Data Pre-Processing in R

L. Torgo

ltorgo@fc.up.pt
Faculdade de Ciências / LIAAD-INESC TEC, LA
Universidade do Porto

Jun, 2017

What is Data Pre-Processing?

Data Pre-Processing

Set of steps that may be necessary to carry out before any further analysis takes place on the available data

Some Motivations for Data Pre-Processing

- Several data mining methods are sensitive to the scale and/or type of the variables
  - Different variables (columns of our data sets) may have rather different scales
  - Some methods are not able to handle either nominal or numeric variables
- We may need to “create” new variables to achieve our objectives
  - Sometimes we are more interested in relative values (variations) than absolute values
  - We may be aware of some domain-specific mathematical relationship among two or more variables that is important for the task
- Frequently we have data sets with unknown variable values
- Our data set may be too large for some methods to be applicable

NYU | STERN
MS in Business Analytics

Some of the Main Classes of Data Pre-Processing

- Data cleaning
  - Given data may be hard to read or require extra parsing efforts
- Data transformation
  - It may be necessary to change/transform some of the values of the data
- Variable creation
  - E.g. to incorporate some domain knowledge
- Dimensionality reduction
  - To make modeling possible

NYU | STERN
MS in Business Analytics

Illustrations of Data Cleaning in R

Making your data tidy

- Properties of tidy data sets:
  - each value belongs to a variable and an observation
  - each variable contains all values of a certain property measured across all observations
  - each observation contains all values of the variables measured for the respective case
- The properties lead to data tables where each row represents an observation and the columns represent different properties measured for each observation

A non tidy data set

	Math	English
Anna	86	90
John	43	75
Catherine	80	82

- This data is about the grades of students on some subjects
- The rows are students
- The columns are the properties measured for each student:
  - name
  - subject
  - grade

Reading the data

Math English
Anna 86 90
John 43 75
Catherine 80 82

The contents of this file could be read as follows:

```
std <- read.table("stud.txt")
std

##           Math English
## Anna        86      90
## John        43      75
## Catherine   80      82
```

Making this data tidy

```
std <- cbind(StudentName=rownames(std),std)
library(tidyr)
tstd <- gather(std,Subject,Grade,Math:English)
tstd

##   StudentName Subject Grade
## 1        Anna    Math    86
## 2        John    Math    43
## 3   Catherine    Math    80
## 4        Anna English    90
## 5        John English    75
## 6   Catherine English    82
```


Handling Dates

- Date/time information are very common types of data
- With real-time data collection (e.g. sensors) this is even more common
- Date/time information can be provided in several different formats
- Being able to read, interpret and convert between these formats is a very frequent data pre-processing task

Package **lubridate**

- Package with many functions related with handling dates/time
- Handy for parsing and/or converting between different formats
- Some examples:

```
library(lubridate)
ymd("20151021")

## [1] "2015-10-21"

ymd("2015/11/30")

## [1] "2015-11-30"

myd("11.2012.3")

## [1] "2012-11-03"

dmy_hms("2/12/2013 14:05:01")

## [1] "2013-12-02 14:05:01 UTC"
```


Examples of using package **lubridate**

```
dates <- c(20120521, "2010-12-12", "2007/01/5", "2015-2-04",
           "Measured on 2014-12-6", "2013-7+ 25")
dates <- ymd(dates)
dates

## [1] "2012-05-21" "2010-12-12" "2007-01-05" "2015-02-04" "2014-12-06"
## [6] "2013-07-25"

data.frame(Dates=dates,WeekDay=wday(dates),nWeekDay=wday(dates,label=TRUE),
           Year=year(dates),Month=month(dates,label=TRUE))

##        Dates WeekDay nWeekDay Year Month
## 1 2012-05-21       2      Mon 2012   May
## 2 2010-12-12       1      Sun 2010   Dec
## 3 2007-01-05       6      Fri 2007   Jan
## 4 2015-02-04       4      Wed 2015   Feb
## 5 2014-12-06       7      Sat 2014   Dec
## 6 2013-07-25       5    Thurs 2013   Jul
```

Conversions between time zones

- Sometimes we get dates from different time zones
- **lubridate** can help with that too
- Some examples:

```
date <- ymd_hms("20150823 18:00:05", tz="Europe/Berlin")
date

## [1] "2015-08-23 18:00:05 CEST"

with_tz(date, tz="Pacific/Auckland")

## [1] "2015-08-24 04:00:05 NZST"

force_tz(date, tz="Pacific/Auckland")

## [1] "2015-08-23 18:00:05 NZST"
```


String Processing

- Processing and/or parsing strings is frequently necessary when reading data into R
- This is particularly true when data is received in a non-standard format

String Processing - some useful packages

- Base R contains several useful functions for string processing
  - E.g. `grep`, `strsplit`, `nchar`, `substr`, etc.
- Package **stringi** provides an extensive set of useful functions for string processing
- Package **stringr** builds upon the extensive set of functions of **stringi** and provides a simpler interface covering the most common needs

String Processing - a concrete example

- Let us work through a concrete example
  - Reading the name of the variables of a problem that are provided within a text file
  - Avoiding having to type them by hand
- The UCI repository contains a large set of data sets
  - Data sets are typically provided in two separate files: one with the data, the other with information on the data set, including the names of the variables
  - This latter file is a text file in a free format
- Let us try to read the information on the names of the variables of the data set named **heart-disease**
  - Information (text file) available at `https://archive.ics.uci.edu/ml/machine-learning-databases/heart-disease/heart-disease.names`

Reading in the file

- Let us start by reading the file

```
d <- readLines(url("https://archive.ics.uci.edu/ml/machine-learning-da
```

- As you may check the useful information is between lines 127 and 235

```
d <- d[127:235]
head(d,2)

## [1] "       1 id: patient identification number"
## [2] "       2 ccf: social security number (I replaced this with a du

tail(d,2)

## [1] "      75 junk: not used"
## [2] "      76 name: last name of patient "
```


Processing the lines

- Trimming white space

```
library(stringr)
d <- str_trim(d)
```

- Looking carefully at the lines (strings) you will see that the lines containing some variable name all follow the pattern
  `ID name ....`
- Where `ID` is a number from 1 to 76
- So we have a number, followed by the information we want (the name of the variable), plus some optional information we do not care
- There are also some lines in the midle that describe the values of the variables and not the variables

NYU STERN | MS in Business Analytics

Processing the lines (cont.)

- Regular expressions are a powerful mechanism for expressing string patterns
- They are out of the scope of this subject
  - Tutorials on regular expressions can be easily found around the Web
- Function `grep()` can be used to match strings against patterns expressed as regular expressions

```
## e.g. line (string) starting with the number 26
d[grep("^26",d)]

## [1] "26 pro (calcium channel blocker used during exercise ECG: 1 =
```


Processing the lines (cont.)

- Lines starting with the numbers 1 till 76

```
tgtLines <- sapply(1:76,function(i) d[grep(paste0("^",i),d)[1]])
head(tgtLines,2)

## [1] "1 id: patient identification number"
## [2] "2 ccf: social security number (I replaced this with a dummy va
```

- Throwing the IDs out...

```
nms <- str_split_fixed(tgtLines," ",2)[,2]
head(nms,2)

## [1] "id: patient identification number"
## [2] "ccf: social security number (I replaced this with a dummy valu
```

Processing the lines (cont.)

- Grabbing the name

```
nms <- str_split_fixed(nms,":",2)[,1]
head(nms,2)

## [1] "id"  "ccf"
```

- Final touches to handle some extra characters (e.g. check `nms[6:8]`)

```
nms <- str_split_fixed(nms," ",2)[,1]
head(nms,2)

## [1] "id"  "ccf"

tail(nms,2)

## [1] "junk" "name"
```


Dealing with Missing/Unknown Values

- Missing variable values are a frequent problem in real world data sets

Some Possible Strategies

- Remove all lines in a data set with some unknown value
- Fill-in the unknowns with the most common value (a statistic of centrality)
- Fill-in with the most common value on the cases that are more "similar" to the one with unknowns
- Explore eventual correlations between variables
- etc.

Some illustrations in R

```
load("carInsurance.Rdata") # car insurance dataset (get it from class web page)
```

```
library(DMwR)
head(ins[!complete.cases(ins),],3)

##   symb normLoss        make fuelType aspiration nDoors   bodyStyle
## 1    3       NA alfa-romero      gas        std    two convertible
## 2    3       NA alfa-romero      gas        std    two convertible
## 3    1       NA alfa-romero      gas        std    two   hatchback
##   driveWheels engineLocation wheelBase length width height curbWeight
## 1         rwd          front      88.6  168.8  64.1   48.8       2548
## 2         rwd          front      88.6  168.8  64.1   48.8       2548
## 3         rwd          front      94.5  171.2  65.5   52.4       2823
##   engineType nrCylinds engineSize fuelSystem bore stroke compressionRatio
## 1       dohc      four        130       mpfi 3.47   2.68                9
## 2       dohc      four        130       mpfi 3.47   2.68                9
## 3       ohcv       six        152       mpfi 2.68   3.47                9
##   horsePower peakRpm cityMpg highwayMpg price
## 1        111    5000      21         27 13495
## 2        111    5000      21         27 16500
## 3        154    5000      19         26 16500
```


Some illustrations in R (2)

```
nrow(ins[!complete.cases(ins),])

## [1] 46

noNA.ins <- na.omit(ins)  # Option 1
nrow(noNA.ins[!complete.cases(noNA.ins),])

## [1] 0

noNA.ins <- centralImputation(ins)  # Option 2
nrow(noNA.ins[!complete.cases(noNA.ins),])

## [1] 0

noNA.ins <- knnImputation(ins,k=10)  # Option 3
nrow(noNA.ins[!complete.cases(noNA.ins),])

## [1] 0
```

Transformations of Variables in R

Standardizing Numeric Variables

Goal

Make all variables have the same scale - usually a scale where all have mean 0 and standard deviation 1

$$y = \frac{x - \bar{x}}{\sigma_x}$$

```
load("carInsurance.Rdata") # car insurance data (check course web page)
```

```
norm.ins <- ins
for(var in c(10:14,17,19:26)) norm.ins[,var] <- scale(ins[,var])
```

Discretization of Numeric Variables

- Sometimes it makes sense to discretize a numeric variable
- This can also reduce computational complexity in some cases
- Let us see an example of discretizing a variable into 4 intervals.
- Two examples of possible strategies
  - Equal-width

```
data(Boston, package="MASS") # The Boston Housing data set
Boston$age <- cut(Boston$age,4)
table(Boston$age)

##
##  (2.8,27.2] (27.2,51.4] (51.4,75.7]  (75.7,100]
##          51          97          96         262
```

  - Equal-frequency

```
data(Boston, package="MASS") # The Boston Housing data set
Boston$age <- cut(Boston$age,quantile(Boston$age,probs=seq(0,1,.25)))
table(Boston$age)

##
##    (2.9,45]   (45,77.5] (77.5,94.1]  (94.1,100]
##         126         126         126         127
```


Creating Variables

Creating Variables

- May be necessary to properly address ou data mining goals
- Several factors may motivate variable creation:
  - Express known relationships between existing variables
  - Overcome limitations of some data mining tools, like for instance:
    - dependencies between cases (rows)
    - etc.

Handling Case Dependencies

- Observations in a data set sometimes are not independent
- Frequent dependencies include time, space or even space-time
- These effects may have a strong impact on the data mining process
- Two main ways of handling this issue:
  - Constrain ourselves to tools that handle these dependencies directly
  - Create variables that express the dependency relationships

Working with relative values instead of absolute values

Why?

Frequent technique that is used in time series analysis to avoid trend effects

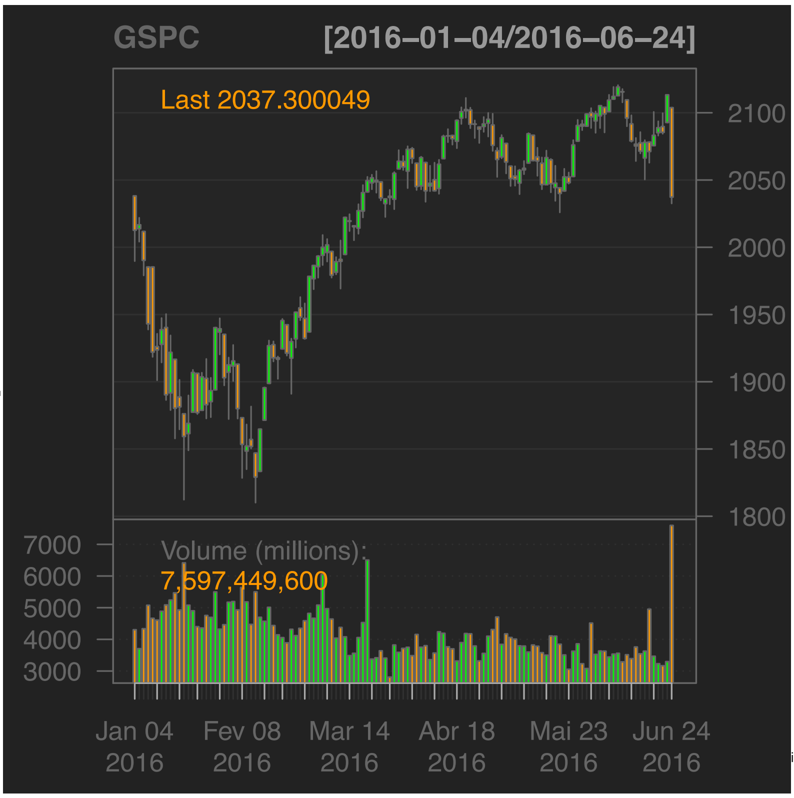
$$y_i = \frac{x_i - x_{i-1}}{x_{i-1}}$$

```
x <- rnorm(100,mean=100,sd=3)
head(x)

## [1]  97.52625 100.19782  99.16785 100.23747 100.38753 101.75377

vx <- diff(x)/x[-length(x)]
head(vx)

## [1]  0.027393332 -0.010279347  0.010785978  0.001496962  0.013609686
## [6] -0.031358624
```


An example with real-world time series data

The S&P 500 stock market index

```
library(quantmod)  # extra package
getSymbols('^GSPC',from='2016-01-01')

## [1] "GSPC"

head(GSPC,3)

##            GSPC.Open GSPC.High GSPC.L
## 2016-01-04   2038.20   2038.20   1989.
## 2016-01-05   2013.78   2021.94   2004.
## 2016-01-06   2011.71   2011.71   1979.
##            GSPC.Adjusted
## 2016-01-04       2012.66
## 2016-01-05       2016.71
## 2016-01-06       1990.26

candleChart(GSPC )
```



An example with real-world time series data (2)

The S&P 500 stock market index

```
head(Cl(GSPC))

##             GSPC.Close
## 2016-01-04    2012.66
## 2016-01-05    2016.71
## 2016-01-06    1990.26
## 2016-01-07    1943.09
## 2016-01-08    1922.03
## 2016-01-11    1923.67

head(Delt(Cl(GSPC)))

##             Delt.1.arithmetic
## 2016-01-04                 NA
## 2016-01-05       0.0020122261
## 2016-01-06      -0.0131153966
## 2016-01-07      -0.0237004430
## 2016-01-08      -0.0108383746
## 2016-01-11       0.0008532723
```

NYU STERN
MS in Business Analytics

Handling Time Order Between Cases

Why?

- There is a time order between the cases
- Some tools shuffle the cases, or are not able to use the information about this order

NYU STERN
MS in Business Analytics

Time Delay Embedding

- Create variables whose values are the value of the time series in previous time steps
- Standard tools find relationships between variables
- If we have variables whose values are the value of the same variable but on different time steps, the tools will be able to model the time relationships with these embeddings
- Note that similar “tricks” can be done with space and space-time dependencies

NYU | STERN
MS in Business Analytics

Reducing Data Dimensionality

Reducing the dimension of the data set

Motivations

- Some data mining methods may be unable to handle very large data sets
- The computation time to obtain a certain model may be too large for the application
- We may want simpler models
- etc.

Some strategies

- Reduce the number of variables
- Reduce the number of cases
- Reduce the number of values on the variables

Reducing the number of variables through PCA

Principal Component Analysis (PCA)

- General Idea : replace the variables by a new (smaller) set where most of the "information" on the problem is still expressed
- Goal : find a new set of axes onto which we will project the original data

- The new set of axes are formed by linear combinations of the original variables
- We search for the linear combinations that "explain" most of the variability on the original axes
- If we are "lucky" with a few of these new axes (ideally two for easy data visualization), we are able to explain most of the variability on the original data
- Each original observation is then "projected" into these new axes

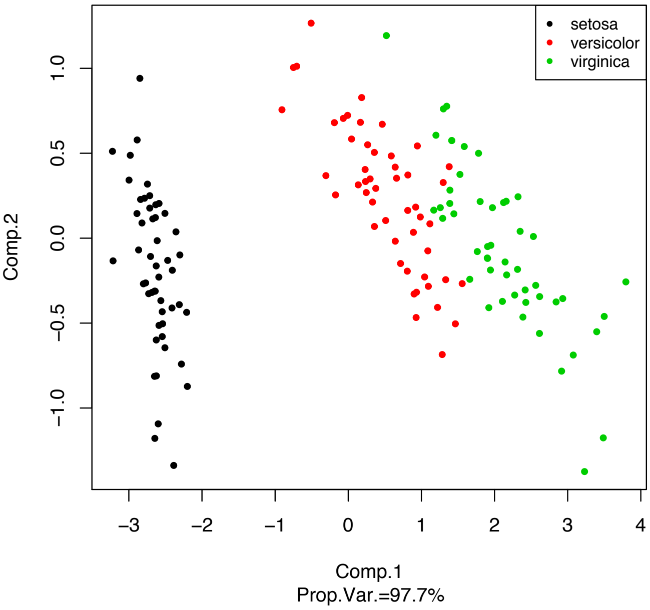
PCA - the method

- Find a first linear combination which better captures the variability in the data
- Move to the second linear combination to try to capture the variability not explained by the first one
- Continue until the set of new variables explains most of the variability (frequently 90% is considered enough)

An illustration with the Iris data set

	Comp.1	Comp.2
Sepal.Length	0.361	−0.657
Sepal.Width	−0.085	−0.730
Petal.Length	0.857	0.173
Petal.Width	0.358	0.075

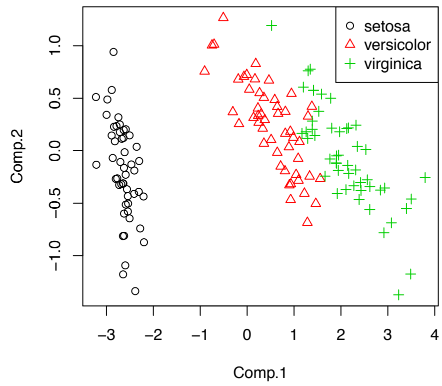
$$\begin{aligned} Comp.1 = 0.361 &\times Sepal.Length \\ - 0.085 &\times Sepal.Width \\ + 0.857 &\times Petal.Length \\ + 0.358 &\times Petal.Width \end{aligned}$$



The example in R

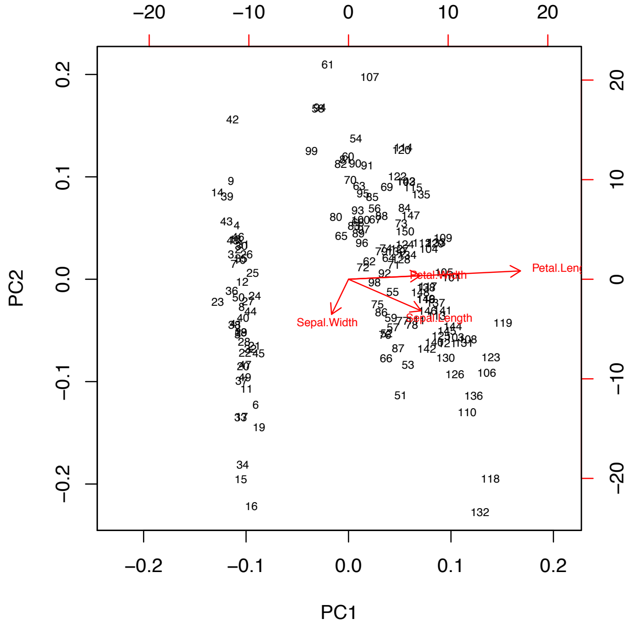
```
pca <- princomp(iris[,-5])
loadings(pca)

##
## Loadings:
##              Comp.1 Comp.2 Comp.3 Comp.4
## Sepal.Length  0.361 -0.657 -0.582  0.315
## Sepal.Width         -0.730  0.598 -0.320
## Petal.Length  0.857  0.173        -0.480
## Petal.Width   0.358         0.546  0.754
##
##                Comp.1 Comp.2 Comp.3 Comp.4
## SS loadings      1.00   1.00   1.00   1.00
## Proportion Var   0.25   0.25   0.25   0.25
## Cumulative Var   0.25   0.50   0.75   1.00
```

```
scs <- pca$scores[,1:2]
plot(scs,col=as.numeric(iris$Species),
     pch=as.numeric(iris$Species))
legend('topright',levels(iris$Species),
       pch=1:3,col=1:3)
```



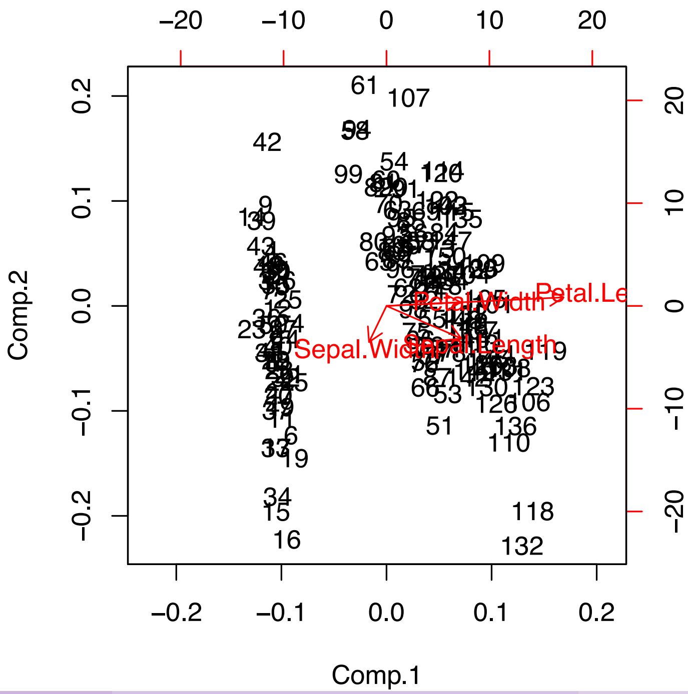
Biplots for visualizating PCAs

- Biplots represent the data points on the two first PCAs
- Each point is represented by its respective score on the components (top and right axes)
- The original variables are also represented as vectors in a scale of loadings within each component (left and bottom axes)



Biplots in R

```
biplot(pca)
```



Reducing the number of cases

Resampling strategies

Reducing the number of cases usually is carried out through some form of random resampling of the original data

Some possible methods:

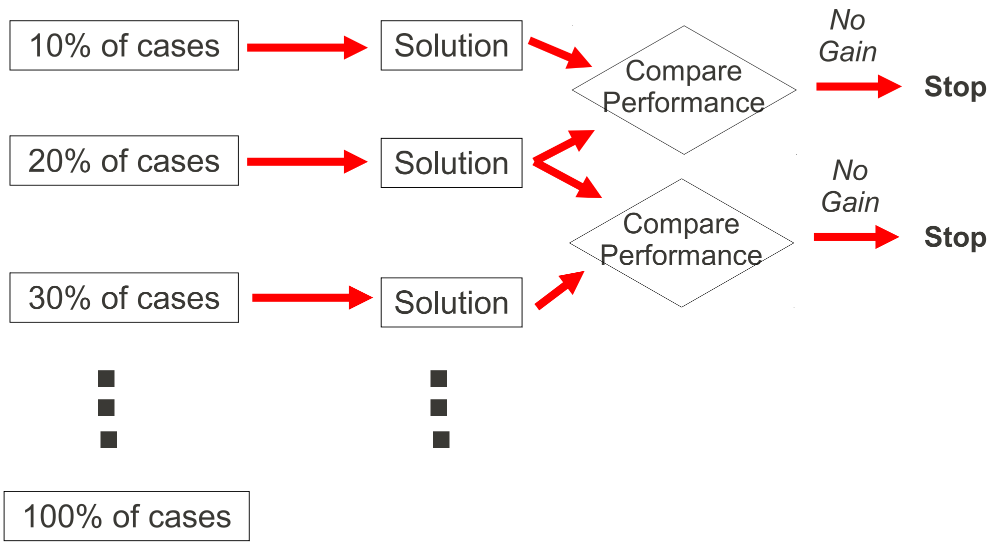
- Random selection of a sub-set of the data set
- Random and stratified selection of a sub-set of the data
- Incremental sampling
- Multiple sample and/or models

Random selection of a sub-set of the data set

Random samples of a data set. Peeking 70% of the rows of one data set:

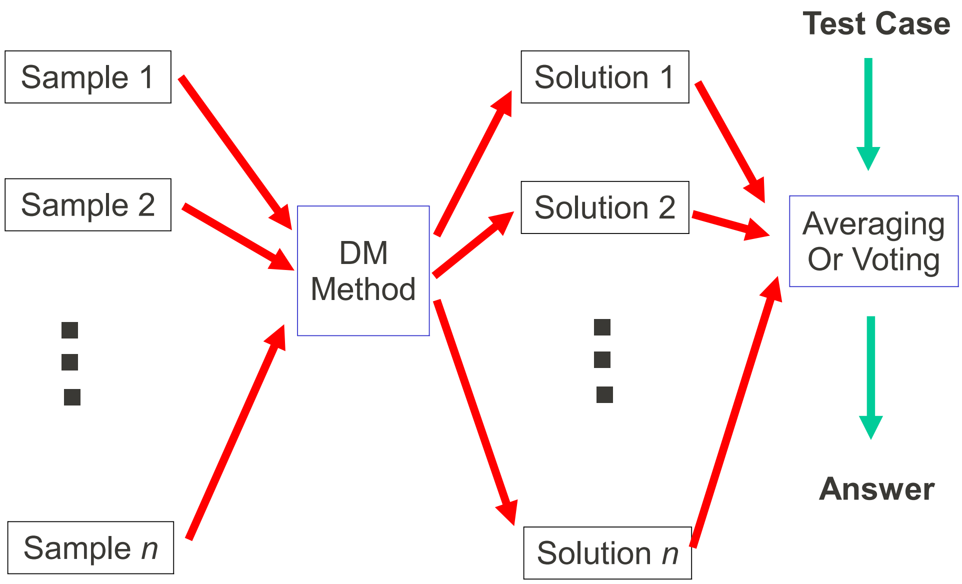
```
data(Boston,package='MASS')
idx <- sample(1:nrow(Boston),as.integer(0.7*nrow(Boston)))
smpl <- Boston[idx,]
rmng <- Boston[-idx,]
nrow(smpl)

## [1] 354

nrow(rmng)

## [1] 152
```

Incremental Sampling



NYU | STERN
MS in Business Analytics

Multiple Samples and/or Models



NYU | STERN
MS in Business Analytics

Reducing the number of values in numeric variables

Main motivation: Some techniques have their computational complexity heavily dependent on the number of values of the numeric variables.
A few simple techniques that may help on these situations:

- Rounding
- Values discretization
  - Grouping values
    - Equal-size groups
    - Equal-frequency groups
    - k-means method
    - etc.

Handling Big Data in R

Big Data

What is Big Data?

- Hadley Wickham (Chief Scientist at RStudio)
  In traditional analysis, the development of a statistical model takes more time than the calculation by the computer. When it comes to Big Data this proportion is turned upside down.
- Wikipedia
  Collection of data sets so large and complex that it becomes difficult to process using on-hand database management tools or traditional data processing applications.
- The 3 V's
  *Increasing **volume** (amount of data), **velocity** (speed of data in and out), and **variety** (range of data types and sources)*

R and Big Data

- R keeps all objects in memory - potential problem for big data
- Still, current versions of R can address 8 TB of RAM on 64-bit machines
- Nevertheless, big data is becoming more and more a hot topic within the R community so new "solutions" are appearing!

Some rules of thumb

- Up to 1 million records - easy on standard R
- 1 million to 1 billion - possible but with additional effort
- More than 1 billion - possibly require map reduce algorithms that can be designed in R and processed with connectors to Hadoop and others

Big Data Approaches in R

- Reducing the dimensionality of data
- Get bigger hardware and/or parallelize your analysis
- Integrate R with higher performing programming languages
- Use alternative R interpreters
- Process data in batches
- Improve your knowledge of R and its inner workings / programming tricks

Get Bigger Hardware

- Buy more memory
- Buy better processing capabilities
- Multi-core, multi-processor, clusters

Some sources of extra information

- CRAN task view on High-performance and Parallel Computing *http://cran.r-project.org/web/views/HighPerformanceComputing.html*
- Explore Revolution Analytics (proprietary) offers for Big Data *http://www.revolutionanalytics.com/revolution-r-enterprise-scaler*

Integrate R with higher performing programming languages

- R is very good at integrating easily with other languages
- You can easily do heavy computation parts in other language
- Still, this requires knowledge about these languages that may not be easily adaptable for data analysis tasks, in spite of their efficiency

Some sources of extra information

- The outstanding package `Rcpp` allows you to call C and C++ directly in the middle of R code

  D. Eddelbuettel (2013): Seamless R and C++ Integration with Rcpp. UserR! Series. Springer.
- Section 5 of the R manual “Writing R Extensions” talks about interfacing other languages

Use alternative R interpreters

- Some special-purpose R interpreters exist

pqR - pretty quick R (*http://www.pqr-project.org/*)

Renjin - R interpreter reimplemented in Java and running on the Java Virtual Machine (*http://www.renjin.org/*)

TERR - TIBCO Enterprise Runtime for R (*http://spotfire.tibco.com/en/discover-spotfire/what-does-spotfire-do/predictive-analytics/tibco-enterprise-runtime-for-r-terr.aspx*)

NYU | STERN
MS in Business Analytics

Process data in batches

- Store data on hard disk
- Load and process data in chuncks
- But, analysis has to be adapted to work by chunk, or methods have to be adapted to work with data types stored on hard disk

Some sources of extra information

- Packages `ff`, `ffbase`, `bigmemory`, `sqldf`, `data.table`, etc.
  http://cran.r-project.org/web/views/HighPerformanceComputing.html
- Explore Revolution Analytics (proprietary) offers for Big Data
  http://www.revolutionanalytics.com/revolution-r-enterprise-scaler

NYU | STERN
MS in Business Analytics

Improve your knowledge of R and its inner workings / programming tricks

Some basic speed up tricks

- Minimize copies of the data
  Hint: learn about the way R passes arguments to functions
  Outstanding source of information at *http://adv-r.had.co.nz/memory.html* of the book “Advanced R Programming” by Hadley Wickham
- Prefer integers over doubles when possible
- Only read the data you really need from files
- Use categorical variables (read factors in R) with care
- Use loops with care particularly if they are making copies of the data along their execution

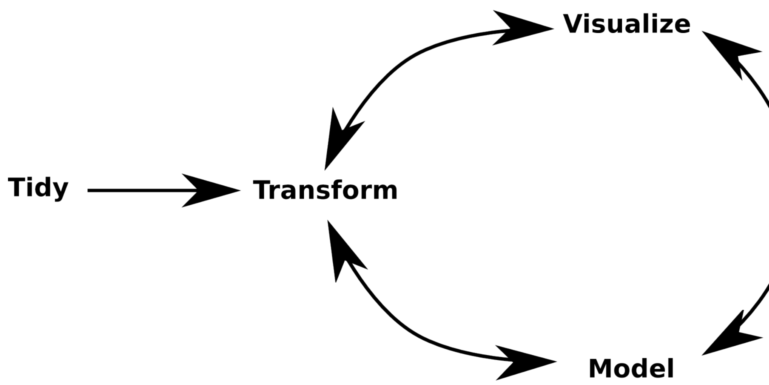
NYU | STERN
MS in Business Analytics

Improve your knowledge of R and its inner workings / programming tricks

Using special purpose packages for frequent tasks

The following is **strongly** inspired by a Hadley Wickham talk (*https://dl.dropboxusercontent.com/u/41902/bigr-data-londonr.pdf*)

- The typical data analysis process



- On each of these steps there may be constraints with big data

Data Transformations

Split-Apply-Combine

- A frequent data transformation one needs to carry out
  1. Split the data set rows according to some criterion
  2. Calculate some value on each of the resulting subsets
  3. Aggregate the results into another aggregated data set

Data Transformations

Split-Apply-Combine - an example

```
library(plyr) # extra package you have to install
data(algae,package="DMwR")
ddply(algae,.(season,speed),function(d) colMeans(d[,5:7],na.rm=TRUE))

##    season  speed      mnO2       Cl      NO3
## 1  autumn   high 11.145333 26.91107 5.789267
## 2  autumn    low 10.112500 44.65738 3.071375
## 3  autumn medium 10.349412 47.73100 4.025353
## 4  spring   high  9.690000 19.74625 2.013667
## 5  spring    low  4.837500 69.22957 2.628500
## 6  spring medium  7.666667 76.23855 2.847792
## 7  summer   high 10.629000 22.49626 2.571900
## 8  summer    low  7.800000 58.74428 4.132571
## 9  summer medium  8.651176 47.23423 3.652059
## 10 winter   high  9.760714 23.86478 2.738500
## 11 winter    low  8.780000 43.13720 3.147600
## 12 winter medium  7.893750 66.95135 3.817609
```

- All nice and clean but ... slow on big data!

NYU | STERN
MS in Business Analytics

Enter “dplyr”

plyr on steroids

- `dplyr` is a new package by Hadley Wickham that re-invents several operations done with `plyr` more efficiently

```
library(dplyr) # another extra package you have to install
data(algae,package="DMwR")
grps <- group_by(algae,season,speed)
summarise(grps,avg.mnO2=mean(mnO2,na.rm=TRUE),
          avg.Cl=mean(Cl,na.rm=TRUE),avg.NO3=mean(NO3,na.rm=TRUE))

##   avg.mnO2   avg.Cl  avg.NO3
## 1 9.117778 43.63628 3.282389
```

NYU | STERN
MS in Business Analytics

Some comments on `dplyr`

- It is extremely fast and efficient
- It can handle not only data frames but also objects of class `data.table` and standard data bases
- New developments may arise as it is a very new package

Data Visualization

- R has excellent facilities for visualizing data
- With big data plotting can become very slow
- Recent developments are trying to take care of this
- Hadley Wickham is developing a new package for this: `bigvis` (*https://github.com/hadley/bigvis*)
  - From the project page:
    The bigvis package provides tools for exploratory data analysis of large datasets (10-100 million obs). The aim is to have most operations take less than 5 seconds on commodity hardware, even for 100,000,000 data points.

Efforts on Modeling with Big Data

- Model construction with Big Data is particularly hard
- Most algorithms include sophisticated operations that frequently do not scale up very well
- The R community is making some efforts to alleviate this problem. A few examples:
  - `bigrf` - a package providing a Random Forests implementation with support for parellel execution and large memory.
  - `biglm, speedglm` - packages for fitting linear and generalized linear models to large data
- A way to face the problem is through streaming algorithms
  - `HadoopStreaming` - Utilities for using R scripts in Hadoop streaming
  - `stream` - interface to MOA open source framework for data stream mining

NYU | STERN
MS in Business Analytics