

INF553 Foundations and Applications of Data Mining

Spring 2020

Assignment 5

Deadline: Apr. 20th 15:00 PM PST

1. Overview of the Assignment

In this assignment, you are going to implement three streaming algorithms. In the first two tasks, you will generate a simulated data stream with the Yelp dataset and implement **Bloom Filtering** and **Flajolet-Martin** algorithm. In the third task, you will do some analysis using **Fixed Size Sample** (Reservoir Sampling).

2. Requirements

2.1 Programming Requirements

- a. **You must use Python to implement all tasks.** There will be a 10% **early bonus** if your last submission is before Apr. 17th 15:00 PST. Be attention, you will only get the 10% bonus only if your python version output is correct.
- b. You are not required to use Spark RDD in this assignment.
- c. You can only use standard Python libraries.

2.2 Programming Environment

Python 3.6

We will use these library versions to compile and test your code. There will be a 20% penalty if we cannot run your code due to the library version inconsistency.

2.3 Important:

1. If we can't call myhashs(s) in task1 and task2 in your script to get the hash value list, there will be a 50% penalty.
2. We will simulate your bloom filter in the grading program simultaneously based on your myhashs(s) outputs. There will be no point if the reported output is largely different from our simulation.
3. Please use integer 553 as the random seed for task3, and use the function **random.randint(0,100000)** to get a random number. If you use the wrong random seed, or discard any obtained random number, or the sequence of random numbers is different from our simulation, there will be a 50% penalty.

2.4 Write your own code

Do not share code with other students!!

For this assignment to be an effective learning experience, you must write your own code! We emphasize this point because you will be able to find Python implementations of some of the required functions on the web. Please do not look for or at any such code!

TAs will combine all the code we can find from the web (e.g., Github) as well as other students' code from this and other (previous) sections for plagiarism detection. We will report all detected plagiarism.

3. Datasets

For this assignment, you need to download the users.txt as the input file. You also need a Python blackbox file to generate data from the input file. We use this blackbox as a simulation of a data stream. The blackbox will return a **list** of User ids which are from file users.txt every time you ask it. Please call the function as this:

```
from blackbox import BlackBox
bx = BlackBox()
stream_users = bx.ask(file_name, stream_size)
```

While doing the tasks, you might need to ask the blackbox multiple times. You may do it by the following sample code:

```
for _ in range(num_of_asks):
    stream_users = bx.ask(file_name, stream_size)
    your_function(stream_users)
```

4. Tasks

4.1 Task1: Bloom Filtering (2.5 pts)

You will implement the Bloom Filtering algorithm to estimate whether the user_id in the data stream has shown before. The details of the Bloom Filtering Algorithm can be found at the streaming lecture slide. **You need to find proper hash functions and the number of hash functions in the Bloom Filtering algorithm.**

In this task, you should keep a **global filter bit array** and **the length is 69997**.

The hash functions used in a Bloom filter should be [independent](#) and [uniformly distributed](#). Some possible the hash functions are:

$$f(x) = (ax + b) \% m \text{ or } f(x) = ((ax + b) \% p) \% m$$

where p is any prime number and m is the length of the filter bit array. You can use any combination for the parameters (a, b, p). The hash functions should keep the same once you created them.

As the user_id is a string, you need to convert it into an integer and then apply hash functions to it., the following code shows one possible solution:

```
import binascii
int(binascii.hexlify(s.encode('utf8')),16)
```

(We only treat the **exact same** strings as the same users. You do not need to consider aliases.)

Execution Details

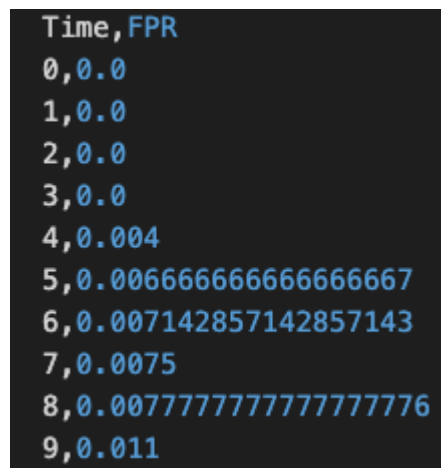
You need to maintain a previous user set in order to calculate the false positive rate (FPR).

For this task, the size of a single data stream will be 100(stream_size). And we will test your code for more than 30 times(num_of_asks), **and your FPRs are only allowed to be larger than 0.5 at most once.**

The run time should be within 100s for 30 data streams.

Output Results

You need to save your results in a **CSV** file with the header "Time,FPR". Each line stores the index of the data batch (starting from 0) and the false positive rate **for that batch of data**. You do not need to round your answer.



Time	FPR
0	0.0
1	0.0
2	0.0
3	0.0
4	0.004
5	0.006666666666666667
6	0.007142857142857143
7	0.0075
8	0.007777777777777776
9	0.011

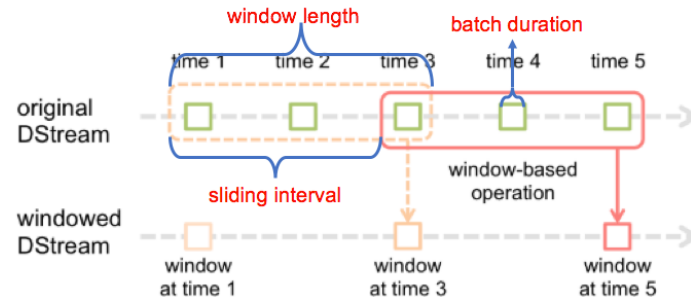
You also need to encapsulate your hash functions into a function called myhashs(s). The input of myhashs(s) is a string(user_id) and the output is a list of hash values (e.g. if you have 3 hash functions, then the size of the output list should be 3 and each element in the list correspond to an output value of your hash function). Below is an example:

```
def myhashs(s):
    result=[]
    for f in hash_function_list:
        result.append(f(s))
    return result
```

Our grading program will also import your python script and call myhashs(s) to test the performance of your hash functions, track your implementation, and compare your result.

4.2 Task2: Flajolet-Martin algorithm (2.5 pts)

In task2, you will implement the Flajolet-Martin algorithm (including the step of combining estimations from groups of hash functions) to estimate the number of unique users within a window in the data stream. The details of the Flajolet-Martin Algorithm can be found at the streaming lecture slide. You need to find proper hash functions and the number of hash functions in the Flajolet-Martin algorithm.



Execution Details

For this task, the size of the stream will be 300(stream_size). And we will test your code more than 30 times(num_of_asks). And for your final result, $0.2 \leq (\text{sum of all your estimations} / \text{sum of all ground truths}) \leq 5$.

The run time should be within 100s for 30 data streams.

Output Results

You need to save your results in a **CSV** file with the header "Time,Ground Truth,Estimation". Each line stores the index of the data batch (starting from 0), the actual number of unique users in the window period, and the estimation result from the Flajolet-Martin algorithm.

```
Time,Ground Truth,Estimation
0,300,248
1,300,304
2,300,267
3,300,324
4,300,280
5,300,290
6,300,296
7,300,292
```

You also need to encapsulate your hash functions into a function called myhashs(s). The input of myhashs(s) is a string(user_id) and the output is a list of hash values (e.g. if you have 3 hash functions, then the size of the output list should be 3 and each element in the list correspond to an output value of your hash function). Below is an example:

```
def myhashs(s):
    result=[]
    for f in hash_function_list:
        result.append(f(s))
    return result
```

Our grading program will also import your python script and call myhashs(s) to test the performance of your hash functions, track your implementation, and compare your result.

4.3 Task3: Fixed Size Sampling (2pts)

In this task you need to implement the fixed size sampling method (Reservoir Sampling Algorithm).

In this task, we assume that the memory can only **save 100 users**, so we need to use the fixed size sampling method to only keep part of the users as a sample in the streaming. When the streaming of the users comes, for the first 100 users, you can directly save them in a list. After that, for the n^{th} (starting from 1) user in the whole sequence of users, you will keep the n^{th} user with the probability of $100/n$, otherwise discard it. If you keep the n^{th} user, you need to randomly pick one in the list to be replaced.

You also need to keep a global variable representing the sequence number of the users.

Execution Details

For this task, the size of the stream will be 100(stream_size). And we will test your code more than 30 times(num_of_asks) .

Be attention here: **Please write your random.seed(553) in the main function. Please do not write random.seed(553) in other places.**

The run time should be within 100s for 30 data streams.

Output Results:

Every time you receive 100 users, you should print the current stage of your reservoir into a CSV file.

E.g. after receiving the 100th user from the streaming, calculate whether the reservoir will replace it with a user in the list or not. Then output the current stage of the reservoir according to the following format, and start a newline.

For each line, the first column is the sequence number (starting from 1) of the latest user in the entire streaming, then the 1th user (with index 0 in your list), 21th user, 41th user, 61th user and 81th user in your reservoir. Below is an example:

```
seqnum,0_id,20_id,40_id,60_id,80_id
100,JDOEDdY30eMfRTjitEnMeg,qGfqAyyPIwoFZ0JC6yC1gw,31wZQG1nKIVv4bxuTJ8CA,X5p43ec6GN4LLoNWUcfZAA,5BIOXJ_vd6uKnGsYH0-oOA
200,JDOEDdY30eMfRTjitEnMeg,k4E9KYqWd2q4hxDgyXT36w,31wZQG1nKIVv4bxuTJ8CA,4oYEAJz9_eg3Y8TiJZLc2g,5BIOXJ_vd6uKnGsYH0-oOA
300,JDOEDdY30eMfRTjitEnMeg,tXQTGAJYZVRzMyFpF0hqJw,31wZQG1nKIVv4bxuTJ8CA,4oYEAJz9_eg3Y8TiJZLc2g,5BIOXJ_vd6uKnGsYH0-oOA
```

streaming printing information example

Please use integer 553 as the random seed, and use the function **random.randint(0,100000)** to get a random number. If you use the wrong random seed, or discard any obtained random number, or the sequence of random numbers is different from our simulation, there will be a 50% penalty.

4.4 Execution Format

```
python task1.py <input_filename> stream_size num_of_asks <output_filename>
```


```
python task2.py <input_filename> stream_size num_of_asks <output_filename>
```


```
python task3.py <input_filename> stream_size num_of_asks <output_filename>
```

5. Submission

You need to submit following files on **Vocareum** with exactly the same name:

 task1.py

 task2.py

 task3.py

6. Grading Criteria

(% penalty = % penalty of possible points you get)

1. You can use your free 5-day extension separately or together but not after one week after the deadline, since the due date is the last day of the class.
2. There will be a 10% bonus if you use both Scala and Python.
3. If we cannot run your programs with the command we specified, there will be an 80% penalty.
4. If we can't call myhashs(s) in your script to get the hash value list, there will be a 50% penalty.
5. [When your program is running, we will simulate your program in our grading program simultaneously based on your myhashs\(s\) outputs.](#) **There will be no point if the reported output is largely different from our simulation.**
6. If you use the wrong random seed, or discard any obtained random number, or the sequence of random numbers is different from our simulation, there will be a 50% penalty.
7. If your program cannot run with the required Python versions, there will be a 20% penalty.
8. If the outputs of your program are unsorted or partially sorted, there will be a 50% penalty.
9. There will be a 20% penalty for late submission within a week and no point after a week.