WEIGHTED RANDOM MESSAGES PROGRAM PROPOSAL

Jun-02, 2019
Hieu Phan
pnthieu@gmail.com

Table of Contents

1. Approach 1
2. Test idea proposal to verify if the function work correctly?..... 2
3. Cases will be covered/tested in this program: 4
4. Framework and program structure:..... 7
5. How to run the program 8
6. Result report: 9

1. Approach

The algorithm of this function is simply described as below

1. *Add all the weights for all the messages into the list (weights)*
2. *Pick a number at random between 1 and the sum of the weights*
3. *Iterate over the items (weights)*
4. *For the current item, subtract the item's weight from the random number that was originally picked*
5. *Compare the result to zero. If less than zero then break and return* ***the index (message version index)****. Otherwise keep iterating. The key is that* ***the larger the weight the more possibility to be less than zero*** *when compared to the random selection between zero and the sum of weights.*
6. *It will return -1 if cannot find any index qualified above condition. E.g: the weights with all min weight value [0, 0, 0]*

My function was programmed in Python as below:

```
7.  '''
8.  Generate a random message base on its weights.
9.  Output:
```

```
    - msg version id or -1 if not found
'''
def generate_random_message_ver(weights):
    is_valid, err_msg = True, False

    is_valid, err_msg = _validate_n_weight(weights)
    if not is_valid:
        raise Exception(err_msg)

    msg_ver = -1

    rnd_no = int(random.random() * sum(weights))
    for i, w in enumerate(weights):
        rnd_no -= w
        if rnd_no < 0:
            msg_ver = i
            break

    return msg_ver
```

Stored in the file: *WeightedRandomChoice/lib/message.py*

And I would propose to develop an automation program to cover all possible cases of this function.

2. Test idea proposal to verify if the function works correctly?

The most important thing is that How to test if random message version is distributed CORRECTLY according to its weights?

TO DO this, I will do as below:

1. Get a random message version distribution list which is generated by the function (`generate_random_message_ver`) called ***actual_msg_dist.***

   Note: Number messages of actual_msg_dist is equal to runtime

2. Generate an expected message distribution list base on its weights called **expected_msg_dist**.

   Ex: With weights = [60, 30, 10]; runtime = 1000 => ***expected_msg_dist = [600, 300, 100]***

3. Compare two lists **actual_msg_dist and expected_msg_dist** one by one and count how many messages are wrongly distributed.

   The way to do is that either count:

   1. Items from actual_msg_list is GREATER THAN expected_msg_list

   OR

   2. Items from actual_msg_list is LESS THAN expected_msg_list

This program uses method #1

Ex: actual_msg_dist = [580, 310, 110]; *expected_msg_dist = [600, 300, 100]*

- ⇨ Items [310, 110] from actual_msg_list are greater than expected_msg_dist.
- ⇨ Number of messages wrongly distributed: **20** (= (310 – 300) + (110 - 100))

4. Acceptable percent wrong distribution (10% by default): It means this unites will accept the wrong ratio in total 10%.

   Because each message version is generated randomly so I would suggest to ACCEPT a percent wrong ratio.

   E.g: runtime = 1000 => number_of_messages = 1000, n = 3

   Accept_percent_wrong_dist = 10% => This program accepts 100 messages (10%) which can be wrongly distributed.

   If number of wrong distribution message is LESS than 10%, we consider our algorithm work CORRECTLY.

   - ⇨ Following distributions will be considered as CORRECT: [590, 310, 110] or [540, 320, 140]

5. Others: I also collect list index of message version which are wrongly allocated
6. Finally, raise fail assertion if the ratio >= 10%

All above idea will be cover in the function (`_test_step_by_step_valid_n_weight`) located in *WeightedRandomChoice/tests/TestRandomMessage.py*

```
'''
    Test step by step to verify if the function generate_random_message_ver works
correctly
    according to provided parameter: n, w, runtime and accept_wrong_percent_in_total
    Input:
        - n: number message version.
        - w: list of weights of each version
        - runtime: number of run to get message version
        - accept_wrong_percent_in_total: Acceptable percent messages are wrongly
distributed. By default: 10%
    Output:
        Ok if pass. Error message if wrong.
    '''
    def (self, n, w, runtime, accept_wrong_percent_in_total):
        #1. Get actual and expected message distribution
        actual_msg_dist = lib_msg.get_actual_msg_distribution(w, runtime)
        expected_msg_dist = lib_msg.generate_expected_msg_distribution(w, runtime)

        list_ver_idx = self._gen_arr_msg_version_idx(n)
```

```
        logging.debug('Test info:\n n: %d \n runtime: %d\n [(idx, weight,
exp_no_msg_version, act_no_msg_version)]: %s' % \
                                (n, runtime, zip(list_ver_idx, w, expected_msg_dist,
actual_msg_dist)))

        #2. Get list message has wrong distribution
        list_msg_idx_wrong_dist =
lib_msg.get_list_msg_idx_wrong_distribution(expected_msg_dist, actual_msg_dist)
        #3. Calc % message distribution wrong in total
        no_msg_wrong_dist = lib_msg.count_msg_wrong_distribution(expected_msg_dist,
actual_msg_dist)

        act_wrong_percent_in_total = float(no_msg_wrong_dist)/float(runtime)*100

        if len (list_msg_idx_wrong_dist) > 0:
            logging.debug('List message wrong distribution (expected, actual): ')
            str = ''
            for idx in list_msg_idx_wrong_dist:
                str += '(%d, %d)' % (expected_msg_dist[idx], actual_msg_dist[idx])
                #logging.info('(%d, %d)' % (expected_msg_dist[idx],
actual_msg_dist[idx]))

            logging.debug(str)

        logging.info(
            'No of msg wrong distribution per total: %d/%d. (%%Expected, %%Actual):
(%0.00f(%%), %0.00f(%%))' %
            (no_msg_wrong_dist, runtime, accept_wrong_percent_in_total,
act_wrong_percent_in_total)
        )
        self.assertLessEqual(
            act_wrong_percent_in_total, accept_wrong_percent_in_total,
            msg= 'Fail: (exp_wrong_pecent, act_wrong_percent): (%0.00f, %0.00f)' %
            (accept_wrong_percent_in_total, act_wrong_percent_in_total)
        )
```

3. Cases will be covered/tested in this program:

To describe what cases will be covered/tested by the unittest, copying its test function here would be enough:

Case 1:

```
'''
    As developer, I want to test the case n < min_n value. So that, check if the
funciton handles invalid n correctly.

    Test info:
        - n: less than min_n
```

```
        - weights: random between min_weight and max_weight
        - runtime: Use default runtime from config/config.json
    Expect:
        The function generate_random_message_ver handles n < min_n correctly
    '''
    def test_invalid_n_less_than_min_n(self):
```

Case 2:

```
'''
    As developer, I want to test the case n > max_n value.  So that, check if the
funciton handles invalid n correctly.

    Test info:
        - n: greater than max_n
        - weights: random between min_weight and max_weight
        - runtime: Use default runtime from config/config.json
    Expect:
        The function generate_random_message_ver handles n > max_n correctly
    '''
    def test_invalid_n_greater_than_max_n(self):
```

Case 3:

```
'''
    As developer, I want to test the case n = min_n. So that, check if the funciton
generate_random_message_ver works correctly.

    Test info:
        - n: equal min_n
        - weights: random between min_weight and max_weight
        - runtime: Use default runtime set in config/config.json (2000)
    Expect:
        The function generate_random_message_ver works correctly with n = min_n
    '''
    def test_n_equal_min_n(self):
```

Case 4:

```
'''
    As developer, I want to test the case n = max_n. So that, check if the function
generate_random_message_id works correctly.

    Test info:
        - n: equal max_n
        - weights: random between min_weight and max_weight
        - runtime: Use default runtime from config/config.json
    Expect:
        The function generate_random_message_ver works correctly with n = max_n
    '''
```

```
    def test_n_equal_max_n(self):
```

Case 5:

```
'''
    As developer, I want to test the case weight  < min_weight.
    So that, check if the funciton generate_random_message_id handles invalid weight
correctly.

    Test info:
        - n: random(min_n, max_n)
        - weights: invalid weight less than min_weight
        - runtime: Use default runtime from config/config.json
    Expect:
        The function generate_random_message_ver works correctly with weight =
max_weight
    '''
    def test_invalid_weight_less_than_min_weight(self):
```

Case 6:

```
'''
    As developer,
    I want to test the case weight  > max_weight.
    So that, check if the function generate_random_message_id handles invalid weight
correctly.

    Test info:
        - n: random(min_n, max_n)
        - weights: invalid weight greater than max_weight
        - runtime: Use default runtime from config/config.json
    Expect:
        The function generate_random_message_ver works correctly with weight =
max_weight
    '''
    def test_invalid_weight_greater_than_max_weight(self):
```

Case 7:

```
'''
    As developer, I want to test the case weight = min_weight. So that, check if this
function works correctly
    Test info:
        - n: random(min_n, max_n)
        - weights: min_weight
        - runtime: Use default runtime from config/config.json
    Expect:
        Function works correctly with weight = min_weight
    '''
```

```
    def test_weight_equal_min_weight(self):
```

Case 8:

```
'''
    As developer, I want to test the case weight = max_weight. So that, check if
function works correctly
    Test info:
        - n: random(min_n, max_n)
        - weights: max_weight
        - runtime: Use default runtime from config/config.json
    Expect:
        The function generate_random_message_ver works correctly with weight =
max_weight
    '''

    def test_weight_equal_max_weight(self):
```

Case 9:

```
'''
    As developer, I want to test the case random n and random weight. So that, check
if function works correctly
    Test info:
        - n: random(min_n, max_n)
        - weights: random (min_weight, max_weight)
        - runtime: Use default runtime from config/config.json
    Expect:
        The function generate_random_message_ver works correctly with weight =
max_weight
    '''

    def test_random_n_and_weight_between_min_max(self):
```

4. **Framework and p**rogram structure:

I implemented and tested this function in **Python** completely. I developed a test class to test all possible cases **automatically** using pyunit framework.

The program has structure as below:

WeightedRandomChoice

|

|

-------config

| |----__init__.py

| |----config.json

|

-------lib

```
|      |----__init__.py
|      |----common.py
|      |----message.py
|
-------tests
       |----__init__.py
       |----runner.py
       |----TestRandomMessage.py
```

In detail:

- config/config.json: Store all needed configuration parameters used in the program.

```
{
  "run_cfg": {
            "min_n": 1,
            "max_n": 100,
            "min_weight": 0,
            "max_weight": 1000000000
    },
    "test_cfg": {
            "accept_wrong_percent_in_total": 10,
            "runtime": 2000,
            "runtime_small": 200,
            "runtime_medium": 2000,
            "runtime_large": 20000
    },
    "err_msg_prefix": {
        "invalid_n": "Invalid n",
        "invalid_weight": "Invalid weight"
    }
}
```

- lib/common.py: Contain common util function used in the program like: get_run_cfg, get_test_cfg…
- lib/message.py: Contain functions to work/support “generate message” like generate_random_message_ver, count_msg_wrong_distribution,…
- tests/TestRandomMessage.py: A unittest class contains all test functions to test all possible cases of this algorithm.
- tests/runner.py: Where to add a test suite for test class

5. How to run the program

1. Download and install Python 2.7
2. Open terminal on MAC or command on Windows
3. Navigate to its folder: WeightedRandomChoice and execute below command: *python tests/runner.py*

Note: Please be noted that I have just tested this program on MacOS only, not test on Windows yet.

6. **Result report**:

Because its limited time and to keep running program as simple as possible (don't need to install some additional libraries in Python) so I only output the test result to terminal as below:

```
abcs-MacBook-Pro:WeightedRandomChoice terryhieu$ cd /Users/terryhieu/Documents/WeightedRandomChoice ; env
PYTHONIOENCODING=UTF-8 PYTHONUNBUFFERED=1
/Library/Frameworks/Python.framework/Versions/2.7/Resources/Python.app/Contents/MacOS/Python
/Users/terryhieu/.vscode/extensions/ms-python.python-2019.5.17517/pythonFiles/ptvsd_launcher.py --default --nodebug --
client --host localhost --port 56294 /Users/terryhieu/Documents/WeightedRandomChoice/tests/runner.py
test_invalid_n_greater_than_max_n (TestRandomMessage.TestGenerateRandomMessageVer) ... ok
test_invalid_n_less_than_min_n (TestRandomMessage.TestGenerateRandomMessageVer) ... ok
test_invalid_weight_greater_than_max_weight (TestRandomMessage.TestGenerateRandomMessageVer) ... ok
test_invalid_weight_less_than_min_weight (TestRandomMessage.TestGenerateRandomMessageVer) ... ok
test_n_equal_max_n (TestRandomMessage.TestGenerateRandomMessageVer) ...
2019-06-03 00:51:48,171 [INFO]: No of msg wrong distribution per total: 196/2000. (%Expected, %Actual): (10(%), 10(%))
ok
test_n_equal_min_n (TestRandomMessage.TestGenerateRandomMessageVer) ...
2019-06-03 00:51:48,836 [INFO]: No of msg wrong distribution per total: 0/2000. (%Expected, %Actual): (10(%), 0(%))
ok
test_random_n_and_weight_between_min_max (TestRandomMessage.TestGenerateRandomMessageVer) ...
2019-06-03 00:51:49,574 [INFO]: No of msg wrong distribution per total: 170/2000. (%Expected, %Actual): (10(%), 8(%))
ok
test_weight_equal_max_weight (TestRandomMessage.TestGenerateRandomMessageVer) ...
2019-06-03 00:51:50,178 [INFO]: No of msg wrong distribution per total: 141/2000. (%Expected, %Actual): (10(%), 7(%))
ok
test_weight_equal_min_weight (TestRandomMessage.TestGenerateRandomMessageVer) ...
2019-06-03 00:51:50,783 [INFO]: No of msg wrong distribution per total: 0/2000. (%Expected, %Actual): (10(%), 0(%))
ok

----------------------------------------------------------------------
Ran 9 tests in 3.281s

OK
<unittest.runner.TextTestResult run=9 errors=0 failures=0>
```

----------End----------