

DSCI553 Foundations and Applications of Data Mining

Spring 2024

Assignment 5

Deadline: April 12th - 11:59 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 the **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 and Spark to implement all tasks.** There will be a 10% bonus for each task if you also submit a Scala implementation and both your Python and Scala implementations are correct.
- b. You are not required to use Spark RDD in this assignment.
- c. You can only use standard Python libraries, which are already installed in the Vocareum.

2.2 Programming Environment

Python 3.6, JDK 1.8, Scala 2.12, and Spark 3.1.2

We will use the above library versions to compile and test your codes. You are required to make sure your codes work and run on Vocareum otherwise we won't be able to grade your code.

2.3 Important things before starting the assignment:

1. If we cannot 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 task 3**, and follow the steps mentioned below 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 codes 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 use users.txt as the input file. You also need a Python blackbox file to generate data from the input file. Both users.txt and blackbox.py can be found in the publicdata directory on Vocareum. We use the blackbox as a simulation of a data stream. The blackbox will return a **list** of user ids from file users.txt every time we call it. **Although it is very unlikely that the user ids returned from the blackbox are not unique, you are required to handle it wherever required.** Please call the blackbox function like the example in the following figure:

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

If you need to ask the blackbox multiple times, you can 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 on the streaming lecture slide. **Please 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 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 remain the same once you create them.

As the user_id is a string, you need to convert the type of user_id to an integer and then apply hash functions to it. The following codes show one possible solution to converting the user_id string to an integer:

```
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

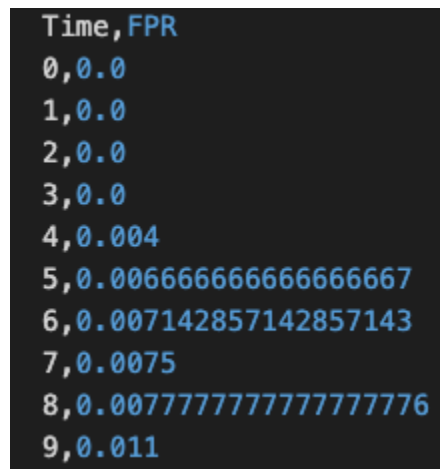
To calculate the false positive rate (FPR), you need to maintain a previous user set.

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
8,0.0077777777777777776
9,0.011
```

You also need to encapsulate your hash functions into a function called myhashs. The input of myhashs function is a user_id (string) and the output is a list of hash values. For example, if you have three hash functions, the size of the output list should be three and each element in the list corresponds to an output value of your hash function. Figure below is a template of myhashs function:

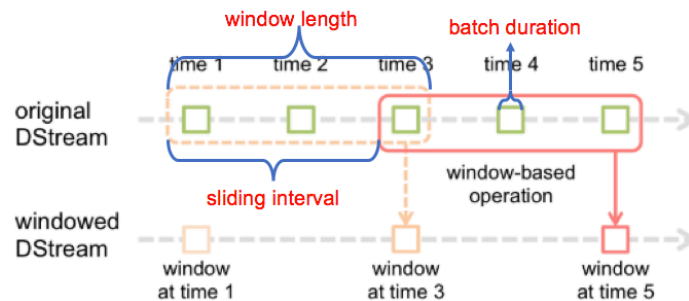
```
def myhashs(s):
    result=[]
    for f in hash_function_list:
        result.append(f(s))
    return result
```

Our grading program will also import your Python script, call myhashs function to test the performance of your hash functions and track your implementation.

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 on 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 function is a user_id (string) and the output is a list of hash values. For example, if you have three hash functions, then the size of the output list should be three and each element in the list corresponds to an output value of your hash function. Figure below is a template of myhashs function:

```
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, call myhashs function to test the performance of your hash functions, and track your implementation.

4.3 Task3: Fixed Size Sampling (2pts)

The goal of task3 is 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} (n starts from 101) 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.

The submission report in Vocareum will show both Python and Scala results only for this task since the outputs generated from Python and Scala scripts would be different.

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 careful 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. For example, after receiving the 100th user from the streaming, your codes should calculate whether the reservoir will replace it with a user in the list or not, and 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. Figure below is an example:

```
seqnum,0_id,20_id,40_id,60_id,80_id
100,JDOEDdY30eMfRTjitEnMeg,qGfQyPyPIwoFZ0JC6yC1gw,31wZQG1nKIVv4bxeuTJ8CA,X5p43ec6GN4LLoNWUcfZAA,5BIOXJ_vd6uKnGsYH0-oOA
200,JDOEDdY30eMfRTjitEnMeg,k4E9KYqWd2q4hxDgyXT36w,31wZQG1nKIVv4bxeuTJ8CA,4oYEAJz9_eg3Y8TiJZLc2g,5BIOXJ_vd6uKnGsYH0-oOA
300,JDOEDdY30eMfRTjitEnMeg,tXQTGAJYZVRzMyFp0hqJw,31wZQG1nKIVv4bxeuTJ8CA,4oYEAJz9_eg3Y8TiJZLc2g,5BIOXJ_vd6uKnGsYH0-oOA
```

streaming printing information example

Important Instructions for task3:

We will compare the output of your codes to the ground truth, your output should be exactly the same as the ground truth output to get the full scores. We will not be providing the ground truth output, but if you follow the following instructions correctly, you will be able to get the correct results easily.

For Python Implementation:

1. Use `random.seed(553)` in the main function
2. For the probability of whether to accept a sample or discard, use `random.random()` which generates a floating point number between 0 and 1. If this randomly generated probability is less than s/n , we accept the sample
3. In case we decide to accept a sample, we need to find an index in the array for replacement. For this purpose use `random.randint()` with appropriate boundaries to generate an index into the array and use this for replacement of the sample.

For Scala Implementation:

1. The scala implementation is very similar to Python, but since the random number generation works differently, the output generated will be different.
2. We will use the `scala.util.Random` class for random number generation. Please only instantiate one instance of this class and use it everywhere in the task3 class.
3. Set the seed to 553 immediately after creating an object of the class.
4. Use `random_object.nextFloat()` to generate a probability for accepting and discarding similar to Python.
5. Use `random_object.nextInt()` with appropriate boundary parameters to generate an index for replacement.
6. Sample codes for BlackBox:

```
class Blackbox {  
  private val r1 = scala.util.Random  
  r1.setSeed(553)  
  def ask(filename: String, num: Int): Array[String] = {  
    val input_file_path = filename  
  
    val lines = Source.fromFile(input_file_path).getLines().toArray  
    var stream = new Array[String](num)  
  
    for (i <- 0 to num - 1) {  
      stream(i) = lines(r1.nextInt(lines.length))  
    }  
    return stream  
  }  
}
```

Since we cannot import the python `blackbox.py` in scala, please use the reference code provided above. You can add this code to your `task3.scala` file itself and make an object of the `Blackbox` class within your main `task3` class. You can then use this object to request a new array of streams of size “num” by calling the `ask` function like Python implementation.

Here is an example:

```
var stream = box.ask(input_file_path, stream_size)
```

Please do not use the “mod” operator to limit the index. Set the appropriate boundaries in the random number generation function itself.

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

Task1:

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

```
/home/local/spark/latest/bin/spark-submit --class task1 --executor-memory 4G --driver-memory 4G  
hw5.jar $<input_filename> stream_size num_of_asks <output_filename>
```

Task2:

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

```
/home/local/spark/latest/bin/spark-submit --class task2 --executor-memory 4G --driver-memory 4G  
hw5.jar <input_filename> stream_size num_of_asks <output_filename>
```

Task3:

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

```
/home/local/spark/latest/bin/spark-submit --class task3 --executor-memory 4G --driver-memory 4G  
hw5.jar <input_filename> stream_size num_of_asks <output_filename>
```

5. Submission

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

- task1.py, [task1.scala]
- task2.py, [task2.scala]
- task3.py, [task3.scala]
- [hw5.jar]
- blackbox.py (copy this file from publicdata directory)
- [Blackbox.scala]

6. Grading Criteria

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

1. You can use your **free 5-day extension** separately or together using the link below:

[Late Day Request Form](#)

This form will record the number of late days you use for each assignment. We will not count late days if no request is submitted. Remember to submit the request **BEFORE the deadline**.

2. There will be a 10% bonus for correct scala implementation provided the Python implementation works correctly.
3. If we cannot run your programs with the command we specified, there will be no regrading
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. We can regrade your assignments within seven days once the scores are released. **No argument after one week.**
8. There will be a **20% penalty** for late submission within a week and **no point** after a week.

7. Common problems causing fail submission on Vocareum/FAQ

(If your program runs successfully on your local machine but fails on Vocareum, please check these)

1. Try your program on the Vocareum terminal. Remember to set Python version as python3.6,

```
"export PYSPARK_PYTHON=python3.6"
```

Use the latest Spark

```
/opt/spark/spark-3.1.2-bin-hadoop3.2/bin/spark-submit
```

Select JDK 8 by running the command

```
"export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-amd64"
```

2. Check the input command line format.
3. Check the output format, for example, the header, tag, and typos.
4. Check the requirements for sorting the results.
5. Your program scripts should be named task1.py task2.py etc.
6. Check whether your local environment fits the assignment description, i.e. version, and configuration.
7. If you implement the core part in Python instead of Spark, or implement it in a high-time complexity way (e.g. search an element in a list instead of a set), your program may be killed on Vocareum because it runs too slowly.
8. Upload a copy of blackbox.py and Blackbox.scala(if applicable) into the "work" folder in Vocareum.