High Performance Computing and Big Data Analytics: Applications

Matthew J. Denny
University of Massachusetts Amherst

mdenny@polsci.umass.edu

8/5/2014

INSTITUTE FOR SOCIAL SCIENCE RESEARCH
UNIVERSITY OF MASSACHUSETTS AMHERST

UMassAmherst

https://polsci.umass.edu/profiles/denny_matthew_j/workshop-materials

Overview

1. General scientific computing in R.
2. HPC R code examples.
3. Basic `bash` commands and `ssh`.
4. Full hands-on example.

1. Scientific Computing in R.

Motivation – the gardener

- How many plants to water?
- Which plants to water?

Overview

- `for( )` and `while( )` loops.
- `if( )` statements.
- Nested loops
- Lists

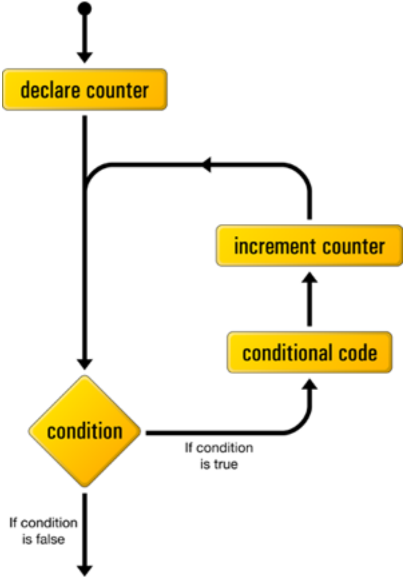
Some preliminaries

```
# create a vector
my_vector <- c(1:10)
print(my_vector)

# get the length of the vector
length(my_vector)

# comparison operators
5 < 6
5 > 6
5 == 5
5 != 6
5 <= 5
```

The `for( )` loop


declare counter
increment counter
conditional code
condition
If condition
is true
If condition
is false

The `for( )` loop

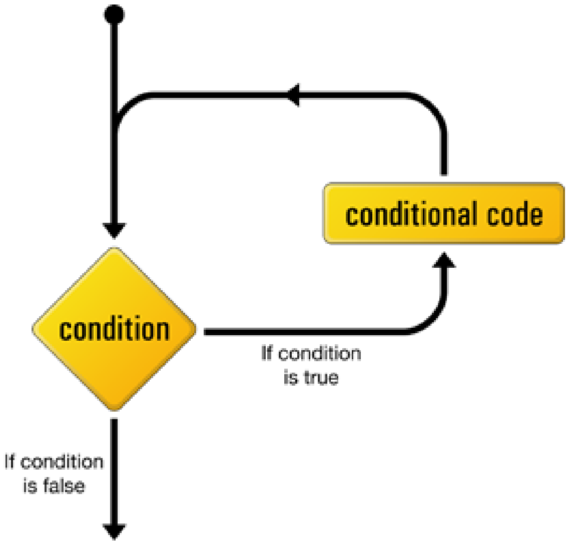
- Do something N times.

```
my_vector <- c(20:30)
for(i in 1:length(my_vector)){
    my_vector[i] <- sqrt(my_vector[i])
}

my_vector

[1] 4.472136 4.582576 4.690416 4.795832 4.898979
[6] 5.000000 5.099020 5.196152 5.291503 5.385165
[11] 5.477226
```

The `while( )` loop



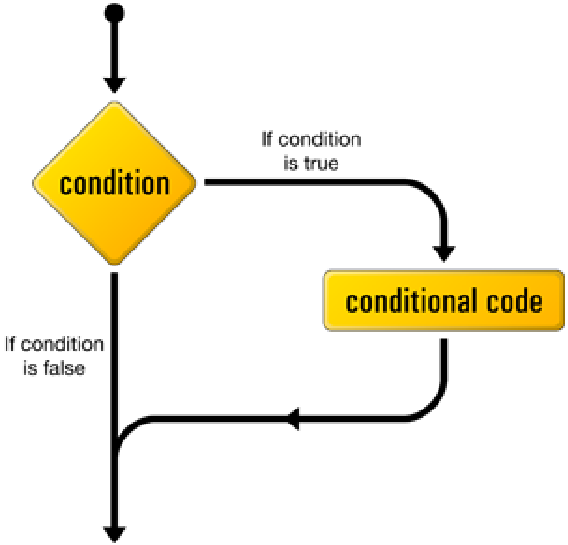
The `while( )` loop

- Do something until a condition is met.
- Useful if you do not know the number of iterations ahead of time.

```
my_vector <- c(20:30)
counter <- 1
while(counter <= length(my_vector)){
    my_vector[counter] <- sqrt(my_vector[counter])
    counter <- counter + 1
}

my_vector
```

The `if( )` statement



The `if( )` statement

- Do something if some condition is met.
- Can be built into a loop.

```
my_vector <- c(20:30)

for(i in 1:length(my_vector)){
    if(my_vector[i] == 25){
        print("The square root of 25 is 5!")
    }
}
```

The `else` statement

- Do something if some condition is not met.

```
my_vector <- c(20:30)

for(i in 1:length(my_vector)){
    if(my_vector[i] == 25){
        print("I am 25!")
    }else{
        print("I am not 25!")
    }
}
```

How do I loop over a matrix?

```
> matrix(1:25,5,5)
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    6   11   16   21
[2,]    2    7   12   17   22
[3,]    3    8   13   18   23
[4,]    4    9   14   19   24
[5,]    5   10   15   20   25
```

Nested loops

- Can loop over entries in higher dimensional data structures.

```
my_matrix <- matrix(1:100,ncol=10,nrow=10)

for(i in 1:length(my_matrix[,1])){
    for(j in 1:length(my_matrix[1,])){
        if(my_matrix[i,j] %% 2 == 0){
            my_matrix[i,j] <- 0
        }
    }
}

my_matrix
```

Lists

- Flexible, can store any kind of data including another list.
- Good for keeping results together.

```
# Create an empty list
my_list <- vector("list", length = 10)

# Create a list from objects
my_list <- list(10, "dog",c(1:10))

# Add a sublist to a list
my_list <- append(my_list, list(list(27,14,"cat")))
```

List contents

```
> my_list
[[1]]
[1] 10

[[2]]
[1] "dog"

[[3]]
 [1]  1  2  3  4  5  6  7  8  9 10

[[4]]
[[4]][[1]]
[1] 27

[[4]][[2]]
[1] 14

[[4]][[3]]
[1] "cat"
```

1. HPC in R.

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( )

```
# 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)
```

High memory regression using `biglm( )`

- Memory efficient implementation of `glm( )`
- Can also read in data in chunks from internet or from 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. Bash and SSH

Overview

- `bash` is a command line terminal.
  - Available for Linux and OS X.
- `VPN`: access campus network.
- `ssh/PuTTY` (Windows) for remote access.
- `ftp`: for file transfer.

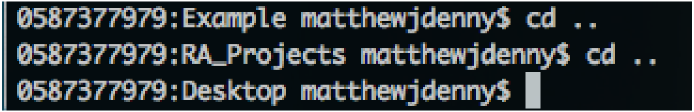
Bash commands

- `cd`: change the current directory.
- `ls` prints contents of current directory.
- `edit/vi/emacs` opens a text editor.

```
0587377979:Desktop matthewjdenny$ cd RA_Projects/
0587377979:RA_Projects matthewjdenny$ ls
Example          Jerry Epstein    Jerry Friedman
0587377979:RA_Projects matthewjdenny$ cd Example/
0587377979:Example matthewjdenny$ ls
Example.R
0587377979:Example matthewjdenny$ edit Example.R
```

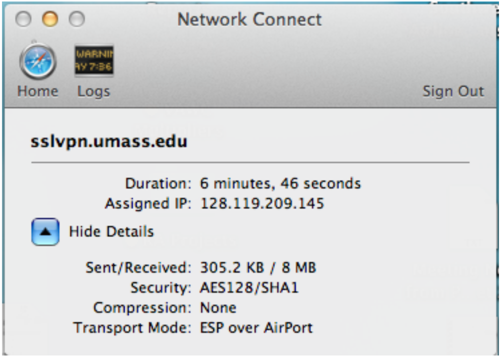
Bash commands – continued

- `python` opens python console.
- `R` opens standard R console.
- `cd ..` moves us back up a level in the directory structure.



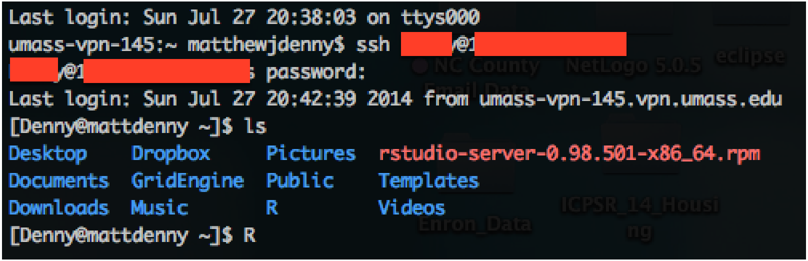
Using VPN

- Provided by university/organization.
- For login to local campus resources.
- Routes all trafic through campus servers

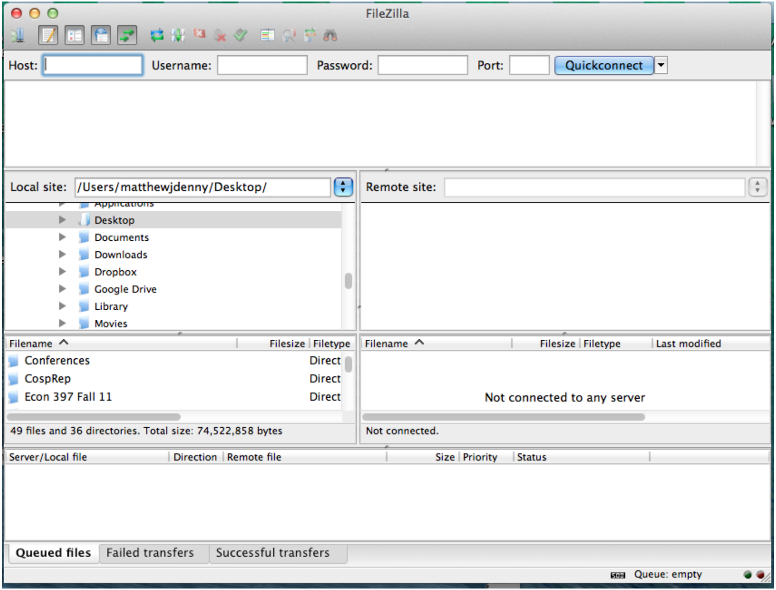


Using SSH

- Must have account on remote machine.
- ssh username@ipaddress
  - static ip: 128.114.64.8
  - dynmaic: somedomain.dyndns.com
- prompt to enter password



FTP using FileZilla



Putting it all together

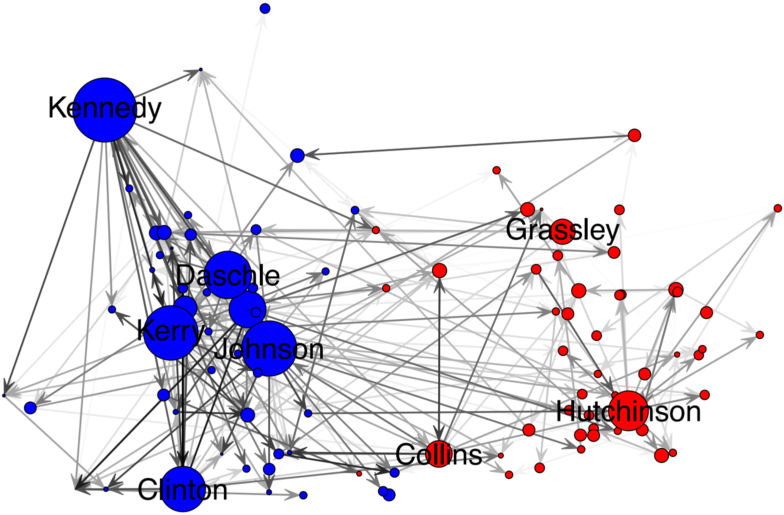
1. Connect to campus network using `VPN`
2. `ssh` into remote computing resource
3. Transfer files to/from machine using `ftp`
4. Navigate directories using `bash` and run analysis in `R` or `Python`.

4. Full Example

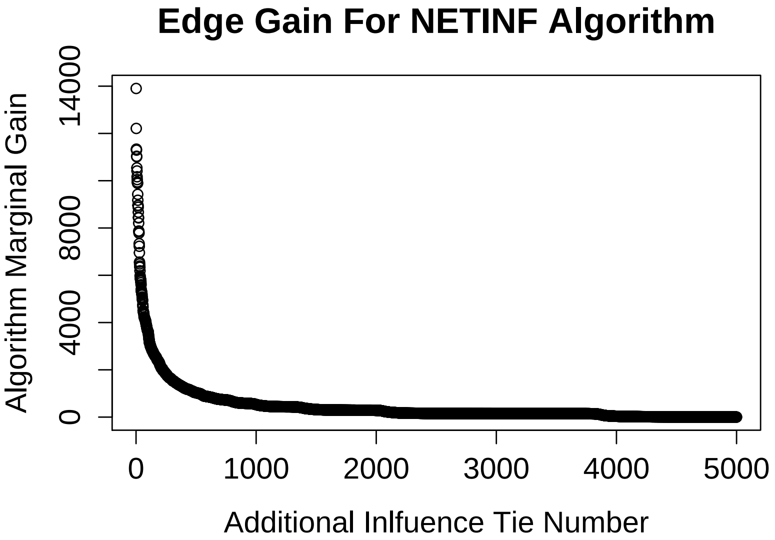
Example

- Want to measure the influence of legislators on each other.
- Use temporal patterns in bill cosponsorship as evidence.
- Gomez Rodriguez, M., Leskovec, J., & Krause, A. (2010). "**Inferring networks of diffusion and influence**". *KDD*

Measuring influence in the Senate



How many ties do I use?

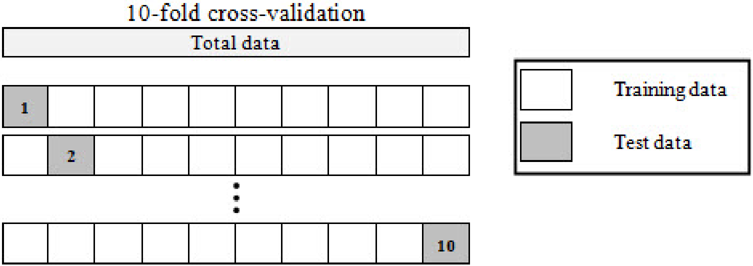


Following along

- Code/data provided with course materials.
- Predict when Senators will cosponsor legislation in held-out sample based on vary number of influence ties.
- Fit event history models for model selection.

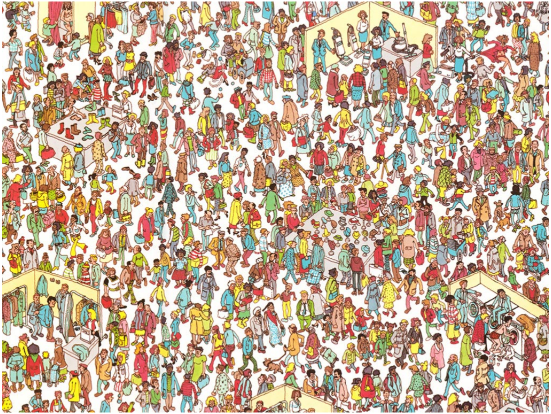
Cross validation

- Jensen, D. D., & Cohen, P. R. (2000). **Multiple Comparisons in Induction Algorithms**. *Machine Learning*, 309338.

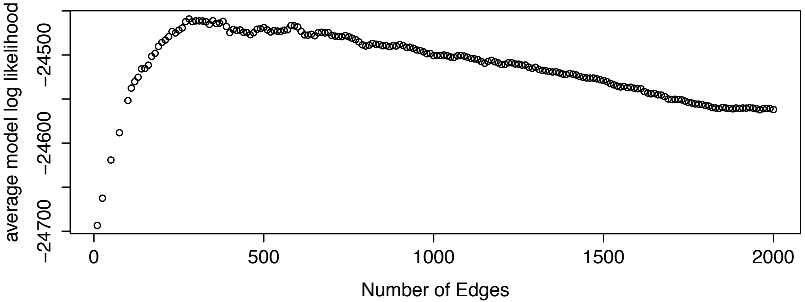


Rare Events Logistic Regression

- King, G., & Zeng, L. (2001). **Logistic regression in rare events data**. *Political Analysis*, 9(2), 137163.



Use model log likelihood for selection.



Link to Materials

https://polsci.umass.edu/profiles/denny_matthew_j/workshop-materials