On a Comprehensive Metadata Framework for Artificial Data in Unsupervised Learning

Rainer Dangl and Friedrich Leisch

Abstract Evaluating new methods and algorithms in unsupervised learning obviously requires thorough benchmarking studies on data sets that most closely reflect performance in actual usage. Designing data sets that do exactly that is quite a challenging task in itself; standing up to the challenge in comparison to other methods is another point which poses a risk of compromising the goal of an objective benchmarking study. We want to address the latter by proposing a framework that standardizes the format of artificial data, or rather its metadata. We intend to introduce a web repository that functions as an exchange for metadata of artificial data and an accompanying R package that can generate actual data from the descriptions obtained from the repository. It is therefore much simpler to find data designed by others and which has been used in previous benchmarking studies. This removes some of the temptation to specifically design artificial data in a way so that a proposed method performs significantly better than existing ones, a claim that might not hold in real life applications.

Rainer Dangl
University of Natural Resources and Life Sciences, Peter-Jordan-Straße 82, 1190 Vienna, Austria
✉ `rainer.dangl@boku.ac.at`

Friedrich Leisch
University of Natural Resources and Life Sciences, Peter-Jordan-Straße 82, 1190 Vienna, Austria
✉ `friedrich.leisch@boku.ac.at`

ARCHIVES OF DATA SCIENCE, SERIES A
(ONLINE FIRST)
KIT SCIENTIFIC PUBLISHING
Vol. 2, No. 1, 2017

DOI 10.5445/KSP/1000058749/22
ISSN 2363-9881

1 Background

When setting up a proper benchmarking study, one of the most important factors that comes to mind intuitively is objectivity. It is certainly inherent to the definition of benchmarking that methods should be compared in a most objective and neutral way in order to determine an unbiased winner. Boulesteix et al (2013) have conducted a survey that investigates the outcome of comparison studies with respect to objectivity. They found that benchmarking studies which are conducted as part of a paper that presents a new method very often identify said new method as the winner, while studies that exclusively compare methods do not always identify a clear winner. This discrepancy may certainly be rooted in many aspects - for example overall benchmarking study design, selection of methods to compare to, or selection of benchmarking data. This paper intends to address the last of the three as one aspect that can be improved upon. Selection of benchmarking data sets is certainly one of the most crucial choices to make, and especially artificial data and its design is a major factor to a successful comparative study.

However, it is not the focus of this paper to introduce guidelines on how artificial data should look like; we rather regard the proposed framework as a contribution to an effort in the computational science community that has gained some momentum in recent years: reproducible research (Mesirov, 2010). In order to substantiate scientific claims, this keyword refers to the necessity that papers should not only need to describe the results of experiments and studies, but also provide a clear protocol which allows replication of those results by the reader (Mesirov, 2010). Unfortunately, computations and the resulting conclusions are quite often taken at face value (Donoho, 2010), which is especially problematic as studies find that frequently either several details essential for successful reproduction are missing (Nekrutenko and Taylor, 2012), or replication is at least difficult (Ioannidis et al, 2009). These are points that are obviously quite problematic in the context of benchmarking. Furthermore, insufficient reproducibility also leads to an increase in retracted papers (Steen, 2011) and has quite severe implications in practice, such as failed clinical trials (Prinz et al, 2011; Begley and Ellis, 2012), which is certainly also not intended by any researcher. Stodden (2010) proposes several points that should help to improve the situation, like providing links to source code and data, keeping track of the computing environment and versions of software used and publishing data and code in non-proprietary formats and under open licenses. With regard to the topic in this paper - artificial data generation - our contribution to reproducible research is an infrastructure making the data set that was used for the computation easily available, a requirement that is also emphasized in Peng et al (2006). One of the primary problems is that quite often a major hindrance to replicating results is that code that generates data is no longer available (Peng, 2011).

We intend to relieve this issue by proposing a development framework for artificial data that allows to easily create, exchange, and generate data sets. The framework makes the development and generation of artificial data more transparent, and more importantly, reproducible. Basically, the framework consists of a web application that is used to store information on data sets and an R package that is used to generate the data sets. The main advantage of introducing a common framework for artificial data is that it is then very easy to obtain data from previous studies, should the author choose to make the code available to others (which of course has to be cited properly - another incentive). Furthermore, if suitable data is already available from previous studies, there is no real argument to develop new data from scratch. This in turn again reduces the tendency to develop data selectively. Also, as noted above, quite often artificial data is insufficiently described in publications, or the code is not available (or only available for another software package), which makes re-coding of already used data quite cumbersome. This is also greatly reduced by an artificial data repository.

2 Framework Design and Terminology

The framework was designed in order to achieve a maximum of platform independence and ease of use, also for users who are not accustomed to programming. Furthermore, an open source approach also greatly increases availability and willingness to actually use this new way of managing artificial data.

The foundation of the whole concept is the R programming language due to its widespread use in the statistical community. While some R experience is necessary to create new benchmarking data sets with this framework, it is quite simple to download and generate data from existing setups.
There are two main parts to the process: an R package and a web repository. The R package implements all functionalities that are needed to generate artificial data sets from R script files that contain the metadata information. It also offers other tools that are described in more detail below. The script files are

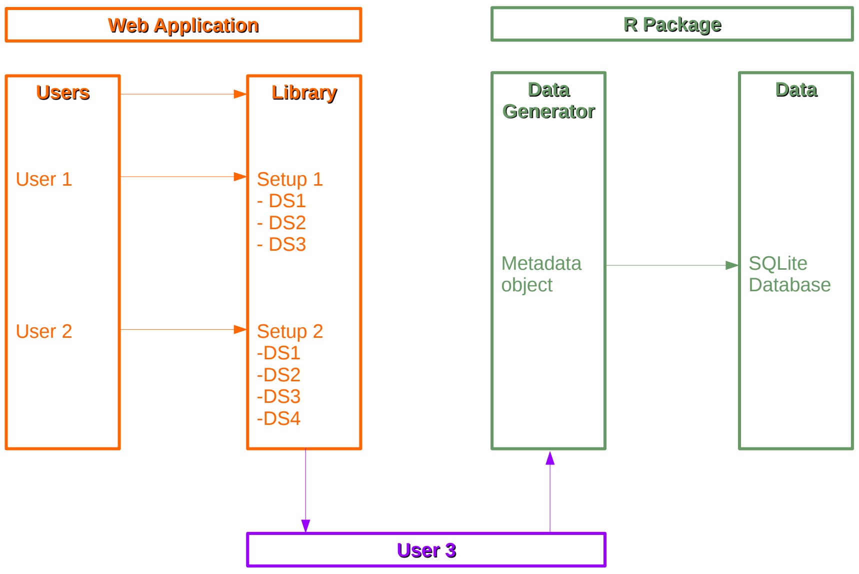


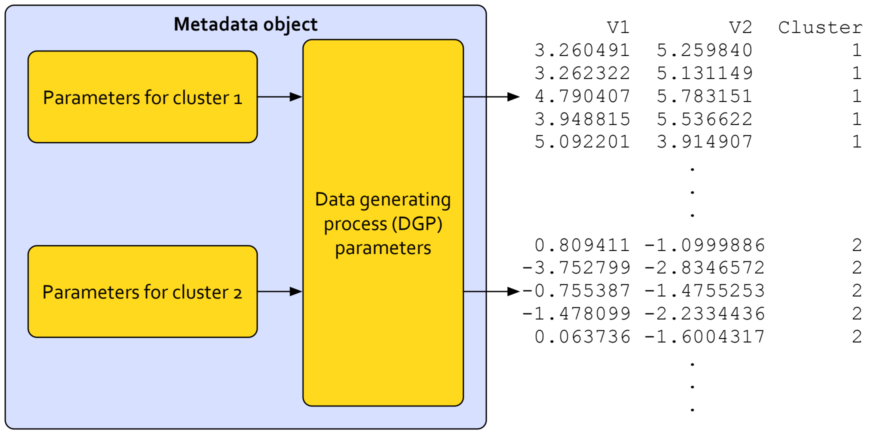
Fig. 1 Framework Layout

the essential part. Basically, they have to be written from scratch by the user. They have to conform to specific rules in order to work with the package. The web repository as the second important part serves as a means to conveniently collect and exchange script files. As illustrated in Fig. 1, the repository hosts data sets that are summarized in so called *benchmarking setups*. One user uploads a script file that contains a benchmarking setup which consists of several data sets. Another user can download the script file and generate the metadata information and in turn the actual data with the R package on the local computer. Consequently, the limit for the kinds of datasets that can be generated (e.g. size, particularly in the big data context), depends on the local hardware of the user, not on the R package or the web repository.

3 Metadata

The framework is based around the notion of using metadata as a means to most efficiently store artificial data. The actual random numbers are not of great relevance; if all parameters of how the data set was generated (data generating process, random number seeds, etc.) are thoroughly documented, the data set can easily be reproduced, eliminating the need to store the actual numbers. Furthermore, this greatly increases transparency, because all essential parameters about the data that could be needed in the further analysis of test results are available. This also highlights why the title of this paper proposes a 'metadata framework' - everything is centered around metadata information. Therefore the script file contains no code that generates actual numbers, it rather produces for each data set included in the setup a metadata object (an S4 object, one of the object systems in R, see (Wickham, 2014)) that complies with the specifications in the R package and that is processed by it to produce actual data. The structure of the metadata object varies to some degree according to which data type is used (metric, binary, ordinal, etc.). Yet the basic structure is approximately the same and is illustrated in Fig. 2. The parameters are assembled cluster-wise by the R package. All information necessary has to be included in the script file, for example with regard to metric data this includes at least the cluster centers, the variance-covariance matrix for each cluster, and the number of observations in each cluster. Furthermore, also a number generating function is needed. This can be just the name of the function, if it already exists in R or in some other package, but also a custom function can be provided. Finally, also information on the random number seed is needed (which one to use, which random number seed, etc.). This way, the metadata object can be processed.

This structure of metadata has several convenient advantages. There is no need to awkwardly sift through programming code to extract information about artificial data that somebody else has implemented. All information needed is encapsulated in a clearly structured way in the metadata object. Moreover, the design of the object is very flexible, it allows a very broad scope of possible data sets. There is no prescribed random number generating function, no prescribed list of obligatory parameters for each cluster. It merely prescribes a certain structure that the metadata has to comply with. This provides a reliable structure that is the same across all data sets of this type and this greatly simplifies understanding and transparency.



```
       V1        V2  Cluster
 3.260491  5.259840        1
 3.262322  5.131149        1
 4.790407  5.783151        1
 3.948815  5.536622        1
 5.092201  3.914907        1
              .
              .
              .
 0.809411 -1.0999886       2
-3.752799 -2.8346572       2
-0.755387 -1.4755253       2
-1.478099 -2.2334436       2
 0.063736 -1.6004317       2
              .
              .
              .
```

Fig. 2 Metadata to actual data

3.1 Data Types

There are various data types for which metadata objects can be created. The most commonly used type is certainly metric data, but there are several more: in order to deal with categories there are binary and ordinal objects available; the functional data type allows implementation of time series data and other scenarios needed for functional clustering; there is also an implementation to generate random string and wordnet data.

3.1.1 Metric Data

The metadata object for metric data primarily consists of two very important slots: the `genfunc` slot that contains the function which generates random numbers (default is `mvrnrom` from package `MASS`), and the `clusters` slot that contains the parameters that the function in slot `genfunc` needs. Consequently, the arguments of the generating function and the list items in the clusters slot have to correspond. The data set is then assembled cluster by cluster by the respective function in the R package of the framework. The complete metadata object has an additional third slot: `seedinfo` contains information on the random number generator. Firstly, which one to use (the default is *Mersenne-Twister*, as it is the default in R), which version of R and thus the generator (default is the current R version) and the random number seed (default is 100). Figure 3 illustrates a simple metadata object for metric data.

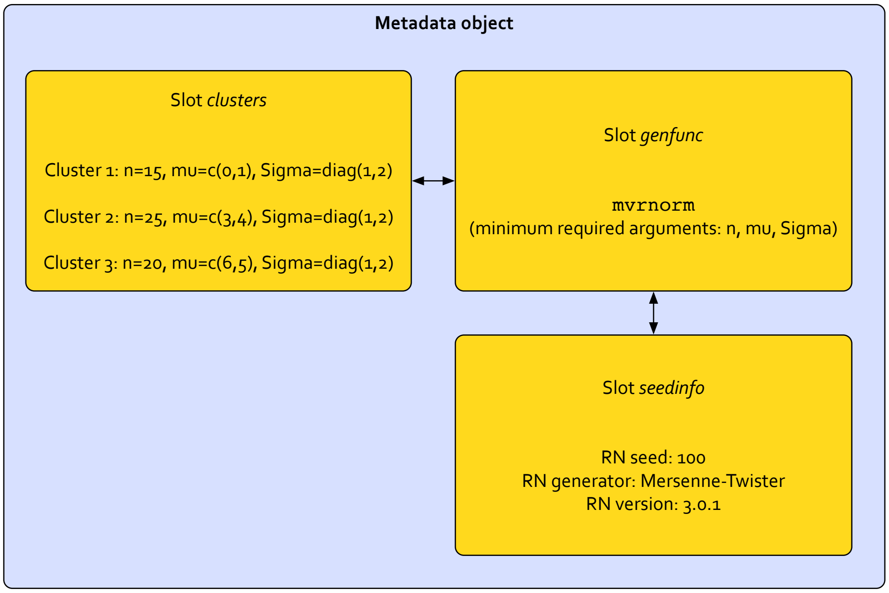

Metadata object
Slot *clusters*
Cluster 1: n=15, mu=c(0,1), Sigma=diag(1,2)
Cluster 2: n=25, mu=c(3,4), Sigma=diag(1,2)
Cluster 3: n=20, mu=c(6,5), Sigma=diag(1,2)
Slot *genfunc*
mvrnorm
(minimum required arguments: n, mu, Sigma)
Slot *seedinfo*
RN seed: 100
RN generator: Mersenne-Twister
RN version: 3.0.1

Fig. 3 Metric metadata object

3.1.2 Ordinal and Binary Data

For binary and ordinal data, the basic structure is essentially the same as for metric data, only the random number generating function is obviously different (the default here is `ordsample` from package `GenOrd` and `generate.binary` from package `MultiOrd` for ordinal and binary data respectively).

3.1.3 Functional Data

Functional data is the only type of metadata whose object differs notably from the other types in terms of overall structure. The cluster centers here are functions, which are stored as a list in a slot of the same name. The other slots provide all other parameters that are needed to evaluate the functions; the primary element being the `gridMatrix`, which encodes for each instance of a cluster center the number of time points and location of evaluation. In slot `interval`, the upper and lower boundaries are given; argument `granularity` determines the steps in-between. Figure 4 for example shows that instance 1 of a particular cluster center function is evaluated on position 3,4,6,7,8 and 10 of the interval. The evaluations are irregular, otherwise each instance would be evaluated at the same time points. Slots `sd` and `sd_distribution` determine the distribution and deviation of the instances around the cluster center function.

```
> samplegrid(total_n=10, minT=4, maxT=7, granularity=10)

      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
 [1,]    0    0    1    1    0    1    1    1    0     1
 [2,]    1    0    1    0    0    0    0    1    0     1
 [3,]    1    1    0    0    0    1    1    0    0     0
 [4,]    0    1    0    1    1    1    1    0    1     0
 [5,]    1    0    1    1    0    1    1    1    1     0
 [6,]    1    1    0    0    1    0    0    0    1     1
 [7,]    0    1    0    0    1    1    1    1    1     1
 [8,]    1    1    0    0    1    1    0    1    1     1
 [9,]    1    1    1    1    0    1    1    0    0     1
[10,]    1    1    0    0    0    1    1    1    1     1
```

Fig. 4 gridMatrix for 10 instances of a cluster center

3.1.4 String Data

String data is implemented to support benchmarking of string distance measures. Currently the concept supports the generation of random string or wordnet data. The latter is an online corpus of English and allows to filter data from the corpus with respect to certain criteria. The structure of the metadata object for random string data is again quite similar to metric data in Fig. 3, except in slot `genfunc` there is a function that generates random strings based on a certain reference string and given a permissible maximum string distance. Those parameters (reference string, which type of distance and the maximum allowed distance) are again stored as a list for each cluster in slot `clusters`.

The wordnet metadata object is strongly linked to the respective R package `wordnet`. The string generating mechanism cannot be changed. For each cluster, a reference string is again provided as for random strings, and additionally a so called *filter type* is required, which determines the relationship of the cluster members to the reference string (e.g. `SoundFilter` to obtain similar sounding words). For a detailed description of available filters please confer the wordnet package vignette (Feinerer and Hornik, 2015; Wallace, 2007; Fellbaum, 1998).

4 The R package bdlp

The bdlp (benchmark data library project) package is the primary tool to develop and generate artificial data (Dangl, 2017). It processes so called benchmarking setup files which basically are R script files that produce the metadata objects in the form described above. The package implements the metadata object classes and provides functions that create templates for script files, check benchmarking setup files before submission and generate data from metadata objects.

A file containing a benchmarking setup (which in turn contains definitions for metadata objects) is in its most basic form an R script file that contains only one function. This function can return two things: Either information on the data sets, or the metadata object for a specific data set. The former is merely intended as information for the user, in order to get an overview and choose a data set to generate. The latter is the basis to generate actual data, as explained above. A very minimal example that can generate metadata for two data sets and the accompanying info table is shown in Fig. 5.
The obligatory function in the script file has to have a specific name: the author of the setup and the year it was created and/or published. The arguments of the function are also fixed. The `info` argument is used to toggle between info table or metadata output, for the latter the respective `setnr` is obviously also necessary. The `seedinfo` and `metaseedinfo` arguments are used to set the random number generator parameters for metadata and actual data generation. As it is of course possible to also have random effects already when generating metadata (e.g. location of cluster centers), two distinct sets of parameters can be defined.

```
require(MASS)

dangl2014 <- function(setnr = NULL,
                      seedinfo = list(100,
                                      paste(R.version$major,
                                            R.version$minor,
                                            sep = "."),
                                      RNGkind()),
                      info = FALSE,
                      metaseedinfo = list(100,
                                          paste(R.version$major,
                                                R.version$minor,
                                                sep = "."),
                                          RNGkind())){

  inf <- data.frame(n = c(50, 40), k = c(2,2),
                    shape = c("spherical", "spherical"))
  ref <- "Dangl R. (2014) A small simulation study.
          Journal of Simple Datasets 10(2), 1-10"
  if(info == T) return(list(summary = inf, reference = ref))

  if(is.null(metaseedinfo)) metaseedinfo <- seedinfo

  set.seed(metaseedinfo[[1]])
  RNGversion(metaseedinfo[[2]])
  RNGkind(metaseedinfo[[3]][1], metaseedinfo[[3]][2])

  if(setnr == 1) {
    return(new("metadata.metric",
      clusters = list(c1 = list(n = 25,
                                mu = c(4,5),
                                Sigma=diag(1,2)),
                      c2 = list(n = 25,
                                mu = c(-1,-2),
                                Sigma=diag(1,2))),
      genfunc = MASS::mvrnorm, seedinfo = seedinfo))
  }
  if(setnr == 2){
    return(new("metadata.metric",
      clusters = list(c1 = list(n = 40,
                                mu = c(0,2,0),
                                Sigma=diag(1,3)),
                      c2 = list(n = 40,
                                mu = c(-1,-2,-2),
                                Sigma=diag(1,3))),
      genfunc = MASS::mvrnorm, seedinfo = seedinfo))
  }
}
```

Fig. 5 Minimal example for an experimental setup file

Contrary to the strictly prescribed structure of the function, its content is highly flexible though - apart from the requirement that either the info table of a valid metadata object has to be returned no specific restrictions are put on the code that can be executed in the function body. It is certainly possible to load required packages, even custom functions that are written from scratch by the developer can be included. In this case these functions are added below and simply sourced at runtime when the function is executed.

There are three ways to arrive at a benchmarking setup file that the package can process. Firstly, one can write the script file completely from scratch, according to the formatting guidelines. Secondly, one can generate a template for the script file using `createFileskeleton`. It takes as arguments some author information, the reference and the info table (basically a stripped-down version of function `saveSetup` described below). The resulting .R file looks like Fig. 5, just without the two metadata objects. Then, the file can be modified further in a text editor. Thirdly, one can generate the script file completely automatically as shown in Fig. 6. Each metadata object is initialized using the function `initializeObject`. The arguments for this function are the data type, number of clusters and the random number generating function. Furthermore, one can also add information on the random number seed. The objects are then modified by the user, i.e. filled with the cluster parameters. Once the metadata objects are complete, the complete script file can be generated using `saveSetup`.

```
require(MASS)
m1 <- initializeObject(type = "metric", genfunc = mvrnorm, k = 2)
m1@clusters$cl1 <- list(n = 25, mu = c(4,5), Sigma = diag(1,2))
m1@clusters$cl2 <- list(n = 25, mu = c(-1,-2), Sigma = diag(1,2))

m2 <- initializeObject(type = "metric", genfunc = mvrnorm, k = 2)
m2@clusters$cl1 <- list(n = 44, mu = c(1,2), Sigma = diag(1,2))
m2@clusters$cl2 <- list(n = 66, mu = c(-5,-6), Sigma = diag(1,2))

saveSetup(name="miller2012.R", author="Mister Miller",
          mail="mister.miller@edu.com",
          inst="Unknown University",
          cit="Simple Data, pp. 23-24",
          objects=list(m1, m2),
          table=data.frame(n = c(50, 110), k = c(2,2),
                           shape = c("spherical", "spherical")))
```

Fig. 6 Creating a script file automatically

Once the script file is complete, data can be generated (Fig. 7). This is done by calling the function `generateDatabase`, where the name of the benchmarking setup, the desired dataset number therein, and the number of draws have to be specified (and optionally of course the random number seed parameters, which overrides the default ones specified in the script file). An SQLite database file is produced that contains the data sets. At this point, the data sets can the be further processed in other software environments that support this file type, not necessarily in R.

```
> generateDatabase(name = "dangl2014.R", setnr = 1, draws = 20)
|=======================================================| 100\%

20 version(s) of set no. 1 of setup dangl2014 generated.
Base seed 100 was used and is included in file name.
```

Fig. 7 Generation of data in R

If the user wishes to contribute the benchmarking setup to the web repository, the user should run function `checkSetup` before uploading. This saves time, because the same check is run on submitted files, and would result in instant rejection if an error is detected. After a successful check and upload, the repository maintainers will have a final review of the new setup before it is made public in the library.

5 The web repository

The website has been built using the R package `shiny` (Chang et al, 2015), which allows the development of R based web applications. There is an open source server environment available by RStudio that runs a web app developed with the `shiny` package. `shiny` allows to implement reactive programming paradigms in R which means that function outputs are immediately updated once one of the input arguments changes. This means that several calculations such as plotting are available in real time on the website and can react to changing inputs. For example, it is possible to select a benchmarking setup and a particular data set therein and look at a plot, switching to another data set triggers rendering of an updated plot. For this purpose, one instance of the data set is generated from the metadata information and a plot is drawn in real time

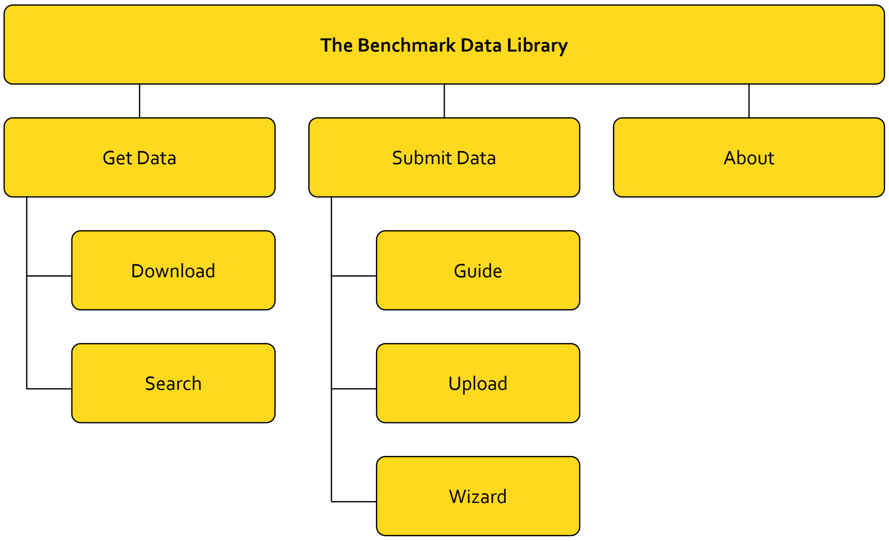


Fig. 8 Repository site map

and displayed on the website. Furthermore, it is possible to select a benchmarking setup and alter the metadata parameters in a web editor and download a modified script file if the user deems it necessary. This also makes using the framework easier for users who have little or no programming experience or who do not usually use R in their work. All this can be conveniently implemented using `shiny`.

The web repository serves as a complement to the R package. It is not strictly necessary in order to generate data - this can be done entirely locally with the R package - but it is a convenient means to collect benchmarking setups and to make them available to the public in an easily manageable manner. Benchmarking setups can be contributed by everybody and after an approval process by the maintainer they are put into the library. Figure 8 shows the structure of the website.

The download section is divided into the actual download page and a search page. The latter features a live search form that uses the obligatory reference part of the info output of the script file to filter available setups based on authors, publication year and keywords in the title. The download page itself features a form to filter by data type and available setups and the included data sets thereof. Data sets can be plotted and their parameters modified in a source code editor.

The contribute section is quite simple: There is a guide that essentially reflects part of this paper in explaining the structure of a metadata object and the format of the script file in order to create a benchmarking setup. The upload page itself only contains a submit form. If the upload is completed successfully, the website maintainer is notified and the script file will be reviewed. The R package includes the aforementioned function `checkSetup` that allows the user to check the formal correctness of the script file already before submitting. If the reviewer determines that the submission is not only correct with regard to syntax but also content (especially whether there is a proper/correct citation), the setup is put into the library. The submitter will be notified in any case, be it acceptance or rejection.

Furthermore, the contribute section also features a web form (wizard) which is simply the web version of the `createFileskeleton` function of the R package. Therefore, it also creates a template script file that can be downloaded which contains the basic layout of the function, the info table and reference, only the code for the metadata objects needs to be inserted.

6 Conclusion and Outlook

The Benchmark Data Library and the accompanying R package `bdlp` are not only an attempt to standardize the format with which artificial data is stored and generated, it is an initiative to promote increasing objectivity in benchmarking studies. The package should not just be a tool to create data, it should help researchers to collaborate and share their work. This certainly cannot be established overnight, but as the project is embedded in the efforts of the IFCS task force on benchmarking, we are confident that the project will receive some attention and does have a promising starting point.

Application in practice will also show further necessary modifications to the platform and the package (e.g. adapting the structure of metadata objects, add new data types, etc.), in order to support as many benchmarking applications as possible.

Acknowledgements I thank the members of the IFCS Task Force on Benchmarking for their valuable input and feedback during development and for their continuing support in implementing the framework in practice. (`https://ifcs.boku.ac.at/repository/`)

References

Begley CG, Ellis LM (2012) Drug development: Raise standards for preclinical cancer research. Nature 483(7391):531–533, DOI 10.1038/483531a

Boulesteix AL, Sabine L, A EMJ (2013) A Plea for Neutral Comparison Studies in Computational Sciences. PLoS ONE 8(4):e61,562, DOI 10.1371/journal.pone.0061562

Chang W, Cheng J, Allaire J, Xie Y, McPherson J (2015) shiny: Web Application Framework for R. URL `http://shiny.rstudio.com`, r package version 0.11.1.9004

Dangl R (2017) bdlp: Transparent and Reproducible Artificial Data Generation. URL `https://CRAN.R-project.org/package=bdlp`, r package version 0.9-1

Donoho DL (2010) An invitation to reproducible computational research. Biostatistics 11(3):385–388, DOI 10.1093/biostatistics/kxq028

Feinerer I, Hornik K (2015) wordnet: WordNet Interface. URL `http://CRAN.R-project.org/package=wordnet`, r package version 0.1-10.

Fellbaum C (1998) WordNet: An Electronic Lexical Database. Bradford Books, Cambridge

Ioannidis JP, Allison DB, Ball CA, Coulibaly I, Cui X, Culhane AC, Falchi M, Furlanello C, Game L, Jurman G, et al (2009) Repeatability of published microarray gene expression analyses. Nature genetics 41(2):149–155, DOI 10.1038/ng.295N1

Mesirov JP (2010) Accessible reproducible research. Science 327(5964), DOI 10.1126/science.1179653

Nekrutenko A, Taylor J (2012) Next-generation sequencing data interpretation: enhancing reproducibility and accessibility. Nature Reviews Genetics 13(9):667–672, DOI 10.1038/nrg3305

Peng RD (2011) Reproducible research in computational science. Science 334(6060):1226, DOI 10.1126/science.1213847

Peng RD, Dominici F, Zeger SL (2006) Reproducible epidemiologic research. American Journal of Epidemiology 163(9):783–789, DOI 10.1093/aje/kwj093

Prinz F, Schlange T, Asadullah K (2011) Believe it or not: how much can we rely on published data on potential drug targets? Nature Reviews Drug Discovery 10(9):712–712, DOI 10.1038/nrd3439

Steen RG (2011) Retractions in the scientific literature: is the incidence of research fraud increasing? Journal of Medical Ethics 37(4):249–253, DOI 10.1136/jme.2010.040923

Stodden VC (2010) Reproducible research. Computing in Science & Engineering 12(5):8–12, DOI 10.1109/MCSE.2010.113

Wallace M (2007) Jawbone Java WordNet API. URL `http://mfwallace.googlepages.com/jawbone`

Wickham H (2014) Advanced R. CRC Press, Boca Raton