



JUEGO DE CAMBIAR UNA LETRA O PERMUTAR

PROCESAMIENTO LENGUAJE NATURAL GRUPO 4



9 DE ENERO DE 2022

UNIVERSIDAD POLITÉCNICA DE MADRID FACULTAD DE INFORMÁTICA CIENCIA DE DATOS E INTELIGENCIA
ARTIFICIAL

09/01/2022

Grado en Ciencia de Datos e Inteligencia Artificial

Procesamiento del Lenguaje Natural – Práctica Grupo 4

Juego de cambiar una letra o permutar

ÍNDICE

| | |
|--|----|
| 1- Programa camper.Rmd..... | 2 |
| 2- Programa camper_interactivo.Rmd | 4 |
| 3- Programa supercamper.Rmd | 7 |
| 4- Anillos | 10 |

1. Programa camper.Rmd

Dada una palabra, haga una secuencia de palabras siguiendo el procedimiento descrito en el enunciado.

Lo primero que vamos a hacer en el programa es descargarnos el diccionario especificado donde están todas las palabras que podemos utilizar. Como el diccionario tiene palabras repetidas, le hemos aplicado `unique()`.

Para hacerlo más eficiente hemos pensado en filtrar el diccionario por el número de caracteres de la palabra que le pasemos, esto nos permitira ser más eficiente. Esto es la función *dicRed*.

LECTURA DE DICCIONARIO

Leemos el diccionario del archivo "dic_es.txt"

```
con <- file(description = '../datos/dic_es.txt',  
            open = 'rt',  
            blocking = TRUE,  
            encoding = 'UTF-8')  
linesDic <- readLines(con)  
close(con)  
linesDic <- unique(linesDic) # por si hay palabras repetidas en el diccionario
```

FUNCIÓN DE REDUCIR LAS LÍNEAS DEL DICCIONARIO

Esta función recibe como parámetros el vector de las líneas del diccionario y una palabra. Y devuelve un diccionario reducido (las palabras que tengan el mismo número de caracteres que la palabra dada). Utilizamos esta función para que cuando apliquemos la función *camper* sobre un ejemplo, el procedimiento vaya más rápido.

```
dicRed <- function(v_dic, word){  
  param <- nchar(word) # número de caracteres de la palabra  
  linesRed <- subset(v_dic, nchar(v_dic) == param)  
  return(linesRed)  
}
```

Creamos 2 funciones, una se encarga encontrar todas las palabras del diccionario relacionadas con otra palabra mediante el intercambio de una sola letra de esa palabra por otras letras (*change_letter*), en cambio la otra, se encarga de encontrar otro grupo de palabras del diccionario que pueden ser creadas a base de permutar una palabra que pasemos(*permutations*).

09/01/2022

Grado en Ciencia de Datos e Inteligencia Artificial

Procesamiento del Lenguaje Natural – Práctica Grupo 4

Juego de cambiar una letra o permutar

FUNCIÓN DE CAMBIAR UNA LETRA DE UNA PALABRA

La función recibe como parámetros de entrada una palabra y un vector (diccionario). Y nos devuelve un vector con todas las palabras del diccionario de la misma longitud que se consigue cambiando una letra a la palabra inicial.

```
change_letter <- function(word, v_dic){  
  library(stringdist)  
  # diccionario reducido (palabras de la misma longitud que word)  
  dic_filtrado <- dicRed(v_dic, word)  
  
  # palabras del diccionario que tienen distancia 1 con la palabra dada.  
  # estas palabras tienen distancia 1 por sustitución (porque el diccionario está reducido).  
  result <- subset(dic_filtrado,  
                    stringdist(dic_filtrado, word, method = 'lv') == 1)  
  
  return(result)  
}
```

FUNCIÓN DE HACER PERMUTACIONES

La función recibe como parámetros de entrada una palabra y un vector (nuestro caso, un diccionario). Nos devuelve un vector con todas las palabras del diccionario con las letras permutadas de la palabra original.

```
permutations <- function(word, v_dic){  
  cadena_word <- unlist(strsplit(word, split = '')) # separamos las letras de la palabra  
  
  if (!require(combinat)) install.packages("combinat")  
  library(combinat) # Usamos la librería "combinat" que contiene una función de hacer permutaciones.  
  # nos devuelve una lista de las permutaciones,  
  # donde cada elemento de la lista es un vector de las letras permutadas.  
  lista_perm <- permn(cadena_word)  
  
  # lista de permutaciones con las letras unidas  
  lista_perm_paste <- lapply(lista_perm, paste, collapse = '')  
  vector_perm <- unlist(lista_perm_paste)  
  
  # de todas las permutaciones, nos quedamos con las que están en el diccionario.  
  result <- subset(vector_perm, vector_perm%in%v_dic&vector_perm!=word)  
  return(result)  
}
```

Juntamos las dos funciones anteriores en una sola para poder operar de forma más sencilla así, ya que pensamos que era mucho lío tener por separado ya que en el programa son las 2 únicas cosas que se pueden hacer. En definitiva, es lo que nos estaba pidiendo el ejercicio por eso lo hemos llamado camper a la función.

FUNCIÓN: CAMBIO DE LETRA + PERMUTACIONES

La función recibe como parámetros de entrada una palabra y un vector (nuestro caso, un diccionario). Devuelve un vector con las palabras del diccionario conseguidas siguiendo el procedimiento de cambiar una letra y permutar.

```
camper <- function(word, v_dic){  
  cambiar_letra <- change_letter(word, v_dic)  
  permutaciones <- permutations(word, v_dic)  
  result <- unique(c(cambiar_letra, permutaciones)) # puede haber palabras repetidas  
  return(result)  
}
```

09/01/2022

Grado en Ciencia de Datos e Inteligencia Artificial

Procesamiento del Lenguaje Natural – Práctica Grupo 4

Juego de cambiar una letra o permutar

Esta es un ejemplo del primer programa, vemos como saca todas las palabras relacionadas con trapo.

Prueba de la función camper:

```
todas_posibilidades <- camper(word = 'trapo', v_dic = linesDic)
todas_posibilidades
```

```
## [1] "grapo" "trabo" "trago" "tramo" "trapa" "trape" "tropo" "trepo" "traps"
## [10] "trajo" "traro" "trato" "trazo" "tropa" "potra" "potar" "optar" "topar"
## [19] "parto" "rpto" "porta"
```

2. Programa camper_interactivo.Rmd

Es un programa que permita jugar con la máquina y sacar palabras ganadoras.

El primer problema de este apartado fue conceptual, ya que al principio pensamos que debíamos desarrollar un programa que permitiera jugar a 2 personas en las que cada uno pone una palabra, pero sin embargo al final dedujimos que teníamos que desarrollar un programa en el que jugaríamos contra la máquina.

Este programa tiene todas las funciones del anterior programa (camper.Rmd), las cuales necesitará la función `camper_iterativo`.

En primer lugar, el jugador introduce una palabra y verificamos la palabra metida. Después le tocaría a la máquina continuar la secuencia. Para ello, se calcula el camper de la palabra del usuario y de todas las palabras calculamos el camper de ellas y elegimos el camper que tenga la menor longitud (la hemos llamado *palabra_ganadora*).

Vas interactuando sucesivamente con la máquina hasta que alguno de los dos consiga un camper con longitud cero (es decir, ya no hay más posibilidades de seguir la secuencia), que quien gana. Según se van escribiendo palabras se van borrando del diccionario para que no se puedan repetir palabras.

09/01/2022

Grado en Ciencia de Datos e Inteligencia Artificial

Procesamiento del Lenguaje Natural – Práctica Grupo 4

Juego de cambiar una letra o permutar

FUNCIÓN: INTERACCIÓN CON EL USUARIO

La función recibe como parámetro de entrada un diccionario y devuelve un vector con las palabras de la secuencia. Lo que hace la función es: el usuario introduce una palabra, el programa le devuelve otra (siguiendo las normas del juego) y así sucesivamente hasta que el programa o el usuario encuentre una palabra ganadora.

```
camper_interactivo <- function(v_dic){
  palabra <- readline('Introduzca la palabra inicial: ')
  if (palabra%in%v_dic == FALSE){ # si la palabra introducida no es válida
    cat('La palabra introducida no se encuentra en el diccionario, pruebe de nuevo.\n')
    camper_interactivo(v_dic)
  }
  v_dic <- dicRed(unique(v_dic), palabra) # reducimos el diccionario
  v_dic <- setdiff(v_dic, palabra)
  opciones <- camper(palabra, v_dic)
  cadena <- c(palabra)
  seguir <- TRUE
  while (seguir == TRUE || length(opciones) > 0){
    # Turno de la máquina
    # la máquina elige la palabra de las opciones que tiene menos longitud en su camper
    min_pal <- length(opciones)
    for (i in 1:length(opciones)){
      # hacemos el camper de todas las opciones
      if (length(camper(opciones[i],v_dic)) < min_pal){
        min_pal <- i
      }
    }
    palabra_ganadora <- opciones[min_pal]
    cat('Turno de la máquina\n')
    cat(paste('Continúe la secuencia: ',palabra_ganadora,'\n'))
    cadena <- c(cadena, palabra_ganadora)
    v_dic <- setdiff(v_dic,palabra_ganadora)
    # todas las palabras que el usuario puede introducir
    opciones_usuario <- camper(palabra_ganadora,v_dic)
    # si no le quedan opciones al usuario: gana la máquina
    if (length(opciones_usuario) == 0){
      cat('¡Ha perdido!Otra vez será.\n')
      break
    }

    # Turno del usuario
    cat('Su turno:')
    palabra_usuario <- readline('Introduzca una palabra: ')
    # si la palabra introducida no sigue las normas del juego
    while (palabra_usuario%in%opciones_usuario == FALSE){
```

09/01/2022

Grado en Ciencia de Datos e Inteligencia Artificial

Procesamiento del Lenguaje Natural – Práctica Grupo 4

Juego de cambiar una letra o permutar

```
if (palabra_usuario%in%cadena){
  cat(';Error! La palabra dada ya se encuentra en la cadena. Intente de nuevo.\n')
  cat(paste('Continúe la secuencia: ',palabra_ganadora,'\n'))
  palabra_usuario <- readline('Introduzca una palabra: ')
}
else{
  cat('La palabra introducida no se encuentra en
      el diccionario o ha realizado más de una cambio de letra
      o permutación simultáneamente, pruebe de nuevo.\n')
  cat(paste('Continúe la secuencia: ',palabra_ganadora,'\n'))
  palabra_usuario <- readline('Introduzca una palabra: ')
}
}
if ((stringdist(palabra_ganadora, palabra_usuario, method = 'lv') == 1)){
  cat(';Correcto! Ha realizado un cambio de letra válido.\n')
}

else{cat(';Correcto! Ha realizado una permutación válida.\n')}
# quitamos del diccionario la palabra introducida por el usuario
v_dic <- setdiff(v_dic,palabra_usuario)

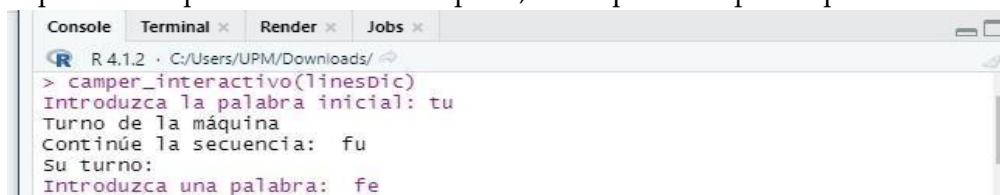
# añadimos la palabra del usuario a la cadena
cadena <- c(cadena, palabra_usuario)
opciones <- camper(palabra_usuario,v_dic)
if (length(opciones) == 0){
  cat(';Ha ganado! Bien jugado.\n')
  seguir <- FALSE
}

}
return(cadena)
}
```

Los principales problemas en esta parte fueron estéticos, lo cual resolvimos utilizando la función `cat()` que muestra mas clara la solución y además utilizamos las `\n` para que no se agolpen las líneas

Otras de las cosas más importantes fue resolver que las palabras ganadoras eran las que tuvieran menor secuencia de palabras en el `camper`, ya que es obvio que así acabarás antes.

Este es un ejemplo de una partida contra la máquina, en la que tu empiezas poniendo una palabra .



```
R 4.1.2 - C:/Users/UPM/Downloads/
> camper_interactivo(linesDic)
Introduzca la palabra inicial: tu
Turno de la máquina
Continúe la secuencia: fu
Su turno:
Introduzca una palabra: fe
```

Aquí vemos como he perdido la partida y vemos toda la secuencia de palabras utilizadas

09/01/2022

Grado en Ciencia de Datos e Inteligencia Artificial

Procesamiento del Lenguaje Natural – Práctica Grupo 4

Juego de cambiar una letra o permutar

```
Console Terminal Render Jobs
R 4.1.2 · C:/Users/UPM/Downloads/
Continúe la secuencia: so
Su turno:
Introduzca una palabra: so
¡Error!, la palabra dada ya se encuentra en la cadena. Intente de nuevo.
Introduzca una palabra: su
¡Correcto!, ha realizado un cambio de letra válido.
Turno de la máquina
Continúe la secuencia: ñu
¡Ha perdido! Otra vez será. [1] "tu" "fu" "fe" "be" "de" "ce" "me" "ge" "le" "h
e" "eh" "oh"
[13] "oi" "oc" "os" "so" "su" "ñu"
> |
```

3. Programa supercamper.Rmd

Es un programa que toma palabras al azar del diccionario y calcule la longitud de la secuencia más larga.

En esta parte también necesitaremos las funciones de camper.R Hemos desarrollado 2 métodos diferentes para este problema.

Método 1:

En resumidas cuentas el primer método se divide en 2 funciones, la primera sec() calcula la secuencia de palabras relacionadas con una palabra dada. Esta función solo calcula secuencias de hasta 1000.

Función supercamper

Como hemos puesto `length(secuencia)<1000` en la función sec, si la cadena máxima fuese de 2345, tendríamos que hacer un bucle para hallar las palabras que faltan. Por lo que esta función supercamper calcula la secuencia más larga de 1000 en 1000 utilizando la función sec si la secuencia superase las 1000 palabras.

```
supercamper<-function(word,v_dic){
  lines<-dicRed(v_dic,word) # Reducimos el diccionario
  secuencia=sec(word,lines,word) # secuencia más 'larga' (máximo 1000 palabras)
  len=length(secuencia)
  result=secuencia
  while (len%1000==0){ #si la secuencia llega a mil palabras, vemos si la secuencia puede
    #hallar secuencia desde la última palabra
    secuencia=sec(secuencia[1000],setdiff(lines,result),secuencia[1000])
    len=length(secuencia) #len de la siguiente secuencia
    result=c(result,secuencia[2:len]) #secuencia[1] es result[length(result)],
    #por lo que empezamos desde 2
  }
  return(result)
}
```


09/01/2022

Grado en Ciencia de Datos e Inteligencia Artificial

Procesamiento del Lenguaje Natural – Práctica Grupo 4

Juego de cambiar una letra o permutar

Método 1:

Función auxiliar

Devuelva la secuencia de palabras más larga a partir de la primera. El método a seguir es el siguiente: Calculamos el camper de la palabra inicial y vemos todas las que nos devuelve. Después, calculamos el camper de todas las posibles soluciones y escogemos la que sea más larga, es decir, la que vaya a tener más posibilidades en el siguiente paso.

La función sec calcula las secuencias hasta las 1000 palabras, porque con 2000 va muy lento

```
sec<-function(word,v_dic,secuencia){
  v_dic=setdiff(v_dic,word) # reducimos el diccionario quitando la palabra de entrada
  res=camper(word,v_dic) # palabras posibles haciendo cambio de letra o permutaciones
  if (length(res)>0 & length(secuencia)<1000){
    long=c() # guardamos el número de posibilidades de cada palabra de res
    for (i in 1:length(res)){
      long=append(long,length(camper(res[i],v_dic)))
    }
    # de todas las palabras de res,
    # cogemos la que tiene mayor longitud de posibilidades
    new_word=res[(which(long==max(long)))[1]] # Ponemos [1],
    #por si hay dos palabras de longitud máxima, así nos selecciona el primero
    secuencia=append(secuencia,new_word)
    sec(new_word,v_dic,secuencia)
  }
  else{
    return(secuencia)}
}
```

La función principal (supercamper), se encarga de ir calculando la secuencia más larga, utiliza la función auxiliar anterior para calcular las secuencias. Va calculando secuencias de 1000 en 1000 (si la secuencia es tiene más de 1000 palabras).

El principal problema de este método es la tardanza. Si la secuencia de palabras es mayor de 1000 palabras, lo solucionamos utilizando bucles para que operara de mil en mil palabras.

Método 2:

Este método es casi idéntico al que teníamos antes, solo que, no utilizamos una función auxiliar para calcular la secuencia, sino que la calculamos directamente en la función supercamper2, es decir, no calcula las secuencias de 1000 en 1000.

Como se puede ver en el código función supercamper2 va cogiendo repetidamente la máxima secuencia hasta llegar a su objetivo.

Hay otra diferencia con respecto a la función sec y, es que vamos a usar otra función camper: lo que cambia en esta función con respecto del camper que definimos al principio es que en esta no aplicamos unique() al resultado final.

Creamos este otro supercamper2 porque cambiando el camper que teníamos al principio, nos daba una secuencia más larga. Sin embargo no sabíamos muy bien por qué, ya que lo único que cambiaba era el unique(). Pero al cargar el diccionario original ya le aplicamos unique(), por lo que pensamos

09/01/2022

Grado en Ciencia de Datos e Inteligencia Artificial

Procesamiento del Lenguaje Natural – Práctica Grupo 4

Juego de cambiar una letra o permutar

que el `unique()` que teníamos al final de la función `camper` original no afectaría. Al final no sabíamos por qué nos daba unos resultados diferentes, por lo que decimos poner los dos métodos.

Método 2

Función `supercamper2`

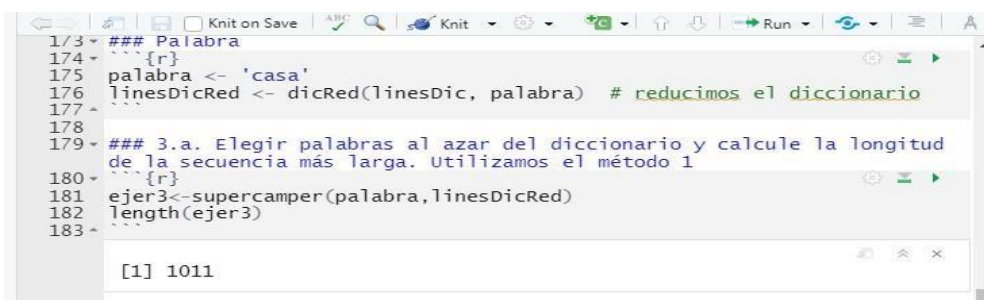
Devuelve la secuencia de palabras más larga a partir de la primera. El método a seguir es el siguiente: calculamos el `camper` de la palabra inicial y vemos todas las que nos devuelve. Después, calculamos el `camper` de todas las posibles soluciones y escogemos la que sea más larga, es decir, la que vaya a tener más posibilidades en el siguiente paso. Esta función hace lo mismo que la función `sec` de antes, solo que calcula la secuencia máxima directamente, es decir, sin restringir la secuencia a las 1000 palabras. Hay otra diferencia con respecto a la función `sec` y, es que vamos a usar otra función `camper`: Lo que cambia en esta función con respecto del `camper` que definimos al principio es que en esta no aplicamos `unique()` al resultado final.

(Creamos este otro `supercamper2` porque cambiando el `camper` que teníamos al principio, nos daba una secuencia más larga. Sin embargo no sabíamos muy bien por qué, ya que lo único que cambiaba era el `unique()`. Pero al cargar el diccionario original ya le aplicamos `unique()`, por lo que pensamos que el `unique()` que teníamos al final de la función `camper` original no afectaría)

```
camper2 <- function(word, v_dic){
  cambiar_letra <- change_letter(word, v_dic)
  permutaciones <- permutations(word, v_dic)
  result <- c(cambiar_letra, permutaciones) # no aplicamos unique()
  return(result)
}

supercamper2 <- function(word, v_dic, secuencia){
  v_dic = setdiff(v_dic, word) # reducimos el diccionario quitando la palabra de entrada
  res = camper2(word, v_dic) # todas las palabras posibles haciendo cambio de letra o permutaciones
  if (length(res)>0){
    longitudes = c() # guardamos el número de posibilidades de cada palabra de res
    for (i in 1:length(res)){
      longitudes = append(longitudes, length(camper2(res[i], v_dic)))
    }
    # de todas las palabras de res, cogemos la que tiene mayor longitud de posibilidades
    new_word = res[(which(longitudes==max(longitudes)))[1]]
    secuencia = append(secuencia, new_word)
    supercamper2(new_word, v_dic, secuencia)
  }
}
```

Este es un ejemplo del método 1 en el que metemos una palabra y nos devuelve la longitud de la secuencia máxima.



```
1/3- ### Palabra
174- {r}
175- palabra <- 'casa'
176- linesDicRed <- dicRed(linesDic, palabra) # reducimos el diccionario
177-
178-
179- ### 3.a. Elegir palabras al azar del diccionario y calcule la longitud
180- de la secuencia más larga. Utilizamos el método 1
181- {r}
182- ejer3<-supercamper(palabra, linesDicRed)
183- length(ejer3)

[1] 1011
```


Este es un ejemplo del método 2 en el que metemos una palabra y nos devuelve la longitud de la secuencia máxima.

09/01/2022

Grado en Ciencia de Datos e Inteligencia Artificial

Procesamiento del Lenguaje Natural – Práctica Grupo 4

Juego de cambiar una letra o permutar



The screenshot shows the RStudio IDE. The script editor on the left contains the following R code:

```
184  
185 ### 3.b. Elegir palabras al azar del diccionario y calcule la longitud  
186 de la secuencia más larga. Utilizamos el método 2  
187 {r}  
188 ejer3b<-supercamper2(palabra,linesDicRed, c(palabra))  
189 length(ejer3b)
```

The console at the bottom shows the output: `[1] 1232`. The Environment pane on the right shows variables: `ejer3b`, `i`, `linesDic`, `linesDicR`, `linesVali`, `n_rand`, and `nmax`.

Vemos como en el segundo método la longitud de la secuencia es más larga , pero no siempre es así , a veces pasa lo contrario con el método 1.

4. (Opcional) Anillos

Secuencia que te lleva a la palabra inicial. Encuentra el anillo más largo.

Utilizamos funciones del programa camper.Rmd y del método 1 del apartado anterior .

Desarrollamos una función auxiliar que nos ayudara a sacar los anillos comenzando por el final para encontrar la palabra inicial.

En función *anillo_aux*, lo que se hace es que se calcula la secuencia más larga con la función *supercamper()* y vamos recorriendo la secuencia desde el final hasta que encontremos una palabra que permutando o cambiando una letra nos lleva a la palabra origen. Esta función saca el índice de la palabra en la secuencia más larga.

En la función *anillo()*, lo que se hace es devolver el anillo más largo utilizando el índice obtenido en la función auxiliar.

Anillo

Para el anillo hemos usado la función *supercamper* que teníamos definida (método 1)

```
anillo_aux<-function(secuencia,word,v_dic){  
  i=length(secuencia) #empezamos desde el final para encontrar el anillo más largo  
  while (i<=length(secuencia)){  
    #si desde esa palabra podemos llegar a la palabra inicial, ya tenemos el anillo  
    if (word%in%camper(secuencia[i],v_dic)){  
      return(i)  
    }  
    else{i=i-1}  
  }  
}  
  
anillo<-function(word,v_dic){  
  lines<-dicRed(v_dic,word)  
  sec=supercamper(word,lines)  
  sol=anillo_aux(sec,word,lines)  
  anillo_largo=append(sec[1:sol],word)  
  cat('El anillo más largo de esta palabra es de',sol,'palabras\n')  
  return(anillo_largo)  
}
```

09/01/2022

Grado en Ciencia de Datos e Inteligencia Artificial

Procesamiento del Lenguaje Natural – Práctica Grupo 4

Juego de cambiar una letra o permutar

Vemos cómo funciona con un ejemplo de un anillo de longitud máxima.

```
linesDicRed <- dicRed(linesDic, 'tu')
anillo_tu<-anillo('tu',linesDicRed)

## El anillo más largo de esta palabra es de 16 palabras
anillo_tu

## [1] "tu" "te" "se" "me" "de" "da" "ea" "ja" "ka" "la" "le" "ce" "fe" "fa" "ta"
## [16] "ti" "tu"
```