Formal Response of its Authors to the Reviewers Comments on the paper
"The bdpar Package: Big Data Pipelining Architecture for R"
(ID: 2020-47) submitted to The R Journal on 22.02.2020

HANDLING EDITOR'S COMMENTS

GENERAL OPINION. - We are pleased to accept your paper "The bdpar Package: Big Data Pipelining Architecture for R" subject to the major revisions described below. We would appreciate a revised version and point-by-point response to the reviewers comments within 2 months. Remember, that when responding to the reviewer's comments, your job is to persuade me, the editor, that either you've dealt with the issue, or that it's not relevant. To this end, please produce a single document that includes all the reviewers comments mingled with your responses. I particularly recommend the strategy described at http://matt.might.net/articles/peer-review-rebuttals/.

GENERAL RESPONSE: Dear Editor, thank you very much for the attention paid to our work. Following your "major revision" decision, we have carefully studied the comments made by the reviewers to improve the weaknesses reported for both the manuscript and the package. Additionally, we have also revised the whole manuscript taking into consideration some general ideas raised during the revision.

We have restructured the architecture of bdpar to provide the functionalities suggested by REVIEWER #2. Concretely the new version (bdpar v3.0.0) provides: (i) a generic debug function able to show (and define) different personalized messages according to a specific log level (DEBUG, INFO, WARN, ERROR, FATAL), (ii) N-RE functionality responsible for avoiding the execution of previously processed instances and(or) tasks, which guarantees that only new input data and(or) pipelines are executed, (iii) the capability to perform both, serial or parallel operations according to the user needs, and (iv) the ability to visualize the intermediate results achieved by each instance after being processed by the tasks comprising the pipeline (white-box implementation paradigm).

Additionally, we have reformatted and rewrote several sections of the manuscript to (i) improve the readability and understanding of several issues and (ii) include the description of the new functionalities provided in bdpar.

Particularly, the last paragraph of the Abstract section has been changed to include a brief enumeration of the functionalities provided by bdpar package. Additionally, we have restructured and rewrote some paragraphs of Section 1 (Introduction) to clarify the gap that bdpar is trying to address in the R ecosystem. It should be noted that we are not trying to present bdpar as a substitute for other pipeline tools (such as repo or drake), but as an alternative that allows users to choose the best option according to their needs (or knowledge).

Moreover, "Package structure and operation" and "Executing bdpar framework" sections have been modified to include the novelties provided in the last version of bdpar. Moreover, we made several modifications in the "Developing new functionalities" section. Concretely, we (i) change the previous text-processing example for another focused on managing and preprocessing images to highlight the ability of bdpar to support any type of input formats and (ii) rewrote the description of DinamicPipeline to facilitate its understanding.

We added in "Managing bdpar configuration options" section the description of the new configuration parameters included in bdpar. Particularly, we included two new configuration categories (N-RE handler and Parallel settings) responsible for enabling (or disabling) the execution of N-RE functionality and customizing the parallelization capabilities (respectively). Finally, we fixed an error in the example code included in "A case study" section.

Below we detail our point-by-point responses to the questions raised by the reviewer. We hope you find our contributions appropriate and interesting for the general audience of The R Journal.

REVIEWER #1:

GENERAL OPINION. - The authors present a new R package, bdpar, as a new highly customizable pipeline-based framework. The code looks great with lots of assertions, documentation and unit tests. However, before I could recommend this package for publication in the R journal, I would like to understand better what this package bdpar adds to the existing R ecosystem. I would therefore ask the authors to make major revisions to their manuscript to address my following comments.

GENERAL RESPONSE: Thank you very much for both, your general appreciation of our work and your constructive criticism. Regarding your recommendations, please find below the point-to-point responses for each of them and refer to the manuscript to evaluate the modifications carried out.

QUESTION 1. - What does this package add to the current R ecosystem with existing packages such as Ruigi and drake? It is not clear from the text why this package was developed; what it adds that was missing before.

RESPONSE 1: We appreciate your pertinent suggestion to improve the quality of the manuscript. To this end, we rewrote the penultimate paragraph of the "Introduction" section to clarify the contributions of our bpdar framework. However, to the best of our knowledge, there are only a few packages able to preprocess heterogeneous data using the pipeline concept (repo, drake and Ruigi). This fact motivates us to create a new alternative (not a substitute) package providing some issues that other alternatives do not implement and we think they are important such as (i) the use of R6 classes which facilitates their understanding and reduces the learning curve for users coming from O.O. programming environments, (ii) pipeline implementation based on the magrittr package, which allows to decrease development time and improve both readability and maintainability of code, (iii) the capability to parallelize the instances instead of tasks, which reduces processing time and maximizes independence in a pre-processing environment, (iv) the ability to execute only the updated instances, which reduces unnecessary computational costs and (v) the capability to show the intermediate results achieved by each instance after being processed by the tasks comprising the pipeline, which allows users to easily trace the status of each instance.

QUESTION 2.- I do not understand how this package is solving the big-data challenge. I am more familiar with the popular R package drake, which allows for caching intermediate results and for easy parallelization of independent tasks through many different parallel frameworks thanks to R package future. Does R package bdpar provide similar features or something else that helps with the big data problem?

RESPONSE 2: We really appreciate your pertinent observation. Following your suggestion, we included three new functionalities to increase the capabilities of our bdpar package: (i) debug mode, which allows users to trace the intermediate results (by using different log levels) achieved by each instance during the whole process, (ii) dual execution mode, capable of performing operations following a parallel or sequential approach according to both, the user needs and available hardware resources and (iii) only-once execution which avoids the execution of previously processed information (instances) guaranteeing that only new input data (instances) and new pipelines are executed. Please examine the changes described in the manuscript.

Minor comments:

QUESTION 3. - The code could be a bit simpler and easier to read if authors could use markdown in the roxygen2 documentation.

RESPONSE 3: You are right. We have updated the package documentation to make the most of the new advantages provided by the latest version of roxygen2 (v.7.1.1). Concretely this last version included a standardised format for documenting R6 classes. Please refer to the bdpar package to evaluate the new documentation to evaluate the changes.

QUESTION 4. - The object-oriented code with R6 is great but is not something that people are used to in R (at the user end), as an R user, I would prefer to avoid writing these $new().

RESPONSE 4: We appreciate your opinion. However, it is important to take into account that object-oriented is the most widespread programming paradigm. In fact, taking a look at CRAN is easy to realise that the number of packages developed using an OOP paradigm is increasing monthly. Taking this fact into account, we decided to develop bdpar using an object-oriented paradigm to reduce the learning curve from users coming from other programming environments. Additionally, we choose R6 instead of other alternatives (such as S3, S4, RC) motivated by two main issues: (i) R6 is the only implementation able to make the most of the main advantages of OOP such as encapsulation, inheritance or semantic references (ii) the efficient management of computer resources concerning the existing OO alternatives (https://r6.r-lib.org/articles/Performance.html#tests-1). We believe that the design of bdpar using R6 and OOP will facilitate its use regardless of the user's programming skills.

QUESTION 5. - If they could make some generic functions for tasks that they perform many times (e.g. assertions of class).

RESPONSE 5: We understand your point of view. To this end, we include some generic functions in bdpar to generalize some operations that are performed recurrently. Concretely bdpar now provides: (i) a generic debug function able to show different personalized messages according to a specific log level (DEBUG, INFO, WARN, ERROR, FATAL), (ii) a clean-cache method responsible of removing the intermediate results stored as a result of executing the pipelines with the debug mode enabled and (iii) cleanCache() functionality (included in the bdpar.Options) in charge of deleting the information of the preprocessed instances required for the proper operation of the debug mode.

REVIEWER #2:

GENERAL OPINION. - It is clear that the authors have designed the package primarily to process unstructured text data with each observation stored in a separate file. All of the built-in pipeline steps are for text-processing, the motivating examples are text processing, and the future directions include more text processing features. There is nothing wrong with this focus. It makes sense to tailor the tool to an intended use case, and users that need to process a large amount of unstructured text data will find this package very useful. Strangely, nothing about text analysis is mentioned in the Abstract, which means potential users may not continue reading. Big Data is a very general term, but it certainly isn't limited to unstructured text data. I consider credit card transactions, ad clicks, electronic medical records, etc. to be Big Data. And what about images, videos, and audio recordings? Thus I have two recommendations: 1) mention in the Abstract that bdpar is especially useful for processing unstructured text data, and 2) if bdpar can handle non-text data sources, provide some explicit examples somewhere in the manuscript (you don't have to provide working code, just state a hypothetical example like "You could use bdpar to analyze image files, with the pipeline steps including cropping, resizing, and rotating each image").

GENERAL RESPONSE: Thank your deep analysis, criticism, extended revision and general appreciation of our work. This manuscript was focused on text pre-processing since the pipelines, tasks and parsers provided by default in bdpar are centred on text-mining. However, bdpar allows users to design and develop new parsers, tasks and pipelines to process the required information. To clarify the capability of our proposal to process other types of data, we modify the example included in "Developing new functionalities" for another which describes how to manage and preprocess images using bdpar. Moreover, the ability to develop personalized parsers allows creating a flexible and adaptable tool regardless of the input data format.

Regarding your appreciation of Big Data, we agree with you. Big Data does not only refer to the type of data to be processed (structured or unstructured) but also the applications, techniques and technologies used to process such information. Our application is proposed as an alternative application to process large volumes of information. The ability to design data-oriented parsers ensures the compatibility of bdpar regardless of the type of input data. To this end, bdpar can be used to analyse together both unstructured information (such as x-ray images or voice recordings) and structured information (such as HTML code or barcodes). Please refer to the manuscript to evaluate the changes.

QUESTION 1.- There is a popular pipeline management tool for R called drake. It is included in the list of awesome pipelines you cited. Here's an example use case that I stumbled across recently. It would be useful to provide some comparison with the drake package (for one, drake pipelines are composed of functions, as compared to bdpar's object-oriented pipes). A less well known R-based pipeline tool is repo.

RESPONSE 1: We understand your appreciation. We know the existence of drake, and we really think that is a powerful generic-purpose pipelining tool based on GNU Make utility with a quite important developer community (with 38 contributors in GitHub) and users. On the other hand, and following the description included in (Napolitano, 2017) repo is a data-centred framework developed to solve data processing problems focused mainly in the bioinformatics research domain (specific-purpose tool). As you can see, both applications are focused on the use of pipelines, however, the target and the implementation concepts are quite divergent.

Taking this into account we decided to present (and develop) an alternative (not a substitute) multi-purpose framework to preprocess heterogeneous data. Our tool differs from the existing alternatives in several aspects (i) the use of R6 classes which facilitates their understanding and reduces the learning curve for users coming from O.O. programming environments, (ii) pipeline implementation based on the magrittr package, which allows to decrease development time and improve readability and maintainability of code, (iii) the capability to parallelize the instances instead of tasks, which reduces processing time and maximizes independence in a pre-processing environment, (iv) the ability to execute only the updated instances, which reduces unnecessary computational costs (v) the capability to show the intermediate results achieved by each instance after being processed by the tasks comprising the pipeline, which allows users to easily trace the status of each instance.

Reference

Napolitano, F (2017). repo: an R package for data-centred management of bioinformatics pipelines. BMC Bioinformatics. Vol 18:112, pp. 1-9. DOI: 10.1186/s12859-017-1510-6

QUESTION 2.- The installation section could use some clarification. The standard install.packages("bdpar") will install the 7 dependencies listed in Imports. As it is currently written, it makes it seem as though the user may have to check to make sure they are installed (and they don't). If the user needs to install the suggested dependencies, you can recommend they run install.packages("bdpar", dependencies = TRUE).

RESPONSE 2: Thank you for your appropriate suggestion. To this end, we add a new sentence at the end of the "Using bdpar package" section to clarify this issue. Please refer to the manuscript to evaluate the changes made.

QUESTION 3.- In the example extractor ExtractorTytb, I found it odd that obtainDate used the file modification time. This would mean all the YouTube video comments would have to be downloaded in real time (and no historical analysis could ever be performed). I searched through the code, and the modification time is only ever used in that one example. Thus I assume you use it for the sake of simplicity, but it confused me more than anything since importing the data is such a crucial step. After reading the paper, my guess was that maybe the date was automatically extracted from a field in each data file. But looking at some of the built-in examples, it appears that email dates are obtained from the filename and Tweet dates are obtained by querying the API. It would be nice if a more realistic example could be displayed, or at minimum mention some examples of how dates are actually extracted in practice.

RESPONSE 3: You are absolutely right. We decided to include that example for sake of simplicity. However, and taking into account your adequate point of view we reformulated the example in order to epitomize a more realistic scenario. Concretely we designed (i) a parser (called ExtractorImage) able to load images and (ii) two pipes (called ImageCroppingPipe, ImageResizePipe) responsible for cropping and resizing the images respectively.

QUESTION 4.- I was confused by the sentence "To reduce the use of computational resources the pipeline is created at the beginning and keeps unaltered while no changes are performed in the pipeline (by adding or removing pipes)." This is referring to the pipeline definition stored in the variable pipeline in the code above, right? What exactly is the computation being saved? An instance of an R6 object, that is itself a collection of R6 instances, shouldn't be very large nor do I see how it would be computationally intensive to add or remove R6 instances (i.e. pipeline steps). I'm probably missing something, so it would be better to be explicit about the potential computational costs.

RESPONSE 4: We appreciate your pertinent suggestion to improve the quality of our paper. DynamicPipeline functionality allows users to build customized pipelines by adding or removing pipes "on the fly". We wrote this sentence to remark that although the pipeline can be dynamically updated according to user needs, both the performance and resource consumption are the same as the "static" one. However, following your advice, we decided to relax the sentence and reformatted the explanation of DynamicPipeline. Please refer to the "Developing new functionalities section" to evaluate the changes.

QUESTION 5.- Since the unstructured text data sources are already saved in files on the local machine, I was confused why the package needed to handle API keys. After investigating the code, I learned that an example is querying the Twitter API for the date of a Tweet (as mentioned above). If it is worth mentioning the API keys in the manuscript, you should also describe some examples of how they are used. If you decide this isn't that important, then the API keys should be removed from the manuscript entirely.

RESPONSE 5: We appreciate your opinion. Our bdpar tool was designed as a generic-purpose pipeline framework to pre-process mainly large amounts of information. Focused on this idea, we decided to provide in bdpar some functionalities able to create, store and manage multiple configuration options according to user needs (by invoking the bdpar.Options method). The API keys are one of the six configuration parameters included by default in bdpar to ensure the proper operation of the extractors responsible for preprocessing Twitter posts and Youtube comments (ExtractorTwtid y ExtractorYtbid respectively). However, it can be deleted (or even ignored) easily by the users (by calling bdpar.Options$remove() function) if it is not used (e.g. when using an image-preprocessing pipeline). The paragraph over Table 2 describes the functions provided in bdpar.Options to handle all the configuration parameters (e.g. bdpar.Options$set(twitter.consumer.key, "example_key")). Nevertheless, due to privacy and security issues both Youtube and Twitter keys are private (each applicant has its own set of API keys), so we consider that it is not appropriate to give a real example.

QUESTION 6.- For the wordcloud example, why aren't all the steps included as part of the pipeline? Would it be possible to convert the "word frequency" step and "wordcloud" steps into pipeline steps? Or does the pipeline only support 1-1 steps? In other words, can I add a pipeline step that combines all the data sources into a single output? In general, the decision of where to end the pipeline would be useful to explain to the reader, so they can evaluate what parts of their data analysis can be converted to a bdpar pipeline.

RESPONSE 6: Thank you for your pertinent opinion to improve the wordcloud example. Is important to highlight that the main concept of the pipeline is based on a set of interconnected tasks (also called pipes) where the output of one task ($task_{n-1}$) is the input of the next one ($task_n$). So, changing the 1-1 step pipeline to an N-1 step pipeline violates the real concept of a pipeline.

Regarding the example, we decided to add a wordcloud to show the dataset generated by bdpar following a simple and graphical way. However, bdpar was developed as a pre-processing framework and not as a data-analysis application.

Your suggestion can be developed in bdpar by adding a new argument to the runPipeline() function used to indicate which task (or tasks) should be executed at the end of the pipeline process. However, we do not implement your suggestion since we think is opposite to the pipeline philosophy. Nevertheless, we have included below an hypothetical implementation example to illustrate how to manage a N-1 step pipe.

```
wordCloud <- function(instanceList) {

        sms <- instanceList[lapply(output, function(instance) {
                        instance$getSpecificProperty("extension") == "tsms"
                })]
        eml <- instanceList[lapply(output, function(instance) {
                        instance$getSpecificProperty("extension") == "eml"
                })]

        #Return the words and frequencies
        word.frec <- function(data){
                corpus <- VCorpus(VectorSource(data))
                corpus <- tm_map(corpus, removePunctuation)
                corpus <- tm_map(corpus, removeNumbers)
                corpus <- tm_map(corpus, stemDocument)
                m <- as.matrix(TermDocumentMatrix(corpus))
                v <- sort(rowSums(m), decreasing = TRUE)
                d <- data.frame(word = names(v), freq = v)
                return(d)
        }
```

```
        sms.words <- word.frec(lapply(sms, function(instance) instance$getData()))
        eml.words <- word.frec(lapply(eml, function(instance) instance$getData()))
        all.words <- word.frec(lapply(instanceList, function(instance) instance$getData()))

        # Wordcloud for SMS and e-mail
        op <- par(mfrow = c(1, 1), mai = rep(0, 4), mar = rep(0, 4))
        wordcloud(words = sms.words$word, freq = sms.words$freq, min.freq = 1, max.words = 100,
                  random.order = FALSE, rot.per = .5, colors = brewer.pal(8, "Dark2"))
        text(x = 0.52, y = 0.15, labels = "Frecuency", vfont = c("serif","bold"))
        legend(x = 0.23, y = 0.13, legend = c("1-3", 4, 10, 28),
               fill = brewer.pal(8, "Dark2")[c(1, 2, 3, 8)], bty = "n", y.intersp = 0.75, horiz = TRUE)

        wordcloud(words = eml.words$word, freq = eml.words$freq, min.freq = 1, max.words = 100,
                  random.order = FALSE, rot.per = .4, colors = brewer.pal(8, "Dark2"))
        text(x = 0.52, y = 0.15, labels = "Frecuency", vfont = c("serif","bold"))
        legend(x = 0.15, y = 0.13, legend = c("1-13", "13-25", 27, 46, 101),
              fill = brewer.pal(8, "Dark2")[c(1, 2, 3, 8)], bty = "n",
                                  x.intersp = 0.25, y.intersp = 0.75, horiz = TRUE)
        par(op)

        #join wordcloud
        op <- par(mai = rep(0,4), mar = rep(0,4))
        wordcloud(words = all.words$word, freq = all.words$freq, min.freq = 1, max.words = 100,
                  random.order = FALSE, rot.per = .5, colors = brewer.pal(8, "Dark2"))
        text(x = 0.5, y = 0.15, labels = "Frecuency", vfont = c("serif","bold"))
        legend(x = 0.19, y = 0.125, legend = c("1-16", "17-29", 56, 129),
               fill = brewer.pal(8, "Dark2")[c(1, 2, 3, 8)], bty = "n",  y.intersp = 0.75, horiz = TRUE)
        par(op)
}

bdpar::runPipeline(path = system.file(file.path("example"), package = "bdpar"), cache = FALSE, verbose =
FALSE, summary = FALSE, finalTask = wordCloud)
```

QUESTION 7.- The demo code displayed in the manuscript does not work:
bdpar::runPipeline(path = system.file(file.path("examples", "testFiles"), package = "bdpar"))
Error in bdpar_object$execute(path, extractors, pipeline): [Bdpar][execute][Error] Path parameter must be an existing file or directory
Looking through your repository, I tried updating the example, which appeared to work (output omitted below). It created the fille teeCSVPipe.output.csv.
bdpar::runPipeline(path = system.file(file.path("example"), package = "bdpar"))
Then in the function word.frec, the call to rowSums is missing its closing parentheses. *sorted <- sort(rowSums(as.matrix(tm::TermDocumentMatrix(corpus)), decreasing = TRUE)*
At this point I stopped trying to run the demo code. It shouldn't be this difficult to try out the package.

RESPONSE 7: You are absolutely right. Concretely, two errors have been detected and fixed: (i) the location of the example file was incorrect (defined in the path parameter) and (ii) the number of parentheses was unbalanced (lack of a closing parenthesis). Please refer to the updated example to evaluate the changes.

Comments on the package

QUESTION 8.- The package functions should check for the availability of a suggested dependency before attempting to use one of its functions. This is described in the section Suggested packages of Writing R

Extensions. For example, the pipeline step FindEmojiPipe uses functions from the suggested dependencies rtweet and textutils. These should be moved to Imports or first checked with requireNamespace.

RESPONSE 8: We understand your appreciation. However, the way we handle this issue was motivated by a suggestion of one of the CRAN members (when submitting the package). Concretely we received a message notifying that we should change the way we proceed with some packages (such as rtweet): "please move it to Suggests and use if conditionally (see §1.1.3.1 of 'Writing R Extensions') and for bdpar use it conditionally.". According to section $1.1.3.1 we believe that the code follows the CRAN indications: "If the intention is to give an error if the suggested package is not available, simply use e.g. rgl::plot3d.". To this end, if the package is not found in CRAN (due to possible removal) our 'bdpar' package has an error handling with the tryCatch function to capture the error and continue with the remaining instances. Moreover, we included the packages (cld2, knitr, rex, rjson, rmarkdown, rtweet, stringi, stringr, testthat, tuber) into the Suggestion section instead of Imports since they are not involved in the main operation of bpdar. Nevertheless, if you consider appropriate to perform any modification in the way 'bdpar' handles this problem, we would appreciate a more specific clarification to update 'bdpar' according to your suggestions.

QUESTION 9.- I was confused by the Python dependency of >= 2.7. While technically correct, I thought that this meant only Python 2 was supported. I tested and confirmed that Python 3 is also supported. Given the backwards incompatibilities between Python 2 and 3, I am accustomed to seeing version requirements such as "Python 2.7+,3.6+" when both Python 2 and 3 are supported by a package.

RESPONSE 9: You are right. We check the compatibility with other Python versions and we can confirm that our tool is fully compatible with Python 2.7+ and Python 3.6+. We reflect this issue clearly in the manuscript.

Having used various pipeline tools listed in the list of awesome pipelines, I felt like there was some crucial features that appear to be missing:

QUESTION 10.- How can I summarize the results of running a pipeline? Is there a summary method available? When I ran runPipeline(), my console was filled with tons of text including 20 warning messages. There wasn't even a final message saying that the pipeline successfully completed. It would be nice to see a summary that shows which samples passed and which failed (and at what step they failed).

RESPONSE 10: We really appreciate your suggestion to improve the functionality of our proposal. Following your advice, we have improved bdpar to show a summary with the obtained results after executing the pipeline. Concretely we included a new optional argument in the runPipeline() function to indicate if a summary should be shown or not (option by default). Below you can see the output generated when executing the example provided in the "A case study" section with the parameter summary=TRUE

```
bdpar::runPipeline(path = system.file(file.path("example"), package = "bdpar"), summary = TRUE)

[2020-11-19 12:11:14][Bdpar][execute][INFO] The pipeline execution has been finished!
[2020-11-19 12:11:14][Bdpar][summary][INFO] Summary after bdpar execution
        Pipeline executed:
                instance %>|%
                        TargetAssigningPipe$new() %>|%
                        StoreFileExtPipe$new() %>|%
                        GuessDatePipe$new() %>|%
                        File2Pipe$new() %>|%
                        MeasureLengthPipe$new(propertyName = "length_before_cleaning_text") %>|%
                        FindUserNamePipe$new() %>|%
                        FindHashtagPipe$new() %>|%
                        FindUrlPipe$new() %>|%
                        FindEmoticonPipe$new() %>|%
                        FindEmojiPipe$new() %>|%
                        GuessLanguagePipe$new() %>|%
                        ContractionPipe$new() %>|%
```

AbbreviationPipe$new() %>|%
SlangPipe$new() %>|%
ToLowerCasePipe$new() %>|%
InterjectionPipe$new() %>|%
StopWordPipe$new() %>|%
MeasureLengthPipe$new(propertyName = "length_after_cleaning_text") %>|%
TeeCSVPipe$new()

Valid instances: 40
Invalid instances: 0
All the possible properties obtained in the different instances: 15

- *target*
- *extension*
- *length_before_cleaning_text*
- *userName*
- *hashtag*
- *URLs*
- *emoticon*
- *Emojis*
- *language*
- *contractions*
- *abbreviation*
- *langpropname*
- *interjection*
- *stopWord*
- *length_after_cleaning_text*

As you can realise, the result of the summary is very descriptive allowing you to see at a glance (i) the pipes executed in the pipeline, (ii) the number of valid and invalid instances, (iii) a list containing the paths associated with the invalid instance (if exist), and (iv) the properties obtained after executing the whole pipeline. We included in the manuscript a description of the summary method but we decided to execute the example included in "A case study" without the summary option to avoid increasing the number of pages excessively.

QUESTION 11.- How can I view the intermediate results of the pipeline steps? I may need to debug an error. Or maybe I am considering adding an intermediate step, and I want to confirm it is doing what I expect. It seems like many of the outputs in the default pipeline end up as columns in the output CSV. But for example ToLowerCasePipe doesn't. What if there was a bug in that step, and I wanted the result immediately after having run that step? From inspecting the nested list returned by runPipeline(), there are 40 elements (20 of ExtractorEml and 20 of ExtractorSms, corresponding to the 20 emails and 20 text messages used as input). It wasn't clear to me how to extract useful information from these classes. And I couldn't find anything in the two vignettes explaining how to inspect the output of running a pipeline.

RESPONSE 11: We thank all the suggestions to improve bdpar. Particularly, we find this question is connected with the previous one focused on improving the traceability of bdpar. To this end, we also included the debug mode which allows users to visualize the intermediate results (by using different granularity levels) achieved by each instance during the whole process. Finally, we updated the description of the vignettes to improve their readability (such as the use of logger mode, the functionality of instance cache handler and parallelization issues).

QUESTION 12.-One of my main motivations for using a pipeline tool is to reduce unnecessary computation. If I add a new input file, only that file is processed through the pipeline. If I add a new intermediate step, only this new step and intermediate steps are executed. Does bdpar support this? From my experimenting, it appears that all the files are re-executed every single time.

```
file.remove("teeCSVPipe.output.csv")
## [1] TRUE
# Run the pipeline
system.time( x1 <- bdpar::runPipeline(path = system.file(file.path("example"), package = "bdpar")) )
```

user system elapsed
33.28 1.25 49.25
 # Run the pipeline a second time. The output already exists, and the input
 # files haven't been modified.

system.time(bdpar::runPipeline(path = system.file(file.path("example"), package = "bdpar")))
user system elapsed
34.89 0.83 51.31

RESPONSE 12: We agree with your point of view. Thus, in the latest version, bpdar is able to detect changes in both the input file and the pipeline. In fact, now bdpar is re-executed only if the input file is modified. Regarding the pipelines, if a new intermediate step is added (called task), bdpar only re-executes the flow from the new task until the end. The ability to discern which part of the flow should remain unchanged and which one should be executed allows minimizing the use of computational resources. Moreover, bdpar is able to associate the input data, the executed pipelines and the achieved results. This fact prevents overwriting the previous flow when a new pipeline is created and executed and avoids the re-execution of the flow if the pipeline has been previously executed over the input file.

Additionally, the last version of bdpar is capable of performing operations following a parallel or sequential approach according to both the user needs and available hardware resources. Below is included a code snippet describing the execution of bdpar using four different configurations: (i) following a sequential execution, (ii) following a parallel execution using 2-cores, (iii) following a parallel execution using 6 cores and finally (iv) using the cache functionality, which allows avoiding unnecessary computation.

#Example 1. Executing bdpar following a sequential paradigm.

path <- system.file(file.path("example"), package = "bdpar")
bdpar.Options$set("numCores", 1)
system.time(runPipeline(path = path, cache = FALSE))
user system elapsed
56.67 0.53 69.64

#Example 2. Executing bdpar using 2 CPU cores for parallelization.

bdpar.Options$set("numCores", 2)
system.time(runPipeline(path = path, cache = FALSE))
user system elapsed
7.84 1.73 44.42

#Example 3. Executing bdpar using 6 CPU cores for parallelization.

bdpar.Options$set("numCores", 6)
system.time(runPipeline(path = path, cache = FALSE))
user system elapsed
9.21 1.98 25.99

#Example 4. Executing bdpar using cache functionality

#The first time bdpar pre-process all the instances

system.time(runPipeline(path = path, cache = TRUE))
user system elapsed
271.95 8.30 294.68

 # *Time saved due to the non execution of previously computed instances.*
system.time(runPipeline(path = path, cache = TRUE))
user system elapsed
20.68 0.84 32.95