Finding if values are in X apart from each other

Asked 7 years ago Modified 7 years ago Viewed 89 times

0

Suppose I have a set of numbers `numbers = [70, 68, 45, 55, 47, 50, 69]`, that sorted gives me (for clarification) `45, 47, 50, 55, 68, 69, 70`.

I want to find the largest group of numbers within a specified `range`, depending of the dataset I am working on. Let's say, for the sake of argument, that I want to find all numbers that are within 3 of each others.

So, with the above input, the output of my program would be : `[70, 68, 69]` as this is the largest group of numbers. If I have multiple equal lists, I choose the one with the highest numbers.

I already wrote a program that solves what I am trying to achieve (or seems to, as I cannot properly see the exception of my program) : what I am seeking from you are :

- links to webpage of...
- name of...
- (or direct help)

... mathematical concepts to help me improve this part of my algorithm. I feel like I am just doing something that works, but this is really, really not optimal **and** which is not looking nice to the eye. I am not looking for the most optimum algorithm, but I want to achieve this task in a more simple, and eloquent way. :)

Proof I did the work:

```
import numpy as np

def find_matching_numbers(numbers):
    tempList = [[numbers[0]]]
    for value in numbers[1:]:
        for x in tempList:
            # tempMean : average of current list
            tempMean = np.mean(x)
             #  if the number is not alone, we're searching from
             #  the average of the list, in both directions (min and max)
             #  with a range of half our initial range search

             #  else if the number is alone, we're searching as if it is
             #  considered as the min AND the max of the range.
            if  len(x) > 1 and value >= tempMean - 1.5 and value <= tempMean +
1.5 or\
                len(x) == 1 and value >= tempMean - 3 and value <= tempMean +
3:
                x.append(value)
```

```
        #   Adding the value to the end of our rangesList,
        #   since it could be a useful value in another context with
        #   other datas.
        tempList.append([value])

    #   Keeping lists with highest length.
    maxLen = max([len(items) for items in tempList])
    ret = []
    for items in tempList:
        if len(items) == maxLen:
            ret.append(items)

    #   If multiple lists with highest length, keep highest one.
    if type(ret[0]) == type([]) and len(ret) > 1:
        ret = max(ret)
    return ret
```

python python-2.7 statistics

Share

Improve this question

Follow

edited Dec 4, 2017 at 22:16

asked Dec 4, 2017 at 15:02

IMCoins

3,276 ● 1 ● 12 ● 27

2 It's a reasonable question, but this is probably not the right forum for it. Maybe try a computer science board like Stack Overflow? – Nik Weaver Dec 4, 2017 at 15:05

As this is more of a mathematical problem than a computing problem, I thought it would be more appropriate here, but if you believe this question deserves to be on stackoverflow, I'll post it there after I let it here a few. :) – IMCoins Dec 4, 2017 at 15:07

This is a computing problem. – Nik Weaver Dec 4, 2017 at 16:05

1 Answer

Sorted by: Highest score (default)

1

Python 3, but I think all you need to do to make it work for python 2 is remove the brackets from the print statement.

```
# uderscore for the _range input so we don't override the builtin range()
function
def find_matching_numbers(numbers, _range=3):
    numbers = sorted(numbers)
    # sorting first allows us a faster algorithm because we know the range
    # is our current number - first in the list
    lists = []
    for num in numbers:
        lists.append([num])
        # don't look at the last list because we just added this number
        for l in lists[:-1]:
            if num - l[0] <= _range:
```

```
                l.append(num)
    # timsort is stable, so the "largest" list will be last
    return sorted(lists, key=len)[-1]

numbers = [70, 68, 45, 55, 47, 50, 69]

print(find_matching_numbers(numbers))
# [68, 69, 70]

visited = set()
for i in range(1, max(numbers) - min(numbers) + 1):
    n = find_matching_numbers(numbers, i)
    if sum(n) not in visited:
                visited.add(sum(n))
                print(i, n)

# 1 [69, 70]
# 2 [68, 69, 70]
# 10 [45, 47, 50, 55]
# 15 [55, 68, 69, 70]
# 20 [50, 55, 68, 69, 70]
# 23 [47, 50, 55, 68, 69, 70]
# 25 [45, 47, 50, 55, 68, 69, 70]
```

Optionally cull lists as you go. This will usually increase performance unless most lists aren't candidates for culling.

```
def find_matching_numbers2(numbers, _range=3):
    numbers = sorted(numbers) # allows us a faster algorithm
    lists = []
    longest = (0, 1)
    for num in numbers:
        remove = set()
        lists.append([num])
        # don't look at the last list because we just added this number
        for l in lists[:-1]:
            ll = len(l)
            if ll > longest[1]:
                longest = (num, ll)
            if num - l[0] <= _range:
                l.append(num)
            elif ll < longest[1]:
                remove.add(l[0])
        lists = [l for l in lists if l[0] not in remove]

    # timsort is stable, so the "largest" list will be last
    return sorted(lists, key=len)[-1]
```

Share

Improve this answer

Follow

edited Dec 5, 2017 at 5:38

answered Dec 5, 2017 at 1:59

Turksarama

1,126 ● 6 ● 13

I love the way you solved this problem. I'm jealous (and a bit ashamed) I haven't thought about it myself. – IMCoins Dec 5, 2017 at 10:10