ePort Vignette:
Academic report generation for statistics instructors

Xiaoyue Cheng, Di Cook, Lindsay Rutter, Amy Froelich

`ePort` version 0.0.1 , 2015-11-27

Contents

[1]This LaTeX vignette document is created using the R function `Sweave` on the R package `ePort`. It is automatically downloaded with the package and can be accessed with the R command `vignette("ePort")`.

Summary

The `ePort` package provides tools for course instructors to generate electronic reports regarding student performance. Instructors can produce reports immediately after homework assignment deadlines, and use them to better understand student performance throughout the teaching semester. The goal is to allow instructors to assess and improve upon their teaching approaches in a fast response cycle.

The tools in this package will be especially beneficial for users who supervise large introductory courses. These courses often consist of multiple topics (groups of learning outcomes) that are taught by multiple instructors across multiple sections (groups of students). To accomodate the various ways that student performance can be examined for such courses, the package can generate various reports that can compare within and between topics and sections.

At its simplest, a report can be generated for one topic and one section. This would allow course coordinators to determine how well a particular section performed on a particular topic.

Reports can also be generated for one topic across multiple sections, with output format that allows course coordinators to quickly determine how well and consistently the multiple sections performed on a particular topic. This could be particularly insightful in cases where discrepancies in student performance are discovered between sections, especially if different instructors and/or teaching methods are being used across the sections.

In addition, reports can be generated for one unit (group of topics), either within one section or between multiple sections. This allows coordinators to assess student performance across all the learning outcomes of the combined topics that form the unit, and to quantify the consistency of how students perform across sections.

In general, both short and long versions of reports can be generated. Short versions of reports provide brief summarizations of student performance without regard to individual problems, whereas long versions of reports provide detailed summarizations of student performance for each individual problem in the assignment. Hence, long reports can also be used to confirm the suitability of assigned problems. For instance, in some courses, problems that assess the same learning outcome and are intended to be of equal difficulty levels are grouped into a problem set, and each student is assigned a random subset from this set of problems. However, sometimes there may be an unexpected discrepancy in student performance between problems in a given problem set, indicating an unintended discrepancy in the clearness or difficulty level of the problems to which students are randomly assigned. This package will allow users to efficiently find and fix such issues.

Introduction

Installation

R is a open source software project for statistical computing, and can be freely downloaded from the Comprehensive R Archive Network (CRAN) website. The link to contributed documentation on the CRAN website offers practical resources for an introduction to R , in several languages. After downloading and installing R , the installation of additional packages is straightforward. To install the `ePort` package from R , use the command:

```
> install.packages("ePort")
```

The ePort package should now be successfully installed. Next, to render it accessible to the current R session, simply type:

```
> library(ePort)
```

Help files

To access help pages with example syntax and documentation for the available functions of the ePort package, please type:

```
> help(package="ePort")
```

To access more detailed information about a specific function in the ePort package, use the following help command on that function, such as:

```
> help(mergeSection)
```

The above command will return the help file for the function. Notice that this help file includes freestanding example syntax to illustrate how function commands are executed. This is the case in help files for most functions. The provided example code can be pasted directly into an R session.

Example data

A directory that contains example data is automatically installed with the ePort package. The name of this directory is extdata. Understanding the location, layout, and content of the extdata directory will be necessary to continue with the examples provided in the vignette.

The absolute pathway to the extdata directory on your local computer can be determined by typing the following command into the R console:

```
> system.file("inst/extdata/", package="ePort")
```

* Add in picture with layout of the example data file * Describe necessary format of each type of example data file * Warn that real data should not be added to this example data file. To view this file, simply open it in a Web Brower (Mozilla Firefox, Google Chrome, Microsoft Internet Explorer, Apple Safari).

Generating Reports

Types of reports - Overview

Currently, the `ePort` package offers six report types, depending on what the user is trying to compare and analyze about student performance. The same function `makeReport()` is used to generate each of these six report types; however, the input parameter `reportType` to the function will be different depending on which of the six report types the user wishes to run. Since it is most efficient for the user to hard-code in the `reportType` parameter, below is a reference for the parameter options (in quotes) for each of the six report types:

- One topic for one section - short version ("secTopicShort")
- One topic for one section - long version ("secTopicLong")
- One topic comparing multiple sections - short version ("crossSecTopicShort")
- One topic comparing multiple sections - long version ("crossSecTopicLong")
- One unit (group of topics) for one section ("secUnit")
- One unit (group of topics) comparing multiple sections ("crossSecUnit")

This information can also be obtained by running `help(makeReport)`, and will be demonstrated in the next two sections immediately below.

One topic for one section - short version

We start by demonstrating how to generate the electronic report for one section one topic. This demonstration will use the example input files provided in the previously-described `extdata` directory, and will output the report to the `OutputFiles` subdirectory of the `extdata` directory. If you have not modified anything in the `extdata` directory, then the `OutputFiles` subdirectory should be empty, as we have not generated any example reports yet.

In this demonstration, we will create a report for `Topic 06` and `Section AB`. Like any individual report, we will require three input files: an answer key file, a data file, and a learning outcome file. There should be two example answer key files in the subdirectory `KeyFiles` (`Topic06.Questions.htm` and `Topic03.Questions.htm`) and we will use the `Topic06.Questions.htm` file. Additionally, there should be four example data files in the subdirectory `DataFiles` (`Topic03.AB.csv`, `Topic03.CD.csv`, `Topic06.AB.csv`, and `Topic06.CD.csv`), and we will use the `Topic06.AB.csv` file. Lastly, there should be two example learning outcome files in the subdirectory `LOFiles` (`Topic03.Outcomes.txt` and `Topic06.Outcomes.txt`), and we will use the `Topic06.Outcomes.txt` file.

The block of code we will use to generate our `Topic 06 Section AB` is as follows:

```
> key_htm = system.file("inst/extdata/KeyFiles/Topic06.Questions.htm", package="ePort")
```

```
> refineKey(key_htm)
> keyPath = gsub("htm$","txt",key_htm)
> dataPath = system.file("inst/extdata/DataFiles/Topic06/Topic06.AB.csv", package="ePort")
> rewriteData(dataPath)
> loPath = system.file("inst/extdata/LOFiles/Topic06.Outcomes.txt", package="ePort")
> outPath = system.file("inst/extdata/OutputFiles", package="ePort")
> makeReport(keyFile=keyPath, dataFile=dataPath, loFile=loPath, outFile=outPath)
```

We now briefly explain each step of the process. First, we must save the absolute pathway of this answer key. Here, we save it to a string variable called `key_htm`.

```
> key_htm = system.file("inst/extdata/KeyFiles/Topic06.Questions.htm", package="ePort")
```

Second, we must parse and clean this .htm answer key file, and convert it to plain text format. We do this by calling the `refineKey()` function of `ePort` on the .htm answer key file. By running the line below, we will create the cleaned .txt file:

```
> refineKey(key_htm)
```

By default, the `refineKey()` function will place the cleaned .txt file into the same directory as the original .html file, and with the same name. Hence, if the you navigate to the `extdata` directory and its `KeyFiles` subdirectory, you should now see the new and clean .txt file we just created, `Topic06.Questions.txt`.

Our third step is to save the absolute pathway of this new and cleaned .txt answer key. Below, we save it to a string variable called `keypath`.

```
> keyPath = gsub("htm$","txt",key_htm)
```

Now that we have the absolute path of our cleaned .txt answer key, our fourth step is to define the absolute pathway of our data file. We save this to a variable called `dataPath`.

```
> dataPath = system.file("inst/extdata/DataFiles/Topic06/Topic06.AB.csv", package="ePort")
```

After this, our fifth step is to prime the data file to be compatible with our next steps of generating the reports. We do this by running the `rewriteData()` function of ePort on the data file. This function changes certain non-meaningful character issues that would otherwise cause a problem when running the reports. For more details about the specific process, please run a help command on the function. Below, we rewrite the data file:

```
> rewriteData(dataPath)
```

Our sixth step is to save the absolute path of our learning outcome file. Below, we save this to a variable called `loPath`.

```
> loPath = system.file("inst/extdata/LOFiles/Topic06.Outcomes.txt", package="ePort")
```

After that, the seventh step is to specify our desired output directory. This is the absolute path of where the reports should be saved. Below, we create a variable called `outPath` to specify that we want to output our report to the `OutputFiles` subdirectory.

```
> outPath = system.file("inst/extdata/OutputFiles", package="ePort")
```

Now that we have primed our three input files (cleaned answer key, data file, and learning outcomes file) and specified our output directory, our last step is to generate the report using the `makeReport()` function.

```
> makeReport(keyFile=keyPath, dataFile=dataPath, loFile=loPath, outFile=outPath)
```

Upon running this code, you will receive the following message and menu asking for your input:

> Please enter integer (1-6) corresponding to desired report type below.
>
> Note: If running many reports, it is more efficient to exit now and hard-code the reportType parameter. See help(makeReport).
>
> 1: One topic for one section - short version ("secTopicShort")
> 2: One topic for one section - long version ("secTopicLong")
> 3: One topic comparing multiple sections - short version ("crossSecTopicShort")
> 4: One topic comparing multiple sections - long version ("crossSecTopicLong")
> 5: One unit (group of topics) for one section ("secUnit")
> 6: One unit (group of topics) comparing multiple sections ("crossSecUnit")

Since we wish to generate the type of report that is the short version of one topic for one section, then we can type the number `1` into the menu console. If we do so, then the report will be generated. However, upon seeing the menu option, we can also escape out of the command, and rerun the last line of code, only this time hard-code specifying the `reportType` parameter as `secTopicShort`. This is shown below:

```
> makeReport(keyFile=keyPath, dataFile=dataPath, loFile=loPath, outFile=outPath,
+ reportType = "secTopicShort")
```

The advantage to specifying by hard-coding the `reportType` parameter is especially pertinent for users who wish to run these reports serially for each of many sections. This will be made more clear in a later section of the vignette (Running sequentially on batch of sections).

Whether we specify the type of report we wish to generate by hard-coding or selecting from the menu, we should have successfully generated the report. We can find our report in the `OutputFiles` subdirectory of the `extdata` example directory. Indeed, we see that we have our short report (`Stat101hwkTopic06ABshort.pdf`) of one topic (`Topic 06`) for one section (`Section AB`).

And a .txt file that contains the list of students who have scored below 80 on this assignment Topic06ABstudentsbelow80.txt.

One topic for one section - long version

The short report we created for one section one topic (`Stat101hwkTopic06ABshort.pdf`) may be helpful for users who want to view a brief and overall summarization of student performance. However, we can also generate a more verbose report summarization for one section one topic that would include additional details, such as separate analysis for each problem in the assignment. We have already defined our three input file pathways and one output file pathway. Hence, if we wish to generate this longer version of the report summarization for one section one topic, all we need to rerun is the `makeReport()` function, only this time specifying the `reportType` parameter with the option of "secTopicLong".

```
> makeReport(keyFile=keyPath, dataFile=dataPath, loFile=loPath, outFile=outPath,
+ reportType="secTopicLong")
```

We can find our output report in the `OutputFiles` subdirectory of the `extdata` example directory. Indeed, we see that we now have a much longer report (`Stat101hwkTopic06AB.pdf`) of one topic (`Topic 06`) for one section (`Section AB`).

One topic comparing multiple sections - short version

Now that we have successfully generated reports for both sections of `Topic 06` separately, we may wish to generate a report that compares the performance between these two sections. In this case, we use the same `makeReport()` function as before. However, for our parameter `dataFile`, we will input a list `dataList` that lists the data files for both sections we wish to compare. Of course, we must also correctly specify the `reportType` parameter to indicate that we wish to generate a short report that compares sections for a given topic:

```
> dataFolder = system.file("inst/extdata/DataFiles/Topic06", package="ePort")
> dataList = list.files(path=dataFolder,full.names=TRUE)[1:2]
> makeReport(keyFile=keyPath,dataFile=dataList,loFile=loPath,outFile=outPath,reportType="crossSecTopi
```

One topic comparing multiple sections - long version

We received a brief comparative summary between the sections of Topic 06 in the previous section. If we would like to see a more detailed version of this report that includes comparative information for each question in the topic of interest, then we can simply run the previous line of code, only now specifying the `reportType` parameter to indicate that we wish to generate a long report that compares sections for a given topic:

```
> makeReport(keyFile=keyPath,dataFile=dataList,loFile=loPath,outFile=outPath,reportType="crossSecTopi
```

One unit (group of topics) for one section

```
> dataFolder = system.file("inst/extdata/DataFiles/Topic03_06", package="ePort")
> dataList = list.files(path=dataFolder,full.names=TRUE)
> for (file in dataList){
+   rewrite(file)
+ }
> dataTable = setDir(dataFolder)
> mergedData = mergeData(dataTable)
> # Add this to makeReport if we are using this file!!!!!!!!!!!!!!!!!!
> for (sctn in unique(tabfiles$section)) {
+   merged = subsetData(mgdata,tabfiles,choice=sctn)
+   knit("/Users/lindz/ePort/inst/Rnw/hw-topic.Rnw",output=paste0('Stat101hwk_Unit1_Section',sctn,'.t
+ }
```

One unit (group of topics) comparing multiple sections

```
> merged = subsetData(mgdata,tabfiles)
> knit("/Users/lindz/ePort/inst/Rnw/hw-topic-section.Rnw",output='Stat101hwk_Unit1_allSections.tex')
```

Running reports sequentially

If a course has multiple sections, and we wish to create an individual report for each of the many sections, then one inconvenient way to accomplish this would be to run the example code above, separately for each section at a time, with new data files each time. However, a more convenient way to accomplish the task would be to run all the reports at once. We are still using the same key and learning outcome files, although we would now need two data files (one with the answers from students in `Section AB` and one with the answers from students in `Section CD`).

We can hard code the two data files needed for the two sections into a vector called `dataListPath` as shown below:

```
> dataListPath = c(system.file("inst/extdata/DataFiles/Topic06/Topic06.AB.csv", package="ePort"),
+ system.file("inst/extdata/DataFiles/Topic06/Topic06.CD.csv", package="ePort"))
```

Next, as demonstrated below, we can generate a report for both sections listed in our `dataListPath` object using a for loop. Our two data files of interest in the for loop (`Topic06.AB.csv` and `Topic06.CD.csv`) are from the same Topic, and so they share the same key file and learning outcome file. Hence, we do not have to run the `refineKey()` function on the key file for this Topic, as we have already completed this step earlier. However, our two data files of interest do not share the same data file. We have already executed the `rewriteData()` function for the `Topic06.AB.csv` data file, but we have not yet done so for the `Topic06.CD.csv` file. Hence, we must include the `rewriteData()` function in our for loop to ensure that both data files have this priming step completed.

```
> for (i in dataListPath){
+   rewriteData(i)
```

```
+   makeReport(keyFile=keyPath, dataFile=i, loFile=loPath, outFile=outPath, reportType="secTopicShort
+ }
```

We can also, however, simply

```
> dataFolder = system.file("inst/extdata/DataFiles/Topic06/", package="ePort")
> #namelist = list.files(path=dataFolder, pattern = "^[^.]*\.[^.]*\.[^.]*$", full.names=FALSE)
> #http://stackoverflow.com/questions/9949176/match-string-with-exactly-2-of-a-given-character-e-g-2-
>
> #dataListPath = c(system.file("inst/extdata/DataFiles/Topic06/Topic06.AB.csv", package="ePort"), sy
> #topic = gsub('.Questions.txt','',gsub('Topic','',basename(key)))
> #namelist = list.files(path=dataPath,full.names=TRUE)
> #namelist = namelist[grep(paste('Topic',topic,'\\.',sep=''),basename(namelist))]
> #namelist
>
```

Report Options

Conclusions

The `ePort` package offers various plotting tools that can assist those studying genealogical lineages in the data exploration phases. As each plot comes with its advantages and disadvantages, we recommend for users to explore several of the available visualization tools.

This vignette briefly introduced some of the capabilities of the `ePort` package. Inevitably, new approaches will necessitate new features in subsequent versions and might reveal unforeseen bugs. Please send comments, suggestions, questions, and bug reports to `amyf@iastate.edu`.