Big Data Analytics in R

Matthew J. Denny
University of Massachusetts Amherst

mdenny@polsci.umass.edu

March 31, 2015

www.mjdenny.com
Overview

1. Overview of High Performance Computing/Big Data Analytics in R.

2. Programming Choices

3. Parallelization/Memory Management Example

4. C++ Example.

5. Big Data Example.
High Performance Computing

- Make use of low overhead, high speed programming languages (C, C++, Fortran, Java, etc.)
- Parallelization
- Efficient implementation.
- Good scheduling.
Big Data Analytics

► Use memory efficient data structures and programming languages.

► More RAM.

► Databases.

► Efficient inference procedures.

► Good scheduling.
How they fit together

High Performance Computing

Big Data
Hardware constraints

- **RAM** = computer working memory – determines size of datasets you can work on.

- **CPU** = processor, determines speed of analysis and degree of parallelization.
Look at your activity monitor!
2. Programming Choices
Efficient R programming

- Loops are slow in R, but fast enough for most things.

- Built-in functions are mostly written in C – much faster!

- Subset data before processing when possible.

- Avoid growing datastructures
Loops are “slow” in R

```r
system.time({
  vect <- c(1:10000000)
  total <- 0
  # check using a loop
  for(i in 1:length(as.numeric(vect))){
    total <- total + vect[i]
  }
  print(total)
})
[1] 5e+13
usersystem elapsed
7.641 0.062 7.701
```
And fast in C

```r
system.time({
    vect <- c(1:10000000)
    #use the builtin R function
    total <- sum(as.numeric(vect))
    print(total)
})

[1] 5e+13

    user  system elapsed
   0.108   0.028   0.136
```
Summing over a sparse dataset

```r
# number of observations
numobs <- 100000000

# observations we want to check
vec <- rep(0,numobs)

# only select 100 to check
vec[sample(1:numobs,100)] <- 1

# combine data
data <- cbind(c(1:numobs),vec)
```
system.time({
  total <- 0
  for(i in 1:numobs){
    if(data[i,2] == 1)
      total <- total + data[i,1]
  }
  print(total)
})

[1] 5385484508

user system elapsed
199.917  0.289  200.350
Subsetting

```r
system.time({
  dat <- subset(data, data[,2] ==1)
  total <- sum(dat[,1])
  print(total)
})

[1] 5385484508
  user  system elapsed
  5.474   1.497   8.245
```
2.a. Pre-Allocation
Adding to a vector vs. pre-allocation

```r
system.time({
  vec <- NULL
  for (i in 1:(10^5)) vec <- c(vec, i)
})
```

```
user  system elapsed
18.495  7.401  25.935
```

```r
system.time({
  vec <- rep(NA, 10^5)
  for (i in 1:(10^5)) vec[i] <- i
})
```

```
user  system elapsed
0.144  0.002  0.145
```
system.time(
{
    vec <- rep(NA,10^6)
    for (i in 1:(10^6)) vec[i] <- i
}
)

user  system elapsed
1.765  0.040  1.872
Adding to a vector – bigger example

```
system.time({
  vec <- NULL
  for (i in 1:(10^6)) vec <- c(vec,i)
})

Timing stopped at: 924.922 120.322 1872.294
I didn’t feel like waiting...
```
Pre-Allocation

- Vectors in R can only hold about 2.1 Billion elements.
- Write to over-allocated vector then subset.
- Speedup is exponential in the vector size and number of additions.
2.b. Parallelization
Parallelization using `foreach`

- Works best when we need to calculate some complex statistic on each row/column of dataset.
- Works just like a regular `for( )` loop as long as operations are independent.
- Good for bootstrapping.
Parallelization using foreach

# Packages:
require(doMC)
require(foreach)

# Register number of cores
nCores <- 8
registerDoMC(nCores)

# iterations
N <- 100

# Run analysis in parallel
results <- foreach(i=1:N,.combine=rbind) %dopar% {
  result <- function(i)
}
Parallelization using a snowfall cluster

- Can run across multiple machines.
- Can run totally different jobs on each thread.
- Requires explicit management by researcher.
Parallelization using a snowfall cluster

# Package:
library(snowfall)

# Register cores
numcpus <- 4
sfInit(parallel=TRUE, cpus=numcpus)

# Check initialization
if(sfParallel()){
  cat( "Parallel on", sfCpus(), "nodes.\n" )
}else{
  cat( "Sequential mode.\n" )
}
Parallelization using a snowfall cluster

# Export all packages
for (i in 1:length(.packages())){
    eval(call("sfLibrary", (.packages()[i]),
    character.only=TRUE))
}

# Export a list of R data objects
sfExport("Object1","Object2","Object3")

# Apply a function across the cluster
result <- sfClusterApplyLB(indexes,Function)

# Stop the cluster
sfStop()
Parallelization using `mclapply()`

- Will not work with Windows machines.
- Simple parallelization.
- Works well with functions written in Rcpp.
Parallelization using `mclapply( )`

```r
# Packages:
library(parallel)

# Wrapper Function
run_on_cluster <- function(i){
  temp <- your_function(i,some other stuff)
  return(temp)
}

# Run analysis
indexes <- 1:Iterations
Result <- mclapply(indexes,
                   run_on_cluster,
                   mc.cores = num_cpus)
```
2.c. Memory Efficient Regression
High memory regression using \texttt{biglm()}

- Memory efficient implementation of \texttt{glm()}.
- Can also read in data in chunks from internet or from elational database.
- Will not take data in matrix form, only data.frame.
High memory regression using `biglm()`

# Data must be of data.frame type
data <- as.data.frame(data)

# Use variable names in formula
str <- "V1 ~ V2 + V4"

# Run model
model <- bigglm(as.formula(str),
    data = full_data,
    family = binomial(),
    maxit = 20)
3. Paralellization/Memory Management Example
Latent Network Inference Example

- Want to measure the influence of legislators on each other.
- Use temporal patterns in bill cosponsorship as evidence.
Inferring influence

Bill Cosponsorship Delay

Temporal Cascades

Time →

Bills

Time →

Time →

Time
How many ties do I use?

Edge Gain For NETINF Algorithm

Additional Influence Tie Number

Algorithm Marginal Gain
Strategy

- Predict when Senators will cosponsor in held-out sample.

- Fit event history models for model selection.

- Optimization over # edges and hyper-parameter (10 80/20 splits)

- Grid Search!
Cross validation

Rare Events Logistic Regression

Use model log likelihood for selection.
4. C++ Example
1. Rcpp is integrated with RStudio – easy C++ coding

2. RcppArmadillo – gives you access to linear algebra libraries.

3. Shallow vs. deep data structures.

4. Best for sampling and looping.
Basic RcppArmadillo C++ function

```cpp
#include <RcppArmadillo.h>
#include <string>
//@[[Rcpp::depends(RcppArmadillo)]]
using namespace Rcpp;
//@[[Rcpp::export]]
List My_Function(
    int some_number,
    List some_vectors,
    arma::vec a_vector,
    arma::mat example_matrix
){
    ...
    List to_return(1);
    to_return[0] = some_data;
    return to_return;
}
```
for(int n = 0; n < number_of_bills; ++n){
    report(n);
    int length = Bill_Lengths[n];
    std::vector<std::string> current = Bill_Words[n];
    for(int i = 0; i < length; ++i){
        int already = 0;
        int counter = 0;
        while(already == 0){
            if(unique_words[counter] == current[i]){
                unique_word_counts[counter] += 1;
                already = 1;
            }
            counter +=1;
        }
        counter +=1;
    }
}
// add to second and third lines of file
#include <random>
#include <math.h>

// set RNG and seed
std::mt19937_64 generator(seed);

// define a uniform distribution and draw from it
std::uniform_real_distribution<double> udist(0.0, 1.0);
double rand_num = udist(generator);

// define a normal distribution and draw from it
std::normal_distribution<double> ndist(mu, sigsq);
my_matrix(k, b) = ndist(generator);
# In R define
Report <- function(string){print(string)}

// In C++ we write (inside function definition)
Function report("Report");

// now we can print stuff back up to R
report(n);

// initialize a vector/matrix to zeros
arma::vec myvec = arma::zeros(len);

// some math operators
double d = exp(log(pow(2,4)));
Things to watch out for

1. Use Armadillo data structures – Rcpp data structures can overflow memory.

2. Cast integers as doubles before dividing.

3. Low latency + faster looping = 50-2,000x speedup.

4. For Linux systems:

   PKG_CPPFLAGS = "-std=c++11"
   Sys.setenv(PKG_CPPFLAGS = PKG_CPPFLAGS)
5. Big Data Example
1. 90,867 final versions of bills introduced in the U.S. Congress form 1993-2012.

2. 293,697,605 tokens (370,445 unique).

3. 90 covariates for every bill.

4. Addition data on amendments, cosponsorships, and floor speeches.
This Act may be cited as the ‘‘EPA Science Act of 2014’’.

SEC. 2. SCIENCE ADVISORY BOARD.

(a) Independent Advice.—Section 8(a) of the Environmental Research, Development, and Demonstration Authorization Act of 1978 (42 U.S.C. 4365(a)) is amended by inserting ‘‘independently’’ after ‘‘Advisory Board which shall’’.

(b) Membership.—Section 8(b) of the Environmental Research, Development, and Demonstration Authorization Act of 1978 (42 U.S.C. 4365(b)) is amended to read as follows:

‘‘(b)(1) The Board shall be composed of at least nine members, one of whom shall be designated Chairman, and shall meet at such times and places as may be designated by the Chairman.

‘‘(2) Each member of the Board shall be qualified by education, training, and experience to evaluate scientific and technical ....
## Taxonomy of Bill Text

<table>
<thead>
<tr>
<th>Category</th>
<th>Definition</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Substantive Language</td>
<td>Confers the intent of a piece of legislation or a particular provision.</td>
<td>{restrict abortion}, {reduce the deficit}</td>
</tr>
<tr>
<td>Topical Language</td>
<td>Confers information about the subject of the Bill.</td>
<td>{alternate academic achievement standards}</td>
</tr>
<tr>
<td>Boilerplate</td>
<td>Gives direction about legal interpretation or implementation.</td>
<td>{Notwithstanding any other provision of this paragraph...}</td>
</tr>
<tr>
<td>Domain Stopwords</td>
<td>Gives no information about intent, legal interpretation or implementation.</td>
<td>{SECTION}, {(c) Title III.–}, {(1) Part a.–}, {(A) Subpart 1.–}, {to adopt}</td>
</tr>
</tbody>
</table>
## Lets Look at N-Grams

<table>
<thead>
<tr>
<th>Unit of Analysis</th>
<th>Topical Text</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tokens</strong></td>
<td>{Should}, {a}, {Federal}, {agency}, {seek}, {to}, {restrict}, {photography}, {of}, {its}, {installations}, {or}, {personnel}, {it}, {shall}, {obtain}, {a}, {court}, {order}, {that}, {outlines}, {the}, {national}, {security}, {or}, {other}, {reasons}, {for}, {the}, {restriction}</td>
</tr>
<tr>
<td><strong>Bigrams</strong></td>
<td>{Should a}, {a Federal}, {Federal agency}, {agency seek}, {seek to}, {to restrict}, {restrict photography}, {photography of}, {of its}, {its installations}, {installations or}, {or personnel}, {personnel it}, {it shall}, {shall obtain}, {obtain a}, {a court}, {court order}, {order that}, {that outlines}, {outlines the}, {the national}, {national security}, {security or}, {or other}, {other reasons}, {reasons for}, {for the}, {the restriction}</td>
</tr>
<tr>
<td>Tag Pattern</td>
<td>Example</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>AN</td>
<td>linear function</td>
</tr>
<tr>
<td>NN</td>
<td>regression coefficients</td>
</tr>
<tr>
<td>AAN</td>
<td>Gaussian random variable</td>
</tr>
<tr>
<td>ANN</td>
<td>cumulative distribution function</td>
</tr>
<tr>
<td>NAN</td>
<td>mean squared error</td>
</tr>
<tr>
<td>NNN</td>
<td>class probability function</td>
</tr>
<tr>
<td>NPN</td>
<td>degrees of freedom</td>
</tr>
</tbody>
</table>

Add in verbs to capture intent...

<table>
<thead>
<tr>
<th>Tag Pattern</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>VN</td>
<td>reduce funding</td>
</tr>
<tr>
<td>VAN</td>
<td>encourage dissenting members</td>
</tr>
<tr>
<td>VNN</td>
<td>restrict federal agencies</td>
</tr>
</tbody>
</table>
## J&K Filtering and Phrase Extraction

<table>
<thead>
<tr>
<th>Unit of Analysis</th>
<th>Topical Text – Verb Phrase Additions</th>
</tr>
</thead>
<tbody>
<tr>
<td>J&amp;K Filtered Bi-grams</td>
<td>{Federal agency}, {restrict photography}, {court order}, {national security}, {other reasons}</td>
</tr>
<tr>
<td>J&amp;K Filtered Trigrams</td>
<td>NONE</td>
</tr>
<tr>
<td>Noun Phrases</td>
<td>{Federal agency}, {court order}, {national security}, {other reasons for the restriction}</td>
</tr>
</tbody>
</table>
Constructing a Document-Term Matrix

- Want Document x Vocabulary matrix.
- Take advantage of sparsity.
- Use C++ for indexing.
- Have to chunk and add.
Constructing a Document-Term Matrix

# gives us simple triplet matrix class
library(slam)

# load in C++ function to generate rows in matrix
Rcpp::sourceCpp('Document_Word_Compiler.cpp')

for(i in 1:chunks){
    dwm <- Generate_Document_Word_Matrix(chunk,..)
    dwm <- as.simple_triplet_matrix(dwm)
    if(j == 1){
        swm <- dwm
    }else{
        swm <- rbind(swm,dwm)
    }
}
## Semi-Supervised Major Topic Tags

<table>
<thead>
<tr>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Macroeconomics</td>
</tr>
<tr>
<td>2. Civil Rights, Minority Issues, and Civil Liberties</td>
</tr>
<tr>
<td>3. Health</td>
</tr>
<tr>
<td>4. Agriculture</td>
</tr>
<tr>
<td>5. Labor and Employment</td>
</tr>
<tr>
<td>6. Education</td>
</tr>
<tr>
<td>7. Environment</td>
</tr>
<tr>
<td>8. Energy</td>
</tr>
<tr>
<td>9. Immigration</td>
</tr>
<tr>
<td>10. Transportation</td>
</tr>
<tr>
<td>12. Law, Crime, and Family Issues</td>
</tr>
<tr>
<td>13. Social Welfare</td>
</tr>
<tr>
<td>14. Community Development and Housing Issues</td>
</tr>
<tr>
<td>15. Banking, Finance, and Domestic Commerce</td>
</tr>
<tr>
<td>16. Defense</td>
</tr>
</tbody>
</table>

.....
J&K Filtered Trigrams for Identifying Topical Area

<table>
<thead>
<tr>
<th>Health</th>
<th>Word</th>
<th>PMI</th>
<th>Local</th>
<th>Global</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>stay</td>
<td>1.754</td>
<td>24425</td>
<td>26428</td>
</tr>
<tr>
<td></td>
<td>start</td>
<td>1.689</td>
<td>12395</td>
<td>14321</td>
</tr>
<tr>
<td></td>
<td>enhance</td>
<td>1.684</td>
<td>22142</td>
<td>25707</td>
</tr>
<tr>
<td></td>
<td>providers</td>
<td>1.684</td>
<td>51656</td>
<td>59976</td>
</tr>
<tr>
<td></td>
<td>mining</td>
<td>1.679</td>
<td>15221</td>
<td>17751</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Filtered Trigram</th>
<th>PMI</th>
<th>Local</th>
<th>Global</th>
</tr>
</thead>
<tbody>
<tr>
<td>health insurance plan</td>
<td>1.340</td>
<td>868</td>
<td>1528</td>
</tr>
<tr>
<td>health benefit plan</td>
<td>1.306</td>
<td>1125</td>
<td>2049</td>
</tr>
<tr>
<td>term care insurance</td>
<td>1.292</td>
<td>3564</td>
<td>6577</td>
</tr>
<tr>
<td>health plan means</td>
<td>1.110</td>
<td>261</td>
<td>578</td>
</tr>
<tr>
<td>alternative dispute resolution</td>
<td>1.079</td>
<td>923</td>
<td>2108</td>
</tr>
</tbody>
</table>
Unit of Analysis? – Keystone XL Pipeline Approval Act

Figure 2: S.1 - Keystone XL Pipeline Approval Act proportion of bill text devoted to each purpose. Note that roughly 5% of the text is devoted to the stated intention of the bill.

- 45 Lines Approve Keystone XL pipeline
- 88 Lines Energy Retrofit for Schools
- 34 Lines “Climate change is real and not a hoax”
- 489 Lines Energy Efficiency Improvements
1. Average conditional mutual information vocabulary partitioning.

2. Political branding and meme-detection.

3. Text/Networks

4. More super secret tech :}
Big Data Challenges

1. Extending R vectors/matricies beyond 2.1 billion elements.

2. More low level datastructures – linked list, queue, stack, etc.
