completion_profiling

December 9, 2017

1 Completion profiling

Profiling avec cProfile, memory_profiler, line_profiler, pyinstrument, snakeviz.

In [1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('ggplot')
from jyquickhelper import add_notebook_menu
add_notebook_menu()
```

Out[1]:
```
<IPython.core.display.HTML object>
```

1.1 Setup

1.1.1 Function to profile

In [2]:
```
from mlstatpy.nlp.completion import CompletionTrieNode

def gain_dynamique_moyen_par_mot(queries, weights):
    per = list(zip(weights, queries))
    total = sum(weights) * 1.0
    res = []
    trie = CompletionTrieNode.build([(None, q) for _, q in per])
    trie.precompute_stat()
    trie.update_stat_dynamic()
    wks = [(w, p, len(w) - trie.min_keystroke0(w)[0]) for p, w in per]
    wks_dyn = [(w, p, len(w) - trie.min_dynamic_keystroke(w)[0])
               for p, w in per]
    wks_dyn2 = [(w, p, len(w) - trie.min_dynamic_keystroke2(w)[0])
                for p, w in per]
    gain = sum(g * p / total for w, p, g in wks)
    gain_dyn = sum(g * p / total for w, p, g in wks_dyn)
    gain_dyn2 = sum(g * p / total for w, p, g in wks_dyn2)
    ave_length = sum(len(w) * p / total for p, w in per)
    return gain, gain_dyn, gain_dyn2, ave_length
```

1.1.2 Data

```
In [3]: from mlstatpy.data.wikipedia import download_titles
        file_titles = download_titles(country='fr')

In [4]: from mlstatpy.data.wikipedia import enumerate_titles
        list_titles = list(sorted(set(_ for _ in enumerate_titles(file_titles) if 'A' <= _[0] <=

In [5]: import random
        sample1000 = random.sample(list_titles, 1000)
        with open("sample1000.txt", "w", encoding="utf-8") as f:
            f.write("\n".join(sample1000))
```

1.2 Standard modules

1.2.1 cProfile

```
In [6]: import cProfile, io, pstats, os

        def toprofile0(lines):
            gain_dynamique_moyen_par_mot(lines, [1.0] * len(lines))

        def doprofile(lines, filename):
            pr = cProfile.Profile()
            pr.enable()
            toprofile0(lines)
            pr.disable()
            s = io.StringIO()
            ps = pstats.Stats(pr, stream=s).sort_stats('cumulative')
            ps.print_stats()
            rem = os.path.normpath(os.path.join(os.getcwd(), "..", "..", ".."))
            res = s.getvalue().replace(rem, "")
            ps.dump_stats(filename)
            return res

In [7]: r = doprofile(sample1000, "completion.prof")
        print(r)

         1276683 function calls in 1.544 seconds

   Ordered by: cumulative time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    1.544    1.544 <ipython-input-6-ff8ba7222a38>:3(toprofile0)
        1    0.001    0.001    1.544    1.544 <ipython-input-2-1cd63cc92e3c>:3(gain_dynamique_mo
        1    0.151    0.151    1.094    1.094 \src\mlstatpy\nlp\completion.py:416(precompute_sta
    16023    0.434    0.000    0.812    0.000 \src\mlstatpy\nlp\completion.py:504(merge_completi
        1    0.070    0.070    0.284    0.284 \src\mlstatpy\nlp\completion.py:451(update_stat_dy
    16023    0.207    0.000    0.211    0.000 {built-in method builtins.__build_class__}
    17023    0.076    0.000    0.135    0.000 \src\mlstatpy\nlp\completion.py:556(update_dynamic
```

```
     1    0.106    0.106    0.131    0.131 \src\mlstatpy\nlp\completion.py:203(build)
 35804    0.087    0.000    0.094    0.000 \src\mlstatpy\nlp\completion.py:524(<listcomp>)
 17023    0.045    0.000    0.051    0.000 \src\mlstatpy\nlp\completion.py:589(second_step)
 16023    0.036    0.000    0.043    0.000 \src\mlstatpy\nlp\completion.py:543(update_minimum
 33850    0.016    0.000    0.038    0.000 {built-in method builtins.all}
 17023    0.024    0.000    0.036    0.000 \src\mlstatpy\nlp\completion.py:625(init_dynamic_m
 16023    0.029    0.000    0.035    0.000 {built-in method builtins.sorted}
110002    0.032    0.000    0.032    0.000 {built-in method builtins.hasattr}
 17024    0.020    0.000    0.031    0.000 \src\mlstatpy\nlp\completion.py:97(unsorted_iter)
 16024    0.004    0.000    0.026    0.000 {method 'extend' of 'collections.deque' objects}
308942    0.026    0.000    0.026    0.000 {built-in method builtins.len}
100538    0.025    0.000    0.025    0.000 \src\mlstatpy\nlp\completion.py:436(<genexpr>)
  3000    0.022    0.000    0.022    0.000 \src\mlstatpy\nlp\completion.py:258(find)
  1001    0.015    0.000    0.022    0.000 \src\mlstatpy\nlp\completion.py:132(leaves)
 17023    0.013    0.000    0.015    0.000 \src\mlstatpy\nlp\completion.py:20(__init__)
 48069    0.012    0.000    0.012    0.000 {method 'extend' of 'list' objects}
 22703    0.012    0.000    0.012    0.000 {built-in method builtins.min}
     1    0.001    0.001    0.011    0.011 <ipython-input-2-1cd63cc92e3c>:10(<listcomp>)
     1    0.001    0.001    0.010    0.010 <ipython-input-2-1cd63cc92e3c>:13(<listcomp>)
     1    0.001    0.001    0.010    0.010 <ipython-input-2-1cd63cc92e3c>:11(<listcomp>)
113965    0.010    0.000    0.010    0.000 {method 'values' of 'dict' objects}
  1000    0.001    0.000    0.010    0.000 \src\mlstatpy\nlp\completion.py:322(min_keystroke0
  1000    0.001    0.000    0.009    0.000 \src\mlstatpy\nlp\completion.py:383(min_dynamic_ke
  1000    0.001    0.000    0.009    0.000 \src\mlstatpy\nlp\completion.py:353(min_dynamic_ke
 17022    0.009    0.000    0.009    0.000 \src\mlstatpy\nlp\completion.py:54(_add)
 19781    0.008    0.000    0.008    0.000 {built-in method builtins.max}
 16023    0.008    0.000    0.008    0.000 \src\mlstatpy\nlp\completion.py:512(<listcomp>)
 51828    0.007    0.000    0.007    0.000 {method 'append' of 'list' objects}
 51069    0.007    0.000    0.007    0.000 {method 'pop' of 'list' objects}
 17023    0.007    0.000    0.007    0.000 \src\mlstatpy\nlp\completion.py:518(<genexpr>)
 16023    0.005    0.000    0.005    0.000 \src\mlstatpy\nlp\completion.py:509(Fake)
 52871    0.004    0.000    0.004    0.000 {method 'popleft' of 'collections.deque' objects}
 34849    0.003    0.000    0.003    0.000 {method 'append' of 'collections.deque' objects}
 32046    0.003    0.000    0.003    0.000 {method 'items' of 'dict' objects}
     5    0.000    0.000    0.002    0.000 {built-in method builtins.sum}
 18023    0.002    0.000    0.002    0.000 {built-in method builtins.isinstance}
  1001    0.000    0.000    0.001    0.000 <ipython-input-2-1cd63cc92e3c>:18(<genexpr>)
  1001    0.000    0.000    0.000    0.000 <ipython-input-2-1cd63cc92e3c>:15(<genexpr>)
  1001    0.000    0.000    0.000    0.000 <ipython-input-2-1cd63cc92e3c>:16(<genexpr>)
     1    0.000    0.000    0.000    0.000 <ipython-input-2-1cd63cc92e3c>:7(<listcomp>)
  1001    0.000    0.000    0.000    0.000 <ipython-input-2-1cd63cc92e3c>:17(<genexpr>)
     1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
```

1.3 Others informations when profiling

1.3.1 memory_profiler

See memory_profiler.

```
In [8]: from memory_profiler import profile
        %load_ext memory_profiler

In [9]: %memit toprofile0(sample1000)

peak memory: 365.09 MiB, increment: 22.69 MiB


In [10]: from io import StringIO
         st = StringIO()
         @profile(stream=st)
         def toprofile(lines):
             gain_dynamique_moyen_par_mot(lines, [1.0] * len(lines))
         toprofile(sample1000)

ERROR: Could not find file <ipython-input-10-7e3f7b9a5136>
NOTE: %mprun can only be used on functions defined in physical files, and not in the IPython env


In [11]: %%file temp_mem_profile.py

         from mlstatpy.nlp.completion import CompletionTrieNode
         from memory_profiler import profile

         @profile(precision=4)
         def gain_dynamique_moyen_par_mot(queries, weights):
             per = list(zip(weights, queries))
             total = sum(weights) * 1.0
             res = []
             trie = CompletionTrieNode.build([(None, q) for _, q in per])
             trie.precompute_stat()
             trie.update_stat_dynamic()
             wks = [(w, p, len(w) - trie.min_keystroke0(w)[0]) for p, w in per]
             wks_dyn = [(w, p, len(w) - trie.min_dynamic_keystroke(w)[0])
                        for p, w in per]
             wks_dyn2 = [(w, p, len(w) - trie.min_dynamic_keystroke2(w)[0])
                         for p, w in per]
             gain = sum(g * p / total for w, p, g in wks)
             gain_dyn = sum(g * p / total for w, p, g in wks_dyn)
             gain_dyn2 = sum(g * p / total for w, p, g in wks_dyn2)
             ave_length = sum(len(w) * p / total for p, w in per)
             return gain, gain_dyn, gain_dyn2, ave_length

         @profile(precision=4)
```

```
        def toprofile():
            with open("sample1000.txt", "r", encoding="utf-8") as f:
                lines = [_.strip("\n\r ") for _ in f.readlines()]
            gain_dynamique_moyen_par_mot(lines, [1.0] * len(lines))
        toprofile()
```

```
Overwriting temp_mem_profile.py
```

```
In [12]: import sys
         cmd = sys.executable
         from pyquickhelper.loghelper import run_cmd
         cmd += " -m memory_profiler temp_mem_profile.py"
         out, err = run_cmd(cmd, wait=True)
         print(out)
```

```
Filename: temp_mem_profile.py

Line #    Mem usage    Increment   Line Contents
================================================
     5  41.0898 MiB   0.0000 MiB   @profile(precision=4)
     6                             def gain_dynamique_moyen_par_mot(queries, weights):
     7  41.1133 MiB   0.0234 MiB       per = list(zip(weights, queries))
     8  41.1133 MiB   0.0000 MiB       total = sum(weights) * 1.0
     9  41.1133 MiB   0.0000 MiB       res = []
    10  48.5898 MiB   7.4766 MiB       trie = CompletionTrieNode.build([(None, q) for _, q in pe
    11  57.7305 MiB   9.1406 MiB       trie.precompute_stat()
    12  68.2422 MiB  10.5117 MiB       trie.update_stat_dynamic()
    13  68.4102 MiB   0.1680 MiB       wks = [(w, p, len(w) - trie.min_keystroke0(w)[0]) for p,
    14  68.4883 MiB   0.0781 MiB       wks_dyn = [(w, p, len(w) - trie.min_dynamic_keystroke(w)[
    15  68.4883 MiB   0.0000 MiB                  for p, w in per]
    16  68.5703 MiB   0.0820 MiB       wks_dyn2 = [(w, p, len(w) - trie.min_dynamic_keystroke2(w
    17  68.5703 MiB   0.0000 MiB                   for p, w in per]
    18  68.5703 MiB   0.0000 MiB       gain = sum(g * p / total for w, p, g in wks)
    19  68.5703 MiB   0.0000 MiB       gain_dyn = sum(g * p / total for w, p, g in wks_dyn)
    20  68.5703 MiB   0.0000 MiB       gain_dyn2 = sum(g * p / total for w, p, g in wks_dyn2)
    21  68.5742 MiB   0.0039 MiB       ave_length = sum(len(w) * p / total for p, w in per)
    22  68.5742 MiB   0.0000 MiB       return gain, gain_dyn, gain_dyn2, ave_length
```

```
Filename: temp_mem_profile.py

Line #    Mem usage    Increment   Line Contents
================================================
    24  40.8281 MiB   0.0000 MiB   @profile(precision=4)
    25                             def toprofile():
    26  40.8281 MiB   0.0000 MiB       with open("sample1000.txt", "r", encoding="utf-8") as f:
    27  41.0898 MiB   0.2617 MiB           lines = [_.strip("\n\r ") for _ in f.readlines()]
```

```
    28   68.5742 MiB   27.4844 MiB       gain_dynamique_moyen_par_mot(lines, [1.0] * len(lines))
```

1.3.2 line_profiler

See line_profiler.

```
In [13]: def lineprofile(lines):
             gain_dynamique_moyen_par_mot(lines, [1.0] * len(lines))

In [14]: from mlstatpy.nlp.completion import CompletionTrieNode

In [15]: from line_profiler import LineProfiler
         prof = LineProfiler()
         prof.add_function(gain_dynamique_moyen_par_mot)
         prof.add_function(CompletionTrieNode.precompute_stat)
         prof.run("lineprofile(sample1000)")
         st = io.StringIO()
         prof.print_stats(stream=st)
         rem = os.path.normpath(os.path.join(os.getcwd(), "..", "..", ".."))
         res = st.getvalue().replace(rem, "")
         print(res)
```

```
Timer unit: 4.27635e-07 s

Total time: 3.28407 s
File: <ipython-input-2-1cd63cc92e3c>
Function: gain_dynamique_moyen_par_mot at line 3

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
     3                                           def gain_dynamique_moyen_par_mot(queries, weigh
     4         1          660    660.0      0.0      per = list(zip(weights, queries))
     5         1           27     27.0      0.0      total = sum(weights) * 1.0
     6         1            5      5.0      0.0      res = []
     7         1       817084 817084.0     10.6      trie = CompletionTrieNode.build([(None, q)
     8         1      5303336 5303336.0     69.1       trie.precompute_stat()
     9         1      1360822 1360822.0     17.7       trie.update_stat_dynamic()
    10         1        62181  62181.0      0.8      wks = [(w, p, len(w) - trie.min_keystroke0(
    11         1           14     14.0      0.0      wks_dyn = [(w, p, len(w) - trie.min_dynamic
    12         1        67281  67281.0      0.9                for p, w in per]
    13         1           19     19.0      0.0      wks_dyn2 = [(w, p, len(w) - trie.min_dynami
    14         1        62370  62370.0      0.8                 for p, w in per]
    15         1         1437   1437.0      0.0      gain = sum(g * p / total for w, p, g in wks
    16         1         1315   1315.0      0.0      gain_dyn = sum(g * p / total for w, p, g in
    17         1         1354   1354.0      0.0      gain_dyn2 = sum(g * p / total for w, p, g i
```

```
    18         1         1700   1700.0      0.0      ave_length = sum(len(w) * p / total for p,
    19         1            5      5.0      0.0      return gain, gain_dyn, gain_dyn2, ave_lengt

Total time: 1.85238 s
File: \src\mlstatpy\nlp\completion.py
Function: precompute_stat at line 416

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
   416                                           def precompute_stat(self):
   417                                               """
   418                                               computes and stores list of completions
   419                                               computes mks
   420
   421                                               @param      clean   clean stat
   422                                               """
   423         1            8      8.0      0.0          stack = deque()
   424         1        84563  84563.0      2.0          stack.extend(self.leaves())
   425     52872       148643      2.8      3.4          while len(stack) > 0:
   426     52871       144391      2.7      3.3              pop = stack.popleft()
   427     52871       141582      2.7      3.3              if pop.stat is not None:
   428     18021        38552      2.1      0.9                  continue
   429     34850        86823      2.5      2.0              if not pop.children:
   430      1000         6388      6.4      0.1                  pop.stat = CompletionTrieNode._
   431      1000         3325      3.3      0.1                  pop.stat.completions = []
   432      1000         3143      3.1      0.1                  pop.stat.mks0 = len(pop.value)
   433      1000         2736      2.7      0.1                  pop.stat.mks0_ = len(pop.value)
   434      1000         2432      2.4      0.1                  if pop.parent is not None:
   435      1000         2752      2.8      0.1                      stack.append(pop.parent)
   436     33850       312043      9.2      7.2              elif all(v.stat is not None for v i
   437     16023        57694      3.6      1.3                  pop.stat = CompletionTrieNode._
   438     16023        40829      2.5      0.9                  if pop.leave:
   439                                                               pop.stat.mks0 = len(pop.val
   440                                                               pop.stat.mks0_ = len(pop.va
   441     16023        61078      3.8      1.4                  stack.extend(pop.children.value
   442     16023      2728426    170.3     63.0                  pop.stat.merge_completions(pop.
   443     16023        61143      3.8      1.4                  pop.stat.next_nodes = pop.child
   444     16023       249495     15.6      5.8                  pop.stat.update_minimum_keystro
   445     16023        46985      2.9      1.1                  if pop.parent is not None:
   446     16022        52475      3.3      1.2                      stack.append(pop.parent)
   447                                               else:
   448                                                   # we'll do it again later
   449     17827        56183      3.2      1.3                  stack.append(pop)
```

1.4 Static Visualization

1.4.1 gprof2dot

See gprof2dot.

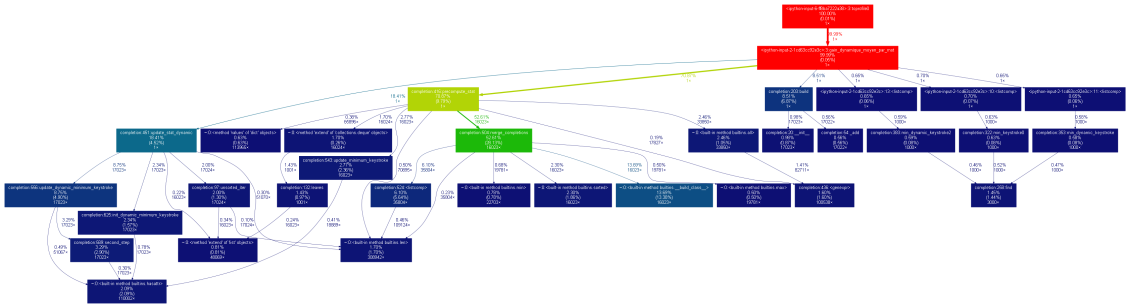
```
In [16]: import gprof2dot
         import sys
         sys.argv=["", "-f", "pstats", "completion.prof", "-o", "completion.dot"]
         gprof2dot.main()
```

```
In [17]: from pyquickhelper.helpgen.conf_path_tools import find_graphviz_dot
         dot = find_graphviz_dot()
```

```
In [18]: from pyquickhelper.loghelper import run_cmd
         out, err = run_cmd('"{0}" completion.dot -Tpng -ocompletion.png'.format(dot), wait=True
         print(out)
```

```
In [19]: from IPython.display import Image
         Image("completion.png")
```

Out[19]:



1.4.2 pyinstrument

See pyinstrument.

```
In [20]: from pyinstrument import Profiler

         profiler = Profiler(use_signal=False)
         profiler.start()

         toprofile0(sample1000)

         profiler.stop()
         out = profiler.output_text(unicode=False, color=False)
         print(out.replace("\\", "/"))
```

```
3.913 gain_dynamique_moyen_par_mot  <ipython-input-2-1cd63cc92e3c>:3
|- 2.628 precompute_stat  mlstatpy/nlp/completion.py:416
|  |- 1.695 merge_completions  mlstatpy/nlp/completion.py:504
|  |  `- 0.289 <listcomp>  mlstatpy/nlp/completion.py:524
|  |- 0.103 leaves  mlstatpy/nlp/completion.py:132
|  |- 0.096 update_minimum_keystroke  mlstatpy/nlp/completion.py:543
|  `- 0.089 <genexpr>  mlstatpy/nlp/completion.py:436
|- 1.022 update_stat_dynamic  mlstatpy/nlp/completion.py:451
|  |- 0.526 update_dynamic_minimum_keystroke  mlstatpy/nlp/completion.py:556
|  |  `- 0.199 second_step  mlstatpy/nlp/completion.py:589
|  |- 0.113 unsorted_iter  mlstatpy/nlp/completion.py:97
|  `- 0.074 init_dynamic_minimum_keystroke  mlstatpy/nlp/completion.py:625
`- 0.192 build  mlstatpy/nlp/completion.py:203
   `- 0.057 __init__  mlstatpy/nlp/completion.py:20
```

1.5 Javascript Visualization

1.5.1 SnakeViz

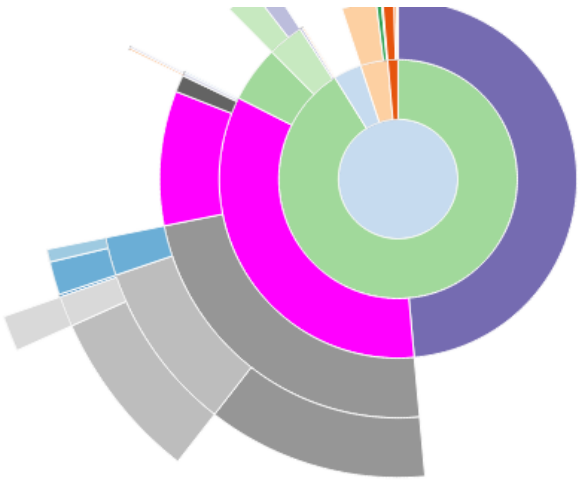
```
In [21]: %load_ext snakeviz
```

L'instruction qui suit lance l'explorateur par d?faut avec les donn?es du profilage.

```
In [22]: # %snakeviz toprofile0(sample1000)
```

```
In [23]: from IPython.display import Image
         Image("https://jiffyclub.github.io/snakeviz/img/func_info.png")
```

Out[23]:



1.5.2 vprof

See vprof.

```
In [24]: from vprof import profiler

         # needs to be run from a file not from a notebook
         # profiler.run(toprofile0, 'cmh', args=(sample1000,), host='localhost', port=8000)
```

```
In [25]: Image("http://i.imgur.com/ijC7wmj.png")
```

Out[25]: