question | Like

747 views

lab4 3d hangs forever

After check out 894, I still don't have any clue how to improve this problem. Within **parseOHEPoint**, I split label, pass point into **parsePoint**, pass rawFeats,OHEDict,numOHEFeats into **oneHotEncoding** and my program hangs forever(more than 2 hours).
My **parsePoint** use 'zip' and **oneHotEncoding** use 'set.intersection' from 894 and these two functions runs within 10 seconds by themselves.

Can anyone help please?

Update:
This method is important since it may have some dependencies with further questions. So I think it may be helpful if I share my experience here.
After making this post, I did some improvement and then 3d took 32 mins, 3e took 3 mins to run. This was good enough for finishing the lab locally but auto-grader gave time out error.
Then I followed "Christopher Cameron" suggestion. 3d and 3e can be finished within 10 seconds.
For **oneHotEncoding**:
Efficient way:

<Converted to Pseudocode>
loop through rawFeats:
    if the feat is in OHEDict:
        add OHEDict[feat] to the output list

Inefficient ways(at least for me):

1. tmp = set.intersection
   for loop to append items in tmp
2. two for loops
3.
   loop through rawFeats:
       if the feat is in OHEDict.keys():
           add OHEDict[feat] to the output list

And How to create 1.0 did not make a huge difference for me.

For **parsePoint**(3b):
I used **zip**

lab4

Updated 7 days ago by Eric Z and Christopher Cameron

the students' answer, *where students collectively construct a single answer*

My answer from @894, I hope it helps:

I solved OneHotEncoding by using a Python map, passing a lambda function that applies a dictionary method (.get). This in one line, then I returned the SparseVector.

For parseOHEPoint:
1) defined the label using the split method in the point string,
2) defined featID-Val tuple using parsePoint,
3) defined OHEData using OneHotEncoding
4) returned the LabeledPoint

parseOHEPoint ran in less than 8s.

Updated 10 days ago by Paula A

the instructors' answer, *where instructors collectively construct a single answer*

Make sure you are not iterating through all items in the OHEDict.

Consider these two examples and their real world analogs:

```
[PhoneBook[name] for name in PhoneBook if name in name_list]
```

I give you a list of ten names and ask you to look up their phone numbers in the phone book. You tear out every page in the phonebook, mix them in a random order and then, choosing pages at random, you look to see if any of the entries on the page match the names I have given you. You continue the search page-by-random-page until you have a list of phone numbers. Some of the people in my list of ten are not even in the phonebook so you have to search every page in the pile. The larger the phone book, the longer it takes!

```
[PhoneBook[name] for name in name_list if name in PhoneBook]
```

I give you a list of ten names and ask you to look up their phone numbers in the phone book. This time you use the fact that the phone book is an indexed collection. For each name, you look it up in the phone book and record the associated phone number. If the name is not in the book, you skip it and move to the next page. Lookup times are constant no matter how large the phone book gets.

Edit: thanks @@qi yang for this second example

```
[PhoneBook[name] for name in name_list if name in PhoneBook.keys()]
```

You might expect this to be the same as the fast example above but it is actually equivalent to the *slow* method. When you call .keys() on a dict, you get a list of the keys in the dict. Since dicts are not ordered, this is the same as searching the torn out pages one at a time to see if any of the target names are on the list. It is a subtle change from the fast example but it makes the code very slow.

--- Closed several threads here. Reopen and comment if the answer does not help you.

Updated 8 days ago by Christopher Cameron

followup discussions *for lingering questions and comments*

Resolved Unresolved

Huỳnh Nhựt Hải 10 days ago
I also faced endless handle exercises 3 by insufficient memory, I actually cleared cache to reduce crash. However, for exercises 4 and 5, noway to address this problem. So, I had to code without running code until I check on autograder.

Miguel Santos Luparelli Mathieu 7 days ago How did you solve the 'cache' problem?

Resolved Unresolved

Jabokoe 10 days ago
I see people using zip and having memory problems. Not entirely clear how people intend to apply `zip` but if you want to automatically get every item in a list numbered, as in `['a', 'b']` -> `[(0, 'a'), (1, 'b')]` try to use `list(enumerate(['a', 'b']))` as it is a common Python way to also number items in `for` loops.

Anonymous 10 days ago yea..... using enumerate never completes the task for me, had to write something else

Paula A 10 days ago I used zip(length of the features, features) for parsePoint, after slicing off the label.

Resolved Unresolved

Anonymous 9 days ago
I get the following output:

```
[LabeledPoint(0.0, (233286,[18,1549,2650,3470,5155,24560,26067,26368,28248,28308,28364,28441,28604,28943,29609,61278,68693
,87063,87179,87778,94600,94639,94663,106593,110579,144519,144972,148940,163701,178588,179986,182182,182198,183861,214092,21
4100,216704,224912,228336],[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0
,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0]))]
```

Resolved Unresolved

Anonymous 9 days ago
I use set.intersection and Python map functions in oneHotEncoding.
I use zip in parsePoint (with xrange).

Still my parseOHEPoint implementation is taking hours! Any other suggestions?

Resolved Unresolved

Wagner Barretto 8 days ago
I had the exact same problem of hanged 3d and solved by rewriting my oneHotEncoding function from a comprehension

```
labels = [OHEDict[k] for k in OHEDict if k in rawFeats]
return SparseVector(numOHEFeats, sorted(labels) ,[1.]*len(labels))
```

to a more imperative solution with a for loop

```
<SOLUTION CODE REDACTED>
```

Both functions pass the 1d test but only the last make 3c not hang forever.
I have absolute no idea why that happens, maybe pySpark has some issues with list comprehensions.

Hope this helps

Himanshu Gupta 8 days ago OMG Wanger! Thanks a lot.

Christopher Cameron 8 days ago
I am not surprised the second one is better but it is not a problem with list comprehensions.

Consider these two examples and their real world analogs:

```
[PhoneBook[name] for name in PhoneBook if name in name_list]
```

I give you a list of ten names and ask you to look up their phone numbers in the phone book. You tear out every page in the phonebook, mix them in a random order and then, choosing pages at random, you look to see if any of the entries on the page match the names I have given you. You continue the search page-by-random-page until you have a list of phone numbers. Some of the people in my list of ten are not even in the phonebook so you have to search every page in the pile. The larger the phone book, the longer it takes!

```
[PhoneBook[name] for name in name_list if name in PhoneBook]
```

I give you a list of ten names and ask you to look up their phone numbers in the phone book. This time you use the fact that the phone book is an indexed collection. For each name, you look it up in the phone book and record the associated phone number. If the name is not in the book, you skip it and move to the next page. Lookup times are constant no matter how large the phone book gets.

Wagner Barretto 8 days ago Thanks Christopher, I completely missed the inefficiency caused by the linear search on a list instead of a constant lookup on a dictionary. And the problem gets way worse as the dict gets bigger.

But it still surprises me that I couldn't complete the code in my laptop (4th gen i7, I left for almost an hour).

Christopher Cameron 8 days ago It sounds like it was taking about 30 mins for some people. If you were really unlucky, the dict would be too big to fit in memory and end up cached to disk where it would be considerably slower to use!

Himanshu Gupta 8 days ago @Wagner,
I was using dict.iteritems() and that was the reason for slow execution. Looking at your post helped me understand that I could in fact do

```
[(*.*) for k in rawFeats if k in Dict]
```

qi yang 8 days ago Initially, I was using

```
indices = {OHEDict[i]:1.0 for i in rawFeats if i in OHEDict.keys()}
```

It run for 38 mins. After I discarded the key() method, it run for 0.8 secs. I just clearly realize the difference between containment test and iteration test.

Christopher Cameron 8 days ago @qi yang

That is good point -- I hadn't thought of students testing membership in .keys(). I will update the example.


Mahrad Zoonemat Kermani 8 days ago @qi yang
thanks a lot mate. it had become a headache.

William Santo 4 days ago Here is a relevant thread, among others:

http://stackoverflow.com/questions/1602934/check-if-a-given-key-already-exists-in-a-dictionary

Wen Jiang 3 days ago Thank you so much. It worked well~ Especially the way you handle those annoying 1.0. I tried so many solutions but only this one solved my problem.

Resolved Unresolved

Jun Xie 8 days ago

Thanks everyone for very informative discussions here. A general question: what is your favorite way of debugging these sometimes slow-execution code chunks? Use small sample sets? If you time it, what is the best time routine? I wish the instructors would include a short introduction of the best practices in this area (even though it might seem basic to those who are well-versed).

Himanshu Gupta 8 days ago mine is to do a sc.parallelize(BID_Data_RDD.take(x))
Then I use x = 2, 10, 100, 1000 and see how the time increases. If it is not linear, good indication that something in not quite right. Of course then there are moments like iterating over Dict using keys() rather than just do a containment check! Need to learn more python!

Erik Stephens 8 days ago I recommend IPython's %timeit magic method. Prefix a cell with %%timeit (note the extra %). That should provide a quick way to compare one cell's performance with another's.

Christopher Cameron 8 days ago You can also use %%time to get the one-shot execution time.

You will avoid many slow implementations if you use idiomatic python. There is a very good talk about python idioms linked from this

page: https://gist.github.com/JeffPaine/6213790

Juan Hernandez 1 day ago The runtime of this cell depends on how efficiently you wrote the parsePoint and the oneHotEncoding functions. To debug it, start by going back up in the notebook and seeing if you can make parsePoint and oneHotEncoding run faster. My advice is just to think creatively. Try to use as few steps as possible.

This cell should run in under 8 seconds, as someone said above (depending on your computer's speed). Mine runs in 6.

Resolved Unresolved

Philip Chen 3 days ago
How can I stop the notebook in the middle of its running, just like Ctr C on command line or kill (process) command?

Christopher Cameron 3 days ago You can use the "interrupt kernel" menu item