Advanced R Programming - Bonus Lecture

Krzysztof Bartoszek
(slides by Leif Jonsson, Måns Magnusson, and Arash Haratian)

Linköping University

krzysztof.bartoszek@liu.se

5 September 2024

Today

Data munging

Machine Learning

Supervised learning in R

Probability in R

Big data

Tidy data

Tidy data and **messy** data

Tidy data

1. Each variable forms a column
2. Each observation forms a row
3. Each type of observational unit forms a table

Tidy data

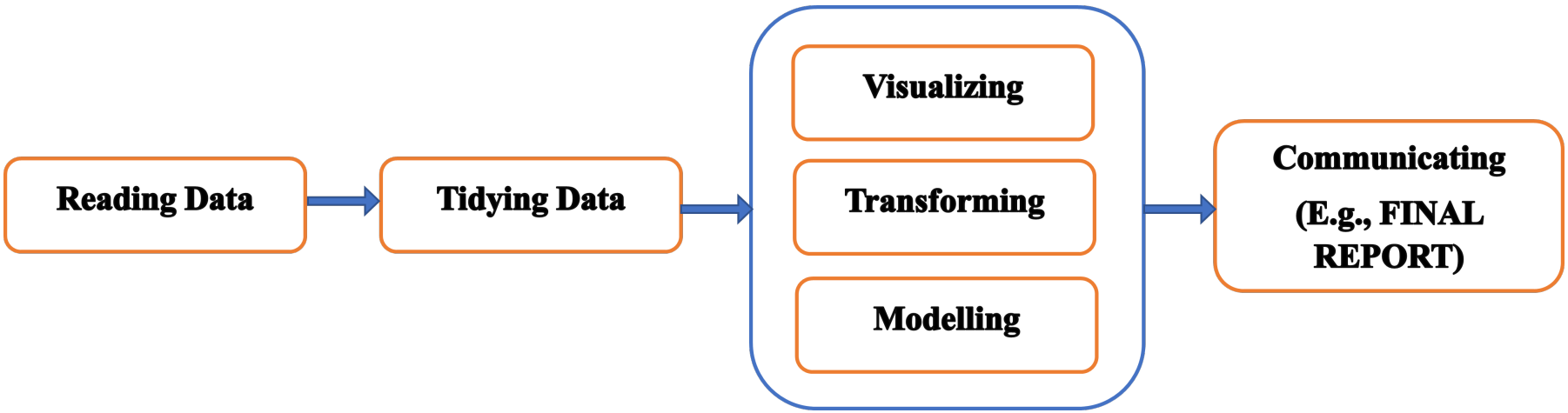
1. Each variable forms a column
2. Each observation forms a row
3. Each type of observational unit forms a table

Examples: `iris`, `faithful` and `gapminder`

	country	continent	year	lifeExp	pop	gdpPercap
1	Afghanistan	Asia	1952	28.801	8425333	779.4453
2	Afghanistan	Asia	1957	30.332	9240934	820.8530
3	Afghanistan	Asia	1962	31.997	10267083	853.1007
4	Afghanistan	Asia	1967	34.020	11537966	836.1971
5	Afghanistan	Asia	1972	36.088	13079460	739.9811
6	Afghanistan	Asia	1977	38.438	14880372	786.1134
7	Afghanistan	Asia	1982	39.854	12881816	978.0114
8	Afghanistan	Asia	1987	40.822	13867957	852.3959
9	Afghanistan	Asia	1992	41.674	16317921	649.3414
10	Afghanistan	Asia	1997	41.763	22227415	635.3414
11	Afghanistan	Asia	2002	42.129	25268405	726.7341
12	Afghanistan	Asia	2007	43.828	31889923	974.5803
13	Albania	Europe	1952	55.230	1282697	1601.0561

Why tidy?

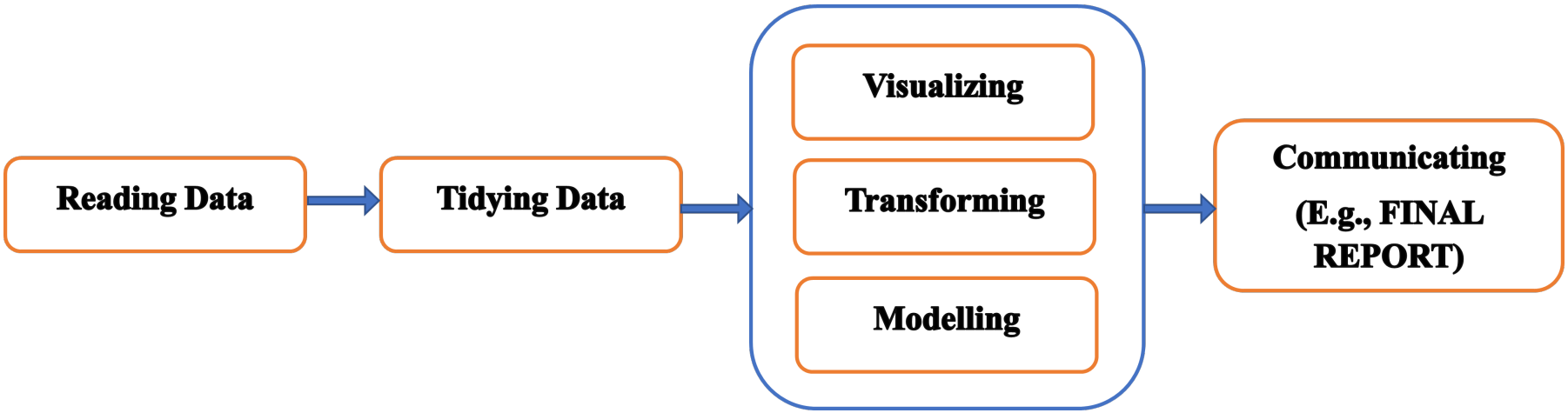
80 % of Big Data work is data munging



Why tidy?

80 % of Big Data work is data munging

Analysis and visualization is based on tidy data

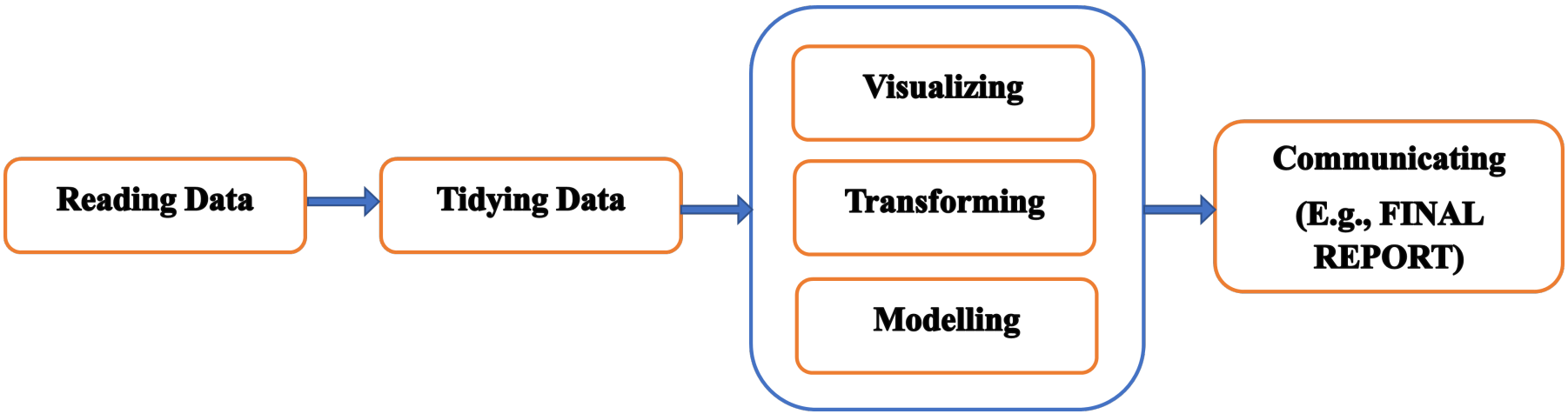


Why tidy?

80 % of Big Data work is data munging

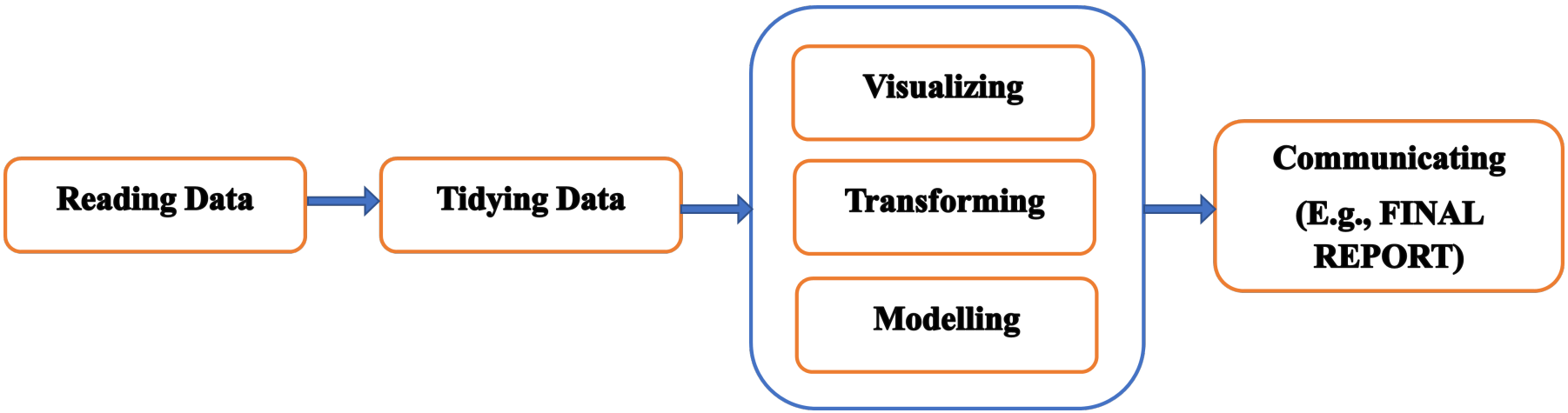
Analysis and visualization is based on tidy data

Performant code



Data analysis pipeline

Messy data $\rightarrow$ Tidy data $\rightarrow$ Analysis



dplyr

Verbs for handling data

Highly optimized C++ code (backend)

Handling larger datasets in R
(no copy-on-modify)

dplyr - Some of the important functions

- mutate(): Adds new columns that are functions of existing columns
- transmute(): Adds new columns that are functions of existing columns, and then it removes the existing ones
- select(): Picks columns based on their names
- filter(): Picks rows based on their values
- group_by() and summarise(): Reduces multiple rows down to a single summary
- arrange(): Changes the ordering of the rows

tidyr

Another package for tidying the data:

- separate(): Separates a character column into multiple columns with a regular expression or numeric position
- extract(): Creates new columns from a character column given a string or Regex
- pivot_wider(): Increases the number of columns by collecting the informations that are spread along the rows
- pivot_longer(): Increases the number of rows by collecting the informations that are spread along the columns

dplyr+tidyr

The cheatsheet: `https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf`

These two packages should be used to turn a messy data into a tidy data.

Messy data

1. Column headers are values, not variable names.
   (`new_sp_m014` - `new_rel_f65` in `who` dataset)
   The column names contain:
   `new_`{method of diagnosis}`_`{gender}{age group}

```
tidyr::who
```

```
#> # A tibble: 7,240 X 60
#>    country     iso2  iso3   year new_s...1 new_s...2 new_s...3 new_s...4
#>    <chr>       <chr> <chr> <int>    <int>    <int>    <int>    <int>
#>  1 Afghanistan AF    AFG    1980       NA       NA       NA       NA
#>  2 Afghanistan AF    AFG    1981       NA       NA       NA       NA
#>  3 Afghanistan AF    AFG    1982       NA       NA       NA       NA
#>  4 Afghanistan AF    AFG    1983       NA       NA       NA       NA
#>  5 Afghanistan AF    AFG    1984       NA       NA       NA       NA

#> # ... with 7,230 more rows, 50 more variables: new_sp_m65 <int>,
#> #   new_sp_f014 <int>, new_sp_f1524 <int>, new_sp_f2534 <int>,
#> #   new_sp_f3544 <int>, new_sp_f4554 <int>, new_sp_f5564 <int>,
#> #   new_sp_f65 <int>, new_sn_m014 <int>, new_sn_m1524 <int>,
#> #   new_sn_m2534 <int>, new_sn_m3544 <int>, new_sn_m4554 <int>,
#> #   new_sn_m5564 <int>, new_sn_m65 <int>, new_sn_f014 <int>,
#> #   new_sn_f1524 <int>, new_sn_f2534 <int>, new_sn_f3544 <int>, ...
```

Messy data

1. Column headers are values, not variable names.
   (`new_sp_m014` - `new_rel_f65` in `who` dataset)
   Solution: The `tidyr::pivot_longer()` can be used:

```
library(tidyverse)
long_who <- who %>%
  pivot_longer(
    cols = new_sp_m014:newrel_f65,
    names_to = "column_info",
    values_to = "cases",
    values_drop_na = TRUE
  )

#> # A tibble: 76,046 X 6
#>    country     iso2  iso3   year column_info  cases
#>    <chr>       <chr> <chr> <int> <chr>        <int>
#>  1 Afghanistan AF    AFG    1997 new_sp_m014      0
#>  2 Afghanistan AF    AFG    1997 new_sp_m1524    10
#>  3 Afghanistan AF    AFG    1997 new_sp_m2534     6
#>  4 Afghanistan AF    AFG    1997 new_sp_m3544     3
#>  5 Afghanistan AF    AFG    1997 new_sp_m4554     5

#> # ... with 76,041 more rows
```

Messy data

1. Column headers are values, not variable names.
2. Multiple variables are stored in one column.
   (The new column `column_info` in previous slide!)
   The column contains:
   - method of diagnosis
   - gender
   - age group

Messy data

1. Column headers are values, not variable names.
2. Multiple variables are stored in one column.
   (The new column `column_info` in `long_who`!)
   Solution: The `tidyr::separate()` can be used:

```
tidy_who <- long_who %>%
  # SOME OBSERVATIONS ARE STORED AS 'newrel' instead of 'new_rel'
  # SO WE HAVE TO TAKE CARE OF THAT FIRST
  mutate(
    column_info = stringr::str_replace(column_info, "newrel", "new_rel")
  ) %>%
  separate(column_info, c("new", "method", "sexage")) %>%
  select(-new, -iso2, -iso3) %>%
  separate(sexage, c("sex", "age"), sep = 1)
```

```
#> # A tibble: 76,046 X 6
#>     country      year method sex   age   cases
#>     <chr>       <int> <chr>  <chr> <chr> <int>
#>  1 Afghanistan  1997 sp     m     014       0
#>  2 Afghanistan  1997 sp     m     1524     10
#>  3 Afghanistan  1997 sp     m     2534      6
#>  4 Afghanistan  1997 sp     m     3544      3
#>  5 Afghanistan  1997 sp     m     4554      5

#> # ... with 76,041 more rows
```

Messy data

1. Column headers are values, not variable names.
2. Multiple variables are stored in one column.
3. Variables are stored in both rows and columns. (`crimetab`)
4. Multiple types of observational units are stored in the same table.
5. A single observational unit is stored in multiple tables.

Regular Expressions

Language for manipulating strings

Find strings that match a pattern

Extract patterns from strings

Replace patterns in strings

Component in many functions
(grep, gsub, stringr::, tidyr::)

Regular Expressions - Syntax

fruit <- c("apple", "banana", "pear", "pineapple")

Symbol	Description	Example
?	The preceding item is optional and will be matched at most once	grep("pi?",fruit)
*	The preceding item will be matched zero or more times	grep("pi*",fruit)
+	The preceding item will be matched one or more times	grep("pi+",fruit)
n	The preceding item is matched exactly n times	grep("p{2}",fruit)

Regex Examples - Finding matching

```
library(gapminder)

grep("we", gapminder$country)

#>   [1] 1465 1466 1467 1468 1469 1470 1471 1472 1473 1474
#>  [11] 1475 1476 1693 1694 1695 1696 1697 1698 1699 1700
#>  [21] 1701 1702 1703 1704

grep("we", gapminder$country, value=TRUE)

#>   [1] "Sweden"   "Sweden"   "Sweden"   "Sweden"
#>   [5] "Sweden"   "Sweden"    "Sweden"    "Sweden"
#>   [9] "Sweden"   "Sweden"   "Sweden"   "Sweden"
#>  [13] "Zimbabwe" "Zimbabwe" "Zimbabwe" "Zimbabwe"
#>  [17] "Zimbabwe" "Zimbabwe" "Zimbabwe" "Zimbabwe"
#>  [21] "Zimbabwe" "Zimbabwe" "Zimbabwe" "Zimbabwe"
```

Regex Examples - Finding matching

Same results with pipe operator (%>%):

```
library(gapminder)
library(tidyverse)

gapminder %>%
  select(country) %>%
  unlist() %>%
  grep(pattern = "we")

#>  [1] 1465 1466 1467 1468 1469 1470 1471 1472 1473 1474
#> [11] 1475 1476 1693 1694 1695 1696 1697 1698 1699 1700
#> [21] 1701 1702 1703 1704

gapminder %>%
  select(country) %>%
  unlist() %>%
  grep(pattern = "we", value=TRUE)

#> country1465 country1466 country1467 country1468 country1469 country1470
#>    "Sweden"    "Sweden"    "Sweden"    "Sweden"    "Sweden"    "Sweden"
#> country1471 country1472 country1473 country1474 country1475 country1476
#>    "Sweden"    "Sweden"    "Sweden"    "Sweden"    "Sweden"    "Sweden"
#> country1693 country1694 country1695 country1696 country1697 country1698
#>  "Zimbabwe"  "Zimbabwe"  "Zimbabwe"  "Zimbabwe"  "Zimbabwe"  "Zimbabwe"
#> country1699 country1700 country1701 country1702 country1703 country1704
#>  "Zimbabwe"  "Zimbabwe"  "Zimbabwe"  "Zimbabwe"  "Zimbabwe"  "Zimbabwe"
```

Regex Examples - Extraction

```
library(tidyverse)

# EXTRACTING THE FIRST WORD WITH `[a-z]+` AS THE EXPRESSION
c("The 13 Cats in the Hats are 17 years",
  "4 scores and 7 beers ago") %>%
str_extract("[a-z]+")

#> [1] "he"     "scores"

# EXTRACTING ALL THE WORDS WITH `str_extract_all()`
c("The 13 Cats in the Hats are 17 years",
  "4 scores and 7 beers ago") %>%
  str_extract_all("[a-z]+")

#> [[1]]
#> [1] "he"    "ats"   "in"    "the"   "ats"   "are"   "years"
#> [[2]]
#> [1] "scores" "and"    "beers"  "ago"

# EXTRACTING ANY DIGITS WITH `[0-9]+` AS THE EXPRESSION
c("The 13 Cats in the Hats are 17 years",
  "4 scores and 7 beers ago") %>%
str_extract("[0-9]+")

#> [1] "13" "4"
```

Regex Examples - Extraction: More complecated Regex

```
library(tidyverse)

#THE STARTING WORD OF THE FIRST 10 SENTENCES
stringr::sentences[1:10] %>%
    str_extract("[A-Z]+[a-z]*")

#>  [1] "The"     "Glue"    "It"      "These"   "Rice"    "The"     "The"
#>  [8] "The"     "Four"    "A"


# NOUNS OF THE FIRST 10 SENTENCES
noun_regex_pattern <- "(a|the) ([^ ]+)"

sentences[1:10] %>%
  str_subset(noun_regex_pattern) %>%
  str_match(noun_regex_pattern)
#>       [,1]          [,2]  [,3]
#> [1,] "the smooth" "the" "smooth"
#> [2,] "the sheet"  "the" "sheet"
#> [3,] "the depth"  "the" "depth"
#> [4,] "a chicken"  "a"   "chicken"
#> [5,] "the parked" "the" "parked"
```

Regex Examples - Tidyr separate()

```
library(tidyverse)

print(tidyr::table3)

#> # A tibble: 6 X 3
#>   country      year rate
#> * <chr>       <int> <chr>
#> 1 Afghanistan  1999 745/19987071
#> 2 Afghanistan  2000 2666/20595360
#> 3 Brazil       1999 37737/172006362
#> 4 Brazil       2000 80488/174504898
#> 5 China        1999 212258/1272915272
#> 6 China        2000 213766/1280428583

# THE 'rate' COLUMN HAS TWO IMPORTANT VARIABLES THAT ARE SEPARATED BY '/'
# ONE SOLUTION CAN BE:
tidyr::table3 %>%
  separate(rate, into = c("cases", "pop"), sep = "/")

#> # A tibble: 6 X 4
#>   country      year cases  pop
#>   <chr>       <int> <chr>  <chr>
#> 1 Afghanistan  1999 745    19987071
#> 2 Afghanistan  2000 2666   20595360
#> 3 Brazil       1999 37737  172006362
#> 4 Brazil       2000 80488  174504898
#> 5 China        1999 212258 1272915272
#> 6 China        2000 213766 1280428583
```

Regex Examples - Tidyr exctract()

```
library(tidyverse)

noun_regex_pattern <- "(a|the) ([^ ]+)"

tibble(sentence = sentences) %>%
  tidyr::extract(
    sentence, c("article", "noun"), noun_regex_pattern,
    remove = FALSE
  )
#> # A tibble: 720 X 3
#>    sentence                                    article noun
#>    <chr>                                       <chr>   <chr>
#>  1 The birch canoe slid on the smooth planks.  the     smooth
#>  2 Glue the sheet to the dark blue background. the     sheet
#>  3 It's easy to tell the depth of a well.      the     depth
#>  4 These days a chicken leg is a rare dish.    a       chicken
#>  5 Rice is often served in round bowls.        <NA>    <NA>
#>  6 The juice of lemons makes fine punch.       <NA>    <NA>
#>  7 The box was thrown beside the parked truck. the     parked
#>  8 The hogs were fed chopped corn and garbage. <NA>    <NA>
#>  9 Four hours of steady work faced us.         <NA>    <NA>
#> 10 Large size in stockings is hard to sell.    <NA>    <NA>
#> # ... with 710 more rows
```

Machine learning?

Automatically detect patterns in data

Machine learning?

Automatically detect patterns in data

Predict future observation

Machine learning?

Automatically detect patterns in data

Predict future observation

Decision making under uncertainty

Types of Machine learning

Supervised learning

Types of Machine learning

Supervised learning

Unsupervised learning

Types of Machine learning

Supervised learning

Unsupervised learning

Reinforcement learning

Supervised learning

(also called predictive learning)

response variable

covariates/features

training set

$$D = (x_i, y_i)_{(i=1)}^{N}$$

Supervised learning examples

If y_i is categorical:
classification

If y_i is real:
regression

Unsupervised learning

(also called knowledge discovery)

dimensionality reduction

latent variable modeling

$$D = (x_i)_{(i=1)}^{N}$$

clustering, PCA, discovering of graph structures

data visualization

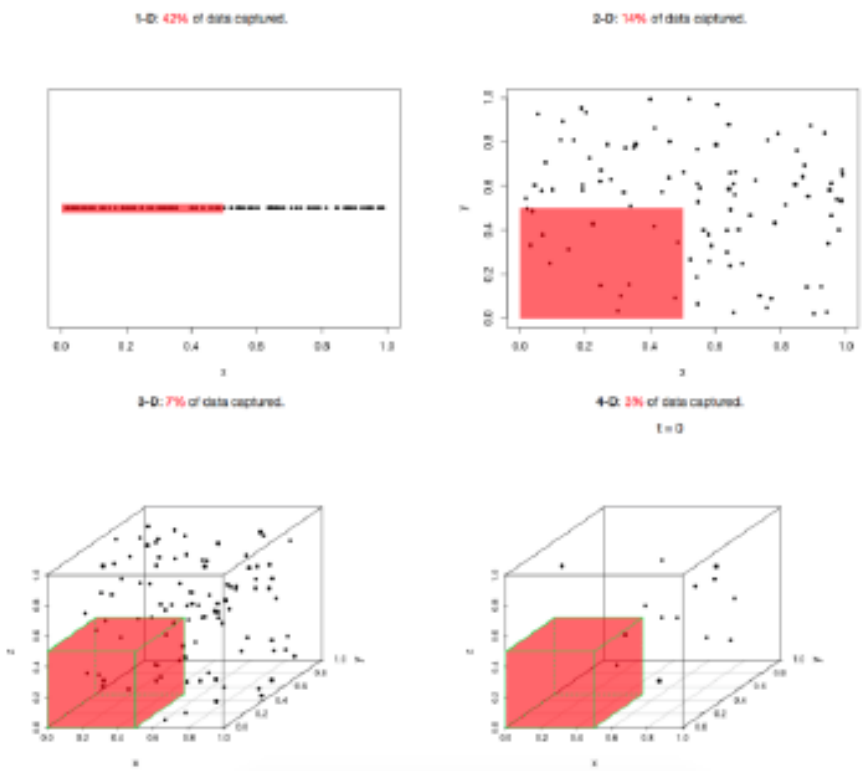
Curse of dimensionality

The more variables the larger distance between datapoints

Euclidian metric

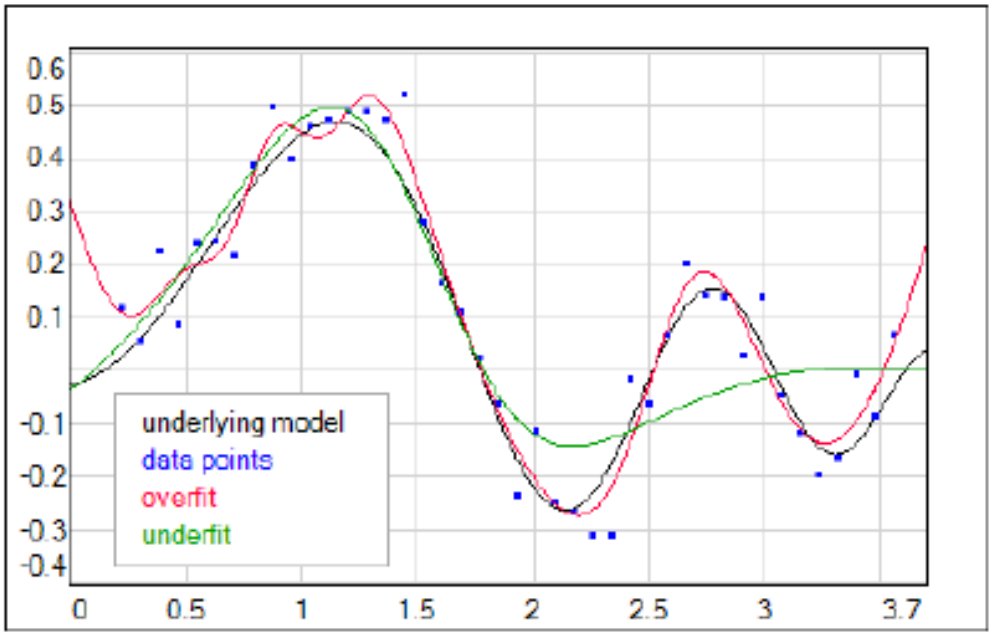
$$d^2(\vec{x}, \vec{y}) = (x_1 - y_1)^2 + (x_2 - y_2)^2 + (x_3 - y_3)^2 + (x_4 - y_4)^2 + \dots$$

Curse of dimensionality



http://www.newsnshit.com/curse-of-dimensionality-interactive-demo/

Fit (bias) and variance in ML



Underfit = bad fit, low variance

Overfit = good fit, high variance

NetMaker (Neural networks simulator and designer by Robert Sulej, Warsaw Univ. Tech.):

`http://www.ire.pw.edu.pl/~rsulej/NetMaker/index.php?pg=e06`

Model selection

fit and variance - tradeoff

Model selection

fit and variance - tradeoff

hyper parameters

Model selection

fit and variance - tradeoff

hyper parameters

generalization error

Model selection

fit and variance - tradeoff

hyper parameters

generalization error

validation set/cross validation

Model selection

fit and variance - tradeoff

hyper parameters

generalization error

validation set/cross validation

information criteria: model fit penalized for model dimensionality

Predictive modeling pipeline

1. Set aside data for test (estimate generalization error)
2. Set aside data for validation (if hyperparams)
3. Run algorithms
4. Find best/optimal hyperparameters (on validation set)
5. Choose final model
6. Estimate generalization error on test set

No free lunch

different models work in different domains

No free lunch

different models work in different domains

accuracy-complexity-intepretability tradeoff

No free lunch theorem

different models work in different domains

accuracy-complexity-intepretability tradeoff

...but more data always wins

the caret package

package for supervised learning

the caret package

package for supervised learning

does not contain methods - a framework

the caret package

package for supervised learning

does not contain methods - a framework

compare methods on hold-out-data

the caret package

package for supervised learning

does not contain methods - a framework

compare methods on hold-out-data

`http://topepo.github.io/caret/`

the caret package

package for supervised learning

does not contain methods - a framework

compare methods on hold-out-data

`http://topepo.github.io/caret/`

specific algorithms are part of other courses

Probability Functions

Prefix	Description	Example
r	Random draw	rnorm
d	Density function	dbinom
q	Quantile function	qbeta
p	CDF	pgamma

Big data

Today's trend:

- whole genome
- surveillance cameras (CCTV)
- Internet traffic
- credit card transactions
- everything communicating with everything

Big data is relative...

... to computational complexity

$O(N) \quad 10^{12}$

Big data is relative...

... to computational complexity

$O(N)$	10^{12}
$O(N^2)$	10^6

Big data is relative...

... to computational complexity

$O(N)$	10^{12}
$O(N^2)$	10^6
$O(N^3)$	10^4

Big data is relative...

... to computational complexity

$O(N)$	10^{12}
$O(N^2)$	10^6
$O(N^3)$	10^4
$O(2^N)$	50

Big data is relative...

... to computational complexity

$O(N)$	10^{12}
$O(N^2)$	10^6
$O(N^3)$	10^4
$O(2^N)$	50

We need algorithms that scale!

Big data is relative...

... to computational complexity

$O(P^2 * N)$	Linear regression
$O(N^3)$	Gaussian processes
$O(N^2)/O(N^3)$	Support vector machines
$O(T(P * N * log(N))$	Random forests
$O(I * N)$	Topic models

Big data in R

R stores data in RAM

Big data in R

R stores data in RAM

integers
4 bytes
numerics
8 bytes

Big data in R

R stores data in RAM

integers
4 bytes
numerics
8 bytes

A matrix with 100M rows and 5 cols with numerics
$100000000 * 5 * 8/(1024^3) \approx 3.8GB$

`help(Memory); help("Memory-limits")`
Genome storage ... ?

How to deal with large data sets

Handle chunkwise
Subsampling
More hardware
C++/Java backend (dplyr)
Reduce data in memory
Database backend

If not enough

Spark and SparkR

Fast cluster computations for ML /STATS

Introduction to Spark:
`https://www.youtube.com/watch?v=_Ss1Cm6W0-I`

The End... for the meantime...

Good luck!

See you next time!