Descifrado mediante el algoritmo Metropolis

Pablo de Arriba Mendizábal

2023-12-03

Esta práctica consiste en el descifrado de un mensaje que ha sido cifrado previamente mediante sustitución. Para ello, se utilizará el algoritmo Metropolis, el cual nos permite generar réplicas de dsitribuciones de probabilidad en casos como este, donde otros métodos de simulación son difíciles de aplicar.

En la resolución de este problema, se han utilizado una serie de funciones facilitadas por el profesor, las cuales se muestran a continuación.

```
#### funciones auxiliares

clave<-c(toupper(letters)[1:14],"Ñ",toupper(letters)[15:26]," ")

letra_a_numero<-function(a) match(a,clave)
numero_a_letra<-function(n) clave[n]

## funcion que transforma mensaje de texto
## a secuencia de numeros

mensaje_a_numero<-function(mensaje){
  mensajenumerado<-NULL
  for(i in 1:nchar(mensaje)){
    mensajenumerado<-c(mensajenumerado,letra_a_numero(substring(mensaje,i,i)))
  }
  return(mensajenumerado)
}

## funcion que transforma secuencia de numeros
## a mensaje de texto

mensaje_a_letra<-function(mensaje){
  mensajetexto<-NULL
  for(i in 1:length(mensaje)){
    mensajetexto<-paste(mensajetexto,numero_a_letra(mensaje[i]),sep='')
  }
  return(mensajetexto)
}

## funcion que devuelve los bigramas de una palabra o grupo de palabras
bigramas_palabra <- function(palabra) {
  aaa<-unlist(strsplit(palabra,''))
  nnn<-length(aaa)
  if (nnn > 1) {
  bigramas<-paste0(aaa[1:(nnn-1)],aaa[2:nnn])
```

```
  }
  return(bigramas)
}

## funcion que devuelve los trigramas de una palabra o grupo de palabras
trigramas_palabra <- function(palabra) {
  aaa<-unlist(strsplit(palabra,''))
  nnn<-length(aaa)
  if (nnn > 2) {
  trigramas<-paste0(aaa[1:(nnn-2)],aaa[2:(nnn-1)],aaa[3:nnn])
  }
  return(trigramas)
}



## funcion que aplica el descifrado de sustitucion
## dado por ordenacion

decodificador<-function(mensaje,ordenacion){
  fun.auxiliar<-function(i) ordenacion[i]
  mensaje_decodificado<-NULL
  for (j in 1:length(mensaje)) mensaje_decodificado<-c(mensaje_decodificado,fun.auxiliar(mensaje[j]))
  return(mensaje_decodificado)
}
```

Además, contamos con 4 archivos necesarios para la resolución de la práctica:

1. long_palabras_espanol.txt
2. bigramas_espanol.txt
3. trigramas_espanol.txt
4. mensaje_cifrado.txt

El primero de ellos contiene las frecuencias de las diferentes longitudes de palabras, de 1 hasta 23 caracteres. En el segundo las frecuencias de gran parte de los bigramas que hay en español, y en el tercero lo mismo pero en este caso con los trigramas.

El cuarto únicamente contiene el mensaje cifrado que debemos resolver.

En primer lugar se deben leer todos los ficheros:

```
mensaje <- readLines("mensaje_cifrado.txt")

frec_longitud <- read.table("long_palabras_espanol.txt")
na_longitud <- data.frame(NA, 10^-8)
colnames(na_longitud) <- c("longitud", "frecuencia")
frec_longitud <- rbind(frec_longitud, na_longitud)

frec_bigramas <- na.omit(read.table("bigramas_espanol.txt", encoding = "UTF-8"))
na_bigrama <- data.frame(NA, 10^-8)
colnames(na_bigrama) <- c("bigrama", "frecuencia")
frec_bigramas <- rbind(frec_bigramas, na_bigrama)

frec_trigramas <- na.omit(read.table("trigramas_espanol.txt", encoding = "UTF-8"))
```

```
na_trigrama <- data.frame(NA, 10^-8)
colnames(na_trigrama) <- c("V1", "V2")
frec_trigramas <- rbind(frec_trigramas, na_trigrama)
```

A cada fichero de frecuencias le he añadido una entrada con el valor 10^{-8}. De esta forma, cuando una palabra no case con ninguna de las entradas de alguno de los ficheros, se le asignará una frecuencia de 10^{-8}.

Tras la lectura de los datos, debemos empezar a tratarlos.

Para ello, paso el mensaje de letra a numérico y a continuación cojo una muestra aleatoria del 1 al 28, que será nuestra ordenación inicial de la sustitución. Después, aplicamos la ordenación al mensaje numérico, lo volvemos a pasar a letra y separamos todas las palabras del mensaje.

```
mensaje_numerico <- mensaje_a_numero(mensaje)

mejor_ordenacion <- sample(1:28)  # Inicializar con una permutación aleatoria
mensaje_decodificado <- decodificador(mensaje_numerico, mejor_ordenacion)
mensaje_texto <- mensaje_a_letra(mensaje_decodificado)
palabras <- unlist(strsplit(mensaje_texto, split = " "))
```

Tal y como se indica en el enunciado de la práctica, en función de la frecuencia de la longitud, y los bigramas y trigramas de cada palabra hay que calcular un score. Para ello he creado 3 funciones, que calculan los scores de las longitudes, los bigramas y los trigramas, y una cuarta que calcula el score total.

Se utiliza la función 'match' para encontrar en cada fichero las coincidencias de longitud, bigramas y trigramas y así extraer sus frecuencias. Si algún elemento no tiene coincidencias, se le asigna el índice de la entrada creada por mí, que tiene la frecuencia 10^{-8}, para de esta forma "penalizar" los casos poco frecuentes.

En el cálculo de los scores, al ser las frecuencias muy grandes, se obtienen valores enormes que no pueden ser manejados de manera sencilla. Por ello, en lugar de multiplicar las frecuencias, tomaremos logaritmos en base 10 y los sumaremos. De esta forma obtendremos valores fáciles de manejar.

```
calc_score1 <- function(palabras) {
  ind <- match(nchar(palabras), frec_longitud$longitud)
  ind[is.na(ind)] <- 24
  valor1 <- sum(log10(frec_longitud[ind,2]))

  return(valor1)
}


calc_score2 <- function(palabras) {
  bigramas <- bigramas_palabra(palabras)
  ind <- match(bigramas, frec_bigramas$bigrama)
  ind[is.na(ind)] <- 708
  valor2 <- sum(log10(frec_bigramas[ind,2]))

  return(valor2)
}



calc_score3 <- function(palabras) {
  trigramas <- trigramas_palabra(palabras)
  ind <- match(trigramas, frec_trigramas$V1)
```

```
  ind[is.na(ind)] <- 14329
  valor3 <- sum(log10(frec_trigramas[ind,2]))

  return(valor3)
}



calc_score <- function(palabras) {
  score1 <- calc_score1(palabras)
  score2 <- calc_score2(palabras)
  score3 <- calc_score3(palabras)
  score <- 3*(score1)+(score2)+(1/2)*(score3)
  return(score)
}
```

Ahora calculamos el score de la ordenación inicial:

```
mejor_score <- calc_score(palabras)
```

Una vez calculado el primer score, comienza la parte más importante del programa.

Vamos a crear un bucle, con un número de iteraciones arbitrario, el cual hará lo siguiente:

En primer lugar, se realiza un intercambio entre dos elementos de la ordenación.

Con esta nueva ordenación, se vuelve a decodificar el mensaje y se calcula el nuevo score.

En este momento hay que decidir, a partir del score calculado, si la nueva ordenación es mejor o peor que la anterior. Para ello he utilizado una condición por la cual si 10 elevado a la diferencia del score antiguo y el nuevo es mayor que un número aletaorio entre 0 y 1, entonces se acepta la nueva ordenación. Debemos elevar 10 a esa diferencia debido a que anteriormente se han transformado los datos con log10, por lo que hay que deshacer la transformación.

Cuando se acepta una ordenación, en la siguiente iteración del bucle se comparará esta ordenación con la nueva.

```
for (iteracion in 1:10000) {
  # Generar una permutaciÃ³n propuesta mediante una trasposiciÃ³n aleatoria
  nueva_ordenacion <- mejor_ordenacion
  indices <- sample(1:28, 2)
  nueva_ordenacion[indices] <- nueva_ordenacion[c(indices[2], indices[1])]

  # Calcular el score de la nueva permutaciÃ³n
  mensaje_decodificado <- decodificador(mensaje_numerico, nueva_ordenacion)
  mensaje_texto <- mensaje_a_letra(mensaje_decodificado)
  palabras <- unlist(strsplit(mensaje_texto, split = " ")); palabras

  nuevo_score <- calc_score(palabras)



  # Decidir si aceptar la nueva permutacion
  if (runif(1) < 10^(nuevo_score - mejor_score)){
    mejor_ordenacion <- nueva_ordenacion
```

```
    mejor_score <- nuevo_score
  }

}
```

Una vez realizadas todas las iteraciones del bucle, obtendremos el mensaje descifrado.

En mi caso, tras realizar numerosas pruebas, con 10000 iteraciones suele descifrar el mensaje correctamente, aunque dependiendo de la ordenación inicial puede tardar menos, más o incluso no llegar a descifrar el mensaje correctamente. En algunos casos con 3000 iteraciones ha sido suficiente para obtener el mensaje completo.

A continuación, muestro el mensaje descifrado tras la ejecución del programa:

TODA PERSONA TIENE DERECHO A LA LIBERTAD DE PENSAMIENTO DE CONCIENCIA Y DE RELIGION ESTE DERECHO INCLUYE LA LIBERTAD DE CAMBIAR DE RELIGION O DE CREENCIA ASI COMO LA LIBERTAD DE MANIFESTAR SU RELIGION O SU CREENCIA INDIVIDUAL Y COLECTIVAMENTE TANTO EN PUBLICO COMO EN PRIVADO POR LA ENSEJANZA LA PRACTICA EL CULTO Y LA OBSERVANCIA TODO INDIVIDUO TIENE DERECHO A LA LIBERTAD DE OPINION Y DE EXPRESION ESTE DERECHO INCLUYE EL DE NO SER MOLESTADO A CAUSA DE SUS OPINIONES EL DE INVESTIGAR Y RECIBIR INFORMACIONES Y OPINIONES Y EL DE DIFUNDIRLAS SIN LIMITACION DE FRONTERAS POR CUALQUIER MEDIO DE EXPRESION