INF553 Foundations and Applications of Data Mining

Spring 2019

Assignment 6

Extended Deadline: Apr. 27th 11:59 PM PST

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 task2. You will use Twitter streaming API streaming for task3: you can use the Python library, **tweepy**, and Scala library, **spark-streaming-twitter** for this task.

c. You can only use Spark RDD and standard Python or Scala libraries except for the ones in (b). 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 should 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: **hw6.jar**

d. You don't need to include your results. We will grade on your code with our testing data (data will be in the same format).

3. Datasets

For grading, we will use **different files** that have the same format as the ones you use during submission period.

3.1 Yelp Business Data

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

3.2 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.

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. Feel free to find other tutorials online for setting up the credentials and adding library dependency as well.

a. Create credentials for Twitter APIs

   - Register on https://apps.twitter.com/ by clicking on "Create new app" and then fill the form click on "Create your Twitter app."

- Go to the newly created app and open the “Keys and Access Tokens”. Click on “Generate my access token.” You will need to use these tokens as arguments when executing the code.

b. Add library dependencies in the code

- You should use Python library, **tweepy**. To install the library, you can use “pip install tweepy” on your local machine and use “pip3.6 install --user tweepy” 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. (See instructions below) Remember to include the libraries to your output jar when building artifacts.

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 (2.5 pts)

You will implement the Bloom Filtering algorithm to estimate whether the city of a business in business_second.json has shown before in business_first.json. 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** with **length 200**.

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.

Since the city 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 cities. You do not need to consider alias.)

Execution Details

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 prediction results in a **CSV** file in one single line. “1” means the city is predicted to have appeared in business_first.json before and “0” means the city is predicted to have not appear in business_first.json. Each number in “0”s and “1”s need to separated by a space. The total number of 0/1s should be the same as the number of rows in business_second.json.

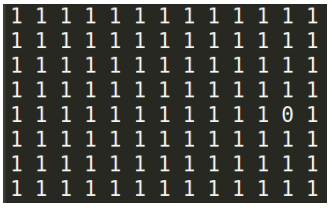

Figure 1: Output file format for task1

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 cities 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 proper number of hash functions in the Flajolet-Martin algorithm.

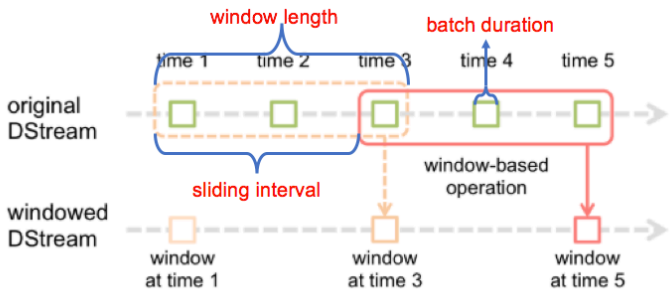

Figure 2: Spark Streaming window

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. The window length should be **30** seconds and the sliding interval should be **10** seconds.

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.

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 cities 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).

```
Time,Ground Truth,Estimation
2020-04-12 20:23:40,20,24
2020-04-12 20:23:50,43,34
2020-04-12 20:24:00,63,40
2020-04-12 20:24:10,72,62
2020-04-12 20:24:20,78,73
2020-04-12 20:24:30,80,73
2020-04-12 20:24:40,80,115
```

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 tweets**, so we need to use the fixed size sampling method to only keep part of the tweets as a sample in the streaming. When the streaming of the Twitter coming, for the first 100 tweets, you can directly save them in a list. After that, for the n^{th} twitter, you will keep the n^{th} tweet with the probability of 100/n, otherwise discard it. If you keep the n^{th} tweet, you need to randomly pick one in the list to be replaced. If the coming tweet has no tag, you can directly ignore it.

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, otherwise the sequence number increases by one.

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

Ouput Results:

You need to save your results in a **CSV** file. Whenever a new tweet arrives, you print the following information block: 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).

Block 1

```
The number of tweets with tags from the beginning: 127
BLACKPINK : 4
KILLTHISLOVE : 4
Aprilandaflower : 3
JoinRishi : 3
BBMAS : 2
BTSBillboardTopGroup : 2
ClubWWN : 2
KaijuBigBattel : 2
MoreThanMania : 2
Shazam : 2
concert : 2

The number of tweets with tags from the beginning: 128
KILLTHISLOVE : 4
Aprilandaflower : 3
BLACKPINK : 3
JoinRishi : 3
BBMAS : 2
BTSBillboardTopGroup : 2
ClubWWN : 2
KaijuBigBattel : 2
MoreThanMania : 2
Shazam : 2
concert : 2
```

Block 2

…

Figure 4: Output file format for task3

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.

4.4 Execution Format

Python:

spark-submit task1.py <first_json_path> <second_json_path> <output_file_path>

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

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

Scala:

spark-submit –class task1 hw6.jar <first_json_path> <second_json_path> <output_file_path>

spark-submit –class task2 hw6.jar <port #> <output_file_path>

spark-submit –class task3 hw6.jar <port #> <output_filename>

Input parameters:

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

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. You can use your **free 8-day** extension separately or together. You must submit a late-day request via https://forms.gle/worKTbCRBWKQ6jqu6. This form is recording the number of late days you use for each assignment. By default, we will not count the late days if no request submitted.

b. 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.

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

d. 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.

e. There will be **15% penalty** for the late submission within one week and no point after that.