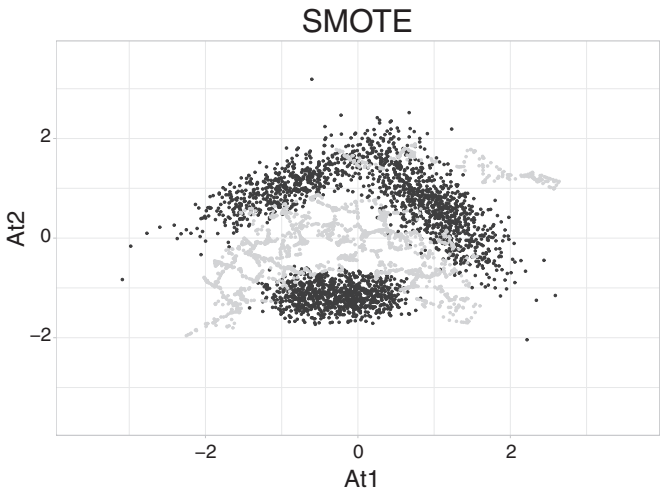


Fig. 11. Banana after resampling with SMOTE.

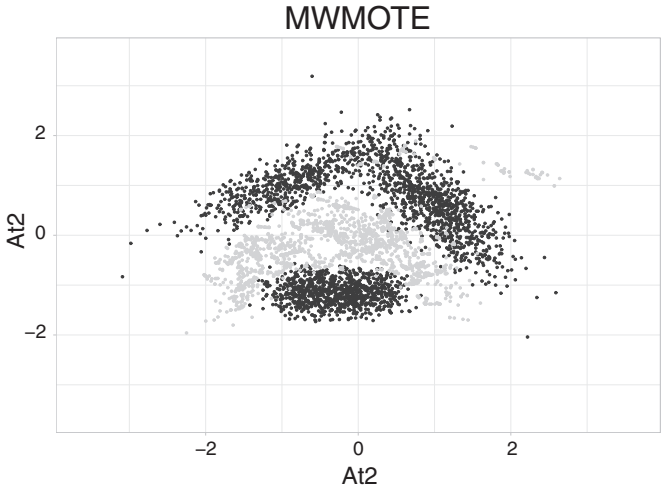


Fig. 12. Banana after resampling with MWMOTE.

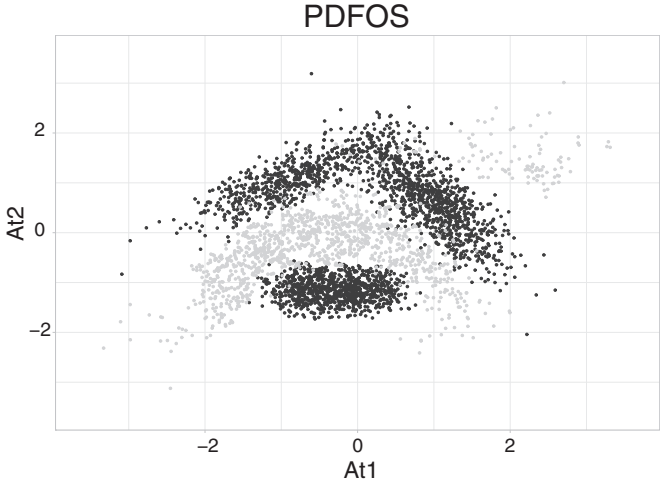


Fig. 13. Banana after resampling with PDFOS.

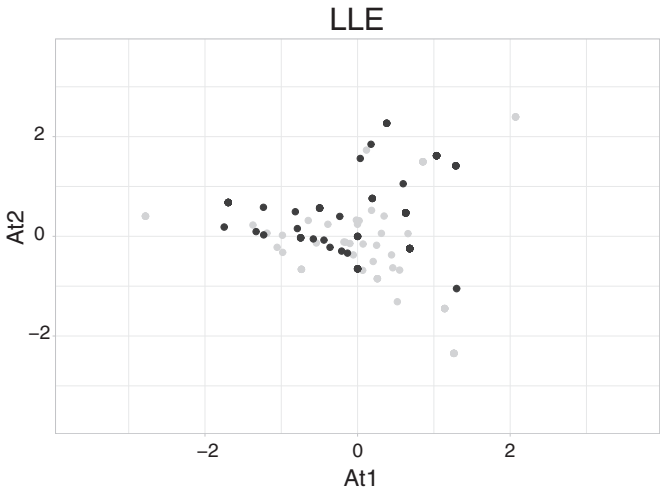


Fig. 14. Sonar dataset transformed to 2D using LLE.

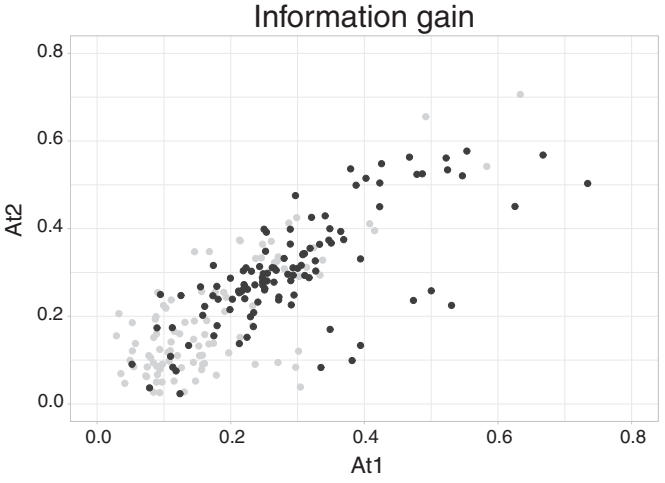


Fig. 15. Sonar data set represented in 2D after using `information gain` feature selection.

3.1.5. Oversampling

Lastly, the example below shows how to apply different oversampling techniques to the dataset aiming to balance the class distributions.

```
banana %>% oversample(method = "SMOTE", ratio = 0.8, filtering = TRUE)
banana %>% oversample(method = "MWMOTE", ratio = 0.9, filtering = FALSE)
banana %>% oversample(method = "PDFOS", ratio = 0.9, filtering = TRUE)
```

In the following figures, the +1 class (which was artificially reduced before) is colored in grey. Figs. 11–13 nicely represent the creation of artificial points of class +1 (this is the class which occupies the central area in Fig. 11, and it was previously decimated to simulate an imbalanced problem) by using SMOTE, MWMOTE and PDFOS, respectively.

3.2. Data reduction

For feature selection and space transformation, the Sonar dataset is going to be used instead, since it contains a large number of input attributes. Using banana is not feasible, as a 2D problem would not need these kind of preprocessing. Since Sonar dataset shows a large dimensionality, only the reduced dimensionality versions, after the space transformation and fea-

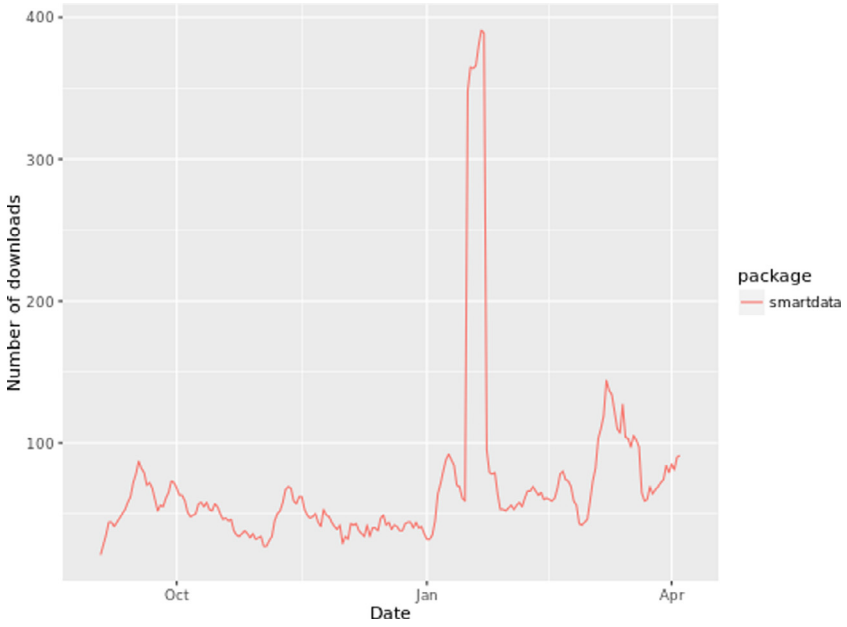


Fig. 16. Downloads for `smartdata` package.

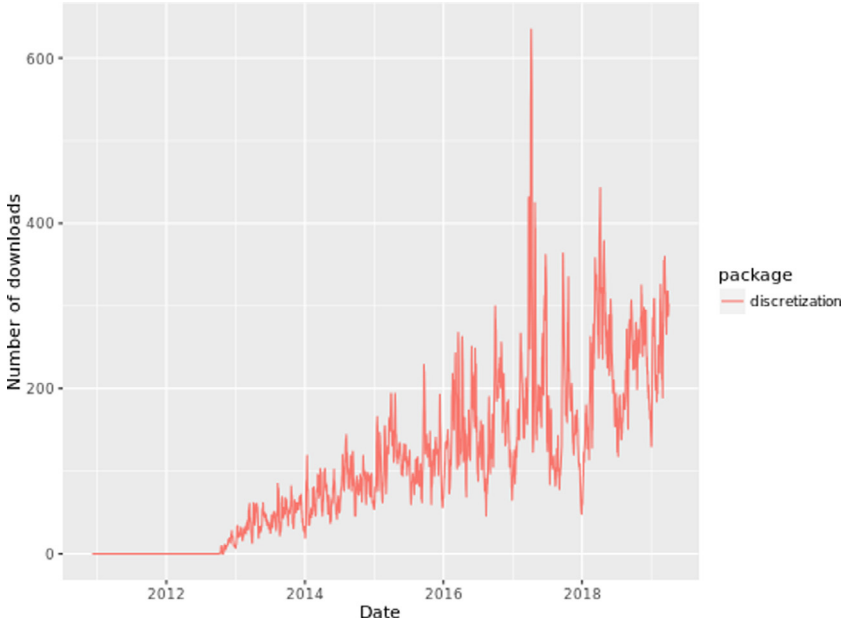


Fig. 17. Downloads for `discretization` package.

ture selection, are represented. First the data is loaded from the `mlbench` package, which needs to be installed before:

```
# Load the previously installed smartdata package
library("smartdata")
# Install and load mlbench
install.packages("mlbench")
library("mlbench")

# Load Sonar dataset from mlbench package
data(Sonar,  package = "mlbench")
```

Then, feature selection and space transformation are performed with ease. On the one hand, Fig. 14 depicts the result of applying LLE, which evaluates data "patches" to estimate the manifolds and the proximity of examples before the transformation. On the other hand, Fig. 15 shows the results of selecting only two attributes among the total of sixty available in `Sonar` by using the `information gain` criteria. Space transformations seem to be better suited to greatly reducing the dimensionality, while feature selection methods are intended to obtain larger subset of attributes with a lesser information loss. Any practitioner can evaluate which approach is better suited to his/her problem with ease with `smartdata`.

```
# Space transformation
Sonar %>% space_transformation("lle_knn", k = 2, regularization = 3, num_features = 2)

# Feature selection
Sonar %>% feature_selection("information_gain", class_attr = "Class", num_features = 2)
```

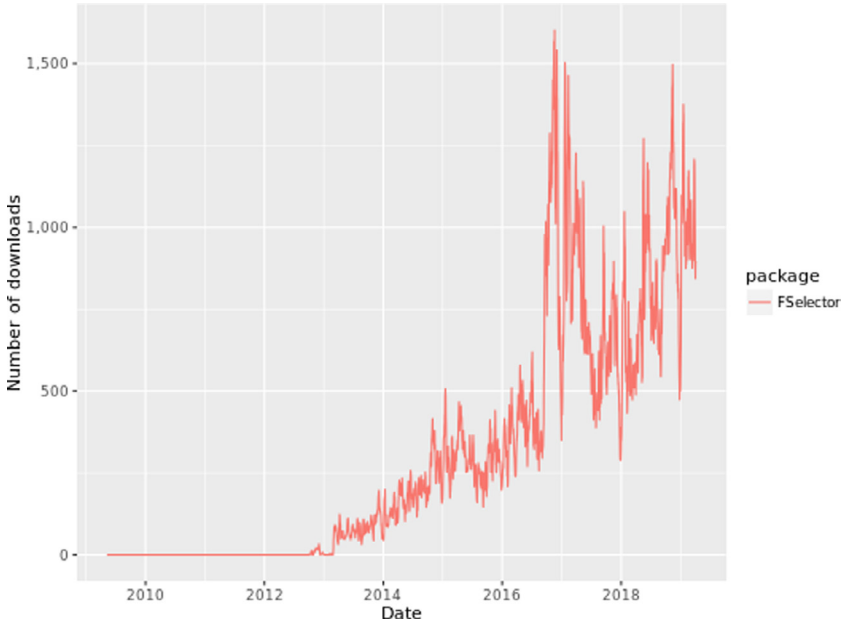


Fig. 18. Downloads for `FSelector` package.

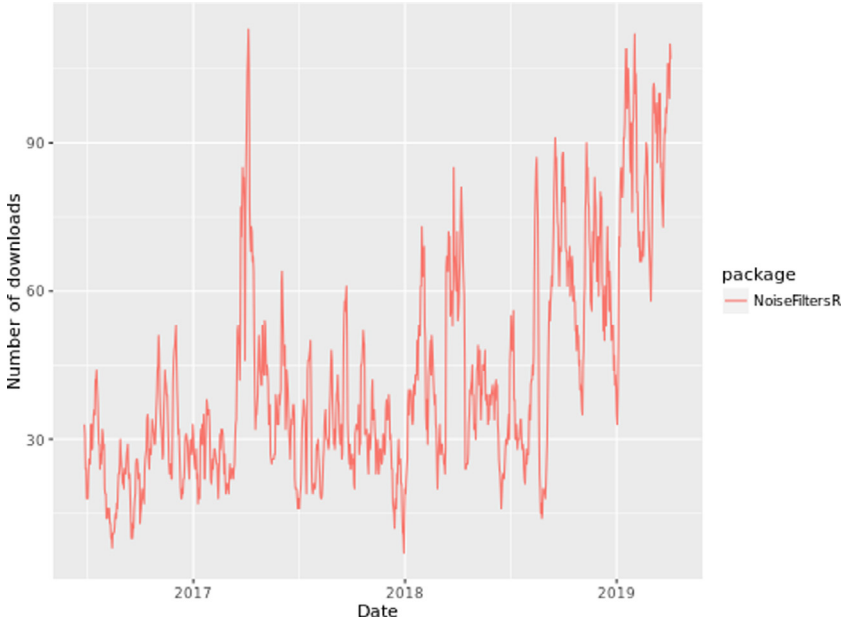


Fig. 19. Downloads for `NoiseFiltersR` package.

4. Conclusions

In this paper the `smartdata` package for R has been presented. It is intended to alleviate some drawbacks that arise in current software solutions. Firstly, to provide an useful implementation of a common interface for preprocessing methods in R, which are scattered through different packages in an heterogeneous way. Secondly, to enable a pipelined workflow that eases the application of several preprocessing steps at once, facilitating the complex treatment of data. Finally, to enable a simpler integration of these methods with the existing preprocessing packages at CRAN.

The package is showing a good reception from data scientists, as shown in the downloads for `smartdata` when compared to other packages included in the proposed wrapper[4]. Figs. 16–21 depict the number of weekly downloads for some of the biggest packages (in terms of included methods) and the `smartdata` package. We have chosen weekly downloads as older packages would present very large cumulative downloads, while weekly

[4] Number of downloads obtained from https://ipub.com/dev-corner/apps/r-package-downloads/.

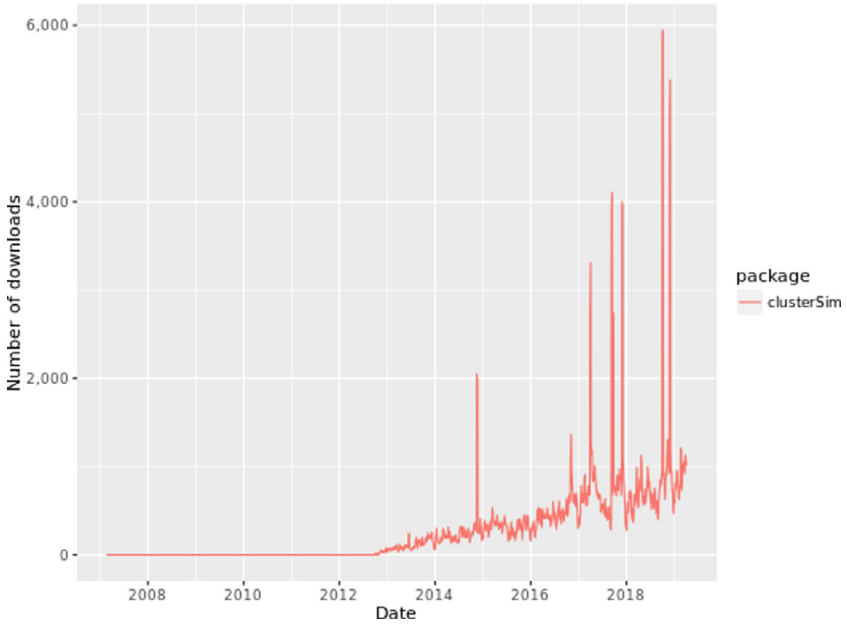


Fig. 20. Downloads for `cluster-sim` package.

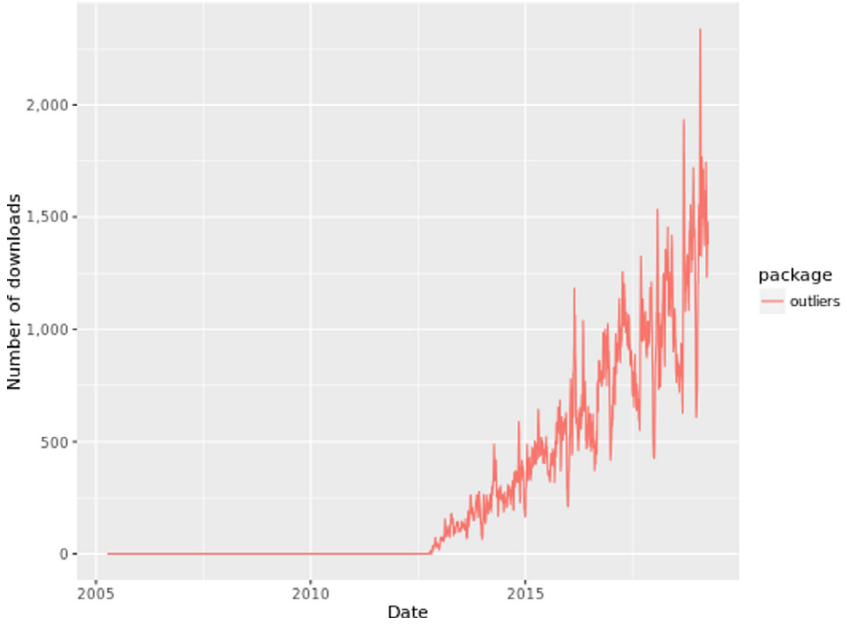


Fig. 21. Downloads for `outliers` package.

downloads will also show the current popularity of the depicted packages. Whereas older packages as `clustser-sim`, `FSelector` and `outliers` currently show higher amounts of weekly downloads, newer packages like `discretization` or `NoiseFiltersR` are on par with `smartdata`. Provided the short life of `smartdata`, we may conclude that its acceptation by the data scientist community is good.

As future work, we propose to keep maintaining and adding functionality to our new `smartdata` package. Specifically, new preprocessing techniques that are regularly proposed in the specialized literature are planned to be included. The extension of the package to cope with nonstandard learning [24] specific preprocessings is also being considered.

Required Metadata

Current executable software version

Table 3
Software metadata.

Nr.	(executable) Software metadata description	Please fill in this column
S1	Current software version	1.0.2
S2	Permanent link to executables of this version	example: https://github.com/ncordon/smartdata
S3	Legal Software License	GPL $\geq$ 2.0
S4	Computing platform/Operating System	Linux, OS X, Microsoft Windows.
S5	Installation requirements & dependencies	R ($\geq$ 3.5.0), Java
S6	If available, link to user manual - if formally published include a reference to the publication in the reference list	https://ncordon.github.io/smartdata/
S7	Support email for questions	nacho.cordon.castillo@gmail.com

Current code version

Table 4
Code metadata.

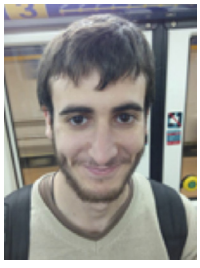
Nr.	Code metadata description	Please fill in this column
C1	Current code version	1.0.2
C2	Permanent link to code/repository used of this code version	https://github.com/ncordon/smartdata
C3	Legal Code License	GPL $\geq$ 2.0
C4	Code versioning system used	git
C5	Software code languages, tools, and services used	R
C6	Compilation requirements, operating environments & dependencies	R ($\geq$ 3.5.0), Java
C7	If available Link to developer documentation/manual	https://ncordon.github.io/smartdata/
C8	Support email for questions	nacho.cordon.castillo@gmail.com

Acknowledgments

This work was funded by the project BigDaP-TOOLS - Ayudas Fundación BBVA a Equipos de Investigación Científica 2016.

References

[1] F. Iafrate, A journey from big data to smart data, in: Digital Enterprise Design & Management, Springer, 2014, pp. 25–33.
[2] A. Lenk, L. Bonorden, A. Hellmanns, N. Rödder, S. Jähnichen, Towards a taxonomy of standards in smart data., in: Proceedings of the 2015 IEEE International Conference on Big Data (Big Data), 2015, pp. 1749–1754.
[3] M. El Arass, N. Souissi, Data lifecycle: from big data to smartdata, in: 2018 IEEE 5th International Congress on Information Science and Technology (CiSt), IEEE, 2018, pp. 80–87.
[4] K. Krishnan, Data Warehousing in the Age of Big Data, Newnes, 2013.
[5] K. Janowicz, F. Van Harmelen, J.A. Hendler, P. Hitzler, Why the data train needs semantic rails, AI Mag. 36 (1) (2015), doi:10.1609/aimag.v36i1.2560.
[6] W. Hu, S.L. Shah, T. Chen, Framework for a smart data analytics platform towards process monitoring and alarm management, Comput. Chem. Eng. 114 (2018) 225–244.
[7] N. Alhakbani, M.M. Hassan, M. Ykhlef, G. Fortino, An efficient event matching system for semantic smart data in the internet of things (iot) environment, Fut. Generat. Comput. Syst. 95 (2019) 163–174.
[8] S. García, J. Luengo, F. Herrera, Data Preprocessing in Data Mining, Springer, 2015.
[9] A. Roy, R.M. Cruz, R. Sabourin, G.D. Cavalcanti, A study on combining dynamic selection and data preprocessing for imbalance learning, Neurocomputing 286 (2018) 179–192.
[10] F. Lauer, Mlweb: a toolkit for machine learning on the web, Neurocomputing 282 (2018) 74–77.
[11] Y. Wu, S.C. Hoi, C. Liu, J. Lu, D. Sahoo, N. Yu, Sol: a library for scalable online learning algorithms, Neurocomputing 260 (2017) 9–12.
[12] I. Triguero, S. González, J.M. Moyano, S. García, J. Alcalá-Fdez, J. Luengo, A. Fernández, M.J. del Jesús, L. Sánchez, F. Herrera, KEEL 3.0: an open source software for multi-stage analysis in data mining, Int. J. Comput. Intell. Syst. 10 (1) (2017) 1238.
[13] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: machine learning in Python, J. Mach. Learn. Res. 12 (2011) 2825–2830.
[14] W. McKinney, pandas: a foundational python library for data analysis and statistics, Python High Perform. Sci. Comput. (2011) 1–9.
[15] J. Demšar, T. Curk, A. Erjavec, Črt Gorup, T. Hočevar, M. Milutinovič, M. Možina, M. Polajnar, M. Toplak, A. Starič, M. Štajdohar, L. Umek, L. Žagar, J. Žbontar, M. Žitnik, B. Zupan, Orange: data mining toolbox in python, J. Mach. Learn. Res. 14 (2013) 2349–2353.
[16] M. Kuhn, Building predictive models in r using the caret package, J. Stat. Softw., Artic. 28 (5) (2008) 1–26.
[17] P. Morales, J. Luengo, L.P. Garcia, A.C. Lorena, A.C. de Carvalho, F. Herrera, The noiseFiltersR package: label noise preprocessing in R, R J. 9 (1) (2017) 219–228.
[18] I. Cordón, S. García, A. Fernández, F. Herrera, Imbalance: oversampling algorithms for imbalanced classification in R, Knowl.-Based Syst. 161 (2018) 329–341.
[19] M. Frasca, G. Valentini, Cosnet: an r package for label prediction in unbalanced biological networks, Neurocomputing 237 (2017) 397–400.
[20] S.v. Buuren, K. Groothuis-Oudshoorn, Mice: multivariate imputation by chained equations in R, J. Stat. Softw. (2010) 1–68.
[21] J. Honaker, G. King, M. Blackwell, et al., Amelia ii: a program for missing data, J. Stat. Softw. 45 (7) (2011) 1–47.
[22] H. Wickham, tidyverse: Easily Install and Load the 'Tidyverse', 2017. R package version 1.2.1.
[23] S. García, J. Luengo, F. Herrera, Tutorial on practical tips of the most influential data preprocessing algorithms in data mining, Knowl. Based Syst. 98 (2016) 1–29.
[24] D. Charte, F. Charte, S. García, F. Herrera, A snapshot on nonstandard supervised learning problems: taxonomy, relationships, problem transformations and algorithm adaptations, Progr. Artif. Intell. (2018), doi:10.1007/s13748-018-00167-7.



Ignacio Cordón, Mathematician and Computer Scientist undergraduate and student of a Master s degree in Physics and Mathematics in the University of Granada. He is currently working for a R&D bioinformatics company, developing data analysis software in Scala and R. He is author of a package for data preprocessing in R: imbalance and co-author of its associated paper Oversampling Algorithms for Imbalanced Classification in R.

Among his interests we can emphasize data analysis, statistics, data preprocessing and software engineering.

Juliín Luengo received the M.S. degree in computer science and the Ph.D. from the University of Granada, Granada, Spain, in 2006 and 2011 respectively.

He currently acts as an Assistant Professor in the Department of Computer Science and Artificial Intelligence at the University of Granada, Spain.

His research interests include machine learning and data mining, data preparation in knowledge discovery and data mining, missing values, noisy data, data complexity and fuzzy systems.

Dr. Luengo has been given some awards and honors for his personal work or for his publications in and conferences, such as IFSA-EUSFLAT 2009 Best Student Paper Award. He belongs to the list of the Highly Cited Researchers in the area of Computer Sciences (2015–2018): http://highlycited.com/ (Clarivate Analytics).

Salvador García received the M.Sc. and Ph.D. degrees in Computer Science from the University of Granada, Granada, Spain, in 2004 and 2008, respectively. He is currently an Associate Professor in the Department of Computer Science and Artificial Intelligence, University of Granada, Granada, Spain.

Salvador García has published more than 70 papers in international journals (more than 50 in Q1), with more than 4500 citations, h-index 26, over 45 papers in international conference proceedings (data from Web of Science). He is a member of the editorial board of the Information Fusion (Elsevier), Swarm and Evolutionary Computation (Elsevier) and AI Communications (IOS Press) journals, and he is co-Editor in Chief of the international journal Progress in Artificial Intelligence (Springer). He is a co-author of the book entitled Data Preprocessing in Data Mining published by Springer. His research interests include data

science, data preprocessing, Big Data, evolutionary learning, Deep Learning, metaheuristics and biometrics.

Dr. García has been given some awards and honors for his personal work or for his publications in and conferences, such as IFSA-EUSFLAT 2015 Best Application Paper Award and IDEAL 2015 Best Paper Award. He belongs to the list of the Highly Cited Researchers in the area of Computer Sciences (2014–2017): http://highlycited.com/ (Clarivate Analytics).

Francisco Herrera (SM'15) received his M.Sc. in Mathematics in 1988 and Ph.D. in Mathematics in 1991, both from the University of Granada, Spain. He is currently a Professor in the Department of Computer Science and Artificial Intelligence at the University of Granada and Director of DaSCI Institute (Andalusian Research Institute in Data Science and Computational Intelligence).

He has been the supervisor of 43 Ph.D. students. He has published more than 400 journal papers, receiving more than 66000 citations (Scholar Google, H-index 128). He is co-author of the books "Genetic Fuzzy Systems" (World Scientific, 2001) and "Data Preprocessing in Data Mining" (Springer, 2015), "The 2-tuple Linguistic Model. Computing with Words in Decision Making" (Springer, 2015), "Multilabel Classification. Problem analysis, metrics and techniques" (Springer, 2016), Multiple Instance Learning. Foundations and Algorithms" (Springer, 2016) and Learning from Imbalanced Data Sets (Springer, 2018).

He currently acts as Editor in Chief of the international journals "Information Fusion" (Elsevier) and Progress in Artificial Intelligence (Springer). He acts as editorial member of a dozen of journals.

He received the following honors and awards: ECCAI Fellow 2009, IFSA Fellow 2013, 2010 Spanish National Award on Computer Science ARITMEL to the "Spanish Engineer on Computer Science", International Cajastur "Mamdani" Prize for Soft Computing (Fourth Edition, 2010), IEEE Transactions on Fuzzy System Outstanding 2008 and 2012 Paper Award (bestowed in 2011 and 2015 respectively), 2011 Lotfi A. Zadeh Prize Best paper Award (IFSA Association), 2013 AEPIA Award to a scientific career in Artificial Intelligence, 2014 XV Andaluca Research Prize Maimónides, 2017 Security Forum I+D+I Prize, 2017 Andaluca Medal (by the regional government of Andaluca), 2018 Granada: Science and Innovation City. Recently he has been nominated as member of the Spanish Royal Academy of Technology (RAIng).

He has been selected as a Highly Cited Researcher http://highlycited.com/ (in the fields of Computer Science and Engineering, respectively, 2014 to present, Clarivate Analytics).

His current research interests include among others, Computational Intelligence (including fuzzy modeling, computing with words, evolutionary algorithms and deep learning), information fusion and decision making, and data science (including data preprocessing, prediction and big data).

Francisco Charte received his B.Eng. degree in Computer Science from the University of Jan in 2010 and his M.Sc. and Ph.D. in Computer Science from the University of Granada in 2011 and 2015. He is an Assistant Professor of Computer Architecture and Computer Technology with the Computer Science Department at the University of Jan (Spain). He is coauthor of the book "Multilabel Classification. Problem analysis, metrics and techniques" (Springer, 2016). His main research interests include machine learning with applications to multilabel classification, high dimensionality and imbalance problems, as well as deep learning algorithms.