Capitulo 8

Los pasos basicos del analisis de textos de datos

A un nivel muy alto, el análisis de datos de texto consta de los siguientes pasos: 1. Obtenga el texto para analizar; 2. Normalice el texto, eliminando detalles irrelevantes, por ejemplo: 2a. Eliminar efectos de mayúsculas y minúsculas: convierte todo el texto a minúsculas o mayúsculas; 2b. Eliminar caracteres molestos: puntuación, símbolos y números; 2c. Analiza el texto en palabras; 2d. Eliminar palabras no informativas; 2e. Deje el texto, eliminando las terminaciones de palabras como “ing”, “s” o “ed”; 3. Convierta el texto normalizado en números; 4. Aplicar herramientas de análisis de datos matemáticos a estos números; 5. Interprete estos resultados de análisis en términos de los datos del texto original.

```
cv <- c("This vector is a character vector.",
"It has three elements.",
"Each element is a text string.")
```

Funciones basicas de char en R

La funcion n char La función nchar cuenta el número de caracteres en una cadena de caracteres o un vector de cadenas de caracteres.

```
#firstNames <- as.character(unclaimedFrame$First.Name)
#firstNames[1:20]
```

La funcion grep

La función grep nos permite buscar la presencia de una subcadena en un vector de caracteres especificado. Específicamente, dada una subcadena subStr y un vector de caracteres charVec, la llamada grep (subStr, charVec) devuelve una lista de los elementos de charVec que contienen subStr una o más veces.

```
#index <- grep("O", firstNames[1:10])
#index
```

Funciones Sub y gsub

Las funciones sub y gsub realizan sustituciones de cadenas, que difieren en que sub reemplaza la primera aparición solo de la subcadena objetivo, mientras que gsub reemplaza todas las ocurrencias. Ambas funciones tienen los mismos tres argumentos requeridos, que aparecen en el mismo orden: 1. patrón es la subcadena a ser reemplazada; 2. reemplazo es el texto de reemplazo para el patrón; 3. x es el vector de caracteres donde se va a reemplazar el patrón.

```
x <- "google search on oolong and oobleck associates"
sub(pattern = "o", replacement = "X", x)
```

```
## [1] "gXogle search on oolong and oobleck associates"
```

```
gsub(pattern = "o", replacement = "X", x)
```

```
## [1] "gXXgle search Xn XXlXng and XXbleck assXciates"
```

La funcion strsplit

La función strsplit divide cada elemento de un vector de caracteres alrededor de una subcadena especificada. Los argumentos requeridos para esta función son:

1. x, el vector de caracteres que se dividirá;
2. split, la subcadena en la que se deben dividir los elementos de x.

Dado que x es un vector de caracteres y cada elemento de este vector se divide en subcadenas separadas por división, el número de estas subcadenas en cada división no se puede determinar de antemano, por lo que

esta función devuelve un objeto de lista en lugar de un vector, con cada elemento de lista que contiene todos los componentes divididos del elemento correspondiente de x.

Al igual que con las funciones grep, sub y gsub, la división del argumento se trata como una expresión regular a menos que el argumento opcional fijo se establezca en TRUE. Los siguientes ejemplos ilustran cómo funciona la función strsplit:

```
x <- c("a comma b comma c", "a, b, c", "a comma (,) b comma (,) c")
strsplit(x, split = ",")
```

```
## [[1]]
## [1] "a comma b comma c"
##
## [[2]]
## [1] "a"  " b" " c"
##
## [[3]]
## [1] "a comma ("   ") b comma (" ") c"
```

Otra aplicacion ConvertAutoMpgRecords

autoMpg del Depósito de aprendizaje automático UCI, descargado como un archivo de texto con registros de ancho fijo que contienen campos numéricos separados por espacios y un campo de texto con comillas incrustadas, separadas por un carácter de tabulación.La función R personalizada ConvertAutoMpgRecords se desarrolló para extraer los datos de estos campos y guardarlos en un marco de datos con el siguiente formato

```
#head(autoMpgFrame, n = 3)
```

La funcion paste

La función pegar combina cadenas de texto más cortas en cadenas más largas, y se ha utilizado en ejemplos presentados en capítulos anteriores. En aplicaciones típicas, esta función se llama con una colección de objetos R, cada uno de los cuales es una cadena de caracteres o se convierte en uno, y estos componentes se concatenan en una cadena de texto más larga. El siguiente ejemplo ilustra este comportamiento básico:

```
n <- 4
textString <- paste("The value of pi to", n, "digits is", round(pi, digits = n))
textString
```

```
## [1] "The value of pi to 4 digits is 3.1416"
```

```
n <- 4
textString <- paste("The value of pi to", n, "digits is", round(pi, digits = n),
sep = ";")
textString
```

```
## [1] "The value of pi to;4;digits is;3.1416"
```

```
paste(c("A", "B"), seq(1, 10, 1), c("x", "y", "z"), sep = ":")
```

```
##  [1] "A:1:x"  "B:2:y"  "A:3:z"  "B:4:x"  "A:5:y"  "B:6:z"  "A:7:x"
##  [8] "B:8:y"  "A:9:z"  "B:10:x"
```

```
str1 <- "mtcars variables:"
str2 <- paste(colnames(mtcars), collapse = ", ")
paste(str1, str2)
```

```
## [1] "mtcars variables: mpg, cyl, disp, hp, drat, wt, qsec, vs, am, gear, carb"
```

Expresiones Regulares Basicas

Esta función acepta varios argumentos opcionales para restringir nuestra búsqueda, lo que puede ser útil si tenemos muchos archivos. Uno de estos argumentos es el patrón, que especifica coincidencias parciales en los nombres de archivo y se interpreta como una expresión regular. Las siguientes especificaciones recuperan los archivos indicados, si los hay, de nuestro directorio de trabajo:

1. list.files (pattern = "R") enumera los nombres de archivo que contienen la letra R.
2. list.files (pattern = "R $") enumera los nombres de archivo que terminan con la letra R.
3. list.files (pattern = ".R") enumera los nombres de archivo que contienen la letra R. precedido por cualquier otra cosa (es decir, no al principio del nombre del archivo).
4. list.files (pattern = "R.") enumera los nombres de archivo que contienen la letra R. seguido de cualquier otra cosa (es decir, no al final del nombre del archivo).

```
x <- c("This is a vector of text strings",
"The first contains no period, but the second one does.",
"The third contains ...", "The fourth is like the first")
grep(".", x, value = TRUE)
```

```
## [1] "This is a vector of text strings"
## [2] "The first contains no period, but the second one does."
## [3] "The third contains ..."
## [4] "The fourth is like the first"
```

```
grep(".", x, value = TRUE, fixed = TRUE)
```

```
## [1] "The first contains no period, but the second one does."
## [2] "The third contains ..."
```

```
x <- "One string ... with both embedded ellipses (...) ... and a final period."
gsub(".", "", x)
```

```
## [1] ""
```

```
x <- "One string ... with both embedded ellipses (...) ... and a final period."
gsub(".", "", x, fixed = TRUE)
```

```
## [1] "One string  with both embedded ellipses ()  and a final period"
```

```
x <- "One string ... with both embedded ellipses (...) ... and a final period."
gsub(".", "", x)
```

```
## [1] ""
```

```
x <- "One string ... with both embedded ellipses (...) ... and a final period."
gsub(".", "", x, fixed = TRUE)
```

```
## [1] "One string  with both embedded ellipses ()  and a final period"
```

Porque las expresiones regulares son extremadamente poderosas y algo contradictorias (al menos hasta que tenga una experiencia sustancial con ellas).

Es importante recordar que las expresiones regulares son pequeñas lenguaje informático propio y debe desarrollarse con solo tanta disciplina y cuidado como aplicamos para escribir cualquier computadora código.

En particular, se debe construir una expresión regular compleja en piezas más pequeñas para comprender cómo cada componente de la expresión regular funciona antes de agregar más componentes.

Char set Los corchetes en una expresión regular, es decir, un corchete de apertura ("["), seguido de un montón de texto, seguido de un corchete de cierre ("]"), definen un conjunto de caracteres que se puede utilizar para que coincida con cualquier carácter en el conjunto. Como ejemplo simple, el siguiente código reemplaza todas las vocales en una cadena de texto con un guión bajo ("_"):

```
x <- "This text string is a typical example, not very interesting."
gsub("[aeiouy]", "_", x)
```

```
## [1] "Th_s t_xt str_ng _s _ t_p_c_l _x_mpl_, n_t v_r_ _nt_r_st_ng."
```

```
x <- "A note from Benny Thornton about Eddie Coyle, 32."
gsub("[a-z]", "_", x)
```

```
## [1] "A ____ ____ B____ T_______ _____ E____ C____, 32."
```

```
gsub("[0-9]", "_", x)
```

```
## [1] "A note from Benny Thornton about Eddie Coyle, __."
```

```
gsub("[A-L]", "_", x)
```

```
## [1] "_ note from _enny Thornton about _ddie _oyle, 32."
```

```
x <- "A note from Benny Thornton about Eddie Coyle, 32."
gsub("[^a-z]", "_", x)
```

```
## [1] "__note_from__enny__hornton_about__ddie__oyle_____"
```

```
gsub("[^0-9]", "_", x)
```

```
## [1] "_____________________________________________32_"
```

```
gsub("[^A-L]", "_", x)
```

```
## [1] "A__________B___________________E_____C_________"
```

En el primer ejemplo, la expresión regular "[^ a-z]" en la función gsub significa "reemplazar cualquier carácter excepto las letras minúsculas con un guión bajo". Por lo tanto, las letras mayúsculas, espacios, signos de puntuación y números se reemplazan con guiones bajos, pero ninguna de las letras minúsculas son.

En el segundo ejemplo, todo excepto los dígitos del 0 al 9 se reemplaza con un guión bajo, y en el ejemplo final, todo excepto las letras mayúsculas A a L se reemplaza con un guión bajo.

Repeticion de MetaCracteres Los metacaracteres "?", "*" y"+" en una expresión regular causan diferentes grados de repetición, y pueden ser particularmente útiles junto con los conjuntos de caracteres definidos al abrir y cerrar corchetes. Estos metacaracteres provocan el siguiente comportamiento de coincidencia de subpatrón:

el metacarácter "?" Coincide con cero o exactamente una vez; el metacarácter "*" coincide con cero o más veces; el metacarácter "+" coincide una o más veces.

```
x <- "This is a text string; it is...it is."
gsub("is[.]", "_", x)
```

```
## [1] "This is a text string; it _..it _"
```

```
gsub("is[.]?", "_", x)
```

```
## [1] "Th_ _ a text string; it _..it _"
```

```
gsub("is[.]*", "_", x)
```

```
## [1] "Th_ _ a text string; it _it _"
```

```
gsub("is[.]+", "_", x)
```

```
## [1] "This is a text string; it _it _"
```

ASCII vs UNICOIDE

Una función útil disponible en el paquete de herramientas es showNonASCII, que se puede aplicar a un vector de caracteres para encontrar cualquier carácter que no sea ASCII. Al aplicar esta función al vector de caracteres de la cuenta bancaria no reclamada, unclaimedLines revela que los primeros 60 caracteres del elemento 4528 de este vector contienen texto no ASCII:

```
#library(tools)
#x <- unclaimedLines[4528]
#y <- substr(x, 1, 60)
#showNonASCII(y)
```

Ejemplo Caracterizacion de un Libro

```
BornAgain <- readLines("LawsonBornAgain.txt")
head(BornAgain)
```

```
## [1] ""
## [2] ""
## [3] ""
## [4] "The Project Gutenberg EBook of Born Again, by Alfred Lawson"
## [5] ""
## [6] "This eBook is for the use of anyone anywhere at no cost and with"
```

```
firstLine <- grep("CHAPTER I$", BornAgain)
lastLine <- grep("(THE END.)", BornAgain, fixed = TRUE)
LawsonText <- BornAgain[firstLine:lastLine]
```

```
## Warning in firstLine:lastLine: numerical expression has 2 elements: only
## the first used
```

El resultado es un vector de caracteres con 6195 elementos y un examen más detallado. muestra que el texto completo puede dividirse en estas partes: 1. Un preámbulo del Proyecto Gutenberg, identificando la fuente y el documento; 2. Una breve sección de Lawson llamada “DEDICACIÓN”; 3. El cuerpo principal del texto consta de 33 secciones, cada una etiquetada con el palabra “CAPÍTULO”, seguido de una sección final llamada “EPÍLOGO”; 4. Una sección de comentarios breves de Lawson llamada “STRAY SHOTS”;

```
firstLine <- grep("CHAPTER I$", BornAgain)
lastLine <- grep("(THE END.)", BornAgain, fixed = TRUE)
LawsonText <- BornAgain[firstLine:lastLine]
```

```
## Warning in firstLine:lastLine: numerical expression has 2 elements: only
## the first used
```

```
head(LawsonText)
```

```
## [1] "CHAPTER I"   ""            "CHAPTER II"  ""            "CHAPTER III"
## [6] ""
```

```
tail(LawsonText)
```

```
## [1] ""
## [2] "\"Here the attendants thought I was crazy, as I rushed into the reception room, crying out to s
## [3] ""
## [4] "\"After a fit of sobbing, Arletta Wright quieted herself long enough to say: 'Telegraph the news
## [5] ""
## [6] "(THE END.)"
```

```
CHindex <- grep("CHAPTER", LawsonText)
EPindex <- grep("EPILOGUE", LawsonText)
```

```
ENDindex <- grep("(THE END.)", LawsonText, fixed = TRUE)
```

El cpu dataframe

```
library(MASS)
head(cpus)
```

```
##             name syct mmin  mmax cach chmin chmax perf estperf
## 1  ADVISOR 32/60  125  256  6000  256    16   128  198     199
## 2  AMDAHL 470V/7   29 8000 32000   32     8    32  269     253
## 3  AMDAHL 470/7A   29 8000 32000   32     8    32  220     253
## 4 AMDAHL 470V/7B   29 8000 32000   32     8    32  172     253
## 5 AMDAHL 470V/7C   29 8000 16000   32     8    16  132     132
## 6  AMDAHL 470V/8   26 8000 32000   64     8    32  318     290
```

```
cpusModelA <- lm(I(log(perf)) ~ I(log(syct)), data = cpus)
summary(cpusModelA)
```

```
##
## Call:
## lm(formula = I(log(perf)) ~ I(log(syct)), data = cpus)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.9779 -0.4859  0.0242  0.4825  2.0564
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   7.40205    0.24299   30.46   <2e-16 ***
## I(log(syct)) -0.70884    0.05001  -14.17   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7486 on 207 degrees of freedom
## Multiple R-squared:  0.4925, Adjusted R-squared:   0.49
## F-statistic: 200.9 on 1 and 207 DF,  p-value: < 2.2e-16
```

```
cpusModelB <- lm(I(log(perf)) ~ I(log(syct)) + mmax + cach, data = cpus)
summary(cpusModelB)
```

```
##
## Call:
## lm(formula = I(log(perf)) ~ I(log(syct)) + mmax + cach, data = cpus)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.43258 -0.32159  0.01254  0.30444  1.07290
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   4.554e+00  2.144e-01  21.240  < 2e-16 ***
## I(log(syct)) -2.625e-01  3.896e-02  -6.739 1.59e-10 ***
## mmax          4.461e-05  3.602e-06  12.385  < 2e-16 ***
## cach          8.055e-03  9.543e-04   8.441 5.68e-15 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## Residual standard error: 0.4586 on 205 degrees of freedom
## Multiple R-squared:  0.8114, Adjusted R-squared:  0.8086
## F-statistic: 293.9 on 3 and 205 DF,  p-value: < 2.2e-16
```

```
cpuNames <- as.character(cpus$name)
cpuNames[1:5]
```

```
## [1] "ADVISOR 32/60"  "AMDAHL 470V/7"  "AMDAHL 470/7A"  "AMDAHL 470V/7B"
## [5] "AMDAHL 470V/7C"
```

```
cpuNameSplits <- strsplit(cpuNames, split = " ")
getFirst <- function(x){x[1]}
firstTerms <- unlist(lapply(cpuNameSplits, getFirst))
table(firstTerms)
```

```
## firstTerms
##      ADVISOR       AMDAHL       APOLLO         BASF          BTI
##            1            9            2            2            2
##    BURROUGHS       C.R.D.       CAMBEX          CDC          DEC
##            8            6            5            9            6
##           DG    FORMATION         FOUR        GOULD       HARRIS
##            7            5            1            3            7
##    HONEYWELL           HP          IBM          IPL     MAGNUSON
##           13            7           32            6            6
##    MICRODATA          NAS          NCR      NIXDORF PERKIN-ELMER
##            1           19           13            3            3
##        PRIME      SIEMENS       SPERRY      STRATUS         WANG
##            5           12           13            1            2
```

```
cpusPlus <- cpus
cpusPlus$mfgr <- firstTerms
cpusModelC <- lm(I(log(perf)) ~ I(log(syct)) + mfgr, data = cpusPlus)
summary(cpusModelC)
```

```
##
## Call:
## lm(formula = I(log(perf)) ~ I(log(syct)) + mfgr, data = cpusPlus)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.7195 -0.3554  0.0000  0.3724  1.6262
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)       9.67168    0.69352  13.946  < 2e-16 ***
## I(log(syct))     -0.90786    0.06243 -14.542  < 2e-16 ***
## mfgrAMDAHL       -0.89966    0.66569  -1.351 0.178256
## mfgrAPOLLO       -0.56906    0.76840  -0.741 0.459925
## mfgrBASF         -1.31285    0.76669  -1.712 0.088574 .
## mfgrBTI          -1.67858    0.76639  -2.190 0.029807 *
## mfgrBURROUGHS    -1.47839    0.66250  -2.232 0.026893 *
## mfgrC.R.D.       -0.80128    0.67718  -1.183 0.238285
## mfgrCAMBEX       -2.40491    0.68659  -3.503 0.000582 ***
## mfgrCDC          -1.80622    0.66151  -2.730 0.006961 **
## mfgrDEC          -0.78139    0.67686  -1.154 0.249866
```

```
## mfgrDG           -0.94245    0.66883  -1.409 0.160553
## mfgrFORMATION    -0.80635    0.69395  -1.162 0.246801
## mfgrFOUR         -1.70475    0.88330  -1.930 0.055200 .
## mfgrGOULD        -0.58654    0.72192  -0.812 0.417605
## mfgrHARRIS       -0.69243    0.66959  -1.034 0.302488
## mfgrHONEYWELL    -1.13837    0.64872  -1.755 0.081013 .
## mfgrHP           -2.00188    0.66785  -2.997 0.003111 **
## mfgrIBM          -1.13803    0.63490  -1.792 0.074759 .
## mfgrIPL          -2.00829    0.67705  -2.966 0.003429 **
## mfgrMAGNUSON     -2.22775    0.67557  -3.298 0.001177 **
## mfgrMICRODATA    -1.72155    0.88337  -1.949 0.052887 .
## mfgrNAS          -1.11859    0.64249  -1.741 0.083410 .
## mfgrNCR          -2.36041    0.65042  -3.629 0.000372 ***
## mfgrNIXDORF      -1.41702    0.72181  -1.963 0.051186 .
## mfgrPERKIN-ELMER -0.97801    0.72251  -1.354 0.177568
## mfgrPRIME        -1.20611    0.68438  -1.762 0.079726 .
## mfgrSIEMENS      -1.58791    0.65108  -2.439 0.015714 *
## mfgrSPERRY       -1.16257    0.64868  -1.792 0.074799 .
## mfgrSTRATUS      -1.33702    0.88330  -1.514 0.131884
## mfgrWANG         -0.06109    0.76956  -0.079 0.936814
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6246 on 178 degrees of freedom
## Multiple R-squared:  0.6962, Adjusted R-squared:  0.645
## F-statistic:  13.6 on 30 and 178 DF,  p-value: < 2.2e-16
```

```
cpusModelD <- lm(I(log(perf)) ~ I(log(syct)) + mmax + cach + mfgr,
data = cpusPlus)
summary(cpusModelD)
```

```
##
## Call:
## lm(formula = I(log(perf)) ~ I(log(syct)) + mmax + cach + mfgr,
##     data = cpusPlus)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.1366 -0.1813  0.0000  0.1991  0.9736
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)       5.238e+00  5.239e-01   9.999  < 2e-16 ***
## I(log(syct))     -4.499e-01  4.688e-02  -9.596  < 2e-16 ***
## mmax              4.206e-05  3.919e-06  10.732  < 2e-16 ***
## cach              7.695e-03  1.015e-03   7.582 1.88e-12 ***
## mfgrAMDAHL        3.658e-02  4.880e-01   0.075   0.9403
## mfgrAPOLLO        9.684e-01  5.301e-01   1.827   0.0694 .
## mfgrBASF          2.824e-01  5.110e-01   0.553   0.5812
## mfgrBTI          -1.375e-01  5.350e-01  -0.257   0.7974
## mfgrBURROUGHS     3.546e-01  4.686e-01   0.757   0.4503
## mfgrC.R.D.        7.807e-01  4.829e-01   1.617   0.1078
## mfgrCAMBEX       -3.293e-01  4.956e-01  -0.664   0.5073
## mfgrCDC           1.569e-01  4.612e-01   0.340   0.7341
## mfgrDEC           6.987e-01  4.835e-01   1.445   0.1502
```

```
## mfgrDG            5.771e-01  4.795e-01   1.204   0.2304
## mfgrFORMATION     2.293e-01  4.971e-01   0.461   0.6452
## mfgrFOUR          4.754e-01  5.939e-01   0.800   0.4245
## mfgrGOULD         6.523e-01  4.783e-01   1.364   0.1744
## mfgrHARRIS        8.453e-01  4.807e-01   1.759   0.0804 .
## mfgrHONEYWELL     3.109e-01  4.712e-01   0.660   0.5102
## mfgrHP            1.203e-01  4.797e-01   0.251   0.8023
## mfgrIBM           3.932e-01  4.628e-01   0.850   0.3967
## mfgrIPL           1.329e-02  4.843e-01   0.027   0.9781
## mfgrMAGNUSON     -3.878e-01  4.837e-01  -0.802   0.4238
## mfgrMICRODATA     2.489e-01  5.956e-01   0.418   0.6765
## mfgrNAS           2.682e-01  4.532e-01   0.592   0.5548
## mfgrNCR          -4.044e-01  4.621e-01  -0.875   0.3827
## mfgrNIXDORF       2.295e-01  4.980e-01   0.461   0.6454
## mfgrPERKIN-ELMER  5.877e-01  5.117e-01   1.149   0.2523
## mfgrPRIME         5.920e-01  4.836e-01   1.224   0.2225
## mfgrSIEMENS       5.214e-02  4.639e-01   0.112   0.9106
## mfgrSPERRY       -5.382e-02  4.673e-01  -0.115   0.9084
## mfgrSTRATUS       5.487e-01  5.989e-01   0.916   0.3608
## mfgrWANG          1.169e+00  5.268e-01   2.220   0.0277 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3804 on 176 degrees of freedom
## Multiple R-squared:  0.8886, Adjusted R-squared:  0.8684
## F-statistic: 43.88 on 32 and 176 DF,  p-value: < 2.2e-16
```