

Data Analysis with R on Stampede

Weijia Xu, Ph.D

Manager, Data Mining & Statistics

Texas Advanced Computing Center

xwj@tacc.utexas.edu

Outline

- Introduction R
- Data mining with R
- Using R on Stampede with big computation

Introduction to R

What R does

R is a programming environment for statistical and data analysis computations.

- Core Package
 - Statistical functions
 - plotting and graphics
 - Data handling and storage
 - predefined data reader
 - textual, regular expressions
 - hashing
 - Data analysis functions
 - Programming support:
 - loops, branching, subroutines
 - Object Oriented
- More additional developed packages.

R-project background

- Origin and History
 - initially written by Ross Ihaka and Robert Gentleman at Dep. of Statistics of U of Auckland, New Zealand during 1990s.
 - International project since 1997
- Open source with GPL license
 - Free to anyone
 - In actively development
 - <http://www.r-project.org/>

Getting Started

- Download and install locally from
 - <http://www.r-project.org/>
- TACC
 - ssh stampede.tacc.utexas.edu
 - % module load R
 - % R

Getting Started

```
login1$ R
```

```
R version 2.15.1 (2012-06-22) -- "Roasted Marshmallows"  
Copyright (C) 2012 The R Foundation for Statistical Computing  
ISBN 3-900051-07-0  
Platform: x86_64-unknown-linux-gnu (64-bit)
```

```
R is free software and comes with ABSOLUTELY NO WARRANTY.  
You are welcome to redistribute it under certain conditions.  
Type 'license()' or 'licence()' for distribution details.
```

```
  Natural language support but running in an English locale
```

```
R is a collaborative project with many contributors.  
Type 'contributors()' for more information and  
'citation()' on how to cite R or R packages in publications.
```

```
Type 'demo()' for some demos, 'help()' for on-line help, or  
'help.start()' for an HTML browser interface to help.  
Type 'q()' to quit R.
```

```
> |
```

Variables

- Numeric

```
> a=49  
> a  
[1] 49
```

- Character String

```
> b="this is a string"  
> b  
[1] "this is a string"
```

- Logical

```
> c=(1+1==3)  
> c  
[1] FALSE
```


NA: Missing Value

- Variables of each data type (numeric, character, logical) can also take the value NA: not available.
 - NA is not the same as 0
 - NA is not the same as ""
 - NA is not the same as FALSE
- Any operations (calculations, comparisons) that involve NA may or may not produce NA:

```
> NA
[1] NA
> 1+NA
[1] NA
> log(NA)
[1] NA
```

```
> NA | TRUE
[1] TRUE
> NA | FALSE
[1] NA
> NA & TRUE
[1] NA
> NA & FALSE
[1] FALSE
```

```
> max(c(1,2,3, NA))
[1] NA
> max(c(1,2,3,NA), na.rm=T)
[1] 3
```

Assigning Values to Variables

- “<-” or “=”

```
> a=4
> a
[1] 4
> a<-40
> a
[1] 40
```

```
> a=c(1, 2, 4, 7, 9)
> a
[1] 1 2 4 7 9
```

- Assign multiple values

- Concatenate, c()
- From stdin, scan()
- Series

- Seq()

```
> a=(1:6)
> a
[1] 1 2 3 4 5 6
```

```
> a=seq(1, 6, 0.5)
> a
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0
```

```
> a=scan()
1: 9
2: 7
3: 4
4: 2
5: 1
6:
Read 5 items
> a
[1] 9 7 4 2 1
```

Basic Data Structure

- Vector

- an ordered collection of data of the same type
- a single number is the special case of a vector with 1 element.
- Usually accessed by index

- Matrix

- A rectangular table of data of the same type

```
> a = c(1, 2, 3)
> a
[1] 1 2 3
> a[1]
[1] 1
> a[2]
[1] 2
> a*2
[1] 2 4 6
```

Basic Data Structure

- List
 - an ordered collection of data of arbitrary types.
 - name-value pair
 - Accessible by name

```
> doe = list(name="john", age=28, married=F)
> doe$name
[1] "john"
> doe$age
[1] 28
> doe$married
[1] FALSE
> doe[1]
$name
[1] "john"
```

Basic Data Structure

- Hash Table
 - In R, a hash table is the same as a workspace for variables, which is the same as an environment.
 - Store Key-value pairs.
 - Value can be accessed by key

```
> tab = new.env(hash=T)
> assign("A", list(id=682, description="six eight two"), env=tab)
> assign("B", list(id=77, description="seven seven"), env=tab)
> assign("C", list(id=77, mykey="the key is C"), env=tab)
> get("A", env=tab)
$id
[1] 682

$description
[1] "six eight two"

> get("C", env=tab)
$id
[1] 77

$mykey
[1] "the key is C"
```

Dataframes

- R handles data in objects known as **dataframes**;
 - rows: data items;
 - columns: values of the different attributes
 - Values in each column should be from the same type.

	Area	Slope	Vegetation	Soil.pH	Damp	Worm.density
Silwood.Bottom	5.1	2	Arable	5.2	FALSE	7
Gunness.Thicket	3.8	0	Scrub	4.2	FALSE	6
Oak.Mead	3.1	2	Grassland	3.9	FALSE	2
North.Gravel	3.3	1	Grassland	4.1	FALSE	1
South.Gravel	3.7	2	Grassland	4.0	FALSE	2
Pond.Field	4.1	0	Meadow	5.0	TRUE	6
Water.Meadow	3.9	0	Meadow	4.9	TRUE	8
Pound.Hill	4.4	2	Arable	4.5	FALSE	5

Read Dataframes From File

- `read.table()`

the first column contains data label

```
> worms<-read.table("worms.txt",header=T,row.names=1)
```

path: in double quotes

the first row contains the variables names

- Read tab-delimited file directly.
 - Variable name in header row cannot have space.
- To see the content of the dataframes (object) just type its name:

```
> worms
```

Selecting Data from Dataframes

- Subscripts within square brackets
 - [, means “all the rows” and
 - ,] means “all the columns”
- To select the first three column of the dataframe

```
> worms[,1:3]
```

	Area	Slope	Vegetation
Silwood.Bottom	5.1	2	Arable
Gunness.Thicket	3.8	0	Scrub
Oak.Mead	3.1	2	Grassland
North.Gravel	3.3	1	Grassland
South.Gravel	3.7	2	Grassland
Pond.Field	4.1	0	Meadow
Water.Meadow	3.9	0	Meadow
Pound.Hill	4.4	2	Arable

Selecting Data from Dataframes

- `names()`
 - Get a list of variables attached to the input name

```
> names(worms)
[1] "Area"          "Slope"         "Vegetation"
[4] "Soil.pH"      "Damp"          "Worm.density"
```

- `attach()`
 - Make the variables accessible by name:
 - > `attach(worms)`

Selecting Data from Dataframes

- Using logic expression while selecting:

	Area	Slope	Vegetation	Soil.pH	Damp	Worm.density
Silwood.Bottom	5.1	2	Arable	5.2	FALSE	7
Gunness.Thicket	3.8	0	Scrub	4.2	FALSE	6
Oak.Mead	3.1	2	Grassland	3.9	FALSE	2
North.Gravel	3.3	1	Grassland	4.1	FALSE	1
South.Gravel	3.7	2	Grassland	4.0	FALSE	2
Pond.Field	4.1	0	Meadow	5.0	TRUE	6
Water.Meadow	3.9	0	Meadow	4.9	TRUE	8
Pound.Hill	4.4	2	Arable	4.5	FALSE	5

```
> worms[Area>4&Slope<1,]
```

	Area	Slope	Vegetation	Soil.pH	Damp	Worm.density
Pond.Field	4.1	0	Meadow	5	TRUE	6

Selecting Data From a Dataframe

More examples:

```
> worms[Damp,]
      Area Slope Vegetation Soil.pH Damp Worm.density
Pond.Field  4.1    0   Meadow    5.0  TRUE           6
Water.Meadow 3.9    0   Meadow    4.9  TRUE           8
> worms$Vegetation
[1] Arable  Scrub  Grassland Grassland Grassland Meadow  Meadow
[8] Arable
Levels: Arable Grassland Meadow Scrub
> worms$Vegetation=="Grassland"
[1] FALSE FALSE  TRUE  TRUE  TRUE FALSE FALSE FALSE
> worms[ worms$Vegetation=="Grassland",]
      Area Slope Vegetation Soil.pH  Damp Worm.density
Oak.Mead  3.1    2  Grassland    3.9 FALSE           2
North.Gravel 3.3    1  Grassland    4.1 FALSE           1
South.Gravel 3.7    2  Grassland    4.0 FALSE           2
```

subset rows by a
logical vector

subset a column

comparison resulting
in logical vector

subset the
selected rows

Sorting Data in Dataframes

- `order()`

State the Area for sorting order

State columns to be sorted

```
> worms[order(worms[,1]),1:6]
```

	Area	Slope	Vegetation	Soil.pH	Damp	Worm.density
Oak.Mead	3.1	2	Grassland	3.9	FALSE	2
North.Gravel	3.3	1	Grassland	4.1	FALSE	1
South.Gravel	3.7	2	Grassland	4.0	FALSE	2
Gunness.Thicket	3.8	0	Scrub	4.2	FALSE	6
Water.Meadow	3.9	0	Meadow	4.9	TRUE	8
Pond.Field	4.1	0	Meadow	5.0	TRUE	6
Pound.Hill	4.4	2	Arable	4.5	FALSE	5
Silwood.Bottom	5.1	2	Arable	5.2	FALSE	7

Flow Control

- If ... else

```
if (logical expression) {  
    statements  
} else {  
    alternative statements  
}
```

* else branch is optional

- loops

```
for(i in 1:10) {  
    print(i*i)  
}
```

```
i=1  
while(i<=10) {  
    print(i*i)  
    i=i+sqrt(i)  
}
```

Flow Control

- `apply(arr, margin, fct)`
 - Applies the function `fct` along some dimensions of the array `arr`, according to `margin`, and returns a vector or array of the appropriate size.

```
> m
      Soil.pH Worm.density
Silwood.Bottom      5.2          7
Pond.Field           5.0          6
Water.Meadow         4.9          8
Pound.Hill           4.5          5
Gunness.Thicket     4.2          6
North.Gravel         4.1          1
South.Gravel         4.0          2
Oak.Mead             3.9          2
> apply(m, 1, sum)
      Silwood.Bottom      Pond.Field      Water.Meadow      Pound.Hill      Gunness.Thicket
           12.2           11.0           12.9           9.5           10.2
      North.Gravel      South.Gravel      Oak.Mead
           5.1           6.0           5.9
> apply(m, 2, sum)
      Soil.pH Worm.density
           35.8           37.0
```

Flow Control

- **lapply (list, fct)** and **sapply (list, fct)**
 - To each element of the list l_i , the function fct is applied. The result is a list whose elements are the individual fct results.
 - **Sapply**, converting results into a vector or array of appropriate size

```
> fct = function(x) { return(c(x, x*x, x*x*x)) }
> sapply(1:5, fct)
      [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    1    4    9   16   25
[3,]    1    8   27   64  125
```

```
> lapply(1:5, fct)
[[1]]
[1] 1 1 1

[[2]]
[1] 2 4 8

[[3]]
[1] 3 9 27

[[4]]
[1] 4 16 64

[[5]]
[1] 5 25 125
```

Create Statistical Summary

- Descriptive summary for numerical variables:
 - arithmetic mean;
 - maximum, minimum, median, 25 and 75 percentiles (first and third quartile);
- Levels of categorical variables are counted

```
> summary(worms)
```

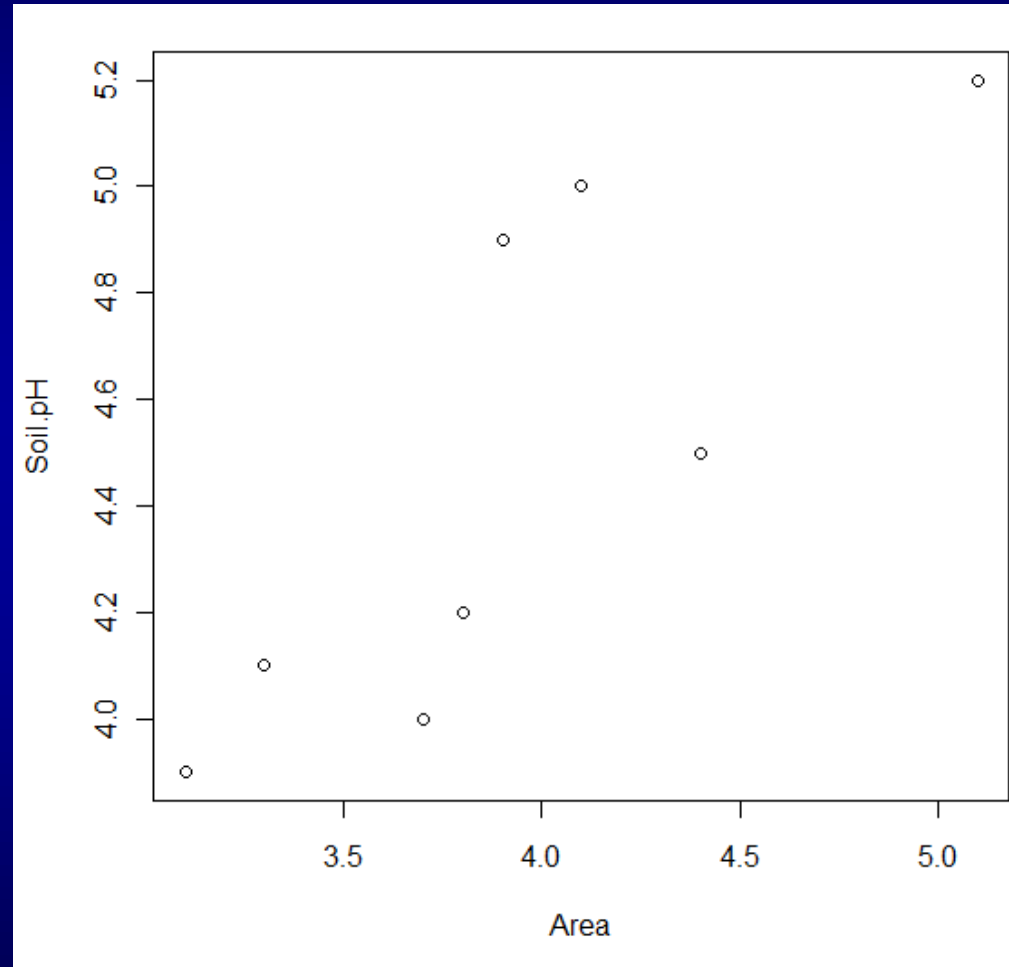
Area	Slope	Vegetation	Soil.pH	Damp	Worm.density
Min. :3.100	Min. :0.000	Arable :2	Min. :3.900	Mode :logical	Min. :1.000
1st Qu.:3.600	1st Qu.:0.000	Grassland:3	1st Qu.:4.075	FALSE:6	1st Qu.:2.000
Median :3.850	Median :1.500	Meadow :2	Median :4.350	TRUE :2	Median :5.500
Mean :3.925	Mean :1.125	Scrub :1	Mean :4.475	NA's :0	Mean :4.625
3rd Qu.:4.175	3rd Qu.:2.000		3rd Qu.:4.925		3rd Qu.:6.250
Max. :5.100	Max. :2.000		Max. :5.200		Max. :8.000

Create Plots

- `plot(...)`
 - Create scatter plot.

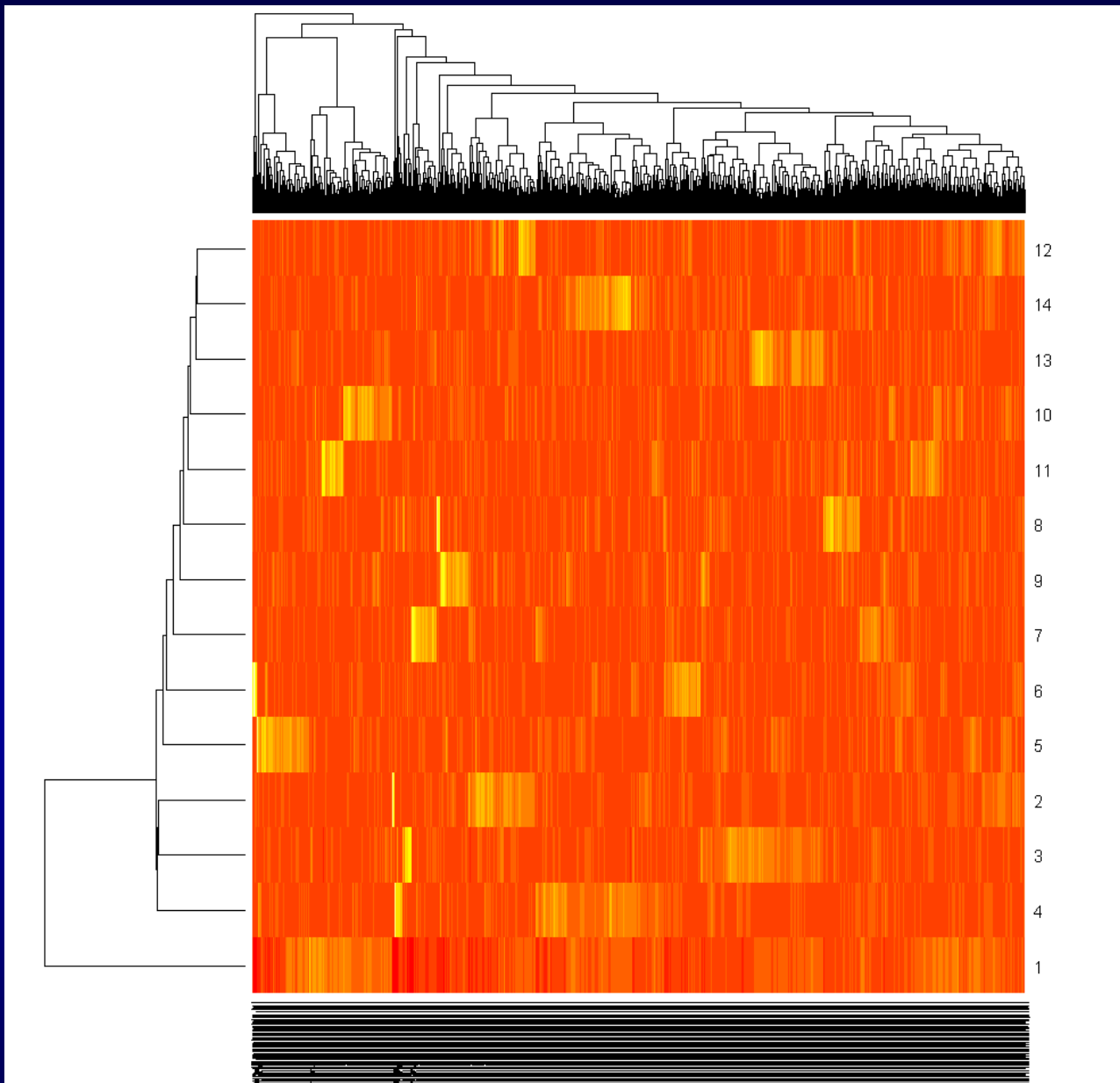
➤ `plot(Area, Soil.pH)`

Automatically create
a postscript file with
default name



Other Common Plots

- Univariate:
 - histograms,
 - density curves,
 - Boxplots, quantile-quantile plots
- Bivariate:
 - scatter plots with trend lines,
 - side-by-side boxplots
- Several variables:
 - scatter plot matrices, lattice
 - 3-dimensional plots,
 - heatmap



Getting help

- “?” Or “help”

Details about a specific command whose name you know (input arguments, options, algorithm, results):

e.g.

`>? t.test`

or

`>help(t.test)`

```
t.test                package:ctest                R Documentation
Student's t-Test
Description:
  Performs one and two sample t-tests on vectors of data.
Usage:
  t.test(x, y = NULL, alternative = c("two.sided", "less", "greater"),
         mu = 0, paired = FALSE, var.equal = FALSE,
         conf.level = 0.95, ...)
  t.test(formula, data, subset, na.action, ...)
Arguments:
  x: a numeric vector of data values.
  y: an optional numeric vector data values.
alternative: a character string specifying the alternative hypothesis,
  must be one of "two.sided" (default), "greater" or
  "less". You can specify just the initial letter.
mu: a number indicating the true value of the mean (or difference
  in means if you are performing a two sample test).
paired: a logical indicating whether you want a paired t-test.
var.equal: a logical variable indicating whether to treat the two
```

Data Mining with R

Data mining with R

- Many data mining methods are also supported in R core package or in R modules
 - Kmeans clustering:
 - `Kmeans()`
 - Decision tree:
 - `rpart()` in `rpart` library
 - Nearest Neighbour
 - `Knn()` in `class` library
 - ...

Additional Libraries and Packages

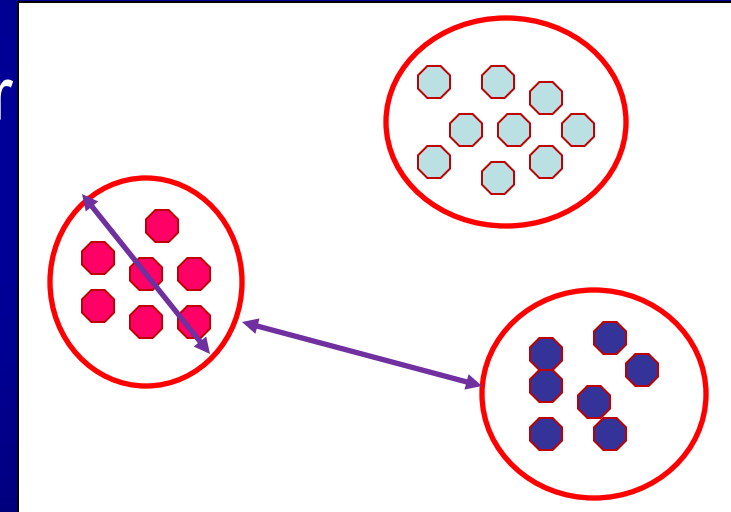
- Libraries
 - Comes with Package installation (Core or others)
 - `library()` shows a list of current installed
 - library must be loaded before use e.g.
 - `library(rpart)`
- Packages
 - Developed code/libraries outside the core packages
 - Can be downloaded and installed separately
 - `install.package("name")`
 - There are currently 4692 packages at <http://cran.r-project.org/web/packages/>
 - E.g. Rweka, interface to Weka.

Common Data Mining Methods

- Clustering Analysis
 - Grouping data object into different bucket.
- Classification
 - Assigning labels to each data object.
 - Requires training data.
- Association Analysis
 - Identifying the connections among different data objects.

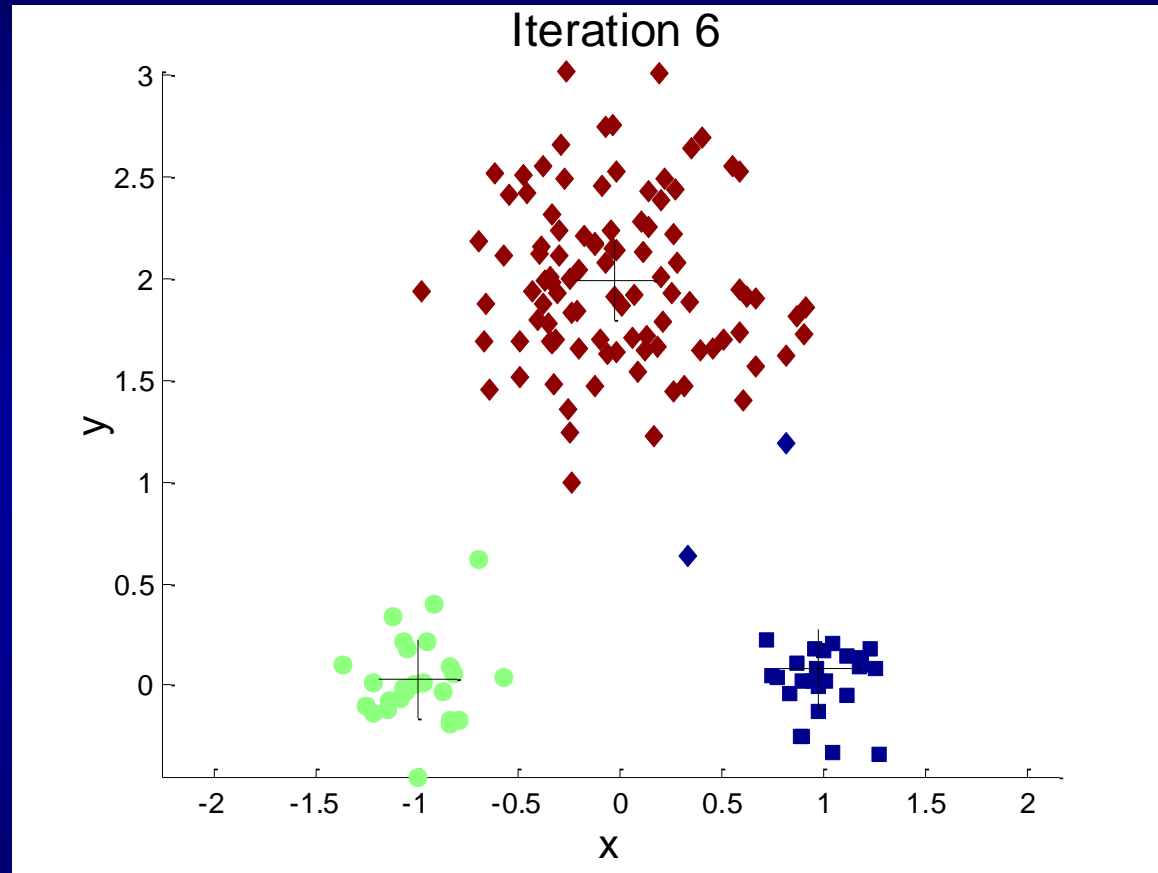
Cluster Analysis

- Finding groups of objects such that the objects in a group will be similar (or related) to one another and different from (or unrelated to) the objects in other groups
 - Inter-cluster distance: maximized
 - Intra-cluster distance: minimized



An Example of k-means Clustering

K=3



Examples are from Tan, Steinbach, Kumar *Introduction to Data Mining*

Kmeans clustering Example

```
% more kmeans.R
x<-read.csv("../data/cluster.csv",header=F)
fit<-kmeans(x, 2)
plot(x,pch=19,xlab=expression(x[1]),
      ylab=expression(x[2]))
points(fit$centers,pch=19,col="blue",cex=2)
points(x,col=fit$cluster,pch=19)
```

> fit

K-means clustering with 2 clusters of sizes 49, 51

Cluster means:

	V1	V2
1	0.99128291	1.078988
2	0.02169424	0.088660

Clustering vector:

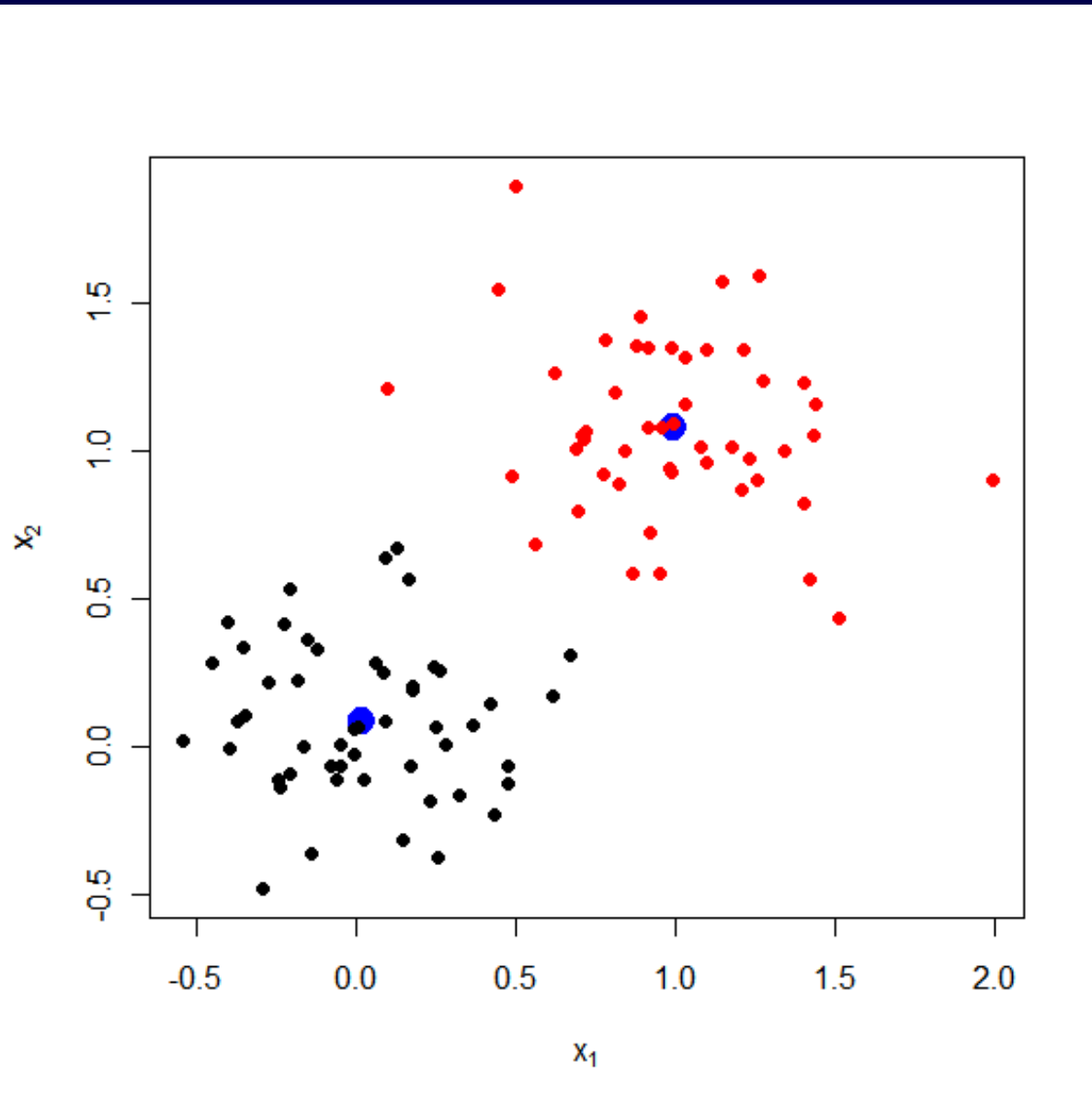
```
[1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
[38] 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

Within cluster sum of squares by cluster:

```
[1] 9.397754 7.489019
```

Available components:

```
[1] "cluster" "centers" "withinss" "size"
>
```



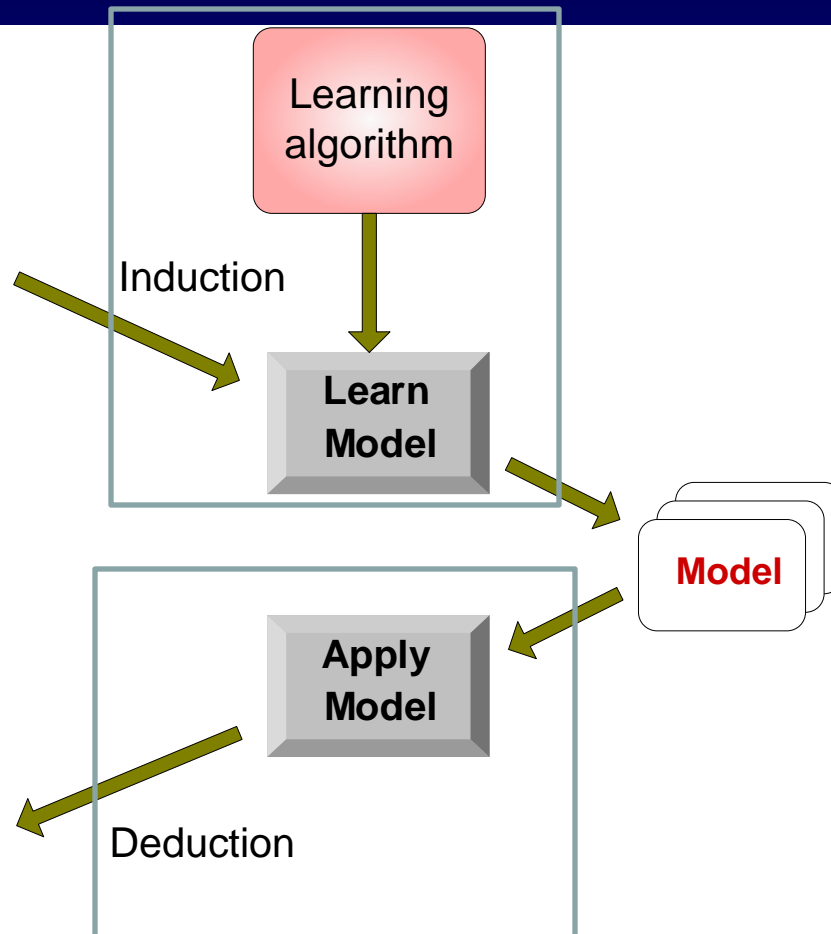
Classification Tasks

Tid	Attrib1	Attrib2	Attrib3	Class
1	Yes	Large	125K	No
2	No	Medium	100K	No
3	No	Small	70K	No
4	Yes	Medium	120K	No
5	No	Large	95K	Yes
6	No	Medium	60K	No
7	Yes	Large	220K	No
8	No	Small	85K	Yes
9	No	Medium	75K	No
10	No	Small	90K	Yes

Training Set

Tid	Attrib1	Attrib2	Attrib3	Class
11	No	Small	55K	?
12	Yes	Medium	80K	?
13	Yes	Large	110K	?
14	No	Small	95K	?
15	No	Large	67K	?

Test Set

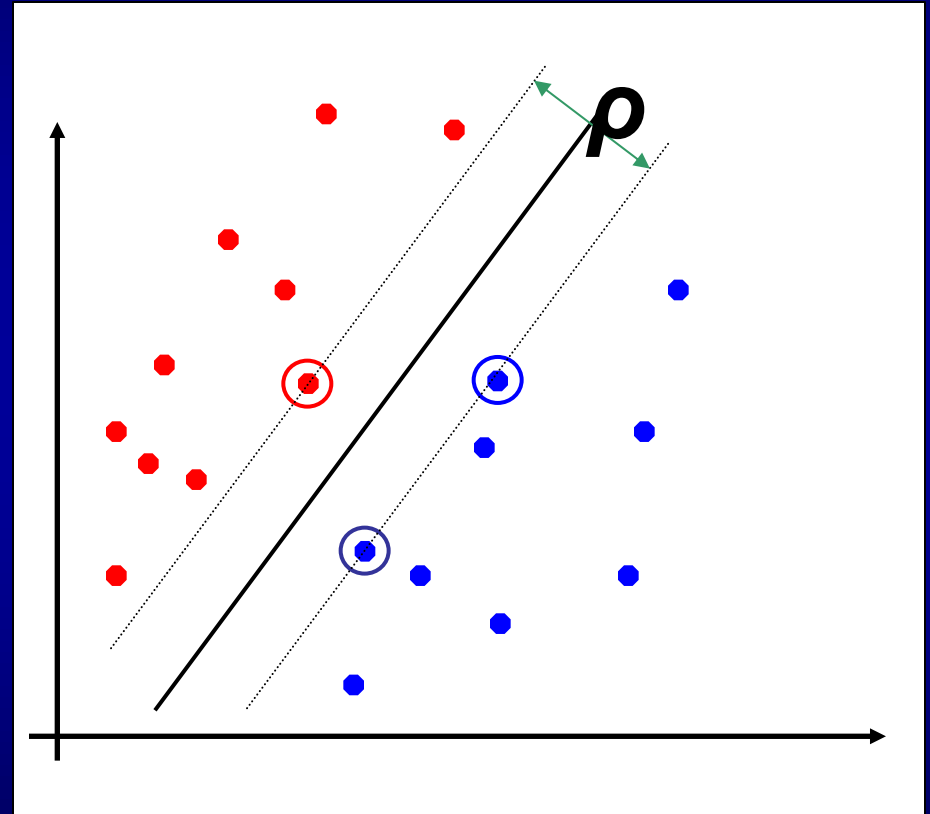


Support Vector Machine Classification

- A distance based classification method.
- The core idea is to find the best hyperplane to separate data from two classes.
- The class of a new object can be determined based on its distance from the hyperplane.

Binary Classification with Linear Separator

- Red and blue dots are representations of objects from two classes in the training data
- The line is a linear separator for the two classes
- The closest objects to the hyperplane is the support vectors.



SVM Classification Example

```
install.packages("e1071")  
library(e1071)  
train<-  
read.csv("sonar_train.csv",header=FALSE)  
y<-as.factor(train[,61])  
x<-train[,1:60]  
fit<-svm(x,y)  
1-sum(y=predict(fit,x))/length(y)
```

SVM Classification Example

```
test<-  
read.csv("sonar_test.csv",header=FALSE)  
y_test<-as.factor(test[,61])  
x_test<-test[,1:60]  
1-  
sum(y_test==predict(fit,x_test))/length  
(y_test)
```

Running R jobs

Running R Session on Stampede

- Interactive mode:
 - Using VNC session
- Batch/script mode:
 - Running R in BATCH mode using job submission script
 - Running R script with parameters.

Running non-interactive R Session

- Interactive R session
 - Good for initial code development and debugging
- Non-interactive R session
 - Including R command and script in the job submission script
 - No need to install/start VNC sessions or java
 - Submitted job and wait for computations to finish

Running R Session in Batch Mode

- ``module`` command:
 - “>module load R”
 - Set up the environment variable for running R
 - make the R executable available in path
- Batch mode
 - “>R CMD BATCH /path/to/R_SCRIPT”
 - Running R script stored in file “R_SCRIPT”
 - By default the result is stored in R_SCRIPTOut

Running R Session in Batch Mode

- R scripts
 - Put the codes you would input when running interactively into a text file. e.g.

```
login1% more mtcars.R
data(mtcars)      # load built-in mtcars data table
attach(mtcars)   # Attaching mtcars names
names(mtcars)    # show column names
summary(mtcars)  # show statistical summary of all columns.
detach()
q()
```

Running R Session in Batch Mode

```
login1% R CMD BATCH mtcars.R
login1% more mtcars.Rout

R version 2.10.1 (2009-12-14)
Copyright (C) 2009 The R Foundation for Statistical Computing
ISBN 3-900051-07-0

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Previously saved workspace restored]

> data(mtcars) # load built-in mtcars data table
> attach(mtcars) # Attaching mtcars names
> names(mtcars) # show column names
 [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"   "gear"
[11] "carb"
```


Running R Session in Batch Mode

```
> names(mtcars) # show column names
[1] "mpg" "cyl" "disp" "hp" "drat" "wt" "qsec" "vs" "am" "gear"
[11] "carb"
> summary(mtcars) # show statistical summary of all columns.
      mpg      cyl      disp      hp
Min.   :10.40  Min.   :4.000  Min.   : 71.1  Min.   : 52.0
1st Qu.:15.43  1st Qu.:4.000  1st Qu.:120.8  1st Qu.: 96.5
Median :19.20  Median :6.000  Median :196.3  Median :123.0
Mean   :20.09  Mean   :6.188  Mean   :230.7  Mean   :146.7
3rd Qu.:22.80  3rd Qu.:8.000  3rd Qu.:326.0  3rd Qu.:180.0
Max.   :33.90  Max.   :8.000  Max.   :472.0  Max.   :335.0

      drat      wt      qsec      vs
Min.   :2.760  Min.   :1.513  Min.   :14.50  Min.   :0.0000
1st Qu.:3.080  1st Qu.:2.581  1st Qu.:16.89  1st Qu.:0.0000
Median :3.695  Median :3.325  Median :17.71  Median :0.0000
Mean   :3.597  Mean   :3.217  Mean   :17.85  Mean   :0.4375
3rd Qu.:3.920  3rd Qu.:3.610  3rd Qu.:18.90  3rd Qu.:1.0000
Max.   :4.930  Max.   :5.424  Max.   :22.90  Max.   :1.0000

      am      gear      carb
Min.   :0.0000  Min.   :3.000  Min.   :1.000
1st Qu.:0.0000  1st Qu.:3.000  1st Qu.:2.000
Median :0.0000  Median :4.000  Median :2.000
Mean   :0.4062  Mean   :3.688  Mean   :2.812
3rd Qu.:1.0000  3rd Qu.:4.000  3rd Qu.:4.000
Max.   :1.0000  Max.   :5.000  Max.   :8.000
> detach()
> q()
> proc.time()
  user  system elapsed
0.192  0.014  0.212
```

Running R Script with Parameters

- Start R in the slave mode
- The R script is provided via stdin
- The parameters is provided as R commnad arguments.
- A simple example:

```
cat sample.R | R --slave --args $1 $2 ...
```

Like
interactive
mode

```
login1% cat sample.R
```

```
arg1 <- as.numeric(commandArgs()[4])
```

```
arg2 <- as.numeric(commandArgs()[5])
```

} Parse arguments

```
paste("Input arguments are ", arg1, arg2, sep=" ")
```

```
paste("The sum is ", arg1+arg2, sep="")
```

} Do something

```
q()
```

```
n
```

```
login1% cat sample.R | R --slave --args 1 2
```

```
[1] "Input arguments are 1 2"
```

```
[1] "The sum is 3"
```

```
login1% cat sample.R | R --slave --args 1231234 54532332
```

```
[1] "Input arguments are 1231234 54532332"
```

```
[1] "The sum is 55763566"
```

Running R Script with Parameters

- Enable more flexibility on computations of same R script.

```
login1% more mtcars.R
data(mtcars)      # load built-in mtcars data table
attach(mtcars)   # Attaching mtcars names
names(mtcars)    # show column names
summary(eval(as.name(commandArgs()[4])))
                 # show statistical summary of object with specified name

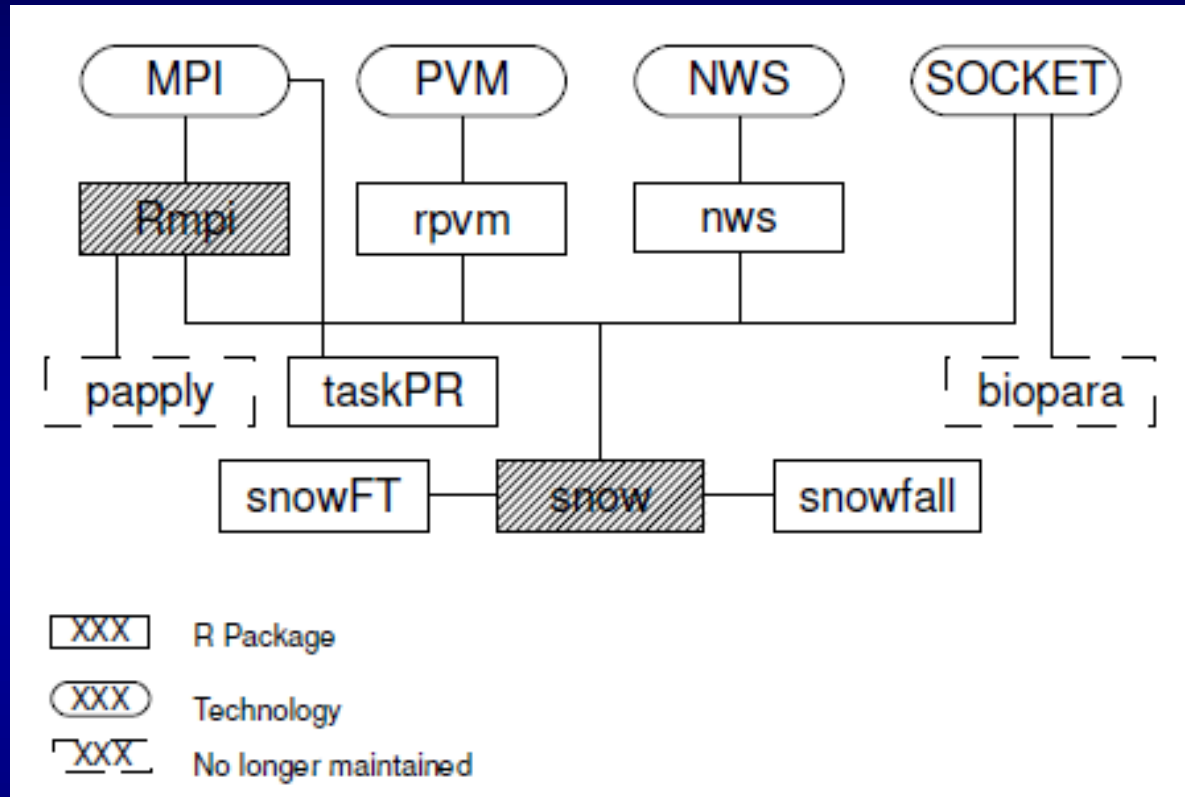
detach()
q()
```

Running R Script with Parameters

```
login1$ cat mtcars2.R | R --slave --args mtcars
[1] "mpg" "cyl" "disp" "hp" "drat" "wt" "qsec" "vs" "am" "gear"
[11] "carb"
      mpg          cyl          disp          hp
Min.   :10.40    Min.   :4.000    Min.   : 71.1    Min.   : 52.0
1st Qu.:15.43    1st Qu.:4.000    1st Qu.:120.8    1st Qu.: 96.5
Median :19.20    Median :6.000    Median :196.3    Median :123.0
Mean   :20.09    Mean   :6.188    Mean   :230.7    Mean   :146.7
3rd Qu.:22.80    3rd Qu.:8.000    3rd Qu.:326.0    3rd Qu.:180.0
Max.   :33.90    Max.   :8.000    Max.   :472.0    Max.   :335.0
      drat          wt          qsec          vs
Min.   :2.760    Min.   :1.513    Min.   :14.50    Min.   :0.0000
1st Qu.:3.080    1st Qu.:2.581    1st Qu.:16.89    1st Qu.:0.0000
Median :3.695    Median :3.325    Median :17.71    Median :0.0000
Mean   :3.597    Mean   :3.217    Mean   :17.85    Mean   :0.4375
3rd Qu.:3.920    3rd Qu.:3.610    3rd Qu.:18.90    3rd Qu.:1.0000
Max.   :4.930    Max.   :5.424    Max.   :22.90    Max.   :1.0000
      am          gear          carb
Min.   :0.0000    Min.   :3.000    Min.   :1.000
1st Qu.:0.0000    1st Qu.:3.000    1st Qu.:2.000
Median :0.0000    Median :4.000    Median :2.000
Mean   :0.4062    Mean   :3.688    Mean   :2.812
3rd Qu.:1.0000    3rd Qu.:4.000    3rd Qu.:4.000
Max.   :1.0000    Max.   :5.000    Max.   :8.000
login1$ cat mtcars2.R | R --slave --args mpg
[1] "mpg" "cyl" "disp" "hp" "drat" "wt" "qsec" "vs" "am" "gear"
[11] "carb"
      Min. 1st Qu. Median      Mean 3rd Qu.      Max.
 10.40  15.42  19.20  20.09  22.80  33.90
login1$ cat mtcars2.R | R --slave --args cyl
[1] "mpg" "cyl" "disp" "hp" "drat" "wt" "qsec" "vs" "am" "gear"
[11] "carb"
      Min. 1st Qu. Median      Mean 3rd Qu.      Max.
   4.000  4.000  6.000  6.188  8.000  8.000
```

Break Big Computations

- Running R with Command Line parameters
 - Similar to run R batch mode
 - Parameters is specified on the command line
 - Good for repeated runs of same computations or running script partially
- Running R with Parallel package
 - Multicore
 - Snow/Rmpi



- M. Schmidberger, M. Morgan, D. Eddebuettel, H. Yu, L. Tierney, and U. Mansmann, "State-of-the-art in Parallel Computing with R," Journal of Statistical Software, vol. 47, 2009.

Multicore

- Utilizes multiple processing core within the same node.
- Replace several common functions with parallel implementations
- No need of significant changes on the existing coding process control.
- Scalability is limited by the number of core and memory available within single node

Multicore -- mcapply

- lapply → mcapply
 - lapply(1:30, rnorm)
 - mclapply(1:30, rnorm)
- mc.cores
 - The maximum number of cores to use
- mc.preschedule
 - TRUE, computation is first divided by the number of cores.
 - FALSE, one job is spawned for each value sequentially

Multicore –parallel and collect

- `parallel(expr, name, mc.set.seed = FALSE, silent = FALSE)`
 - Starts a parallel process for evaluating `expr`,
- `collect(jobs, wait = TRUE, timeout = 0, intermediate = FALSE)`
 - Collects the result from the parallel process.

```
p <- parallel(1:10)
```

```
q <- parallel(1:20)
```

```
collect(list(p, q)) # wait for jobs to finish and collect all results
```

Snow

- Developed Based on Rmpi package,
- Simplify the process to initialize parallel process over cluster.

```
cl <- makeCluster(4, type='SOCK')
```

```
birthday <- function(n) {  
  ntests <- 1000  
  pop <- 1:365  
  anydup <- function(i)  
    any(duplicated(  
      sample(pop, n, replace=TRUE)))  
  sum(sapply(seq(ntests), anydup)) / ntests}  
}
```

```
x <- foreach(j=1:100) %dopar% birthday (j)
```

```
stopCluster(cl)
```

Ref: <http://www.rinfinance.com/RinFinance2009/presentations/UIC-Lewis%204-25-09.pdf>

Snow

- Provide similar MPI functions on snow cluster:
 - clusterSplit, clusterCall, ClusterEvalQ, clusterApply,

```
clusterApply(cl, 1:2, get("+"), 3)
```

```
clusterEvalQ(cl, library(boot))
```

```
x<-1
```

```
clusterExport(cl, "x")
```

```
clusterCall(cl, function(y) x + y, 2)
```

Snow

- Provide parallel version of common functions:
 - parLapply, parApply, parSapply
 - Similar to mcapply from mutlicore
 - Need to setup the snow cluster first

```
cl <- makeCluster(4, type='SOCK')  
parSapply(cl, 1:20, get("+"), 3)
```

Running R with MPI

- Packages are available to provide interface from R script to the MPI calls
 - Rmpi
 - <http://www.stats.uwo.ca/faculty/yu/Rmpi/>
 - Basic and good interface to MPI library
 - More on this in lab session.
 - Pbd-R
 - <http://r-pbd.org/>
 - Include pdb-MPI, and several other specialized packages.
 - More on this in lab session.
- Other similar/derived module
 - doMPI
 - Parallel R

Further references

- R
 - M. Crawley, *Statistics An Introduction using R*, Wiley
 - J. Verzani, *SimpleR Using R for Introductory Statistics*
<http://cran.r-project.org/doc/contrib/Verzani-SimpleR.pdf>
 - Programming manual:
 - <http://cran.r-project.org/manuals.html>
- Using R for data mining
 - *Data Mining with R: Learning with case studies*, Luis Togo