Assignment 7

edelsonc

September 19, 2016

This assignment will look at the same data set as assignment 6. However, in the arff file from assignment 6 will be converted into a csv. Following this, the quality of the data will be assessed, with emphasis on the subset of blood pressure, blood urea, creatinine and potassium.

Converting arff to csv

Arff files have a fairly rigid format. Because of this, small error in the formatting can create large problems when attempting to import data. So although arff can be viewed as more "sophisticated", they are not quite as robust as a csv. Therefore, often times it is easier to first convert the arff to a csv.

In order to do this, a small bash executable was created, aptly named `arff_to_csv`. The code is below and can also be found at https://github.com/edelsonc/arff_to_csv.

```
#!/usr/local/bin/bash
# the above shebang should be the one appropriate for your bash
#created: 09/13/2016
# this script will transform an arff file ($1) into a csv file ($2)

# this finds the match for attributes and writes the header for the csv
header=$( grep "@attribute" "$1" | awk '{print $2","}' | sed "s/['\"]//g" | sed '$s/,$//' )
echo $header > "$2"

# since the @data tag marks the beginning of the data block, it can be used
# to copy over the data, which is all the lines after it.
line=$( grep -n '@data' "$1" | cut -d':' -f1 )
total_lines=$( wc -l "$1" | awk '{ print $1 }' )
want_line=$(( total_lines - line ))

# use tail to look at the end file numbers and append to our new csv
# these results are piped to sed, where lines beginning with % are deleted
cat "$1" | tail -"$want_line" | sed '/^%/d' | sed 's/\(%.*$\)//' >> "$2"
```

The above script initially looks at each line of the input arff (`$1`) and selects those containing `@attribute`, and then prints the second field from then with `awk`, added a comma to the end of each entry. `sed` then goes through these lines, removing single or double quotes. Finally, the comma at the end of the last entry is removed before the `header` is written to the csv file ($2).

`grep` is once again employed to find the line where `@data` occurs. All of the data follows after this tag, and spans to the end of the document. So the lines that need to be copied are selected by finding the number of lines which remain after the `@data` tag, and then selecting those using `tail`. Finally, `sed` is used to remove comments and the data is appended to the csv.

The above script can then be used to convert to a csv as follows

```
# make the script executable
chmod 755 arff_to_csv
```

```
# convert using the new executable
./arff_to_csv chronic_kidney_disease_full.arff ckdf.csv
```

Reading the csv and Creating a Datafram

This part is fairly simple, and has been covered in previous assignments

```
# create dataframe by reading in csv
kidney_all <- read.csv("ckdf.csv")
str(kidney_all)
```

```
## 'data.frame':    401 obs. of  25 variables:
##  $ age  : Factor w/ 78 levels "?","11","12",..: 38 62 54 38 42 52 60 13 43 44 ...
##  $ bp   : Factor w/ 12 levels "","?","100","110",..: 11 8 11 10 11 12 10 2 3 12 ...
##  $ sg   : Factor w/ 7 levels "","?","1.005",..: 6 6 4 3 4 5 4 5 5 6 ...
##  $ al   : Factor w/ 8 levels "","?","0","1",..: 4 7 5 7 5 6 3 5 6 5 ...
##  $ su   : Factor w/ 8 levels "","?","0","1",..: 3 3 6 3 3 3 3 7 3 3 ...
##  $ rbc  : Factor w/ 4 levels "","?","abnormal",..: 2 2 4 4 4 2 2 4 4 3 ...
##  $ pc   : Factor w/ 4 levels "","?","abnormal",..: 4 4 4 3 4 2 4 3 3 3 ...
##  $ pcc  : Factor w/ 4 levels "","?","notpresent",..: 3 3 3 4 3 3 3 3 4 4 ...
##  $ ba   : Factor w/ 4 levels "","?","notpresent",..: 3 3 3 3 3 3 3 3 3 3 ...
##  $ bgr  : Factor w/ 148 levels "","?","100","101",..: 24 2 117 20 9 124 3 115 38 123 ...
##  $ bu   : Factor w/ 120 levels "","?","1.5","10",..: 66 35 84 87 54 53 85 60 90 7 ...
##  $ sc   : Factor w/ 88 levels "","?","0.4","0.5",..: 12 7 18 58 14 11 51 11 19 79 ...
##  $ sod  : Factor w/ 36 levels "","?","104","111",..: 2 2 2 4 2 28 3 2 2 6 ...
##  $ pot  : Factor w/ 44 levels "","?","2.5","2.7",..: 2 2 2 3 2 8 18 2 2 13 ...
##  $ hemo : Factor w/ 120 levels "","?","10.0",..: 60 16 117 15 19 26 28 28 11 116 ...
##  $ pcv  : Factor w/ 46 levels "","\t?","\t43",..: 35 29 22 23 26 30 27 35 24 20 ...
##  $ wbcc : Factor w/ 94 levels "","\t?","\t6200",..: 75 59 73 65 71 75 5 67 91 21 ...
##  $ rbcc : Factor w/ 51 levels "","\t?","?","2.1",..: 37 3 3 22 30 28 3 34 24 20 ...
##  $ htn  : Factor w/ 4 levels "","?","no","yes": 4 3 3 4 3 4 3 3 4 4 ...
##  $ dm   : Factor w/ 7 levels "","\tno","\tyes",..: 7 6 7 6 6 7 6 7 7 7 ...
##  $ cad  : Factor w/ 5 levels "","\tno","?",..: 4 4 4 4 4 4 4 4 4 4 ...
##  $ appet: Factor w/ 5 levels "","?","good",..: 3 3 5 5 3 3 3 3 3 5 ...
##  $ pe   : Factor w/ 5 levels "","?","good",..: 4 4 4 5 4 5 4 5 4 4 ...
##  $ ane  : Factor w/ 4 levels "","?","no","yes": 3 3 4 4 3 3 3 3 4 4 ...
##  $ class: Factor w/ 5 levels "","ckd","ckd\t",..: 2 2 2 2 2 2 2 2 2 2 ...
```

```
# subset for blood pressure, blood urea, creatinine, and potassium
sub_kidney <- kidney_all[c(2,11,13,14)]
str(sub_kidney)
```

```
## 'data.frame':    401 obs. of  4 variables:
##  $ bp : Factor w/ 12 levels "","?","100","110",..: 11 8 11 10 11 12 10 2 3 12 ...
##  $ bu : Factor w/ 120 levels "","?","1.5","10",..: 66 35 84 87 54 53 85 60 90 7 ...
##  $ sod: Factor w/ 36 levels "","?","104","111",..: 2 2 2 4 2 28 3 2 2 6 ...
##  $ pot: Factor w/ 44 levels "","?","2.5","2.7",..: 2 2 2 3 2 8 18 2 2 13 ...
```

Assessing Quality

Since the data is numeric, first we'll look find the number of missing values, which are denoted with a `?`.

```
# replace the ? with NA
sub_kidney[sub_kidney == "?"] <- NA

# count instances of NA
number_missing <- sum(is.na(sub_kidney))
number_missing
```

```
## [1] 206
```

So there are 206 missing values in our data. This means that 12.84% of the data is missing. Furthermore, there are 299 complete rows, 75% of the total.

Currently all four columns are factors. In order to gauge the believability of the remaining data, it would help to coerce them into numeric types and look at some summary statistics.

```
# use the apply function to turn each column of the dataframe into numeric type
sub_kidney <- data.frame(apply(sub_kidney, 2, as.numeric))
summary(sub_kidney)
```

```
##        bp               bu              sod             pot
##  Min.   : 50.00   Min.   :  1.50   Min.   :  4.5   Min.   : 2.500
##  1st Qu.: 70.00   1st Qu.: 27.00   1st Qu.:135.0   1st Qu.: 3.800
##  Median : 80.00   Median : 42.00   Median :138.0   Median : 4.400
##  Mean   : 76.47   Mean   : 57.43   Mean   :137.5   Mean   : 4.627
##  3rd Qu.: 80.00   3rd Qu.: 66.00   3rd Qu.:142.0   3rd Qu.: 4.900
##  Max.   :180.00   Max.   :391.00   Max.   :163.0   Max.   :47.000
##  NA's   :13       NA's   :20       NA's   :88      NA's   :89
```

Now we can see that potassium and creatinine have more `NA`s than blood pressure of blood urea. Additionally, the range for blood pressure is realistic. However, the other columns seem odd. For instantce, a hight of 47 for potassium, where all the other values are tightly clustered around 4, seems odd, especially since there is only one other value that high. This might suggest an outlier, or a missing decimal place between the 4 and the 7.

For creatinine, the values are all very close except the min of 4.5. We can easily check to see if there are other value below 100

```
sod_vec <- sub_kidney$sod[complete.cases(sub_kidney$sod)]
sod_vec[sod_vec < 100]
```

```
## [1] 4.5
```

So there is only one value less than 100, and that is our minimum of 4.5. Doing the same for blood urea, with a cut off of 15 and an upper cut off of 80, we find

```
bu_vec <- sub_kidney$bu[complete.cases(sub_kidney$bu)]
sort(bu_vec[bu_vec <= 15])
```

```
##  [1]  1.5 10.0 10.0 15.0 15.0 15.0 15.0 15.0 15.0 15.0 15.0 15.0
```

```r
sort(bu_vec[bu_vec > 200])
```

```
##  [1] 202 208 215 217 219 223 235 241 309 322 391
```

Showing that both the minimum and the maximim are extreme values conpared to the rest of the range, with the minimum more so. However, as compared with potassium, its far more believable that there values are outliers rather than insert problems.

In summary, the overal quality of the data is mixed. With a fair number of missing fields and a number of suspect entries, it is hard to trust this data completely. However, overall it appears to be intact, and with only 3 or 4 suspect values out of 1604, it is probably good enough to use for most basic analyses.