Text Processing in R

Professor Bear

September 1, 2018

In this lesson we'll learn text processing and regular expressions in R.

Additional packages needed

- If necessary install the followings packages.

```
install.packages("stringr");
```

```
library(stringr)
```

Text Processing in R

Lets start with an example string.

```
my_string <- "Find a goal you are passionate about. Then pursue it.
Relentlessly."
my_string
```

```
## [1] "Find a goal you are passionate about. Then pursue it. Relentlessly."
```

```
lower_string <- tolower(my_string)
lower_string
```

```
## [1] "find a goal you are passionate about. then pursue it. relentlessly."
```

```
my_string_vector <- str_split(my_string, " ")
my_string_vector
```

```
## [[1]]
##  [1] "Find"          "a"             "goal"          "you"
##  [5] "are"           "passionate"    "about."        "Then"
##  [9] "pursue"        "it."           "Relentlessly."
```

Here's a quick cheat-sheet on string manipulation functions in R, from Quick-R's list of String Functions.

```
substr(x, start=n1, stop=n2)
grep(pattern,x, value=FALSE, ignore.case=FALSE, fixed=FALSE)
gsub(pattern, replacement, x, ignore.case=FALSE, fixed=FALSE)
gregexpr(pattern, text, ignore.case=FALSE, perl=FALSE,
fixed=FALSE)
strsplit(x, split)
```

```
paste(…, sep="", collapse=NULL)
sprintf(fmt, …)
toupper/tolower(x)
nchar(x)
```

Regular Expressions and Grep

In theoretical computer science and formal language theory, a regular expression (abbreviated regex or regexp and sometimes called a rational expression) is a sequence of characters that define a search pattern, mainly for use in pattern matching with strings, or string matching, i.e. "find and replace"-like operations. The concept arose in the 1950s, when the American mathematician Stephen Kleene formalized the description of a regular language, and came into common use with the Unix text processing utilities ed, an editor, and grep (global regular expression print), a filter.

grep is a command-line utility for searching plain-text data sets for lines matching a regular expression. Grep was originally developed for the Unix operating system, but is available today for all Unix-like systems and is built in to languages like python and Perl.

Regular Expressions Examples

Basic regex syntax

```
.   Normally matches any character except a newline.

When you match a pattern within parentheses, you can use any of $1, $2, ...
later to refer to the previously matched pattern.

+   Matches the preceding pattern element one or more times.
?   Matches the preceding pattern element zero or one times.
*   Matches the preceding pattern element zero or more times.
|   Separates alternate possibilities.

\w  Matches an alphanumeric character, including "_";  same as [A-Za-z0-9_]
in ASCII, and
[\p{Alphabetic}\p{GC=Mark}\p{GC=Decimal_Number}\p{GC=Connector_Punctuation}]

\W  Matches a non-alphanumeric character, excluding "_";
same as [^A-Za-z0-9_] in ASCII, and
[^\p{Alphabetic}\p{GC=Mark}\p{GC=Decimal_Number}\p{GC=Connector_Punctuation}]

\s  Matches a whitespace character,
which in ASCII are tab, line feed, form feed, carriage return, and space;

\S  Matches anything BUT a whitespace.

\d  Matches a digit;
```

```
same as [0-9] in ASCII;

\D  Matches a non-digit;

^   Matches the beginning of a line or string.

$   Matches the end of a line or string.
```

Some simple regex examples

```
 {^[-+]?[0-9]*\.?[0-9]+([eE][-+]?[0-9]+)?$}  # Floating Point Number

{^[A-Za-z]+$}   # Only letters.

 {^[[:alpha?:]]+$} # Only letters, the Unicode way.

 {(.)\1{3}} $string {\1} result # Back References

(\[0-9]{1,3})\.(\[0-9]{1,3})\.(\[0-9]{1,3})\.(\[0-9]{1,3}) # IP Numbers
```

Regular Expressions in R

```
letters
```

```
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q"
## [18] "r" "s" "t" "u" "v" "w" "x" "y" "z"
```

```
grep("[a-z]", letters)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
## [24] 24 25 26
```

```
grep("[x-z]", letters)
```

```
## [1] 24 25 26
```

```
grep("[bear]", letters)
```

```
## [1]  1  2  5 18
```

```
grep("[A-Z]", letters)
```

```
## integer(0)
```

```
grep("[AB]", letters)
```

```
## integer(0)
```

```
grep("[AB]", letters,ignore.case = TRUE)
```

```
## [1] 1 2
```

```r
grep("[a-zA-Z]", letters)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
## [24] 24 25 26
```

```r
grep("[azAZ]", letters)
```

```
## [1]  1 26
```

```r
seuss <- c("You have brains in your head.",
           "You have feet in your shoes.",
           "You can steer yourself any direction you choose.",
           "You're on your own.",
           "And you know what you know.",
           "And YOU are the one who'll decide where to go...",
           "- Dr. Seuss")
grep("you", seuss)
```

```
## [1] 1 2 3 4 5
```

```r
if(length(i <- grep("in", seuss)))
  cat("'in' appears at least once in\n\t", seuss[i], "\n")
```

```
## 'in' appears at least once in
##   You have brains in your head. You have feet in your shoes.
```

```r
i # 1 and 2
```

```
## [1] 1 2
```

```r
seuss[i]
```

```
## [1] "You have brains in your head." "You have feet in your shoes."
```

```r
 ## Modify all 'a' or 'b's;  "\" must be escaped
s<-gsub("([ab])", "\\1-z\\1_s", "abc vs, ABC  - Oh, A B C. It's easy as, 1 2
3. As simple as, do re mi. a b c, 1 2 3")
s
```

```
## [1] "a-za_sb-zb_sc vs, ABC  - Oh, A B C. It's ea-za_ssy a-za_ss, 1 2 3. As
simple a-za_ss, do re mi. a-za_s b-zb_s c, 1 2 3"
```

```r
s<-gsub("([ab])", "-z_s", "abc vs, ABC  - Oh, A B C. It's easy as, 1 2 3. As
simple as, do re mi. a b c, 1 2 3")
s
```

```
## [1] "-z_s-z_sc vs, ABC  - Oh, A B C. It's e-z_ssy -z_ss, 1 2 3. As simple
-z_ss, do re mi. -z_s -z_s c, 1 2 3"
```

```r
s<-gsub("([ab])", "\\a\\b", "abc vs, ABC  - Oh, A B C. It's easy as, 1 2 3.
As simple as, do re mi. a b c, 1 2 3")
s
```

```
## [1] "ababc vs, ABC  - Oh, A B C. It's eabsy abs, 1 2 3. As simple abs, do
re mi. ab ab c, 1 2 3"

stop.words <- c("The", "for", "are",
          "to",  "your",
         "to", "and", "the",
         "is", "your",  "and", "is",
         "it", "for", "all", "its", "it's")

seuss.re<-regexpr("[ ]+",seuss)
seuss.re

## [1] 4 4 4 7 4 4 2
## attr(,"match.length")
## [1] 1 1 1 1 1 1 1
## attr(,"index.type")
## [1] "chars"
## attr(,"useBytes")
## [1] TRUE

seuss.re<-gregexpr("[ ]+",seuss)
seuss.re

## [[1]]
## [1]  4  9 16 19 24
## attr(,"match.length")
## [1] 1 1 1 1 1
## attr(,"index.type")
## [1] "chars"
## attr(,"useBytes")
## [1] TRUE
##
## [[2]]
## [1]  4  9 14 17 22
## attr(,"match.length")
## [1] 1 1 1 1 1
## attr(,"index.type")
## [1] "chars"
## attr(,"useBytes")
## [1] TRUE
##
## [[3]]
## [1]  4  8 14 23 27 37 41
## attr(,"match.length")
## [1] 1 1 1 1 1 1 1
## attr(,"index.type")
## [1] "chars"
## attr(,"useBytes")
## [1] TRUE
##
## [[4]]
```

```
## [1]  7 10 15
## attr(,"match.length")
## [1] 1 1 1
## attr(,"index.type")
## [1] "chars"
## attr(,"useBytes")
## [1] TRUE
##
## [[5]]
## [1]  4  8 13 18 22
## attr(,"match.length")
## [1] 1 1 1 1 1
## attr(,"index.type")
## [1] "chars"
## attr(,"useBytes")
## [1] TRUE
##
## [[6]]
## [1]  4  8 12 16 20 27 34 40 43
## attr(,"match.length")
## [1] 1 1 1 1 1 1 1 1 1
## attr(,"index.type")
## [1] "chars"
## attr(,"useBytes")
## [1] TRUE
##
## [[7]]
## [1] 2 6
## attr(,"match.length")
## [1] 1 1
## attr(,"index.type")
## [1] "chars"
## attr(,"useBytes")
## [1] TRUE

## trim trailing white space
str <- " You have brains in your head      "
str

## [1] " You have brains in your head      "

str<-sub("[ ]+$", "", str)  ## trailing spaces only
str

## [1] " You have brains in your head"

str<-sub("^[ ]+", "", str)  ## leading spaces only
str

## [1] "You have brains in your head"
```

```r
sub("[[:space:]]+$", "", str) ## white space, POSIX-style
```

```
## [1] "You have brains in your head"
```

```r
sub("\\s+$", "", str, perl = TRUE) ## Perl-style white space
```

```
## [1] "You have brains in your head"
```

```r
## capitalizing
txt <- "You have brains in your head"
gsub("(\\w)(\\w*)", "\\U\\1\\L\\2", txt, perl=TRUE)
```

```
## [1] "You Have Brains In Your Head"
```

```r
gsub("\\b(\\w)",    "\\U\\1",       txt, perl=TRUE)
```

```
## [1] "You Have Brains In Your Head"
```

```r
txt2 <- "You have feet in your shoes."
gsub("(\\w)(\\w*)(\\w)", "\\U\\1\\E\\2\\U\\3", txt2, perl=TRUE)
```

```
## [1] "YoU HavE FeeT IN YouR ShoeS."
```

```r
sub("(\\w)(\\w*)(\\w)", "\\U\\1\\E\\2\\U\\3", txt2, perl=TRUE)
```

```
## [1] "YoU have feet in your shoes."
```

```r
## Decompose a URL into its components.
url <- "http://nikbearbrown/machine/learning/"
m <- regexec("^(([^:]+)://)?([^:/]+)(:([0-9]+))?(/.*)", url)
m
```

```
## [[1]]
## [1]  1  1  1  8  0  0 20
## attr(,"match.length")
## [1] 37  7  4 12  0  0 18
## attr(,"index.type")
## [1] "chars"
## attr(,"useBytes")
## [1] TRUE
```

```r
regmatches(url, m)
```

```
## [[1]]
## [1] "http://nikbearbrown/machine/learning/"
## [2] "http://"
## [3] "http"
## [4] "nikbearbrown"
## [5] ""
## [6] ""
## [7] "/machine/learning/"
```

```r
## parts of a URL:
URL_parts <- function(u) {
  m <- regexec("^(([^:]+)://)?([^:/]+)(:([0-9]+))?(/.*)", u)
  parts <- do.call(rbind,
                   lapply(regmatches(u, m), `[`, c(3L, 4L, 6L, 7L)))
  colnames(parts) <- c("protocol","host","port","path")
  parts
}
URL_parts(url)
```

```
##      protocol host           port path
## [1,] "http"   "nikbearbrown" ""   "/machine/learning/"
```

```r
# Spilt text
seuss
```

```
## [1] "You have brains in your head."
## [2] "You have feet in your shoes."
## [3] "You can steer yourself any direction you choose."
## [4] "You're on your own."
## [5] "And you know what you know."
## [6] "And YOU are the one who'll decide where to go..."
## [7] "- Dr. Seuss"
```

```r
seuss[1]
```

```
## [1] "You have brains in your head."
```

```r
sp<-strsplit(seuss[1], "a") # "You have brains in your head."
s<-"You    have   brains  in  your head. "
sp<-strsplit(seuss[1], " ")
sp
```

```
## [[1]]
## [1] "You"    "have"   "brains" "in"     "your"   "head."
```

```r
sp<-strsplit(s," ")
sp
```

```
## [[1]]
##  [1] "You"    ""       ""       ""       "have"   ""       ""
##  [8] "brains" ""       "in"     ""       "your"   "head."
```

```r
sp<-strsplit(s,"[ ]+")
sp
```

```
## [[1]]
## [1] "You"    "have"   "brains" "in"     "your"   "head."
```

```r
sp<-strsplit(s,"[ ]+",perl = TRUE)
sp
```

```
## [[1]]
## [1] "You"    "have"   "brains" "in"     "your"   "head."
```

Cleaning text example.

Text Processing in R

```
clean.string <- function(string){
    # Lowercase
    temp <- tolower(string)
    #' Remove everything that is not a number or letter (may want to keep
more
    #' stuff in your actual analyses).
    temp <- stringr::str_replace_all(temp,"[^a-zA-Z\\s]", " ")
    # Shrink down to just one white space
    temp <- stringr::str_replace_all(temp,"[\\s]+", " ")
    # Split it
    temp <- stringr::str_split(temp, " ")[[1]]
    # Get rid of trailing "" if necessary
    indexes <- which(temp == "")
    if(length(indexes) > 0){
      temp <- temp[-indexes]
    }
    return(temp)
}
clean.string(my_string)
```

```
##  [1] "find"         "a"            "goal"         "you"
##  [5] "are"          "passionate"   "about"        "then"
##  [9] "pursue"       "it"           "relentlessly"
```