Data Mining with R part 2

วิชา การค้นพบองค์ความรู้และการทำเหมืองข้อมูลชั้นสูง

Veerasak Kritsanapraphan

Chulalongkorn University

Email : veerasak.kr568@cbs.chula.ac.th

@veerasakk

Agenda

- Data Mining Techniques using R
  - Neural Network
  - Clustering
    - K-Means Clustering
    - Hierarchical Clustering
  - Association Rules
  - Multi-mobile Learning
- Parallel Computing using R
  - Hadoop

Slide and Source Codes

https://github.com/vkrit/chula_datamining

Prepare Data

- iris <- read.csv("iris.data.csv", header=TRUE)
- # Prepare iris
- set.seed(567)
- ind <- sample(2, nrow(iris), replace=TRUE, prob=c(0.7, 0.3))
- trainData <- iris[ind==1,]
- testData <- iris[ind==2,]

Neural Network

Training of neural networks

Description

`neuralnet` is used to train neural networks using backpropagation, resilient backpropagation (RPROP) with (Riedmiller, 1994) or without weight backtracking (Riedmiller and Braun, 1993) or the modified globally convergent version (GRPROP) by Anastasiadis et al. (2005). The function allows flexible settings through custom-choice of error and activation function. Furthermore the calculation of generalized weights (Intrator O. and Intrator N., 1993) is implemented.

Usage

```
neuralnet(formula, data, hidden = 1, threshold = 0.01,
          stepmax = 1e+05, rep = 1, startweights = NULL,
          learningrate.limit = NULL,
          learningrate.factor = list(minus = 0.5, plus = 1.2),
          learningrate=NULL, lifesign = "none",
          lifesign.step = 1000, algorithm = "rprop+",
          err.fct = "sse", act.fct = "logistic",
          linear.output = TRUE, exclude = NULL,
          constant.weights = NULL, likelihood = FALSE)
```

Neural Network

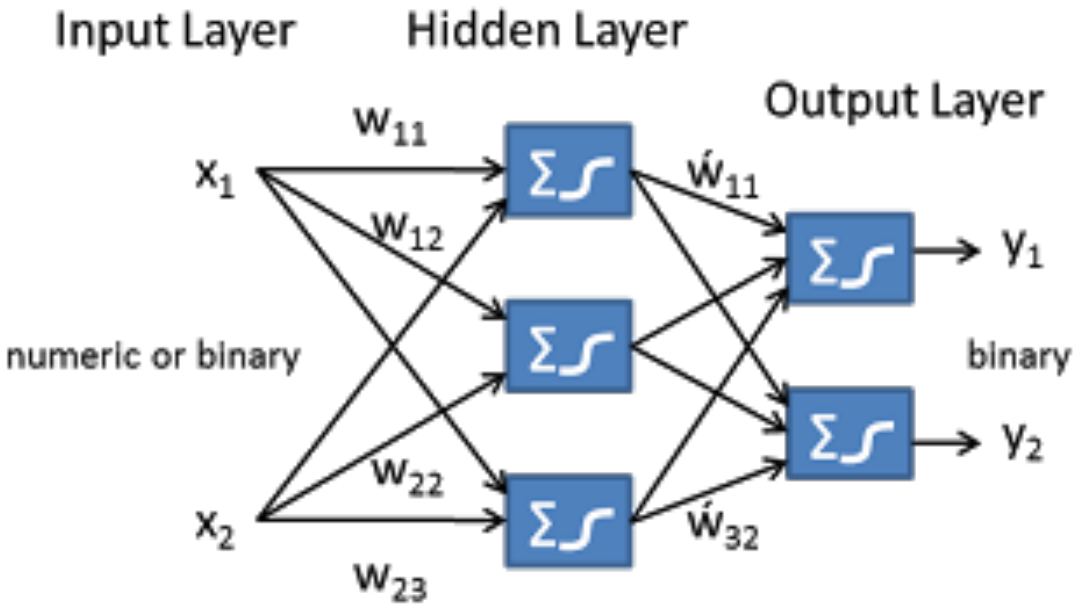
Arguments

`formula`	a symbolic description of the model to be fitted.
`data`	a data frame containing the variables specified in `formula`.
`hidden`	a vector of integers specifying the number of hidden neurons (vertices) in each layer.
`threshold`	a numeric value specifying the threshold for the partial derivatives of the error function as stopping criteria.
`stepmax`	the maximum steps for the training of the neural network. Reaching this maximum leads to a stop of the neural network's training process.
`rep`	the number of repetitions for the neural network's training.
`startweights`	a vector containing starting values for the weights. The weights will not be randomly initialized.
`learningrate.limit`	a vector or a list containing the lowest and highest limit for the learning rate. Used only for RPROP and GRPROP.
`learningrate.factor`	a vector or a list containing the multiplication factors for the upper and lower learning rate. Used only for RPROP and GRPROP.
`learningrate`	a numeric value specifying the learning rate used by traditional backpropagation. Used only for traditional backpropagation.
`lifesign`	a string specifying how much the function will print during the calculation of the neural network. 'none', 'minimal' or 'full'.

Neural Network

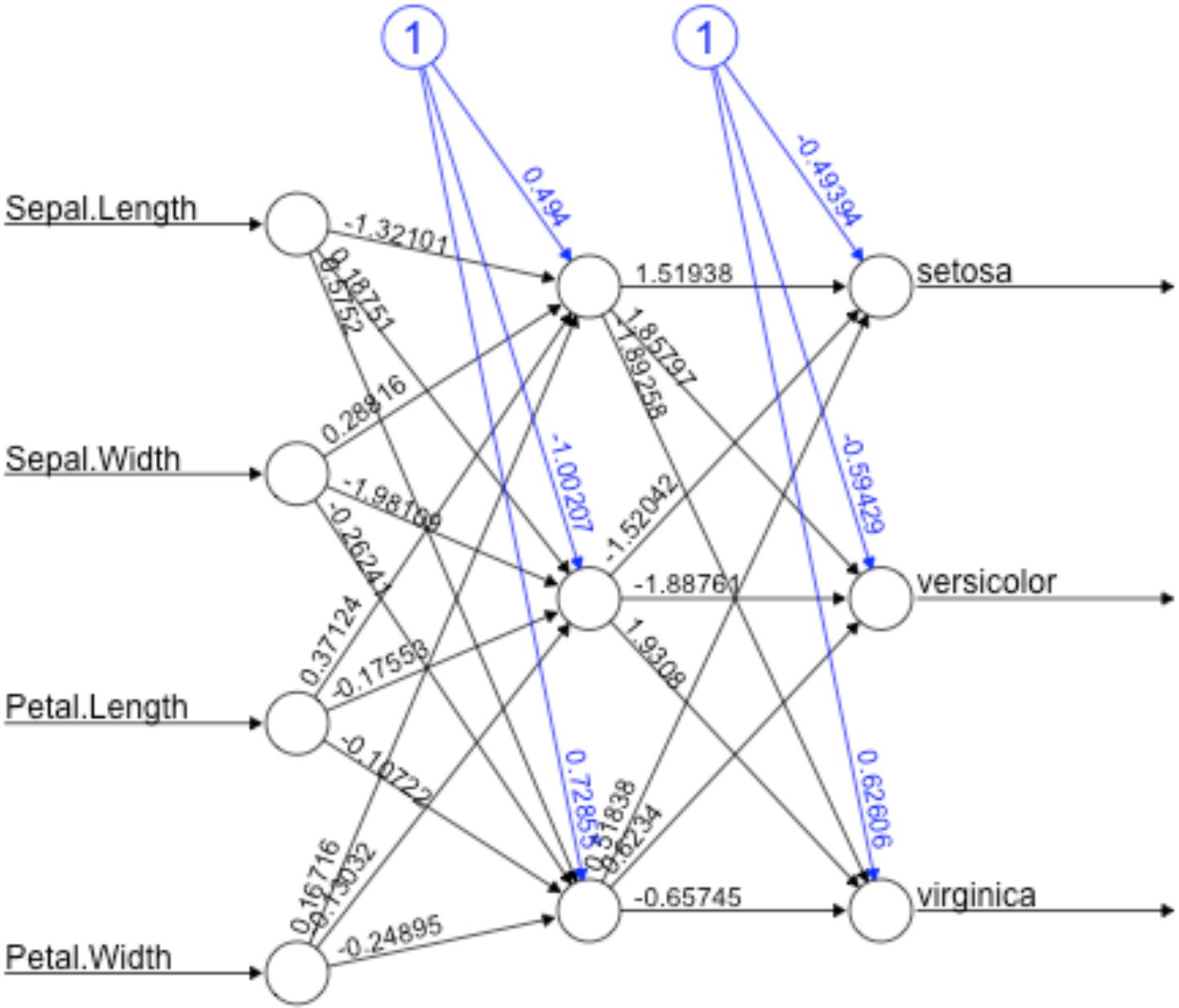
`lifesign.step`	an integer specifying the stepsize to print the minimal threshold in full lifesign mode.
`algorithm`	a string containing the algorithm type to calculate the neural network. The following types are possible: 'backprop', 'rprop+', 'rprop-', 'sag', or 'slr'. 'backprop' refers to backpropagation, 'rprop+' and 'rprop-' refer to the resilient backpropagation with and without weight backtracking, while 'sag' and 'slr' induce the usage of the modified globally convergent algorithm (grprop). See Details for more information.
`err.fct`	a differentiable function that is used for the calculation of the error. Alternatively, the strings 'sse' and 'ce' which stand for the sum of squared errors and the cross-entropy can be used.
`act.fct`	a differentiable function that is used for smoothing the result of the cross product of the covariate or neurons and the weights. Additionally the strings, 'logistic' and 'tanh' are possible for the logistic function and tangent hyperbolicus.
`linear.output`	logical. If act.fct should not be applied to the output neurons set linear output to TRUE, otherwise to FALSE.
`exclude`	a vector or a matrix specifying the weights, that are excluded from the calculation. If given as a vector, the exact positions of the weights must be known. A matrix with n-rows and 3 columns will exclude n weights, where the first column stands for the layer, the second column for the input neuron and the third column for the output neuron of the weight.
`constant.weights`	a vector specifying the values of the weights that are excluded from the training process and treated as fix.
`likelihood`	logical. If the error function is equal to the negative log-likelihood function, the information criteria AIC and BIC will be calculated. Furthermore the usage of confidence.interval is meaningfull.

Neural Network



```
> library(neuralnet)
> nnet_iristrain <-iristrain
> #Binarize the categorical output
> nnet_iristrain <- cbind(nnet_iristrain,
                          iristrain$Species == 'setosa')
> nnet_iristrain <- cbind(nnet_iristrain,
                          iristrain$Species == 'versicolor')
> nnet_iristrain <- cbind(nnet_iristrain,
                          iristrain$Species == 'virginica')
> names(nnet_iristrain)[6] <- 'setosa'
> names(nnet_iristrain)[7] <- 'versicolor'
> names(nnet_iristrain)[8] <- 'virginica'
> nn <- neuralnet(setosa+versicolor+virginica ~
                  Sepal.Length+Sepal.Width
                              +Petal.Length
                              +Petal.Width,
                  data=nnet_iristrain,
                  hidden=c(3))
> plot(nn)
> mypredict <- compute(nn, iristest[-5])$net.result
> # Put multiple binary output to categorical output
> maxidx <- function(arr) {
      return(which(arr == max(arr)))
  }
> idx <- apply(mypredict, c(1), maxidx)
> prediction <- c('setosa', 'versicolor', 'virginica')[idx]
> table(prediction, iristest$Species)

prediction   setosa versicolor virginica
  setosa         10          0         0
  versicolor      0         10         3
  virginica       0          0         7
```



Error: 0.001277 Steps: 281

Clustering

- K-Means Clustering
- Hierarchical Clustering

K-Means Clustering

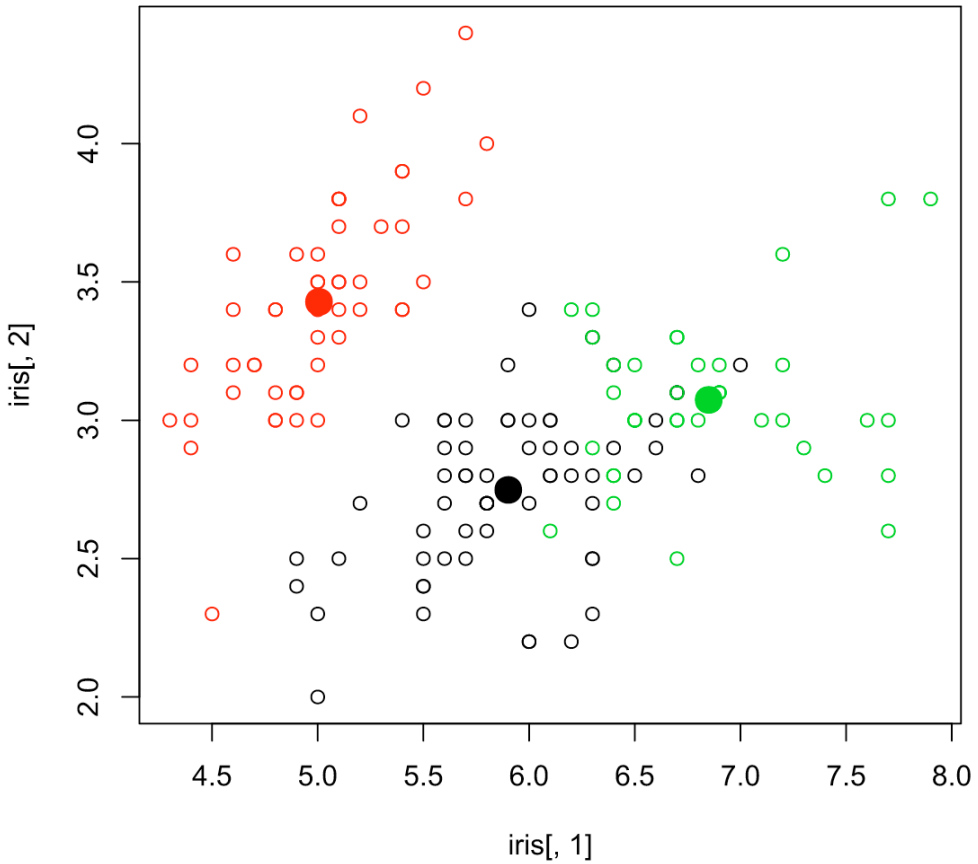
1. Pick an initial set of K centroids (this can be random or any other means)

2. For each data point, assign it to the member of the closest centroid according to the given distance function

3. Adjust the centroid position as the mean of all its assigned member data points. Go back to (2) until the membership isn't change and centroid position is stable.

4. Output the centroids.

- library(stats)
- set.seed(101)
- km <- kmeans(iris[,1:4], 3)
- plot(iris[,1], iris[,2], col=km$cluster)
- points(km$centers[,c(1,2)], col=1:3, pch=19, cex=2)

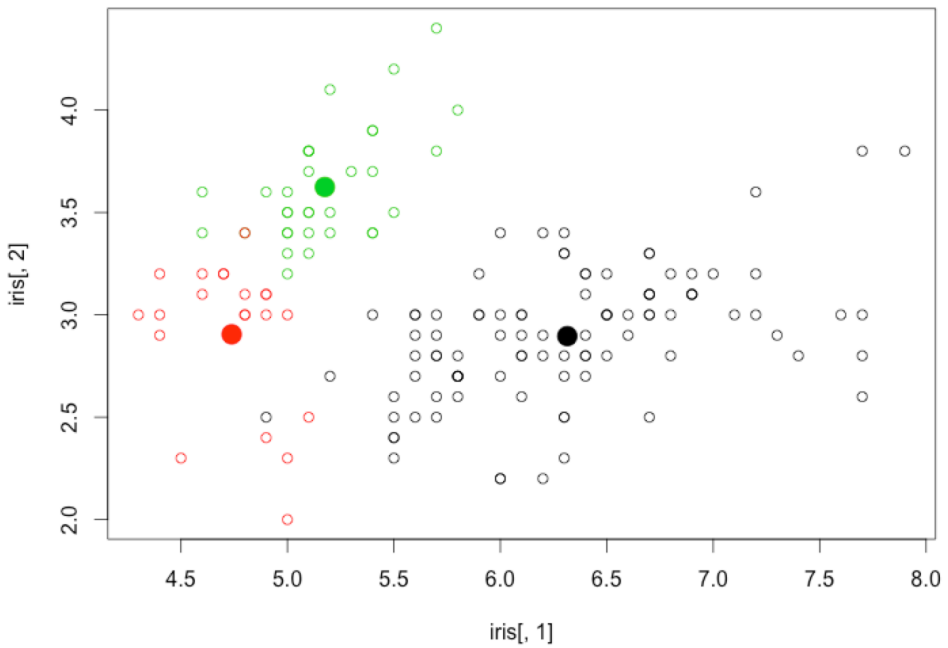


- table(km$cluster, iris$Species)

```
   setosa versicolor virginica
1       0         48        14
2      50          0         0
3       0          2        36
>
```

Another round

- set.seed(900)
- km <- kmeans(iris[,1:4], 3)
- plot(iris[,1], iris[,2], col=km$cluster)
- points(km$centers[,c(1,2)], col=1:3, pch=19, cex=2)



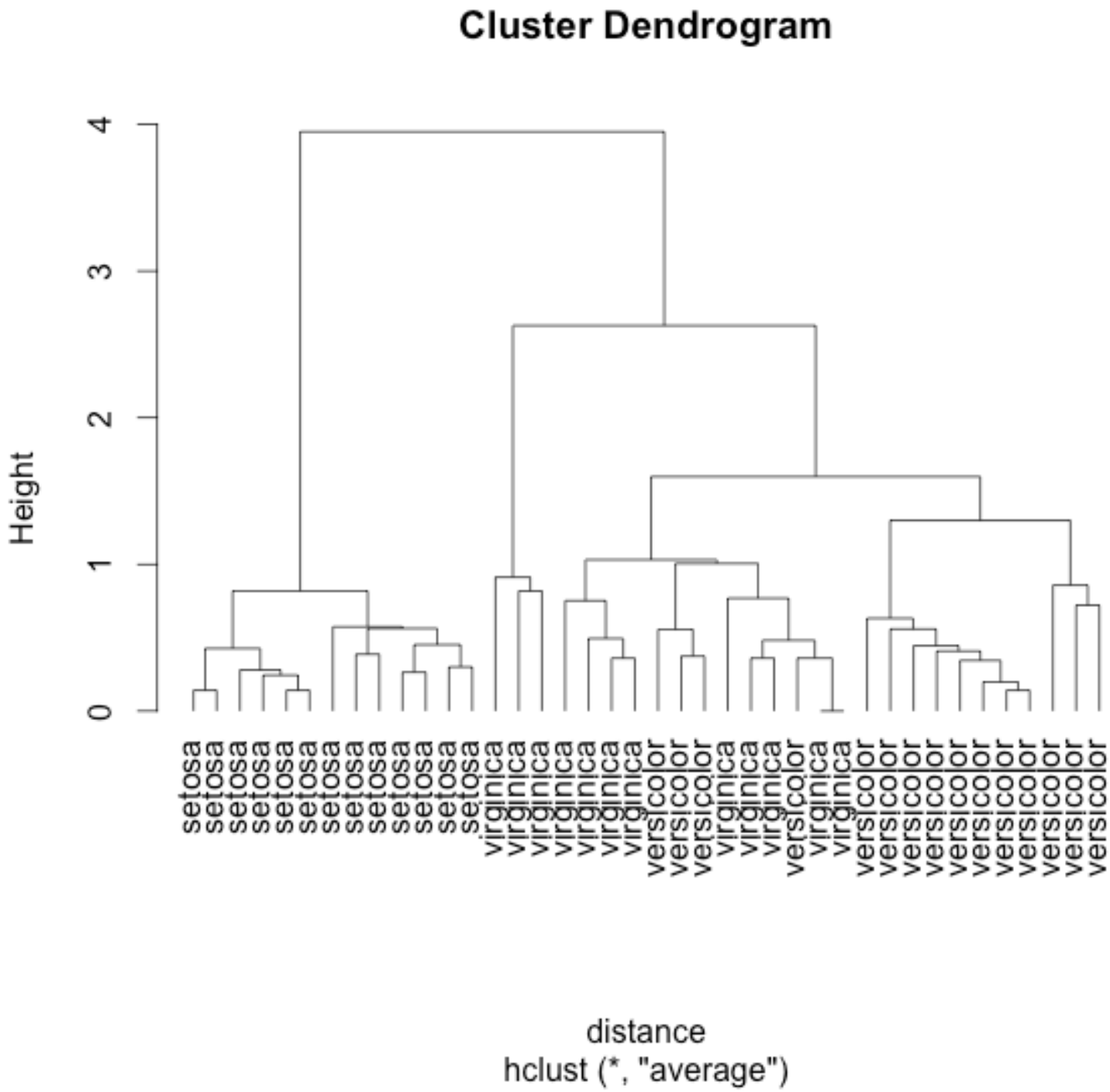
- table(km$cluster, iris$Species)

```
   setosa versicolor virginica
1       0         46        50
2      17          4         0
3      33          0         0
>
```

Hierarchical Clustering

- Compute distance between every pairs of point/cluster.
    - (a) Distance between point is just using the distance function.
    - (b) Compute distance between pointA to clusterB may involve many choices (such as the min/max/avg distance between the pointA and points in the clusterB).
    - (c) Compute distance between clusterA to clusterB may first compute distance of all points pairs (one from clusterA and the other from clusterB) and then pick either min/max/avg of these pairs.
- Combine the two closest point/cluster into a cluster. Go back to (1) until only one big cluster remains

- set.seed(101)
- sampleiris <- iris[sample(1:150, 40),] # get samples from iris dataset
- # each observation has 4 variables, ie, they are interpreted as 4-D points
- distance <- dist(sampleiris[,-5], method="euclidean")
- cluster <- hclust(distance, method="average")
- plot(cluster, hang=-1, label=sampleiris$Species)

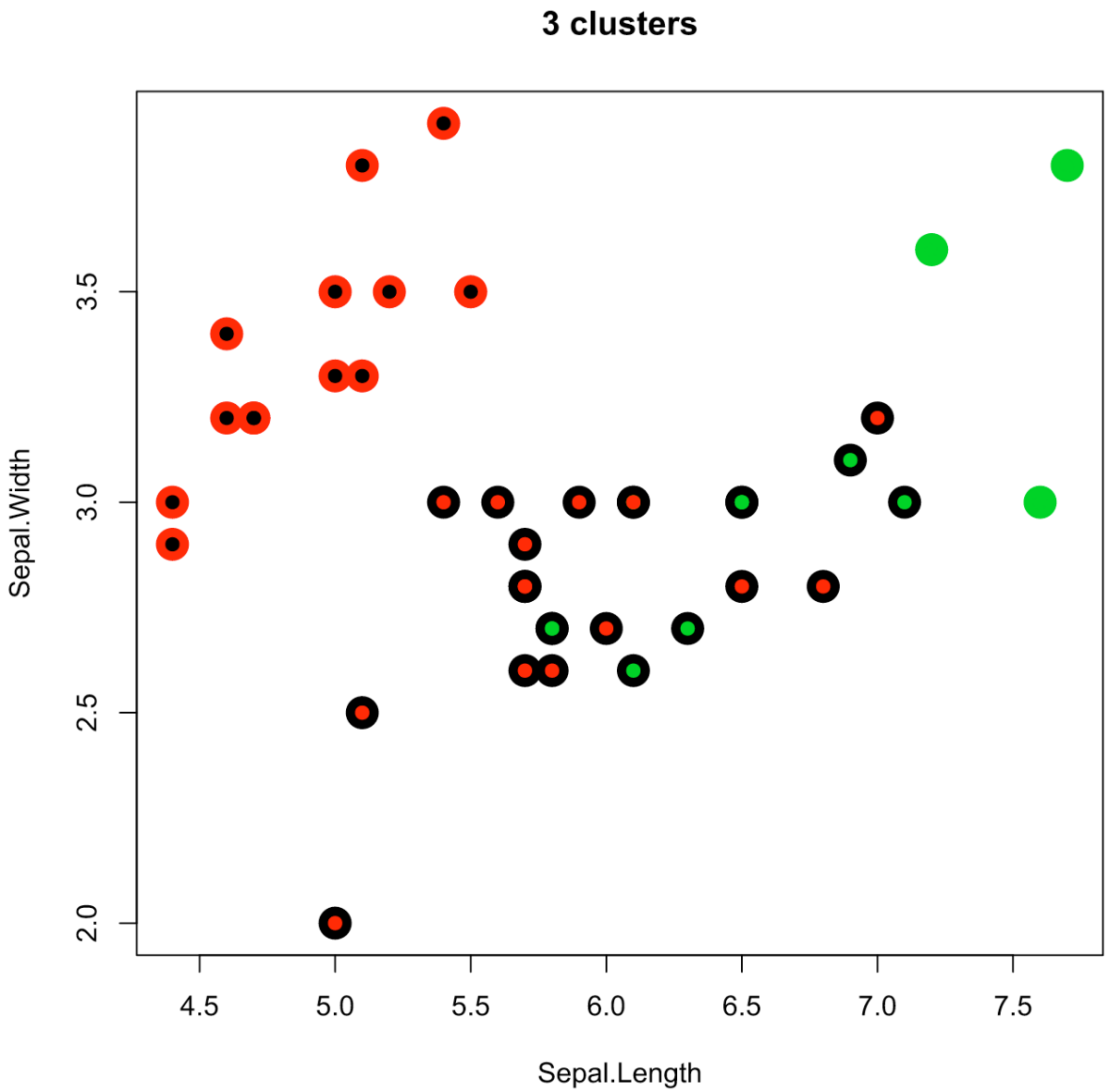

Cluster Dendrogram
Height
0
1
2
3
4
setosa
virginica
versicolor
distance
hclust (*, "average")

It’s possible to prune the result tree.

- par(mfrow=c(1,2))
- group.3 <- cutree(cluster, k = 3) # prune the tree by 3 clusters
- table(group.3, sampleiris$Species) # compare with known classes

```
group.3 setosa versicolor virginica
      1      0         15         9
      2     13          0         0
      3      0          0         3
> 
```

- plot(sampleiris[,c(1,2)], col=group.3, pch=19, cex=2.5, main="3 clusters")
- points(sampleiris[,c(1,2)], col=sampleiris$Species, pch=19, cex=1)



Association Rules (Market Basket Analysis)

- **Support**: The fraction of which our item set occurs in our dataset.
- **Confidence**: probability that a rule is correct for a new transaction with items on the left.
- **Lift**: The ratio by which by the confidence of a rule exceeds the expected confidence.
- **Note**: if the lift is 1 it indicates that the items on the left and right are independent

Apriori Algorithm

- ?apriori

Usage

```
apriori(data, parameter = NULL, appearance = NULL, control = NULL)
```

Arguments

`data`	object of class `transactions` or any data structure which can be coerced into `transactions` (e.g., a binary matrix or data.frame).
`parameter`	object of class `APparameter` or named list. The default behavior is to mine rules with support 0.1, confidence 0.8, and maxlen 10.
`appearance`	object of class `APappearance` or named list. With this argument item appearance can be restricted. By default all items can appear unrestricted.
`control`	object of class `APcontrol` or named list. Controls the performance of the mining algorithm (item sorting, etc.)

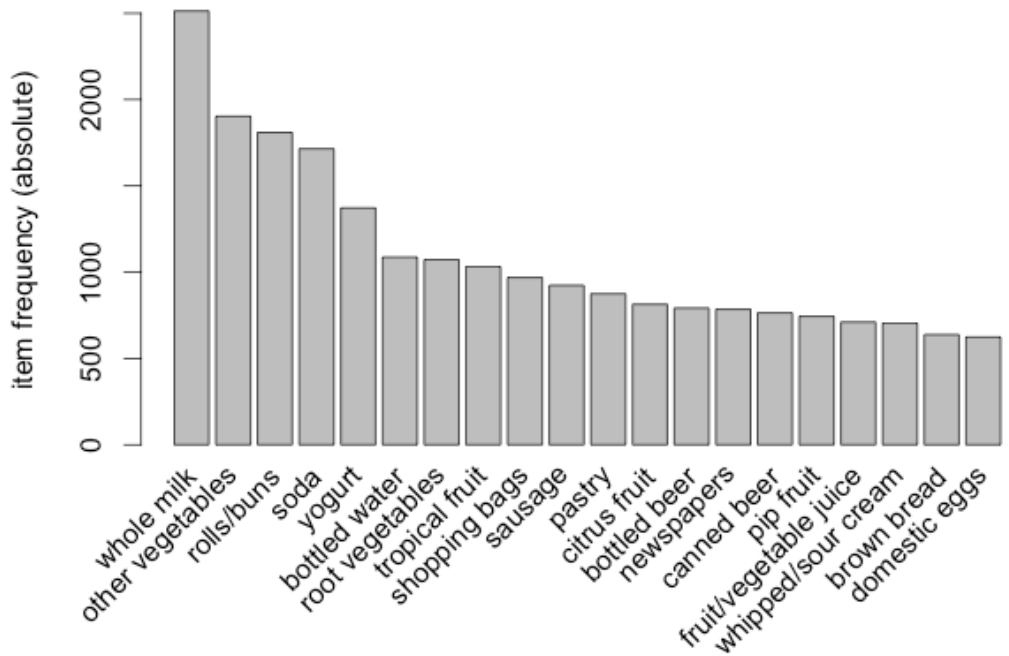
Apriori Algorithm

So lets get started by loading up our libraries and data set.

- # Load the libraries
- library(arules)
- library(arulesViz)
- library(datasets)
-
- # Load the data set
- data(Groceries)

Explore Data

- # Create an item frequency plot for the top 20 items
- itemFrequencyPlot(Groceries, topN=20,type="absolute")



- **rules <- apriori(Groceries, parameter = list(supp = 0.001, conf = 0.8))**
- # Show the top 5 rules, but only 2 digits
- options(digits=2)
- inspect(rules[1:5])

```
  lhs                     rhs            support confidence lift
1 {liquor,
   red/blush wine} => {bottled beer}  0.0019       0.90 11.2
2 {curd,
   cereals}        => {whole milk}    0.0010       0.91  3.6
3 {yogurt,
   cereals}        => {whole milk}    0.0017       0.81  3.2
4 {butter,
   jam}            => {whole milk}    0.0010       0.83  3.3
5 {soups,
   bottled beer}   => {whole milk}    0.0011       0.92  3.6
>
```

- summary(rules)

```
set of 410 rules

rule length distribution (lhs + rhs):sizes
  3   4   5   6
 29 229 140  12

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    3.0     4.0     4.0     4.3     5.0     6.0

summary of quality measures:
    support           confidence        lift
 Min.   :0.00102   Min.   :0.80   Min.   : 3.1
 1st Qu.:0.00102   1st Qu.:0.83   1st Qu.: 3.3
 Median :0.00122   Median :0.85   Median : 3.6
 Mean   :0.00125   Mean   :0.87   Mean   : 4.0
 3rd Qu.:0.00132   3rd Qu.:0.91   3rd Qu.: 4.3
 Max.   :0.00315   Max.   :1.00   Max.   :11.2

mining info:
      data ntransactions support confidence
 Groceries          9835   0.001        0.8
```

- # Sort Rules
- rules<-sort(rules, by="confidence", decreasing=TRUE)
- inspect(rules[1:5])

```
  lhs                        rhs           support confidence lift
1 {rice,
   sugar}                 => {whole milk}  0.0012          1  3.9
2 {canned fish,
   hygiene articles}      => {whole milk}  0.0011          1  3.9
3 {root vegetables,
   butter,
   rice}                  => {whole milk}  0.0010          1  3.9
4 {root vegetables,
   whipped/sour cream,
   flour}                 => {whole milk}  0.0017          1  3.9
5 {butter,
   soft cheese,
   domestic eggs}         => {whole milk}  0.0010          1  3.9
>
```

Change to have limit association in one rule

- # change to have maximum of 3
- rules <- apriori(Groceries, parameter = list(supp = 0.001, conf = 0.8,maxlen=3))
- inspect(rules[1:5])

```
  lhs                     rhs             support confidence lift
1 {liquor,
   red/blush wine} => {bottled beer}  0.0019       0.90 11.2
2 {curd,
   cereals}        => {whole milk}    0.0010       0.91  3.6
3 {yogurt,
   cereals}        => {whole milk}    0.0017       0.81  3.2
4 {butter,
   jam}            => {whole milk}    0.0010       0.83  3.3
5 {soups,
   bottled beer}   => {whole milk}    0.0011       0.92  3.6
```

Rules pruned

- subset.matrix <- is.subset(rules, rules)
- subset.matrix[lower.tri(subset.matrix, diag=T)] <- NA
- redundant <- colSums(subset.matrix, na.rm=T) >= 1
- rules.pruned <- rules[!redundant]
- rules<-rules.pruned
- summary(rules)

```
set of 330 rules

rule length distribution (lhs + rhs):sizes
  3   4   5   6
 29 216  84   1

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    3.0     4.0     4.0     4.2     5.0     6.0

summary of quality measures:
    support         confidence        lift
 Min.   :0.00102   Min.   :0.80   Min.   : 3.1
 1st Qu.:0.00102   1st Qu.:0.82   1st Qu.: 3.3
 Median :0.00122   Median :0.85   Median : 3.6
 Mean   :0.00127   Mean   :0.86   Mean   : 3.8
 3rd Qu.:0.00132   3rd Qu.:0.91   3rd Qu.: 4.3
 Max.   :0.00315   Max.   :1.00   Max.   :11.2

mining info:
      data ntransactions support confidence
 Groceries          9835   0.001        0.8
```

Targeting Items

- What are customers likely to buy before buying whole milk?
- What are customers likely to buy if they purchase whole milk?
- This essentially means we want to set either the Left Hand Side and Right Hand Side. This is not difficult to do with R!

Likely to buy after buy whole milk

- rules<-apriori(data=Groceries, parameter=list(supp=0.001,conf = 0.08), appearance = list(default="lhs",rhs="whole milk"), control = list(verbose=F))

- rules<-sort(rules, decreasing=TRUE,by="confidence")

- inspect(rules[1:5])

```
  lhs                        rhs          support confidence lift
1 {rice,
   sugar}                 => {whole milk} 0.0012          1  3.9
2 {canned fish,
   hygiene articles}      => {whole milk} 0.0011          1  3.9
3 {root vegetables,
   butter,
   rice}                  => {whole milk} 0.0010          1  3.9
4 {root vegetables,
   whipped/sour cream,
   flour}                 => {whole milk} 0.0017          1  3.9
5 {butter,
   soft cheese,
   domestic eggs}         => {whole milk} 0.0010          1  3.9
>
```

Find whole milk's antecedents

- rules<-apriori(data=Groceries, parameter=list(supp=0.001,conf = 0.15,minlen=2), appearance = list(default="rhs",lhs="whole milk"), control = list(verbose=F))

- rules<-sort(rules, decreasing=TRUE,by="confidence")

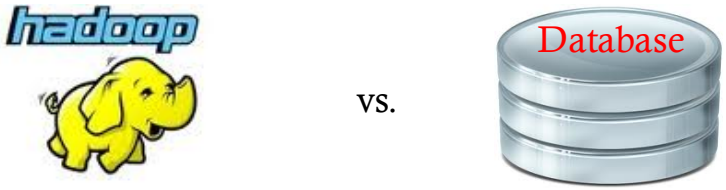
- inspect(rules[1:5])

```
  lhs                rhs                    support confidence lift
1 {whole milk} => {other vegetables}   0.075       0.29  1.5
2 {whole milk} => {rolls/buns}         0.057       0.22  1.2
3 {whole milk} => {yogurt}             0.056       0.22  1.6
4 {whole milk} => {root vegetables}    0.049       0.19  1.8
5 {whole milk} => {tropical fruit}     0.042       0.17  1.6
>
```

Hadoop and Map-reduce Paradigm

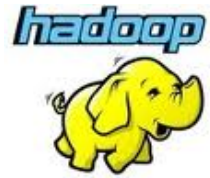
Large-Scale Data Analytics

- MapReduce computing paradigm (E.g., Hadoop) vs. Traditional database systems

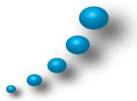


- Many enterprises are turning to Hadoop
  - Especially applications generating big data
  - Web applications, social networks, scientific applications

Why Hadoop is able to compete?



vs.



Scalability (petabytes of data, thousands of machines)

Flexibility in accepting all data formats (no schema)

Efficient and simple fault-tolerant mechanism

Commodity inexpensive hardware

Performance (tons of indexing, tuning, data organization tech.)

Features:

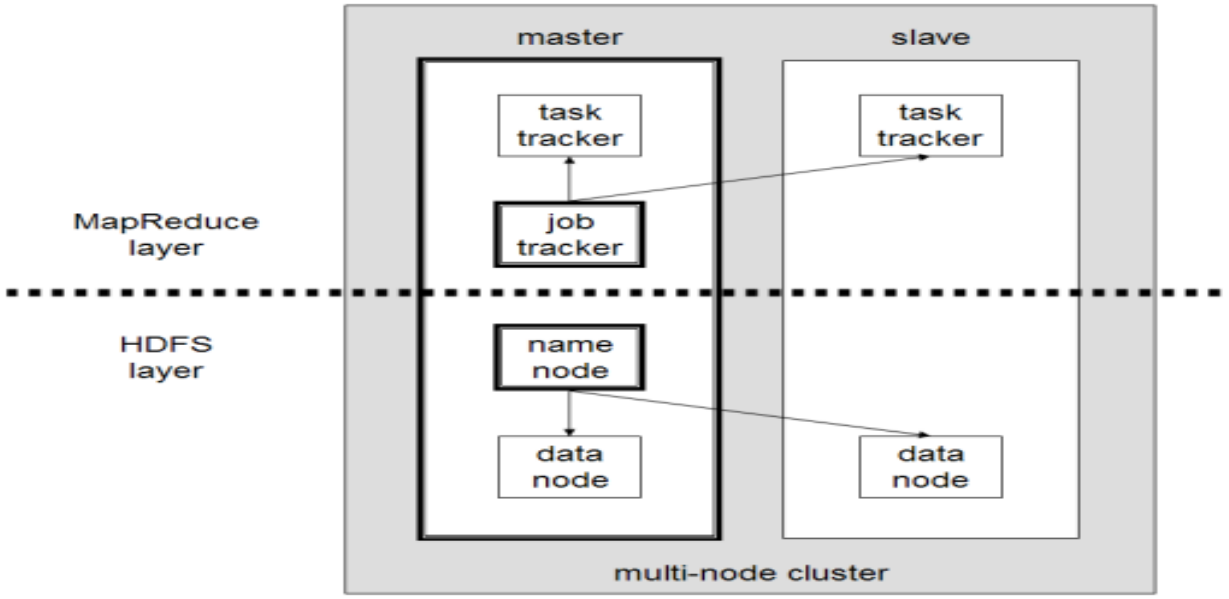
- Provenance tracking
- Annotation management
- ….

What is Hadoop?

- Hadoop is a software framework for *distributed processing* of *large datasets* across *large clusters* of computers
  - ***Large datasets*** → Terabytes or petabytes of data
  - ***Large clusters*** → hundreds or thousands of nodes
- Hadoop is open-source implementation for Google ***MapReduce***
- Hadoop is based on a simple programming model called *MapReduce*
- Hadoop is based on a simple data model, *any data will fit*

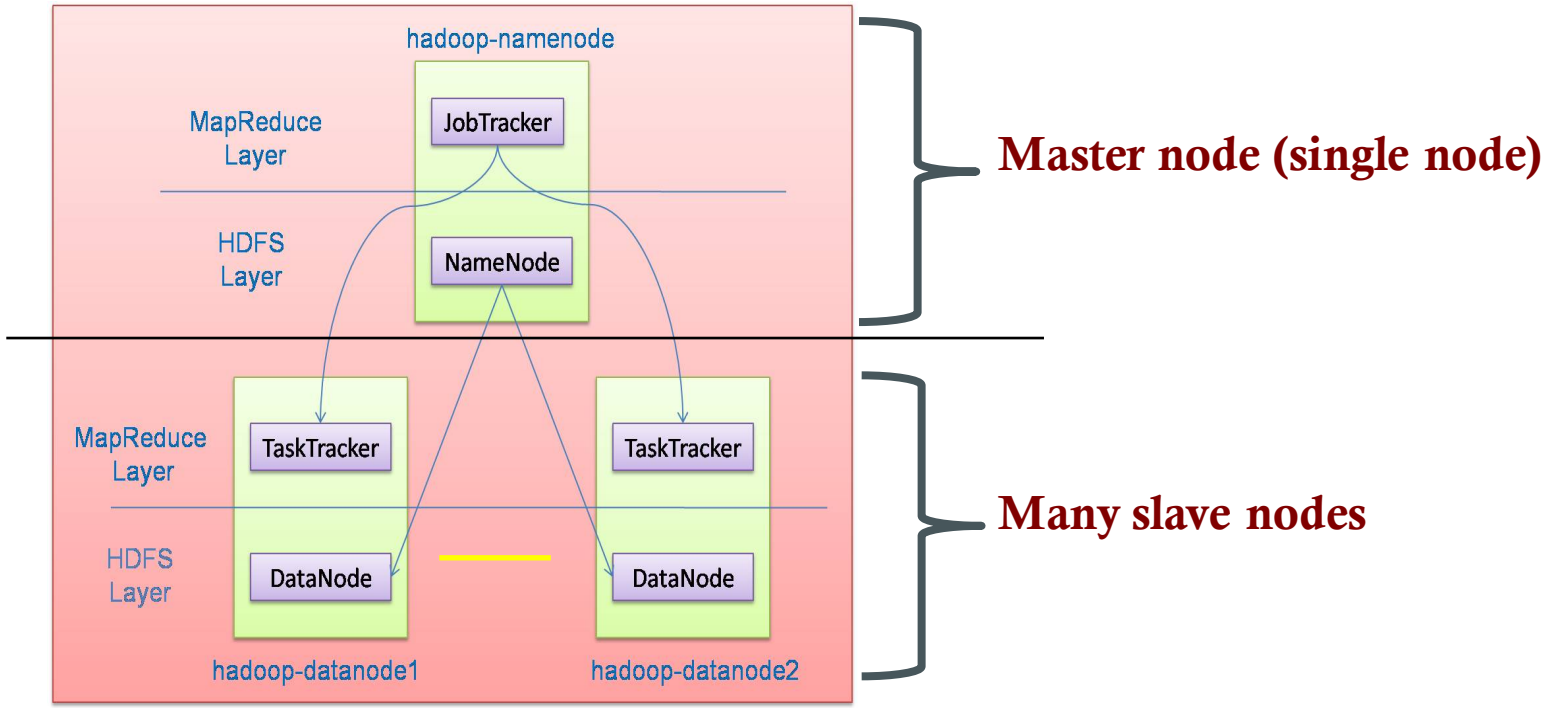
What is Hadoop?

- **Hadoop framework consists on two main layers**
  - Distributed file system (HDFS)
  - Execution engine (MapReduce)



Hadoop Architecture

- Hadoop is designed as a *master-slave shared-nothing* architecture



Design principles of Hadoop

- Need to process big data
- Need to parallelize computation across thousands of nodes
- **Commodity hardware**
  - Large number of low-end cheap machines working in parallel to solve a computing problem
- This is in contrast to **Parallel DBs**
  - Small number of high-end expensive machines

Design principles of Hadoop

- **Automatic parallelization & distribution**
  - Hidden from the end-user
- **Fault tolerance and automatic recovery**
  - Nodes/tasks will fail and will recover automatically
- **Clean and simple programming abstraction**
  - Users only provide two functions “map” and “reduce”

RHadoop

- install.packages(c('rJava','RJSONIO', 'itertools', 'digest','Rcpp','httr','functional','devtools', 'plyr','reshape2')

- Sys.setenv("HADOOP_CMD"="/usr/local/Cellar/hadoop/2.6.0/bin/hadoop")

- Sys.setenv("HADOOP_STREAMING"="/usr/local/Cellar/hadoop/2.6.0/libexec/share/hadoop/tools/lib/hadoop-streaming-2.6.0.jar")

- Sys.getenv("HADOOP_CMD")

- Sys.setenv("HADOOP_HOME"="/usr/local/Cellar/hadoop/2.6.0")

Install RHadoop

- Installing RHadoop [rhdfs, rmr, rhbase]
  1. Download RHadoop packages from GitHub repository of Revolution Analytics: https://github.com/RevolutionAnalytics/RHadoop
    - rmr: [rmr-2.2.2.tar.gz]
    - rhdfs: [rhdfs-1.6.0.tar.gz]
    - rhbase: [rhbase-1.2.0.tar.gz]

2. Installing packages.

- For rmr we use: R CMD INSTALL rmr-2.2.2.tar.gz
- For rhdfs we use: R CMD INSTALL rmr-2.2.2.tar.gz
- For rhbase we use: R CMD INSTALL rhbase-1.2.0.tar.gz

gdp data

- library(rmr2)
- library(rhdfs)
- gdp <- NA
- gdp <- read.csv("~/Downloads/GDP.csv")
- gdp <- gdp[,1:4]
- gdp$GDP <- as.double(gsub(",","",gdp$GDP))
- head(gdp)

Setup Map-Reduce Function

- hdfs.init()
- gdp.values <- to.dfs(gdp)
- aaplRevenue = 181890

- gdp.map.fn <- function(k,v) {
- key <- ifelse(v[4] < aaplRevenue, "less", "greater")
- keyval(key, 1)
- }
- count.reduce.fn <- function(k,v) {
- keyval(k, length(v))
- }

Run Map-Reduce Function

- count <- mapreduce(input=gdp.values, map = gdp.map.fn, reduce = count.reduce.fn)

- from.dfs(count)

Q&A