Problem Set 2

Alexander Brandt
SID: 24092167

September 19 2015

1 An Analysis of US Census Data

1.1

To create a subset of the file, we first determine the file's number of lines (here given by a call to the external unix command "wc" using the -l flag). The file is then opened via bzip2 stream, where it is read in 100,000 line segments. These are subsampled based on 10,000 entries from the range [1, lines], sampled without replacement.

```
# Our function declaration that will make below steps much simpler
# to call
readcenus <- function(bz2filename, lines, CHUNKSIZE, SAMPLE_SIZE,
                      use_readcsv, debug) {
# Open the file from a bzip2 stream using our file name argument
bz2file <- bzfile(bz2filename,"r")
# This ensures our sampling will be consistent every time.  Which
# is important for computationally reproduceable results
set.seed(0)
# Our selection, which needs to be sorted to work with our for
# loop, is generated from the uniformly on [1, lines]
# (but without replacement).
selection <- sort(sample.int(lines, size=SAMPLE_SIZE,
                             replace=FALSE, prob=NULL))
# Alternating methods will be used for timing purposes later, which
# is why we include this boolean option.  Here we read the header
# which will inform the column names for our data frame.
if (use_readcsv) {
  header=readLines(bz2file,n=1)
}
else {
  header=c(unlist(strsplit(scan(bz2file,what="character"
                                ,nlines=1),",")))
```

```
}
# Initialize the empty array, which will allow us to avoid using
# the append function later
to_store <- matrix("NA", nrow=SAMPLE_SIZE, ncol=length(header))
counter <- 1
# Count the number of blocks by dividing lines by the block size,
# and then rounding up
print(system.time(for (i in seq(ceiling(lines/CHUNKSIZE))) {
  start <- ((i-1)*CHUNKSIZE)+1;
  end <- start+CHUNKSIZE-1;
  if (use_readcsv) {
    x <- as.matrix(read.csv(bz2file,nrows=CHUNKSIZE));
  }
  else {
    x <- readLines(bz2file,n=CHUNKSIZE)
  }
  head(x,n=2)
  # Select our subset, and then ensure the numbers are
  # appropriately scaled back so we can collect in the
  # block.
  subset =
  (start:end)[start:end %in% selection] - (i-1)*CHUNKSIZE;
  for (n in subset) {
    if (use_readcsv) {
      to_store[counter,] <- x[n]
    }
    else {
      to_store[counter,] <-
        c(unlist(strsplit(x[n], "," ,fixed=TRUE)))
    }
    counter <- counter+1
  }
  # This debug feature is very helpful for examinging the
  # time each "step" or block reading takes.  It will allow
  # us to debug quickly, and with the use_readcsv flag,
  # compare various R functions.
  if (debug)
  {
    if (i==5){break}

  }
}))
  # Finally store the information as a data frame,
  # populate the headers of the file, and then close the
  # bz2file stream before returning the finished data frame.
```

```
  to_store_df = data.frame(to_store)
  colnames(to_store_df) <- c(header)
  close(bz2file)
  return(to_store_df)
}
```

We invoke the function with a few short lines of code:

```
bz2filename <- "ss13hus.csv.bz2"
mylines <- as.numeric(system(sprintf("bzcat %s |
wc -l", bz2filename),intern=TRUE))-1
# Used for testing: mylines <-7219001-1
MYCHUNKSIZE <- 100000
MYSAMPLE_SIZE <- 10000
to_store_df <-
  readcenus(bz2file=bz2filename, lines=mylines,
            CHUNKSIZE=MYCHUNKSIZE, SAMPLE_SIZE=MYSAMPLE_SIZE,
            use_readcsv=FALSE, debug=FALSE)

##     user   system  elapsed
## 1369.373    4.573 1375.229

# Subset with fields of interest
myvars <-
  c("ST","NP","BDSP","BLD","RMSP", "TEN", "FINCP",
    "FPARC", "HHL", "NOC", "MV", "VEH", "YBL")
write.csv(to_store_df[,myvars],file="cenus_selection.csv")
```

1.2

We now use alternating use_readcsv flags, and activating the debug flag (which allows us to just go through one iteration of the 100,000 read process), we see that indeed readLines() is a much faster function than read.csv(). I investigated using "skip=" to speed up the file, but this actually adds a significant amount of time to the routine.

```
readcsv_df <-
  readcenus(bz2file=bz2filename, lines=mylines,
            CHUNKSIZE=MYCHUNKSIZE, SAMPLE_SIZE=MYSAMPLE_SIZE,
            use_readcsv=TRUE, debug=TRUE)

##    user  system elapsed
## 131.785   2.445 134.350

readcsv_df <-
```

```
  readcenus(bz2file=bz2filename, lines=mylines,
            CHUNKSIZE=MYCHUNKSIZE, SAMPLE_SIZE=MYSAMPLE_SIZE,
            use_readcsv=FALSE, debug=TRUE)

##    user  system elapsed
##  93.914   0.473  94.483
```

1.3

The best way to preprocess the file, in my opinion, is to select only the columns we are interested in for analysis, as defined by the problem statement. We create a simple bash script to loop through strings of the column names, then use a precisely bounded grep to find the line numbers of these strings, where the line numbers now correspond to the column numbers (1-indexed) in our original .csv.bz2 file. These columns are selected with a simple cut command from a bzcat input, and then the stdout stream is directed back into a .bz2 compression. This preprocessing reduces the file from the original 564M to a far more reasonable 26M.

Listing 1: preprocess.sh

```
# Extract the header so we can find our columns of interest
bzcat ss13hus.csv.bz2 | head -n 1 > ss13hus.header.txt

# We will use the file line coordinates as the proxy for index columns
sed -e $'s/,/\\\n/g' ss13hus.header.txt > ss13hus.header.nsv

# Our desired headers
for i in "ST" "NP" "BDSP" "BLD" "RMSP" "TEN" "FINCP" \
    "FPARC" "HHL" "NOC" "MV" "VEH" "YBL"
do
    x=`grep -n ^$i$ ss13hus.header.nsv | cut -d':' -f 1`
    v="$v $x"
done
# Now $v contains our columns of interest, which we just need
# to separate by commas to use with cut.  A sed command will
# accomplish this with ease.
bzcat ss13hus.csv.bz2 | \
    cut -d, -f`echo $v | \
    sed 's/ /,/g'` | \
    bzip2 > preprocessed.csv.bz2
```

1.4

Finally, we propose a simple test of our data based on the definitions of each column in the census. Specifically, how does the region of the united states that

one lives in correlate with the languages spoken in the home? Note to Harold: Chris OK'd different fields than the ones chosen for our subset file. I would prefer to use common SNPs, but something tells me that isn't in the cards quite yet.

```
# Construct a table based on our subsampling routine, looking at the
# region of the country with respect to the languages spoken in the home
con_table<-table(to_store_df$DIVISION,to_store_df$HHL)
# We populate the row/column names based on the .pdf explaining the
# various codes and their meanings for each country
colnames(con_table) <-
c("NA", "English Only", "Spanish",
  "Other Indo-Euro",
  "Asian/Pac", "Other")
rownames(con_table) <-
c("New England", "Mid Atlantic", "E N Central",
  "W N Central", "S Atlantic", "E S Central",
  "W S Central", "Mountain", "Pacific")
# Print the table, and a summary of the statistical significances
con_table
```

```
##
##                  NA English Only Spanish Other Indo-Euro Asian/Pac Other
##   New England   122          314      20              46         5     1
##   Mid Atlantic  222          868      93              73        34     2
##   E N Central   241         1161      71              45        26     7
##   W N Central   124          503      23              12         7     4
##   S Atlantic    363         1374     126              63        36    10
##   E S Central   105          525      17              10         3     4
##   W S Central   204          732     167              28        21     6
##   Mountain      115          483      71              20        18     8
##   Pacific       219          809     229              70       126    14
```

```
summary(con_table)
```

```
## Number of cases in table: 10000
## Number of factors: 2
## Test for independence of all factors:
##  Chisq = 662.2, df = 40, p-value = 1.046e-113
##  Chi-squared approximation may be incorrect
```

Based on our low p-value, we cannot reject the null hypothesis that region has no affect on language (as we might expect).