Explaining the newAbbrvReport Function

Mark White | markhwhiteii@gmail.com

5/16/2017

I will discuss the `newAbbrvReport` function in this document. The function can be loaded by running the `newAbbrvReport.R` document, and the `newAbbrvReportGuide.pdf` file explains how to operate it. In case people are interested in improving, changing, or otherwise understanding the code, I will explain here line-by-line what the code means. I will first present a segment of code and then explain it below.

The `newAbbrvReport` function

```
newAbbrvReport <- function(channel, month, year) {
```

This first line names the function `newAbbrvReport` and specifies the three different arguments that it will take (see `newAbbrvReportGuide.pdf` for a description of these arguments).

```
  suppressMessages(require(RODBC))
  suppressMessages(require(dplyr))
  suppressMessages(require(stringr))
  suppressMessages(require(stringi))
  suppressMessages(require(tidytext))
```

This chunk loads the necessary packages, while suppressing annoying startup messages that they might tell the user.

```
  year_ch <- str_sub(year,start=-2)
```

The user provides a four digit year (e.g., "2017"); however, we will want to shorten this to just "17" for variable naming purposes. This line only takes the last two characters of the `year` argument. So what was "2017" now is saved as "17" in the object named `year_ch`.

```
  get_ch <- data.frame(
    month=c("01","02","03","04","05","06",
            "07","08","09","10","11","12"),
    ch=c("jan","feb","mar","apr","may",
         "jun","jul","aug","spt","oct",
         "nov","dec"))
  month_ch <- as.character(get_ch[which(get_ch$month %in% month),"ch"])
```

The user also provides a two digit number that represents the month. For variable naming purposes, we want to convert this to a month name. These lines of code create a small data frame where one variable is the month number, and the other variable is a month name. This is a "key" for the function to convert month numbers into month names. Here's what this data frame looks like:

```
kable(data.frame(month=c("01","02","03","04","05","06",
                         "07","08","09","10","11","12"),
                 ch=c("jan","feb","mar","apr","may","jun",
                      "jul","aug","spt","oct","nov","dec")))
```

month	ch
01	jan
02	feb
03	mar
04	apr
05	may
06	jun
07	jul
08	aug
09	spt
10	oct
11	nov
12	dec

The last part of the code above takes whatever row includes the correct `month` number and returns the column that holds the month names. This is saved as a character into the `month_ch` object. This will come in handy in a bit.

```
  newraw <- sqlQuery(channel,
                     query=paste0(
  "SELECT C_BioSense_ID, Admit_Reason_Description, Chief_Complaint_Text
   FROM KS_PR_Processed
   WHERE MONTH(C_Visit_Date_Time) = ",
   month, " AND YEAR(C_Visit_Date_Time) = ", year))
```

This is a SQL query that pulls down the BioSense ID, admit reason, and chief complaint for every record appearing in the month and year specified and saves it into a new data frame called `newraw`.

```
  newraw <- within(newraw,
      list(Admit_Reason_Description <-
str_replace_na(Admit_Reason_Description, ""),
      Chief_Complaint_Text <- str_replace_na(Chief_Complaint_Text, ""),
      textraw <- paste(Admit_Reason_Description, Chief_Complaint_Text),
      text <- str_replace_all(textraw, "[^[a-zA-Z ]]", " ")))
  colnames(newraw) <-
c("id","admit_reason","chief_complaint","text","textraw")
```

This chunk cleans up that new bit of data called `newraw`. All `NA` values in the admit reason and chief complaint fields will be replaced with nothing (i.e., ""), which will help in the next line when we create a new variable, called `textraw`, that concatenates the two fields (if `NA`s

are not replaced, then the character "NA" would have been pasted). The next line takes the textraw variable, removes everything that isn't a letter, and saves it in a new variable called text. Lastly, the column names for this data frame are renamed to be easier to work with.

```
  newword <- unnest_tokens(newraw, word, text)
```

This function takes the text variable and replaces it with a word variable. For every word in the text field, a new row is created that contains just one word. So if the text field read, "pt is in pain", the text variable would be replaced with a word variable, and that case would have four rows: one for "pt", another for "is", another for "in", and another for "pain". This allows us to do analyses for each word. This new data frame is saved into an object called newword.

```
  newword <- filter(group_by(newword, id), !duplicated(word))
```

This line first groups the newword data by id (i.e., the BioSense ID). For each group, the data are then filtered to remove any word that has been duplicated. This trimmed data file is re-assigned to the object newword.

```
  data("stop_words")
  newword <- anti_join(newword, stop_words, by="word")
```

These two lines read in the stop_words data from the tidytext package. The next line then removes any of these stop words from the newword data. Stop words are things we are generally uninterested in: Words like, "of", "the", "and", etc.

```
  newcounts <- count(ungroup(newword), word, sort=TRUE)
```

This line ungroups the data by id so that counts are done acros the whole dataset. We count up words and sort them from most used to least used. This new dataset, which includes a column word and a column n (i.e., how many times it appeared), is saved into the newcounts object.

Now we are ready to start joining these new counts with the old data we had!

```
  oldcounts$new <- factor("not new")
```

The first thing that happens is a new variable named new is created in the oldcounts data. Since none of these words will be considered new (because they all appeared in the old data), every entry for this is saved as a factor, with the level being "not new".

```
  allcounts <- full_join(oldcounts, newcounts, by="word")
```

This code joins the old and new counts by the word column.

```
  colnames(allcounts)[which(names(allcounts)=="n")] <- paste0("n_", month_ch,
year_ch)
```

Merging them left the count column for the current month named `n`. We can change this by looking for which column name is named `"n"`, and then we replace that with `"n_"` and then the month and year characters we extracted at the very beginning of the code.

```
  allcounts$new <- ifelse(is.na(allcounts$new)==TRUE, "new", "not new")
```

The `new` column was not present in the `newcounts` data frame, so all words that were not present in the old data frame that *are* present now (i.e., the new words!) are registered as `NA` values. This code looks for `NA` values in the `allcounts$new` column and replaces them with the word "new"; otherwise, it is kept as being called "not new".

```
  allcounts[is.na(allcounts)] <- 0
```

Any other `NA` values are blank counts. These can simply be replaced with the number zero.

```
  allcounts$length <- stri_length(allcounts$word)
  newabrv <- c(allcounts[which(allcounts$new=="new" &
allcounts$length<4),"word"]$word)
```

We only want to include words that are short (i.e., under four words), as the reports can get *very* long if we don't. The first line here creates a new variable that simply calculates the length of the word. The second line creates a list: It takes data from `allcounts`, but only words that are `new` and where the `length` is under four words. We then extract just what is in the `word` column. This is the list of new abbreviations.

Now how do we get the ID and context to match it?

```
  report <- data_frame()
  for (i in 1:length(newabrv)) {
    report <- bind_rows(report, unique(newword[which(newword$word %in%
newabrv[i]),c("id","word","textraw")]))
  }
```

The first line creates an empty data frame that we will fill with the results of our report. The next three lines are a `for` loop. For every abbreviation in the list of new abbreviations, we go back to the `newword` data and look for the row in which the new abbreviation appears. We return the `id`, `word`, and `textraw` columns from these rows, and we bind them to the `report` data frame. This loops through for every abbreviation.

```
  output <- list(report, subset(allcounts, select=-c(new,length)))
  names(output) <- c("report", "allcounts")
  return(output)
}
```

Here is where we organize and return the output. The first line here takes the `report` and `allcounts` data (removing the `new` and `length` columns), and it puts them into a list named `output`. The next line names these two elements of the `output`: `"report"` and `"allcounts"`. Next, we `return` the output, and the closed curly bracket indicates that we are done specifying our function!