Analyzing Big (Survey) Data with Limited Computing Resources in R: A Case Study

Anthony Damico, MHS and Rachel Licata, MPH

Kaiser Family Foundation

Background

Since its inception, working with R has proven to be a challenge for researchers using large data sets. Like the statistical packages Stata and SPSS, R natively pulls files directly into Random Access Memory (RAM), effectively capping the workable file size by the available memory of a user's workstation. Record counts for databases used in genomics, medical claims, and even population research can easily number in the millions, potentially rendering each of these software packages inadequate. Analysts accustomed to the line-by-line memory handling technique of SAS and SUDAAN – both of which read in a single row, execute all computations, and then release that row from memory – commonly cite this drawback as their central concern when asked why they have not converted to R. Thanks to its dynamic package system, however, the R language can harness the line-by-line functionality of Structured Query Language in tandem with Open Database Connectivity (ODBC), enabling it to replicate many RAM-insensitive processes that were previously only known to a subset of the major statistical systems.

Objective

The objective of this presentation is to describe the steps required to convert large government survey data files into a SQLite database and then produce the principle set of statistical estimates and accompanying error terms, while accommodating computer systems with limited amounts of RAM. While proprietary software packages such as SAS and SUDAAN have the capacity to analyze large survey data sets in a memory-insensitive fashion, researchers can employ the techniques outlined in this poster to utilize the free R statistical computing platform and produce equivalent results.

Acknowledgements

The authors would like to thank Seth Falcon of Opscode Inc. for coding assistance and Thomas Lumley of the University of Washington for survey methodology advice.

Read large CSV files into SQL DB

```
#set to the number of GB of RAM on computer
gbram <- 0.5

#set to CSV file directory
setwd("C:\\American Community Survey\\2009\\")

#program start
start <- Sys.time()
chunk_size <- gbram * 100000
table_name <- "acs09"

library(RSQLite)

file_list <- c("ss09pusa.csv", "ss09pusb.csv")
file_list
input <- file(file_list[1], "r")

db <- dbConnect(SQLite(), dbname="acs09.db")
header <- readLines(input, n = 1)
fields <- strsplit(header, ",")[[1]]
colTypes <- rep("INTEGER", length(fields))
colDecl <- paste(fields, colTypes)
sql <- sprintf(
        paste("CREATE TABLE",table_name,"(%s)") ,
        paste(colDecl, collapse = ", ") )
dbGetQuery(db, sql)
colClasses <- rep("character", length(fields))
sql.in <- sprintf(
        paste("INSERT INTO",table_name,"VALUES (%s)"),
        paste(rep("?", length(fields)), collapse = ","))

dbBeginTransaction(db)
for (i in 1:length(file_list)){
        input <- file(file_list[i], "r")
        header <- readLines(input, n = 1)
        tryCatch({
        while (TRUE) {
                part <- read.table(input, nrows=chunk_size, sep=",",
                        colClasses = colClasses,
                        comment.char = "")
                dbGetPreparedQuery(db, sql.in, bind.data = part)
        }
        }, error = function(e) {
        if (grepl("no lines available", conditionMessage(e)))
                TRUE
        else
                stop(conditionMessage(e))
        })
}
dbCommit(db)
dbGetInfo(db)

Sys.time() - start
```

Means, Distributions, and Medians

```
dec_places <- "1.00000000"

# simple weighted mean on a linear variable – Age – of all US residents
dbGetQuery( db , paste("SELECT" , dec_places , "* SUM(AGEP*PWGTP) /
                        SUM(PWGTP) as wgtage
                        from acs09") )

# simple weighted mean on a linear variable – Age – of all US residents, by state
dbGetQuery( db , paste("SELECT ST," , dec_places , "* SUM(AGEP*PWGTP) /
                        SUM(PWGTP) as wgtage
                        from acs09
                        GROUP BY ST") )

# simple weighted mean on a factor variable
# % with Public Health Insurance Coverage – of all US residents
dbGetQuery( db , paste("SELECT PUBCOV," , dec_places , "* sum(PWGTP) /
                        (SELECT sum(PWGTP) from acs09) as pctPUBCOV
                        FROM acs09
                        GROUP BY PUBCOV") )

# % with Public Health Insurance Coverage – of all US residents, by state
try(dbGetQuery( db , "DROP TABLE totals_temp"),silent=T)
dbGetQuery( db , paste( "CREATE TABLE totals_temp AS
                        SELECT ST," , dec_places , "* sum(PWGTP) as PWGTP FROM
acs09
                        GROUP BY ST") )

dbGetQuery( db , paste( "SELECT b.ST, PUBCOV,", dec_places , "* sum(b.PWGTP) /
                        (a.PWGTP) as pctPUBCOV
                        FROM acs09 b INNER JOIN totals_temp a
                        ON a.ST == b.ST
                        GROUP BY PUBCOV, b.ST")

#rough median and other quantiles on a linear variable – Income – of adult US residents
s <- dbGetQuery( db , "SELECT SUM(PWGTP) FROM acs09 WHERE AGEP > 17 AND
                        PINCP != "")

#find record with the desired quantile point
p_w <- s * .5
#p_w <- s * .75

#reorder table by variable of interest
try(dbGetQuery( db , "DROP TABLE ordered_temp"),silent=T)
dbGetQuery( db , "CREATE TABLE ordered_temp AS
                        SELECT CAST(PINCP AS INTEGER) AS PINCP , PWGTP
                        FROM acs09 WHERE PINCP != '' AND AGEP > 17
                        ORDER BY PINCP ASC")

#pull single record at appropriate point in data set containing weighted median
sql.in <- paste( "SELECT PINCP , (SELECT SUM(PWGTP) FROM ordered_temp b
                        WHERE b.rowid <= a.rowid) as sum_wgts
                        FROM ordered_temp a
                        WHERE sum_wgts >=" , p_w , "AND AGEP > 17 LIMIT 1")

wgtd_median <- dbGetQuery( db , sql.in)
```

Computing Standard Errors with SQL

```
# weighted mean and confidence interval of a linear variable – Age – by state
i <- 1:80
replicate_sums <- paste(", ",dec_places," * SUM(AGEP * pwgtp",i,") / ",
                dec_places , "* SUM(pwgtp",i,") AS PWGTP_",i,sep="",collapse="")
sql.in <- paste("SELECT ST,", dec_places ,"* SUM(AGEP*PWGTP) /" ,
                 dec_places , "* SUM(PWGTP) AS PWGTP_A" , replicate_sums ,
                "FROM acs09 GROUP BY ST")
z <- dbGetQuery( db , sql.in )

for (i in 1:80){
     z[,paste("DIFFSQ",i,sep="")] <- ( z[,"PWGTP_A"] - z[,paste("PWGTP_",i,sep="")])^2
}
z[,"SE"] <- sqrt( rowSums(z[,grepl("DIFFSQ",names(z))]) * 4 / 80 )
z[,"UB"] <- z[,"PWGTP_A"] + 1.645 * z[,"SE"]
z[,"LB"] <- z[,"PWGTP_A"] - 1.645 * z[,"SE"]

#state code, mean, SE, 90% lower bound, 90% upper bound
z[,c("ST","PWGTP_A","SE","LB","UB")]

# weighted mean and confidence interval of a factor variable – % with Public Ins. – by state
i <- 1:80
try(dbGetQuery( db , "DROP TABLE totals_temp"),silent=T)
replicate_sums <- paste(", SUM(pwgtp",i,")  AS PWGTP_",i,sep="",collapse="")
sql.in <- paste("CREATE TABLE totals_temp AS SELECT ST, SUM(PWGTP) AS PWGTP_A",
                replicate_sums ,
                "FROM acs09 GROUP BY ST")
dbGetQuery( db , sql.in )

replicate_sums <- paste("," , dec_places , "*sum(b.PWGTP",i,") / (a.PWGTP_",i,") as
                pct",i,sep="",collapse="")
sql.in <- paste("SELECT b.ST, PUBCOV," , dec_places ,
                "* SUM(b.PWGTP)/(a.PWGTP_A) as pctA" , replicate_sums,
                "FROM acs09 b INNER JOIN totals_temp a ON a.ST == b.ST",
                "GROUP BY PUBCOV, b.ST")
z <- dbGetQuery( db , sql.in )

for (i in 1:80){
     z[,paste("DIFFSQ",i,sep="")] <- ( z[,"pctA"] - z[,paste("pct",i,sep="")])^2
}
z[,"SE"] <- sqrt( rowSums(z[,grepl("DIFFSQ",names(z))]) * 4 / 80 )
z[,"UB"] <- z[,"pctA"] + 1.645 * z[,"SE"]
z[,"LB"] <- z[,"pctA"] - 1.645 * z[,"SE"]

#state code, public coverage category, percent, SE, 90% lower bound, 90% upper bound
z[,c("ST","PUBCOV","pctA","SE","LB","UB")]
```

Limitations and Future Research

- Medians (especially when grouped) run very slowly and should be optimized

- Replicate-Weighted Regressions are currently only possible using the *survey* package, which requires somewhat larger amounts of RAM to load, even when combined with ODBC.