A Brief Primer on Software Tools

authored by Thomas Scofield

This document gives an example of what a data processing session might look like using the revised tools found in `loadTransactions2.R` and `processDat2.R`. It is assumed that the data file `tbdatUpdated.csv` is in the working directory, and has a particular format matching the data files which have been supplied by TBD Solutions, Inc. for data analysis related to this project before now. This format includes patient claim data for which three different levels of specifity, called `CCS.Level.1`, `CCS.Level.2` and `CCS.Level.3`.

The present mode of analysis relies on the data in `CCS.Level.3` as the primary indicator of patient conditions, which is natural as it provides the greatest specificity. While some steps have been taken in the **R** code to make it easy to switch to an analysis *based* on `CCS.Level.2`, this does not happen and will not without some further changes to code not explained here.

There are several settings the user can tweak which appear in the first few lines of code in `processDat2.R`. The main purpose of this document is to explain why one tweaks these settings, and what changes are appropriate. As a first run, we use the scripts with the current settings, sourcing them:

```
source("loadTransactions2.R")
source("processDat2.R")


parameter specification:
 tidLists support minlen maxlen            target   ext
    FALSE    0.03      2     20 frequent itemsets FALSE

algorithmic control:
 sparse sort verbose
      7   -2    TRUE

eclat - find frequent item sets with the eclat algorithm
version 2.6 (2004.08.16)         (c) 2002-2004   Christian Borgelt
create itemset ...
set transactions ...[260 item(s), 23444 transaction(s)] done [0.02s].
sorting and recoding items ... [42 item(s)] done [0.00s].
creating sparse bit matrix ... [42 row(s), 23444 column(s)] done [0.00s].
writing  ... [61 set(s)] done [0.02s].
Creating S4 object  ... done [0.00s].
```

This may result in a non-trivial amount of processing time; with current settings, perhaps 10 minutes or more. when processing is complete, new data structures/variables will exist.

The main variable containing data of interest is called `freqItemsets`. It is a data structure, in **R** called a *data frame*, having two variables: `items` and `support`.

```
names(freqItemsets)

[1] "items"   "support"
```

If we look at the first frequent itemset in this data frame, it may not be as informative as one expects.

```
freqItemsets[1, ]

     items    support
47 {70,99} 0.07148951
```

The value of `items`, rather than *giving* the names of morbidity conditions that occur frequently together

in the dataset, *point to* those conditions, which are stored separately in the list (another data frame, to be technical) called `ccs3`; the main avenue provided in this software to seeing the details of these items is via the `elaborateItemset()` function:

```
elaborateItemset(1)  # arg '1'' indicates 1st frequent itemset is desired

               lev3.desc lev3.idx lev2.idx lev1.idx
1   DEPRESSIVE DISORDERS       70       65       15
2 ESSENTIAL HYPERTENSION       99       54        5
```

Moving on to the `support` column of `freqItemset`, this indicates what portion of patients included in the dataset have conditions found in this particular itemset. If you view a larger portion of the list of `freqItemsets`

```
head(freqItemsets)

      items    support
47  {70,99} 0.07148951
50 {237,99} 0.06257465
48  {16,99} 0.06048456
49  {10,99} 0.05131377
44  {72,99} 0.04845589
7  {139,70} 0.04670705
```

you note that they have been ordered by descending value in the `support` column. This brings us to one of the settings a user may wish to tweak. The user can specify a lower *bound* for the supports of itemsets designated as *frequent*. One generally doesn't want to call a comorbidity condition "frequent" if there is only one patient in the entire dataset who *has* it. If we consider comorbidity conditions *frequent* when, say, they appear in at least 3% of patients, we can include a line at the start of `processDat2.R` which says

```
supportThreshold = 0.03
```

Tweaking this value to be higher will result in fewer itemsets designated as "frequent", decreasing processing time; setting it lower produces more, increasing processing time.

Continuing with user settings, there are two other lines at the start of `processDat2.R` which similarly affect the search for frequent itemsets.

```
itemsetMinLength = 2
itemsetMaxLength = 20
```

The problem of focus in this dataset is to find *comorbidity* conditions; we consider them interesting because more than one morbidity condition is being experienced by patients who might benefit from treatment which takes these multiple conditions into account. Thus, an itemset with just one condition should not be flagged for us. We may even decide that having just two morbidity conditions in an itemset is too few to be of interest. On the other hand, patients with the same 20 morbidity conditions may occur regularly, but we may not feel the research into treatment of this many conditions at once warrants including them in our search. The lines above are used to set upper and lower bounds on the number of morbidity conditions an itemset that is flagged by the software can contain.

The last user setting, found, again, at the start of `processDat2.R`, is

```
ccsLevelToCross = 1
```

Valid settings are 0, 1 or 2. The idea here is that, while many patients may have a combination of morbidity conditions such as

PULMONARY ARTERY ANOMALIES

CONGENITAL INSUFFICIENCY OF AORTIC VALVE
ENDOCARDIAL CUSHION DEFECTS

but, as these are all conditions of the heart and may not call for the same scrutiny as other comorbidity conditions, as a (heart) medical specialist may already have sufficient expertise to consider them together. Our dataset provides indicators of this, as these three conditions all have the same `CCS.Level.2` value (and, necessarily, the same `CCS.Level.1` value). Assuming we want the software to flag itemsets only when they include morbidity conditions with differing `CCS.Level.1` values, we will set the value of `ccsLevelToCross` as above. If, instead, we wish to relax things so that an itemset can be flagged if it contains morbidity conditions of differing `CCS.Level.2` values, we will set `ccsLevelToCross` to 2; this expresses a greater openness to flagging frequent itemsets than when it is set to 1. The greatest openness occurs when the value is set to 0 for, in this case, an itemset can be *frequent* even if all its morbidity conditions come from the same CCS Level 1 and CCS Level 2 specifications.

Along with `elaborateItemset()`, two more support functions have been included in the software to help one explore the results. These are `itemsetPatientIDs()` and `patientItemsets()`. The former is used to list those patients, by patient id, who are part of the *support* of a particular frequent itemset. That is, the list produced by the command

```
itemsetPatientIDs(1)
```

is the ids of the patients in the dataset whose morbidity conditions include those of the first frequent itemset `freqItemset[1,]`. On the other hand, one may wish to know, for a particular patient, which frequent itemsets she belongs to, if any. This is where the other support function comes in. We simply need a way to specify which patient. If, say, the desired patient has id 61238, then we type

```
patientItemsets(id = 61238)

[1] "No patient with this id fits any frequent itemset."
NULL
```

It turns out this function allows us to specify a patient by an alternate method, by `index`. The patients from the dataset are in a list called `pIDs`; here we list the first few:

```
head(pIDs)

[1]  24951 106355 360023 377423 382450 383046
```

The third one down has id 360023. We can use either that patient's id, or his number in the list (3), when we call the above function—that is, both of these lines produce the same output:

```
patientItemsets(id = 360023)
patientItemsets(index = 3)
```