Universidad
Internacional
Menéndez Pelayo

Máster Universitario en Investigación en Inteligencia Artificial

Curso 2020-2021

Datos temporales y complejos

Evaluación Data Streams: Metodologías

Día/Mes/Año

Laura Rodríguez Navas
DNI: 43630508-Z
e-mail: rodrigueznavas@posgrado.uimp.es

Índice

1. Ejercicio 1

Sea un fichero de texto que contiene una secuencia desordenada de valores enteros del 1 al N, pero al que le falta uno de los valores. Escriba un programa en cualquier lenguaje de programación tal que lea un fichero como el anterior y dé en el menor tiempo posible y con el menor uso de memoria respuesta a cuál es el número que falta en la secuencia. Pruébelo para N suficientemente grande. Haga una tabla con el tiempo de computación necesario y la memoria usada para al menos 10 ejecuciones con distintos valores de N.

```
import os
import psutil

from time import time

from code.exercise_one.generate_data import createInputFile


def get_process_memory():
    # return the memory usage in MB
    process = psutil.Process(os.getpid())
    return process.memory_info().rss / float(2 ** 20)


def findMissingNumber(ds):
    # data stream length
    n = len(ds)

    # sum of the first N natural numbers, that is, the sum of the natural numbers from 1 to
        N
    sum_total = (n + 1) * (n + 2) // 2

    # sum of all the elements of the data stream specified by ds
    sum_de_ds = sum(ds)

    # subtract the first N natural numbers from the sum of all the elements of the data
        stream,
    # the result of the subtraction will be the value of the missing element
    return sum_total - sum_de_ds


if __name__ == '__main__':
    start_time = time()
    start_mem = get_process_memory()
    N = 100000
    x = 99

    # create sequence from 1 to N without x
    createInputFile(N, x)

    # find missing number
    input_ds = list(map(int, open('data.txt', 'r').read().splitlines()))
    print("Missing number: {}".format(findMissingNumber(input_ds)))

    end_time = time() - start_time
    end_mem = get_process_memory()
    print("Memory used in MB: {} \nExecution time in seconds: {:.10f}".format(end_mem -
        start_mem, end_time))
```

Algoritmo 1: Búsqueda del número que falta en la secuencia.

A continuación, podemos ver la tabla con el tiempo de computación (tCompu) necesario y la memoria usada (uMem) de 10 ejecuciones con distintos valores de N. El tiempo de computación se obtiene en segundos y la memoria usada en MB.

id	N	tCompu	uMem
1	100	0.0010001659	0.0234375
2	1000	0.0030002594	0.0546875
3	10000	0.0190010071	0.74609375
4	50000	0.1000056267	1.46875
5	100000	0.2060117722	3.11328125
6	500000	1.1400651932	9.69921875
7	1000000	2.2931311131	19.5625
8	5000000	12.1846969128	97.375
9	10000000	24.5054016113	193.1328125
10	15000000	37.9671714306	289.20703125

Nota: Genere los datos del fichero de entrada con otro programa que lo escriba, esto es dado dos enteros x y N (x entre 1 y N) construya un programa que escriba en un fichero los números del 1 al N excepto x de manera desordenada.

```
import random


def createInputFile(N, x):
    # check if N and x are integers
    if isinstance(N, int) and isinstance(x, int):
        # check if 1 <= x <= N
        if 1 <= x <= N:
            try:
                # generate permutations from 1 to N
                permutations = random.sample(range(1, N + 1), N)

                # remove x from permutations
                permutations.remove(x)

                # write permutations to TXT file
                with open('data.txt', 'w') as outfile:
                    outfile.write("\n".join(str(item) for item in permutations))

                outfile.close()

            except Exception as e:
                raise Exception("Cannot generate the input file, {}".format(e))
    else:
        raise Exception("N and x must be integer values")
```

Algoritmo 2: Generador de los datos del fichero de entrada.

2. Ejercicio 2

Sea una secuencia desordenada de N números (N suficientemente grande). Queremos encontrar mediante muestreo un número en ella que sea mayor que el tercer cuartil, esto es, que esté en el 25 % superior de los valores de la secuencia. Para ello una posibilidad es escoger aleatoriamente k valores y quedarnos con el mayor

de esos k. ¿Cuál es el menor k posible para que el error sea menor que 0.3? Es decir, que en media de cada 10 veces que repitamos el experimento nos equivoquemos en menos de 3. ¿Y si queremos que la tasa de error baje al 0.1, cuánto debe valer k?

El menor k posible para que el error sea menor que 0.3:

$$k = \log_2(1/0.3) \approx 2$$

El menor k posible para que el error sea menor que 0.1:

$$k = \log_2(1/0.1) \approx 4$$

Haga un programa similar al mostrado en el vídeo que demuestre experimentalmente los valores de k calculados de forma teórica.

```
import random


def main(k):
    acierto = 0
    num_valores = 20
    mitad = num_valores / 2
    num_intentos = 1000000
    ls = list()

    for i in range(0, num_valores):
        ls.append(i)

    for j in range(0, num_intentos):
        random.shuffle(ls)
        if k == 2:
            if ls[0] >= mitad or ls[1] >= mitad:
                acierto += 1
        elif k == 4:
            if ls[0] >= mitad or ls[1] >= mitad or ls[2] >= mitad or ls[4] >= mitad:
                acierto += 1

        print("intentos: {}, aciertos: {}".format(j, acierto))

    print("probabilidad de error {}".format((num_intentos - 1. * acierto) / num_intentos))


if __name__ == '__main__':
    main(2)  # k = 2
    main(4)  # k = 4
```

Algoritmo 3: Demostración experimental de los valores de k calculados de manera teórica.

Con k = 2, el resultado obtenido de la ejecución del algoritmo anterior es 0.236167, y de esta manera se demuestra que la probabilidad del error es menor a 0.3.

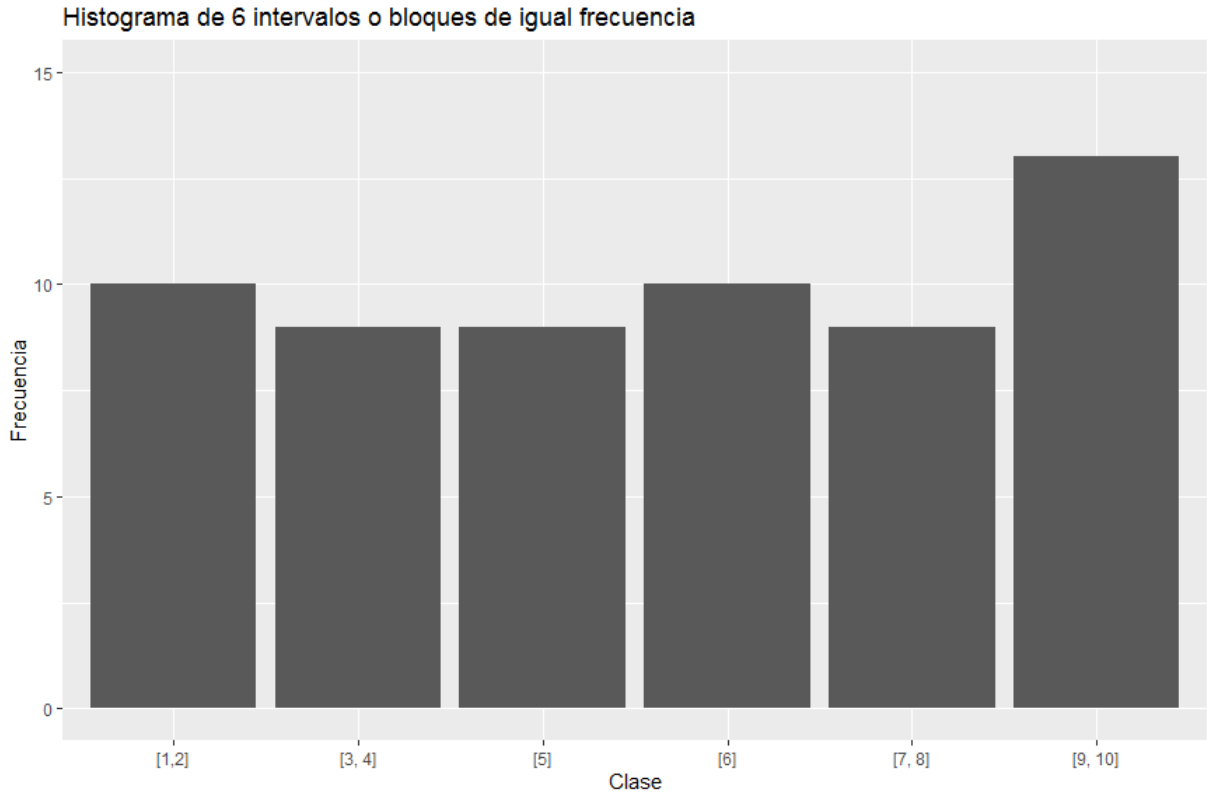
Con k = 4, el resultado obtenido de la ejecución del algoritmo anterior es 0.043217 y se demuestra que la probabilidad del error es menor a 0.1.

3. Ejercicio 3

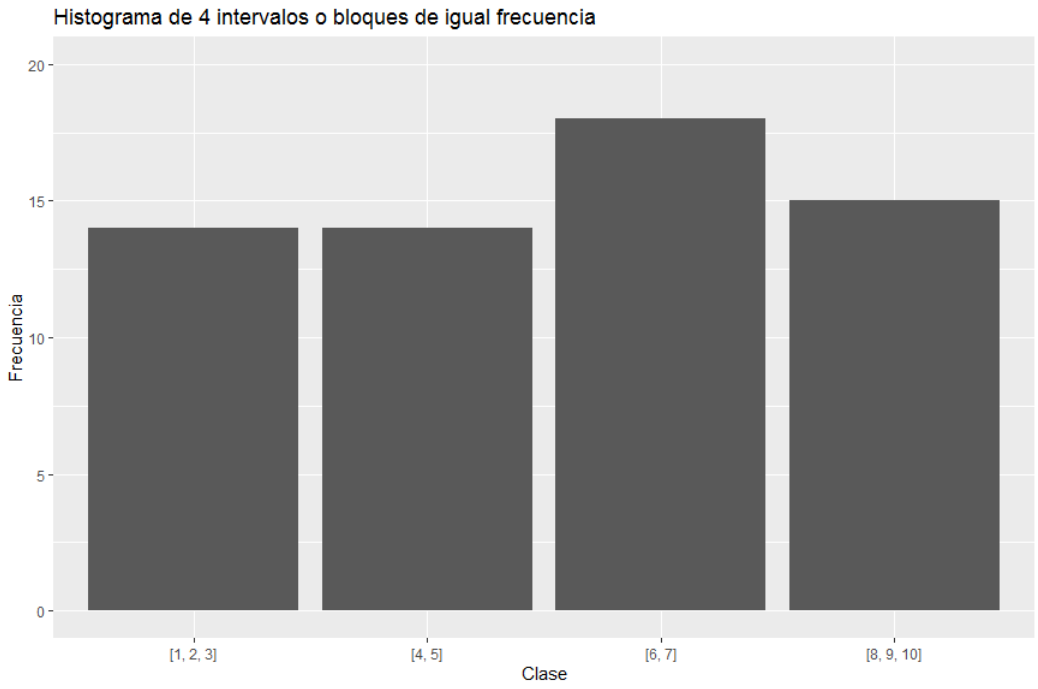
Supongamos tenemos una secuencia de valores con la frecuencia que se indica:

Valor: 1 Frecuencia: 7
Valor: 2 Frecuencia: 3
Valor: 3 Frecuencia: 4
Valor: 4 Frecuencia: 5
Valor: 5 Frecuencia: 9
Valor: 6 Frecuencia: 10
Valor: 7 Frecuencia: 8
Valor: 8 Frecuencia: 1
Valor: 9 Frecuencia: 9
Valor: 10 Frecuencia: 4

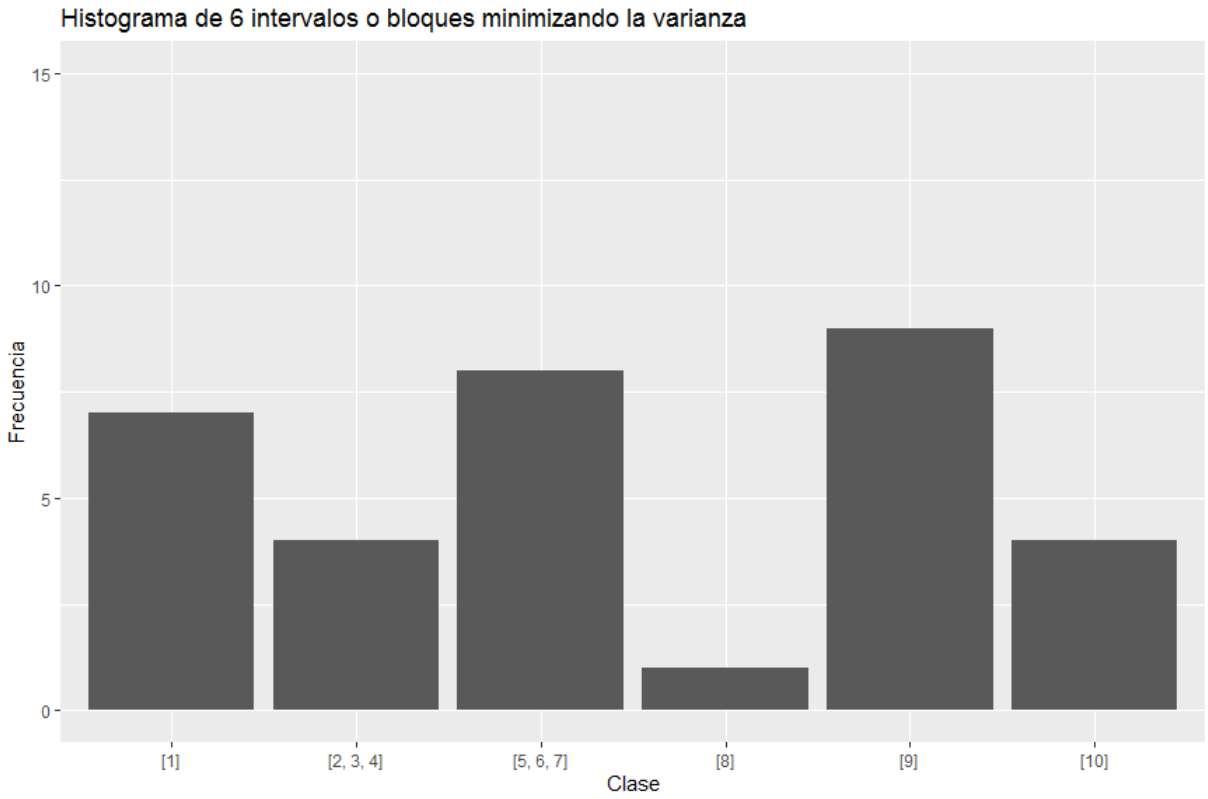
Si agrupamos en un histograma de 6 intervalos o bloques de igual frecuencia la salida sería:



Si agrupamos en un histograma de 4 intervalos o bloques de igual frecuencia la salida sería:



Si agrupamos en un histograma de 6 intervalos o bloques minimizando la varianza sería:



4. Ejercicio 4

Calcula en el ejemplo de Count-Min Sketch que se puede ver en el vídeo los conjuntos sobre los que habría que hallar el mínimo para calcular la frecuencia de D, E y F. ¿En cuál se está cometiendo un error sobre el valor real?

Lista de Algoritmos

Para esta actividad se han realizado una serie de desarrollos en Python y R que se pueden encontrar **AQUÍ**.