Tracking New Diagnosis Codes (newDxCodes)

Mark White | markhwhiteii@gmail.com

5/18/2017

This function will generate a report of diagnosis codes that have not been used since before 2016. The fucntion works on a monthly basis: You enter in the most recent month, and it returns what diagnosis codes we have not seen since before 2016. It is very similar to the `newAbbrvReport` function, so I recommend reading that guide and function explainer first. In addition to returning new diagnosis codes, this function will also return a new dataset that can be used as the comparison dataset for the *next* months check.

To use this function, you will need to first run the code in the `newDxCodes.R` document. (*Note*: For those interested in editing the code, I discuss it line-by-line in the `newDxCodesExplainer.pdf` document).

Preparing to use the `newDxCodes` function

First, you will have to load in the old data that to which you will be comparing the new month's data. This is located in the same file in which the `newDxCodes.R` document is stored, and this data file is named `codeCounts.csv`. When you load this file, depending on how you import the data, it may save the `code` variable as a factor; if this occurs, the function will coerce this variable to a `character` string at a certain point. If you get this error message, don't worry.

After this, you will have to set up a connection using the `odbcConnect` function in the `RODBC` package. Load this package using the `library(RODBC)` command. An example of this code looks like:

```
channel <- odbcConnect("Biosense_Platform", "BIOSENSE\\username", "password")
```

Lastly, you will have to install the `dplyr`, `tidytext`, `stringr`, and `stringi` packages before using these functions, since the functions discussed here depend on those packages (as well as the `RODBC` package). After you have this set up, you are ready to run the functions.

The `newDxCodes` function

This function will return two data frames: first, a report of new codes used in the specified month; second, an updated version of the `codeCounts` data file that now contains the specified month's codes and counts.

This function has four arguments:
1. channel = the channel you set up through `odbcConnect`.
2. month = the month you want the report for (i.e., `"01"`, `"02"`, ..., `"12"`).
3. year = the year you want the report for (e.g., `"2017"`, `"2018"`).

4. oldcounts = the `codeCounts` data that you will be using as a comparison dataset for this month's data.

Note: The arguments for `month` and `year` must be put between quotation marks.

Example

Today is May 18th, 2017. Let's say that I wanted to generate a report for this month so far. I set up the first `codeCounts.csv` file to go from January of 2016 up through April 2017, so this month is the first month a report can be generated. To generate this report, I would first set up my `channel` using the `odbcConnect()` code above, and then I would need to load in the `codeCounts.csv` data (again, this file is stored in the same place as this document):

```
codeCounts <- read.csv("~/mhwii/data/codeCounts.csv", stringsAsFactors=FALSE)
```

We can then look at the first few rows and columns of the data:

code	n_jan16	n_feb16	n_mar16	n_apr16	n_may16
r109	32	41	146	1344	1794
i10	31	24	212	3235	4268
j069	26	12	52	464	576
r51	22	19	85	1001	1291
f17210	19	3	56	1083	1220
n390	19	15	93	1144	1503

We can see counts for six codes over five different months. For example, there were 1,144 uses of the code "n390" in April of 2016. It is very important to note at this point that *all counts generated only count ONE use of the code per patient visit at a facility.* If multiple messages were sent contaning the same text for a patient visit, all code utilized in this document counts each of the codes *once*; similarly, if a message used a code twice in it, the code would only be counted *once*.

Now that we (a) have set up a channel and (b) loaded in the data, then we are ready to generate a new report. The code is simple:

```
may <- newDxCodes(channel=channel, month="05", year="2017",
oldcounts=codeCounts)
```

`may` returns a list of two data frames: one called `report` and another called `allcounts`. First, let's take a look at what a report looks like. This can be done by typing in `may$report`. Let's see what the beginning of this report looks like:

C_BioSense_ID	code
2017.05.11.3823_058427	h15109
2017.05.16.3866_583912	h15109
2017.05.05.3793_E510390	m65839

2017.05.15.3793_E86326	m65839
2017.05.06.3868_5292	r93429
2017.05.10.3893_000271222	r93429

The first column reports the BioSense ID, the second column reports the new code that was used. To export this report, one executes the code:

```
write.csv(may$report, "report_may17.csv", row.names=FALSE)
```

`may` also contains a data frame called `allcounts`. This is an updated version of the `codeCounts` file you loaded at the beginning of the script. Let's compare the column names for `codeCounts` and `may$allcounts`:

```
names(codeCounts)

##  [1] "code"    "n_jan16" "n_feb16" "n_mar16" "n_apr16" "n_may16" "n_jun16"
##  [8] "n_jul16" "n_aug16" "n_spt16" "n_oct16" "n_nov16" "n_dec16" "n_jan17"
## [15] "n_feb17" "n_mar17" "n_apr17"

names(may$allcounts)

##  [1] "code"    "n_jan16" "n_feb16" "n_mar16" "n_apr16" "n_may16" "n_jun16"
##  [8] "n_jul16" "n_aug16" "n_spt16" "n_oct16" "n_nov16" "n_dec16" "n_jan17"
## [15] "n_feb17" "n_mar17" "n_apr17" "n_may17"
```

We can see that they are exactly the same, except that `may$allcounts` now includes a column for the number of times a code was used in May of 2017. This file can be downloaded as a .csv and replace the `codeCounts.csv` file that resides there now. Next month, this new file will become the file one uploads for the `oldcounts` arguments to compare for new words used in *June*. Then the report for June will generate an updated file, which can be saved as the new `codeCounts.csv` for July, etc. To save the new `codeCounts.csv` file, one executes the code:

```
write.csv(may$allcounts, "codeCounts.csv", row.names=FALSE)
```