DSCI553 Foundations and Applications of Data Mining

Fall 2020

Assignment 5

Deadline: Nov. 17th 11:59 PM PST

NO LATE SUBMISSIONS

1. Overview of the Assignment

In this assignment, you are going to implement three algorithms: the Bloom filtering, Flajolet-Martin algorithm, and reservoir sampling. For the first task, you will implement **Bloom Filtering** for off-line Yelp business dataset. The "off-line" here means you do not need to take the input as streaming data. For the second and the third task, you need to deal with on-line streaming data directly. In the second task, you need to generate a *simulated* data stream with the Yelp dataset and implement **Flajolet-Martin** algorithm with Spark Streaming library. In the third task, you will do some analysis on Twitter stream using fixed size sampling **(Reservoir Sampling)**.

2. Requirements

2.1 Programming Requirements

a. **You must use Python and Spark to implement all tasks.** 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** for task1 and task2. In task3, you will use Twitter API of streaming. You can use the Python library, **tweepy**, and Scala library, **spark-streaming-twitter**.

c. You can only use Spark RDD and standard Python or Scala libraries. I.e. no point if using Spark DataFrame or DataSet

2.2 Programming Environment

Python 3.6, Scala 2.11 and Spark 2.3.2

We will use Vocareum to automatically run and grade your submission. You must test your scripts on **the local machine** and **the Vocareum terminal** before submission.

2.3 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.

2.4 What you need to turn in

You need to submit the following files on Vocareum: (all lowercase)

a. [REQUIRED] three Python scripts, named: **task1.py, task2.py, task3.py**
b. [REQUIRED FOR SCALA] three Scala scripts, named: **task1.scala, task2.scala, task3.scala**
c. [REQUIRED FOR SCALA] one jar package, named: **hw5.jar**
d. You don't need to include your results.

3. Datasets

3.1.1 Yelp Business Data

For task1, you need to download business_first.json and business_second.json from Vocareum. The first file is used to set up the bit array for Bloom filtering, and the second file is used for prediction.

3.1.2 Yelp Streaming Data Simulation

For task2, you need to download the business.json file and the generate_stream.jar on the Vocareum. 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 "business.json" with the command:

   java -cp <generate_stream.jar file path> StreamSimulation <business.json 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.
   - You can add " > /dev/null" at the end of command to suppress the output

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)

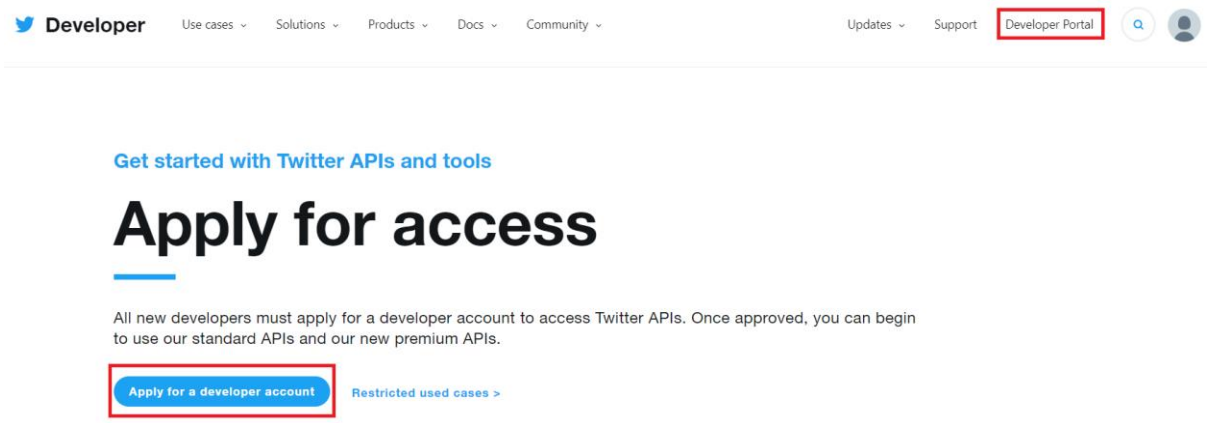
   - The first argument is the host name, which is "localhost" in this case.
   - The second argument is the port number in step 1), which is 9999 in this case.

3.2 Twitter Stream Data

For task3, you need to analyze the twitter streaming data using Twitter APIs. Please follow the instruction to set up Twitter APIs.

a. Create credentials for Twitter APIs

Register on https://developer.twitter.com/en/apply-for-access
- by clicking on "Apply for a developer account" and then fill the form.

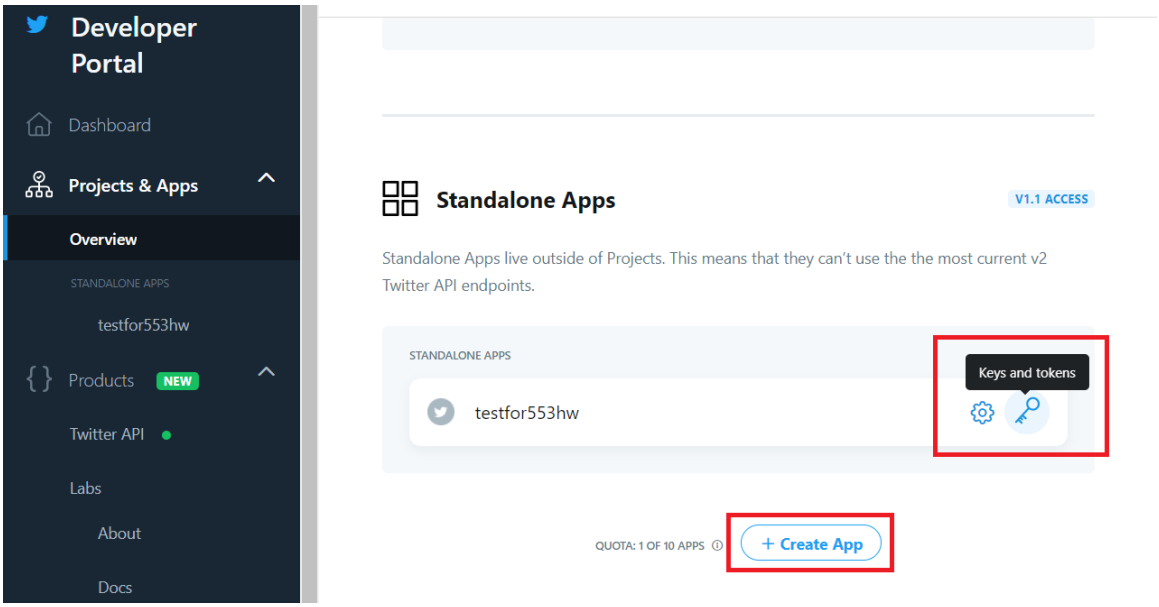


-

- **Apply it ASAP** because it takes time for Twitter to review and approve your application.

  (Tips: use your USC email account, fill the form in detail otherwise Twitter request more information)

- After approval, go to the "Developer Portal" and create App, and then you can generate and access keys and tokens. You will need them for task3.



b. Add library dependencies in the code

- You can use Python library, **tweepy**. To install the library on your laptop, you can use "pip install tweepy". (Tweepy is already installed on Vocareum.)
- You can use Scala libraries, **spark-streaming-twitter and spark-streaming**. To install the libraries, you can add the library dependencies in the sbt.

http://docs.tweepy.org/en/3.7.0/streaming_how_to.html

http://bahir.apache.org/docs/spark/current/spark-streaming-twitter/

4. Tasks

4.1 Task1: Bloom Filtering (3 pts)

You will implement the Bloom Filtering algorithm to estimate whether the name of a coming business 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.

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 name of a business 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 names. You do not need to consider alias.)

Execution Details

For task1, you need to download the business_first.json and business_second.json from Vocareum. The first input file is used to set up the bit array for Bloom filtering, and the second input file is used for prediction. More specifically, you don't need to use Spark for generating predictions. Instead, you can just iterate through each record in the second input file and apply Bloom Filtering algorithm to estimate whether the coming name appeared in the first input file.

While iterating through the second input file, you need to build an array of predictions, e.g., "predictions = []". For a given record, if your Bloom Filter predicts that the name appeared before, append "T" to the predictions array (e.g., predictions.append("T")). Otherwise, append "F".

Your code should finish within **60 seconds**. We will evaluate on the false positive rate (FPR). And the false negative rate (FNR). For the definition of FPR and FNR, you might find this helpful: https://en.wikipedia.org/wiki/Sensitivity_and_specificity.

Output Results

You need to save your generated predictions array to the output file we will provide. The output file does not need to have any headers or newlines. Simply generate whitespace-delimited string from your predictions array and write it to the file. For example, if your predictions array is ["T", "T", "F", "T"], your output file should contain "T T F T". The total number of F/Ts should be the same as the number of rows in the second input file.

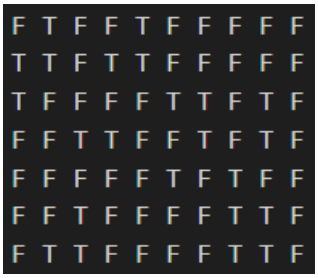


Figure 1: Output file format for task1

4.2 Task2: Flajolet-Martin algorithm (3.6 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 states 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.

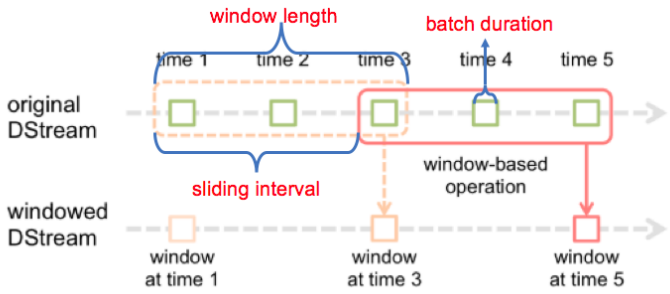


Figure 2: Spark Streaming window

Execution Details

In Spark Streaming, set the batch duration to **5** seconds:

```
ssc=StreamingContext(sc , 5)
```

You will get a batch of data in spark streaming every 5 seconds. You should also set the window length to 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,Gound Truth,Estimation". Each line stores the timestamp when you receive the batch of data, the actual number of unique states 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 3 shows an example). You do not need to round your answer.

```
Time,Gound Truth,Estimation
2020-10-27 12:37:17,10,10
2020-10-27 12:37:28,11,11
2020-10-27 12:37:40,11,11
2020-10-27 12:37:51,12,12
2020-10-27 12:38:01,12,12
2020-10-27 12:38:11,12,12
2020-10-27 12:38:21,10,10
2020-10-27 12:38:31,10,10
2020-10-27 12:38:41,10,10
```

Figure 3: Flajolet-Martin output file format

4.3 Task3: Fixed Size Sampling on Twitter Streaming (3pts)

You will use Twitter API of streaming to implement the fixed size sampling method (Reservoir Sampling Algorithm) and find popular tags on tweets based on the samples.

In this task, we assume that the memory can only **save 100 tags**, so we need to use the fixed size sampling method to only keep a subset of tags as a sample in the streaming. When the streaming of the Twitter coming, for the first 100 tags, you can directly save them in a list. After that, for the n^{th} tag, you will keep the n^{th} tag with the probability of 100/n, otherwise discard it. If you keep the n^{th} tag, you need to randomly pick one in the list to be replaced. Keep in mind that a tweet can have multiple tags. If that's the case, you should process each tag independently.

You also need to keep a global variable representing the sequence number of the tweet. If the coming tweet has no tag, the sequence number will not increase, else the sequence number increases by one.

Every time you receive a new tweet (so, every time you increment the sequence number), you need to find the tags in the sample list with the top 3 frequencies.

Output Results:

You need to save your results in a CSV file. Whenever a new tweet arrives, you append the following information block to the file: In the first line, you should print the sequence number of this new tweet as shown in the example. Then, you should print the tags and frequencies in the descending order of frequency. Each block is separated by an empty line. If some tags share the same frequency, you should print them all and ordered in lexicographic order (Figure 4).

During submission period, we will test your code for 50 seconds. During grading period, we will test your code for 10 minutes. You should test your code on the local machine to make sure it runs smoothly as long as there is data stream comes in.

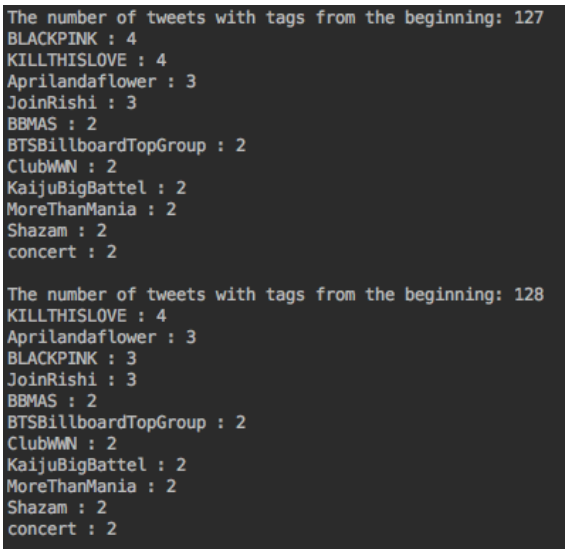


Figure 4: Output file format for task3

4.4 Execution Format

Python:

spark-submit task1.py <first_json_path> <second_json_path> <output_ filename >

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

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

Scala:

spark-submit –class task1 hw5.jar <first_json_path> <second_json_path> <output_ filename >

spark-submit –class task2 hw5.jar <port #> <output_ filename >

spark-submit –class task3 hw5.jar <port #> <output_filename>

Input parameters:

1. <port #>: the simulated streaming port your listen to.

2. < output_ filename >: the output file including file path, file name, and extension.

Vocareum:

export PYSPARK_PYTHON=python3.6

spark-submit task1.py $ASNLIB/publicdata/business_first.json $ASNLIB/publicdata/business_second.json task1_res

java -cp $ASNLIB/publicdata/generate_stream.jar StreamSimulation $ASNLIB/publicdata/business.json 9999 100

spark-submit 9999 task2_res

spark-submit task3.py 9999 task3_res

spark-submit --class task1 hw5.jar $ASNLIB/publicdata/business_first.json $ASNLIB/publicdata/business_second.json task1_scala_res

java -cp $ASNLIB/publicdata/generate_stream.jar StreamSimulation $ASNLIB/publicdata/business.json 9999 100

spark-submit --class task2 hw5.jar 9999 task2_scala_res

spark-submit --class task3 hw5.jar 9999 task3_scala_res

Note:

It's OK to have the following error in your submission log:

```
Exception in thread "receiver-supervisor-future-0" java.lang.Error: java.lang.InterruptedException: sleep interrupted
    at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1155)
    at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
    at java.lang.Thread.run(Thread.java:748)
Caused by: java.lang.InterruptedException: sleep interrupted
    at java.lang.Thread.sleep(Native Method)
    at org.apache.spark.streaming.receiver.ReceiverSupervisor$$anonfun$restartReceiver$1.apply$mcV$sp(ReceiverSupervisor.scala:196)
    at org.apache.spark.streaming.receiver.ReceiverSupervisor$$anonfun$restartReceiver$1.apply(ReceiverSupervisor.scala:189)
    at org.apache.spark.streaming.receiver.ReceiverSupervisor$$anonfun$restartReceiver$1.apply(ReceiverSupervisor.scala:189)
    at scala.concurrent.impl.Future$PromiseCompletingRunnable.liftedTree1$1(Future.scala:24)
    at scala.concurrent.impl.Future$PromiseCompletingRunnable.run(Future.scala:24)
    at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149)
    ... 2 more
```

5. Grading Criteria

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

a. There will be 10% bonus for each task if your Scala implementations are correct. Only when your Python results are correct, the bonus of Scala will be calculated. There is no partial point for Scala.

b. There will be no point if your submission cannot be executed on Vocareum.

c. There is no regrading. Once the grade is posted on the Vocareum, we will only regrade your assignments if there is a grading error. No exceptions.

d. No late submissions allowed.