Cálculo de la mediana

Luis Enrique López Nerio
Reynolds de Jesús HernÃąndez FernÃąndez
Daniel Honorato
Daniela Carrizales
Yadira Marroquin
Universidad Autonoma de Nuevo Leon

23 de Noviembre del 2017

Abstract

Dentro de la estadística es normal encontrarnos con mediadas de tendencia central, las medidas de tendencia central nos pueden dar información útil acerca de un conjunto de datos, es por esta razón que son importantes dentro de la estadística, algunas de las medidas de tendencia central más usadas son la media, mediana y moda.

En este reporte nos propondremos describir brevemente lo que es la mediana, su importancia y diferencia con otras medidas de tendencia central y lo más importante, describir algunos algoritmos para encontrar la mediana además de proponer un algoritmo que no puede dar este valor de una manera muy eficiente.

I. Introduccíon

Las medidas de tendencia central dentro de la estadística son básicamente características o información de un conjunto de dados, de manera más específica las medidas de tendencia central nos dan pistas acerca de la posición central de nuestra serie de datos. Existen diferentes medidas de tendencia central y cada una puede tener ciertas diferencias con las otras, algunas medidas de tendencia central que existen son las siguientes:

- Media
- Mediana
- Moda

La media es probablemente es una de las medidas más usadas en la estadística, probablemente hayas escuchado hablar de ella ya, tal vez con otro nombre, promedio, media aritmética, esperanza matemática, básicamente nos da el valor de la suma de una serie de datos dividido entre el número de datos, uele representarse de la siguiente manera $\bar{x}$.
Sean $x_1, x_2, x_3, ..., x_n$ una serie de datos, entonces la media es:

$$\bar{x} = \sum_{i=1}^{n} x_i$$

En estadistica se denomina mediana a estadística al valor que se encuentra en el lugar central de todos los datos de un estudio cuando éstos están ordenados de menor a mayor. El símbolo de la mediana se representa por M_e. La mediana es por tanto el número central de un grupo de

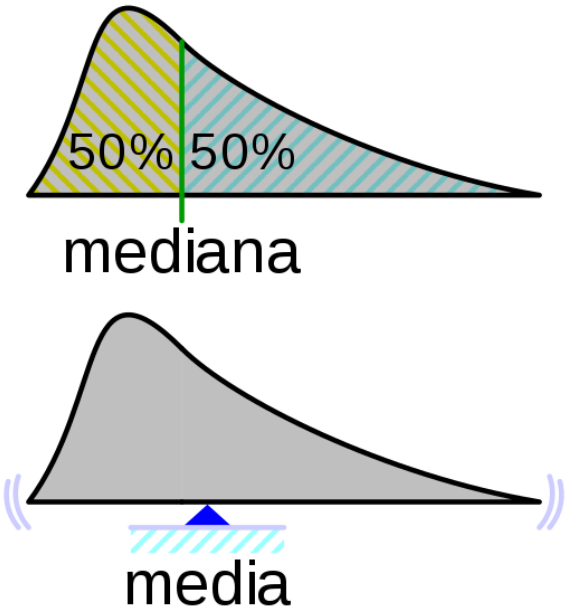

Figure 1: *Visualización de las medidas de tendencia central.*

números ordenados por su tamaño.
Para hallar la mediana en estadística, se ordenan los números de una muestra según su valor y se determina el que queda en el medio. Si la cantidad de términos es impar, la mediana es el valor central. Si la cantidad de términos es par, suma los dos términos del medio y divide entre 2.
Es el número medio en una serie ordenada de números, es decir, que la mitad son mayores que él y la mitad menores, en caso de que sea un grupo par de números, la mediana es el promedio entre los 2 valores centrales, en la figura 1 podemos ver una forma de visualizar la diferencia entre la mediana y la media.
Sean $x_1, x_2, x_3, ..., x_n$ una serie de datos ordenados de forma creciente, entonces la mediana es:

n impar

$$M_e = x_{\frac{n+1}{2}}$$

n par

$$M_e = (x_{\frac{n}{2}} + x_{\frac{n}{2}+1})/2$$

II. Diferencia entre mediana y moda

Aunque las dos son medidas de tendencia central es importante notar que hay diferencias substanciales entre la mediana y la media, algunas veces para diferentes conjuntos de datos el valor de la mediana y la moda puede que sea el mismo, sin embargo es común que este no sea el caso, para enfatizar la diferencia entre la mediana y la media introduciremos un ejemplo.

Supongamos que unos ejecutivos de una canal de televisión quieren lanzar un nuevo programa, para decidir qué programa van a lanzar se hacen la siguiente pregunta: ¿Cuál es el programa que

atraerá la mayor audiencia?, para responder esta pregunta tienen que saber cuál es la edad de su audiencia de manera general.

Para simplificar el ejemplo supongamos que su audiencia son 5 personas, y estas personas tienen las edades 8, 6, 6, 10, 85 años respectivamente, ¿Cuál es la medida de tendencia central que mejor nos ayuda a tomar la decisión de los ejecutivos?

Media

$$\bar{x} = \frac{8+6+6+10+85}{5} = \frac{115}{5} = 23$$

Mediana

$$M_e = 8$$

Las media nos dice que la edad promedio de nuestra audiencia es de 23 años, sim embargo si observamos a nuestra audiencia podemos notar que esta es una población con un sesgo muy grande debido a una sola persona, si elegimos un programa basado en la media de 23 años probablemente no cubramos a nadie en nuestra audiencia.

Sin embargo la mediana nos da una buena medida para tomar una decisión, elegir un programa basado en la mediana nos permitiría cubrir a la mayor audiencia posible, aunque este es un ejemplo simplificado es común encontrarse situaciones en los que la media no es una medida muy buena para tomar decisiones debido a poblaciones sesgadas por algunos datos muy grandes o muy pequeños.

III. Mediana

Ahora que observamos que la mediana es una medida de tendencia central que puede darnos información útil acerca de nuestra población, el siguiente paso es encontrar un algoritmo que nos ayude a encontrarla, si regresamos a nuestra definición de mediana nos podemos dar cuenta de que para calcularla es necesario ordenar los datos, por lo tanto regresamos a un problema recurrente dentro de las ciencias computacionales, el ordenamiento de datos.

Cuando estudiamos los algoritmos de ordenación describimos brevemente su idea general, complejidad computacional, ventajas y desventajas, unos eran más rápidos que otros, dentro de los que tenían un mejor desempeño estudiamos el algoritmo de Quicksort.

Este algoritmo fue creado por Tony Hoare en el año 1959, este algoritmo entra de los que son clasificados de “divide y conquista”, al poder dividir en problemas más pequeños del mismo tipo, por lo tanto este algoritmo usa recursión para ordenar el arreglo.

La idea básica del algoritmo consiste en elegir un elemento del arreglo, a este elemento elegido lo llamaremos elemento pivote, lo siguiente será reordenar el arreglo de manera que todos los elementos más pequeños del pivote estén del lado izquierdo de este y los mismo para los elementos mayores.

Una vez que termina de ordenar el arreglo en torno al pivote, el pivote se encontrara en su posición correspondiente, entonces se realizar el mismo proceso para la parte derecha e izquierda del arreglo que están sin ordenar.

```
1
2  """
       ********************************************************************************************
3  ###############################################################################################
4                                          Quicksort
5  Funci\'on para ordenar un arreglo de longitud arbitraria con el algoritmo quicsort , toma
       como
6  parametro el arreglo, el indice menor y mayor del arreglo
7  ###############################################################################################
8  ********************************************************************************************
       """
9  def quicksort(arreglo,low,high ):
10     if low < high:
11         m = particion(arreglo, low, high)
12         quicksort(arreglo,low,m-1)
13         quicksort(arreglo, m+1 , high)
14
15
16 """
       ********************************************************************************************
17 ###############################################################################################
18                                         Partici\'on
19 Funci\'on para realizar una subrutina del algoritmo quicksort
20 la funci\'on regresa el indice del valor pivote en su orden respectivo en el arreglo
21 ###############################################################################################
22 ********************************************************************************************
       """
23 def particion(arreglo, low, high):
24     pivote = arreglo[high]
25     wall = low-1
26     for j in range(low, high):
27         if arreglo[j]<pivote:
28             wall = wall + 1
29             swap(arreglo,wall,j)
30     if(arreglo[high]< arreglo[wall+1]):
31         swap(arreglo,wall+1,high)
32     return wall+1
33
34 """
       ********************************************************************************************
35 ###############################################################################################
36                                             swap
37 Funci\'on que realiza un intercambio de posici\'on de valores dentro de un arreglo, toma
       como parametros
38 el arreglo  y los indices de los valores a cambiar
39 ###############################################################################################
40 ********************************************************************************************
       """
41 def swap(arreglo, a, b):
42     aux = arreglo[a]
43     arreglo[a]= arreglo[b]
44     arreglo[b]= aux
```

```
45
46
47
48
49 def mediana1(arr):
50         tiempo1=time.time()
51         quicksort(arr,0,len(arr)-1)
52         if (((len(arr))%2)!=0):
53             med=          arr[int(len(arr)/2)]
54         else:
55             med =     (arr[int(len(arr)/2)-1]+arr[int(len(arr)/2)] ) /2
56         tiempo2 = time.time()
57         return (med,tiempo2-tiempo1)
```

Codigo 1.-Algoritmo Quicksort

Matemáticamente se puede demostrar que la complejidad del algoritmo quicksort es de $\mathcal{O}(nlog(n))$, aunque para los peores casos puede llegar a tomar $\mathcal{O}(n^2)$, hay esquemas en los que el valor pivote se elige tomando diferentes valores al azar y de esos valores tomar la mediana. Para arreglos que están hasta cierto punto ordenados o que contienen demasiados valores repetidos su desempeño disminuye.

La complejidad del algoritmo bubble sort es de $\mathcal{O}(n^2)$ ya que en el peor de los casos, en el cual el arreglo está completamente invertido,se tendrá que recorrer el arreglo en el while (n-1) veces y tiene que realizar una iteración for dentro del while n veces. El algoritmo bubble sort por lo tanto no tiene un muy buen desempeño algunas ventajas son que, su interpretación es sencilla y no requiere espacio en memoria adicional, lo cual es de gran ayuda cuando el espacio en memoria es reducido.

Utilizaremos este algoritmo para ordenar una serie de datos y poder obtener la mediana.

IV. Algoritmo de Mediana más eficiente

Como encontrar la mediana se reduce a ordenar una serie de datos nos dimos cuenta de que teníamos que utilizar un algoritmo de ordenación, sin embargo este proceso se puede reducir y hacer más eficiente, tenemos que recordar que la mediana se puede calcular si ordenamos una serie de datos y tomamos los datos en el centro, pero la mediana también se podía pensar como el dato que dividía a nuestro con junto de datos de manera que el número de datos antes y después de la mediana fuera el mismo, ósea que antes y después de la mediana están la mitad de los datos.

Podemos utilizar este hecho a nuestro favor, si recordamos el algoritmo Quicksort tiene como paso particionar nuestro datos, haremos uso de este proceso para generar un algoritmo para la mediana que sea más eficiente.

La idea general es particionar nuestro datos con la primera parte de nuestro algoritmo de Quicksort tomando un valor arbitrario, después hacer esto todos los valores que estén a la izquierda de nuestro valor pivote serán menor a él y todos los datos a la derecha de nuestro pivote serán mayor a él.

El siguiente paso será preguntarnos en que índice se encuentra, dependiendo de si nuestro valor se encuentra antes de la mitad de los datos, después de la mitad de los datos o en la mitad exactamente volveremos a particionar nuestro arreglo pero ahora podemos ignorar una parte del

arreglo y concentrarnos en la otra, de esta forma no es necesario ordenar todo el arreglo sino que solo debemos partir nuestro arreglo hasta llegar a un pivote que se encuentre justo en la mitad.

A continuación se muestra el código para realizar este proceso:

```
1
2  """
       ********************************************************************************************
3  ##############################################################################################
4                                                  Mediana
5  Funci\'on que encuentra la mediana de un arreglo
6  ##############################################################################################
7  ********************************************************************************************
       """
8
9  def split(arreglo,n, pivote,iz, de ):
10     while(iz<de):
11         while ( arreglo[iz]<pivote):
12             iz+=1
13         while (arreglo[de]>pivote):
14             de-=1
15         aux = arreglo[iz]
16         arreglo[iz] = arreglo[de]
17         arreglo[de] = aux
18     return (arreglo,pivote,iz, len(arreglo)//2)
19
20 def mediana2(arreglo ):
21     tiempo1=time.time()
22     k=len(arreglo)//2
23     I=0
24     D = len(arreglo) -1
25     lon = len(arreglo)
26     if(len(arreglo)%2!=0):##len(arreglo)%2!=0
27         while(True):
28             pivote = arreglo[(I+D)//2]
29             s=split(arreglo,lon,pivote,I,D)
30             if(s[2]==k):
31                 tiempo2=time.time()
32                 return (s[1], tiempo2-tiempo1)
33             elif (s[2]<k):
34                 I = s[2]+1
35             else:
36                 D = s[2]-1
37     else:
38         mediana=0
39         while(True):
40             pivote = arreglo[(I+D)//2]
41             s=split(arreglo,lon,pivote,I,D)
42             if(s[2]==k):
43                 #print(arreglo)
44                 mediana=mediana+s[1]
45                 arreglo.pop()
46                 #print(arreglo)
47                 break
48             elif (s[2]<k):
49                 I = s[2]+1
50             else:
```

```
            D = s[2]-1
        k=len(arreglo)//2
        I=0
        D = len(arreglo) -1
        lon = len(arreglo)
        while(True):
            pivote = arreglo[(I+D)//2]
            s=split(arreglo,lon,pivote,I,D)
            if(s[2]==k):
                mediana=mediana+s[1]
                tiempo2= time.time()
                return (mediana/2, tiempo2-tiempo1)
                break
            elif (s[2]<k):
                I = s[2]+1
            else:
                D = s[2]-1
```

Codigo 2.-Algoritmo para mediana eficiente

V. Experimento

Para comparar nuestros algoritmos que encuentran la mediana se realizara el siguiente experimento, se calcula la mediana de una serie de datos aleatorios, comenzando con 1 dato e incrementando el número de datos aleatorios en 10,000 hasta llegar a 100,001 datos aleatorios, este proceso se realizara 10 veces, cada vez que se calcule la mediana se calculara el tiempo en segundo que tomo encontrar el valor para cada uno de los 2 algoritmos.

```

longitud=[]
tiempo1=[]
tiempo2=[]
for i in range(1, 100005,10000):
    for m in range (20):
        longitud.append(i)
        arr1 = ran_num(i)
        arr2= copy.deepcopy(arr1)
        med1=mediana1(arr1)
        med2=mediana2(arr2)
        tiempo1.append(med1[1])
        tiempo2.append(med2[1])
```

Codigo 3.-Algoritmo para experimento

Una vez realizado el experimento los datos fueron ingresados a una base de datos y con codigo en R se realizaron las siguientes graficas:

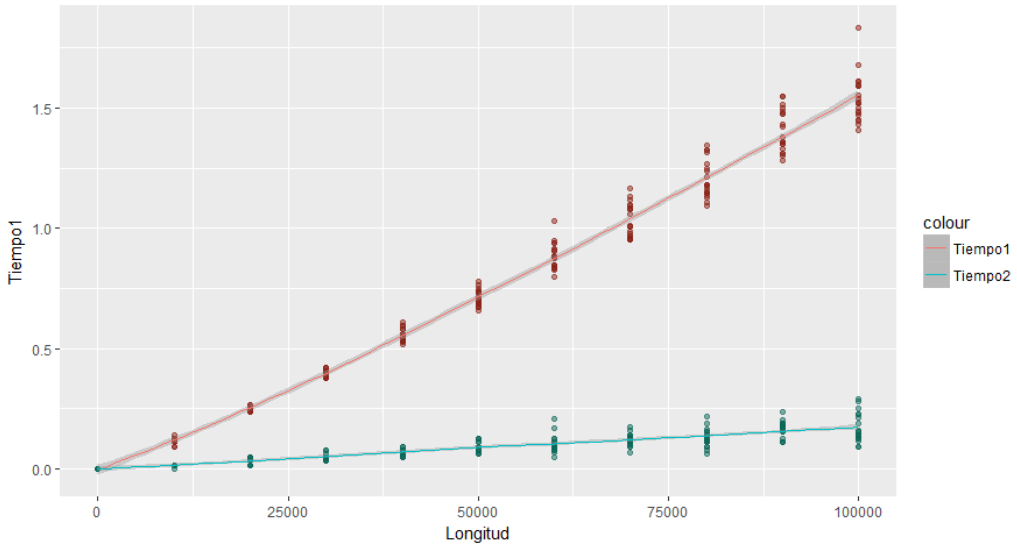

Figure 2: *Scatterplot de el tiempo en el eje y, longitud de arreglo en el eje x*

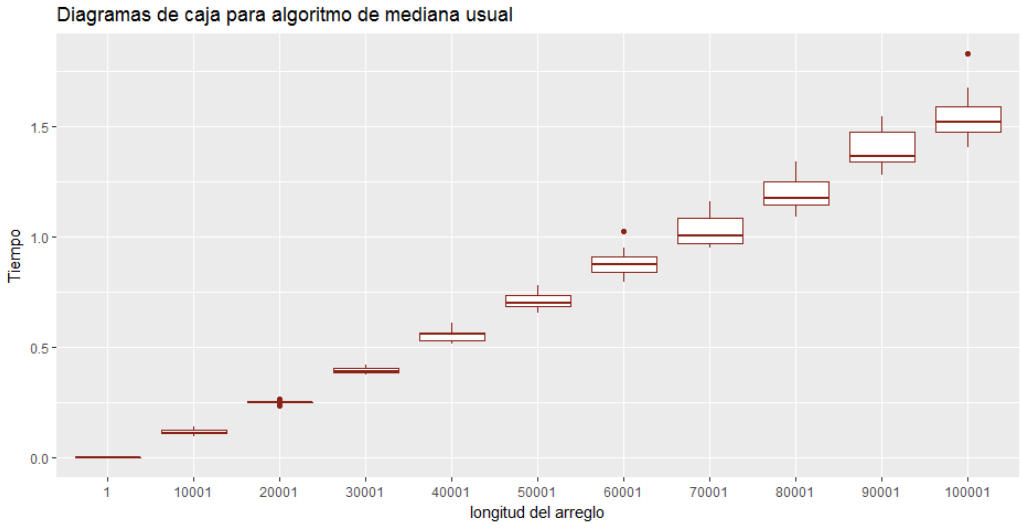

Figure 3: *Diagramas de caja para el calculo de la mediana primero ordenando los datos*

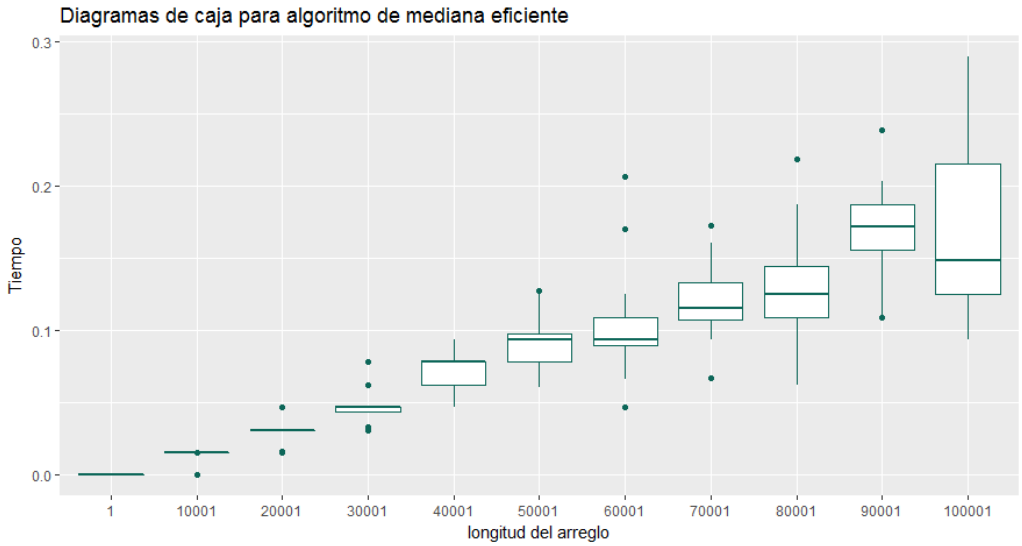

Figure 4: *Diagramas de caja para el calculo de la mediana de manera eficiente*

VI. Conclusión

Los resultados del experimento muestran que el algoritmo donde ordenamos primero el arreglo con el algoritmos Quicksort para calcular la mediana toman significativamente mas tiempo que utilizando el algoritmo donde no se ordena completamente el arreglo, esto era de esperarse ya que el segundo algoritmo reduce el problema de manera sustancial en cada paso. Como futuro problema queda encontrar estadisticos de orden diferentes a la mediana.