

# INF553 Foundations and Applications of Data Mining

Fall 2019

## Assignment 5

**Deadline: Nov. 20<sup>th</sup> 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 with Spark Streaming. 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, the required algorithms must be implemented in Spark. There will be **10% bonus** for each task if you also submit a Scala implementation and both your Python and Scala implementations are correct.
- b. You will need **Spark Streaming library**
- c. **You can only use Spark RDD and standard Python or Scala libraries.**

#### 2.2 Programming Environment

**Python 3.7, Scala 2.11, JDK 1.8 and Spark 2.4.4**

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.

For this homework, the workers and drivers of Spark will run in Python3.7.

#### 2.3 Important:

1. If we can't call myhashs(s) in your script to get the hash value list, there will be 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, and use 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 number is different from our simulation, there will be 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 task1 and task2, you need to download the users.json file and the generate\_stream.jar on the Blackboard. Please follow the instructions below to simulate streaming on your machine:

- 1) Run the generate\_stream.jar in the terminal to generate Yelp streaming data from the "users.json" with the command:

```
java -cp <generate_stream.jar file path> StreamSimulation <users.txt file path> 9999 100
```

- 9999 is a port number on the localhost. You can assign any available port to it.
  - 100 represents 100 milliseconds (0.1 second) which is the time interval between items in the simulated data stream.
- 2) Keep step 1) running while testing your code. Use "Ctrl+C" to terminate if necessary.
  - 3) Add the following code to connect the data stream in your Spark Streaming code:

```
ssc.socketTextStream("localhost", 9999)
```

- The first argument is the host name, which is "localhost" in this case.
- The second argument is the port number in step 1), which is 9999 in this case.

### 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 the same** strings as the same users. You do not need to consider alias.)

### Execution Details

In Spark Streaming, set the batch duration to **10** seconds:

```
ssc=StreamingContext(sc, 10)
```

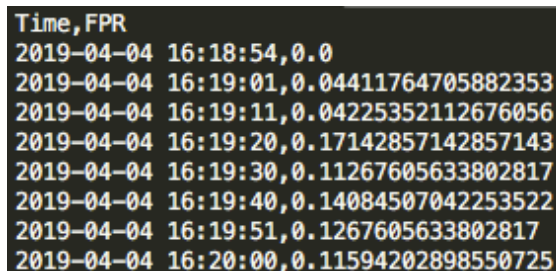
You will get a batch of users in spark streaming every 10 seconds and you will use the Bloom Filtering algorithm to estimate whether the coming user appeared before or not since the beginning of your code.

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

**We will test your code for 5 minutes and your FPRs are only allowed to be larger than 0.05 at most once.**

### Output Results

You need to save your results in a **CSV** file with the header “Time,FPR”. Each line stores the timestamp when you receive the batch of data and the false positive rate **for that batch of data**. The time format should be “YYYY-MM-DD hh:mm:ss” (Figure 2 shows an example). You do not need to round your answer.



Time, FPR
2019-04-04 16:18:54,0.0
2019-04-04 16:19:01,0.04411764705882353
2019-04-04 16:19:11,0.04225352112676056
2019-04-04 16:19:20,0.17142857142857143
2019-04-04 16:19:30,0.11267605633802817
2019-04-04 16:19:40,0.14084507042253522
2019-04-04 16:19:51,0.1267605633802817
2019-04-04 16:20:00,0.11594202898550725

Figure 2: Output file format for task1

**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 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.

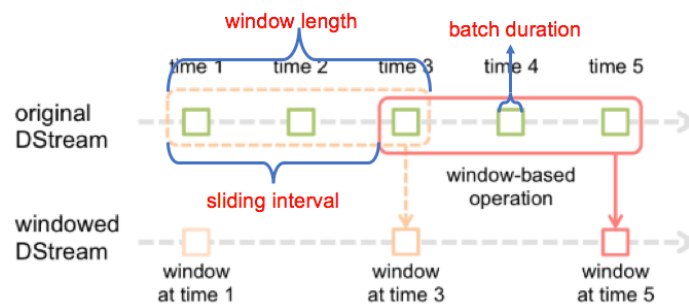


Figure 3: Spark Streaming window

### Execution Details

For this task, the batch duration should be **5** seconds, the window length should be **30** seconds and the sliding interval should be **10** seconds. We will test your code for **10** minutes.

### Output Results

You need to save your results in a **CSV** file with the header "Time,Ground Truth,Estimation". Each line stores the timestamp when you receive the batch of data, the actual number of unique users in the window period, and the estimation result from the Flajolet-Martin algorithm. The time format should be "YYYY-MM-DD hh:mm:ss" (Figure 4 shows an example). You do not need to round your answer.

```
Time,Ground Truth,Estimation
2019-04-04 18:03:21,24,41
2019-04-04 18:03:30,42,41
2019-04-04 18:03:41,63,41
2019-04-04 18:03:50,65,26
2019-04-04 18:04:00,73,26
2019-04-04 18:04:10,74,41
2019-04-04 18:04:20,78,42
```

Figure 4: Flajolet-Martin output file format

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 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 coming, for the first 100 users, you can directly save them in a list. After that, for the  $n^{\text{th}}$  (starting from 1) user in the whole sequence of users, you will keep the  $n^{\text{th}}$  user with the probability of  $100/n$ , otherwise discard it. If you keep the  $n^{\text{th}}$  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.

**Every time you receive a new user, you need to find the user in the sample list with the top 3 frequencies.**

#### Output Results: you just need to print your results in the terminal

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

For example, after receiving the 100<sup>th</sup> user from the streaming, calculate whether the reservoir will keep it and replace 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 1<sup>th</sup> user (with index 0 in your list), 21<sup>th</sup> user, 41<sup>th</sup> user, 61<sup>th</sup> user and 81<sup>th</sup> 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,tXQTGAJYZVRzMypFn0hqJw,31wZQG1nKIVv4bxuTJ8CA,4oYEAJz9_eg3Y8TiJZLc2g,5BIOXJ_vd6uKngsYH0-oOA
```

Figure 5: streaming printing information example

**Please use integer 553 as the random seed**, and use 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 number is different from our simulation, there will be 50% penalty

## 4.4 Execution Format

### Python:

```
spark-submit task1.py <port #> <output_filename>
```

```
spark-submit task2.py <port #> <output_filename>
```

```
spark-submit task3.py <port #> <output_filename>
```

### Scala:

```
spark-submit --class task1 hw5.jar <port #> <output_file_path>
```

```
spark-submit --class task2 hw5.jar <port #> <output_file_path>
```

```
spark-submit --class task3 hw5.jar <port #> <output_file_path>
```

Input parameters:

1. <port #>: the simulated streaming port your listen to.
2. <output\_filename>: the output file including file path, file name, and extension.

## 5. Submission

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

- a. Three Python scripts:
  - task1.py
  - task2.py
  - task3.py
- b. [OPTIONAL] Three Scala scripts and a jar:
  - task1.scala
  - task2.scala
  - task3.scala
  - hw5.jar

Because this homework is about online algorithms, Vocareum will only accept the submission without running the simple test case. Please double check the input/output format and the execution carefully before submitting.

### Important:

1. If we can't call myhashs(s) in your script to get the hash value list, there will be 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, and use 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 number is different from our simulation, there will be 50% penalty.

## 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 10% bonus if you use both Scala and Python.
3. If we cannot run your programs with the command we specified, there will be 80% penalty.
4. If we can't call myhashs(s) in your script to get the hash value list, there will be 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 number is different from our simulation, there will be 50% penalty.
7. If your program cannot run with the required Scala/Python/Spark versions, there will be 20% penalty.
8. If the outputs of your program are unsorted or partially sorted, there will be 50% penalty.
9. We can regrade on your assignments within seven days once the scores are released. No argue after one week. There will be 20% penalty if our grading is correct.
10. There will be 20% penalty for late submission within a week and no point after a week.
11. Only when your results from Python are correct, the bonus of using Scala will be calculated. There is no partially point for Scala. See the example below:

Example situations

Task	Score for Python	Score for Scala (10% of previous column if correct)	Total
Task1	Correct: 3 points	Correct: 3 * 10%	3.3
Task1	Wrong: 0 point	Correct: 0 * 10%	0.0
Task1	Partially correct: 1.5 points	Correct: 1.5 * 10%	1.65
Task1	Partially correct: 1.5 points	Wrong: 0	1.5