INF553 Foundations and Applications of Data Mining

Fall 2019

Assignment 5

Deadline: Dec. 5th 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 **Bloom Filtering** and **Flajolet-Martin** algorithm with Spark Steaming. In the third task, you will do some analysis on Twitter stream using **Fixed Size Sample** (Reservoir Sampling).

2. Requirements

2.1 Programming Requirements

a. **You must use Python 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.

2.2 Programming Environment

Python 3.6, Scala 2.11 and Spark 2.3.3

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 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

Your submission must be a **zip file** with the name convention: **firstname_lastname_hw5.zip** (all lowercase, e.g., xin_yu_hw5.zip). You should pack the following required (and optional) files in a folder named **firstname_lastname_hw5** (all lowercase, e.g., xin_yu_hw5) in the zip file (see Figure 1):

a. [REQUIRED] three Python scripts, named: (all lowercase)

firstname_lastname_task1.py
firstname_lastname_task2.py
firstname_lastname_task3.py

b. [OPTIONAL] three Scala scripts, named: (all lowercase)

firstname_lastname_task1.scala
firstname_lastname_task2.scala
firstname_lastname_task3.scala

c. [OPTIONAL] one jar package, named: (all lowercase)

firstname_lastname_hw5.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).

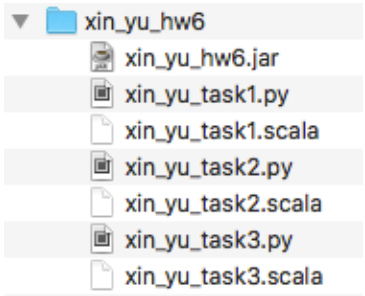

Figure 1: The folder structure after your submission file is unzipped.

3. Datasets

3.1 Yelp Streaming Data Simulation

For task1 and task2, you need to download the yelp.json file and the 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 "yelp.json" with the command:

java -cp < stream.jar file path> StreamSimulation <yelp.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.

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 can use Python library, **tweepy**. To install the library, you can use "pip install tweepy".
- 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 (2.5 pts)

You will implement the Bloom Filtering algorithm to estimate whether the **US state** 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.

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

As the state of a business is a string convert the state 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 states. 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 data in spark streaming every 10 seconds and you will use the Bloom Filtering algorithm to estimate whether the coming state appeared before or not since the beginning of your code.

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

We will test your code for 10 minutes.

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. 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

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 number of hash functions in the Flajolet-Martin algorithm.

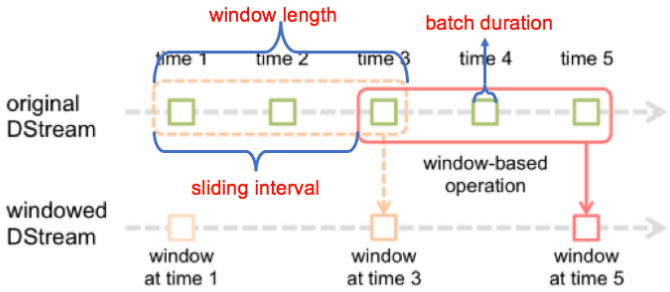

Figure 3: Spark Streaming window

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,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 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

4.3 Task3: Fixed Size Sampling on Twitter Streaming (2pts)

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 150 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 150 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 150/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, else the sequence number increases one.

Every time you receive a new tweet, you need to find the tags in the sample list with the top 5 frequencies.

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

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. If some tags share the same frequency, you should print them **all** and ordered in lexicographic order (Figure 5).

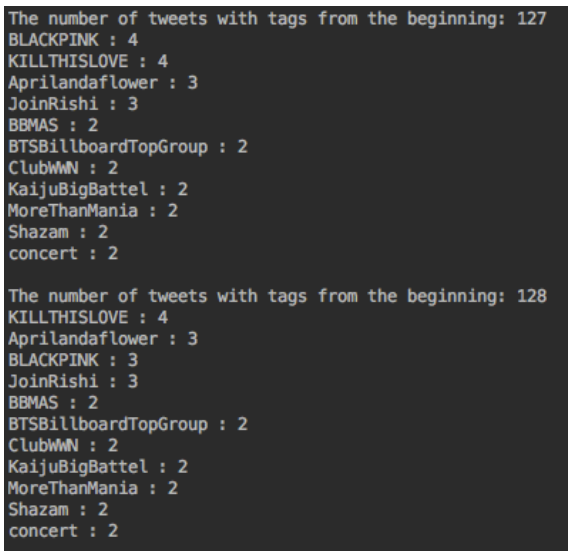

Figure 5: Twitter streaming printing information example

4.4 Execution Format

Python:

spark-submit firstname_lastname_task1.py <port #> <output_filename>

spark-submit firstname_lastname_task2.py <port #> <output_filename>

spark-submit firstname_lastname_task3.py

Scala:

spark-submit –class firstname_lastname_task1 firstname_lastname_hw5.jar <port #> <output_file_path>

spark-submit –class firstname_lastname_task2 firstname_lastname_hw5.jar <port #> <output_file_path>

spark-submit –class firstname_lastname_task3 firstname_lastname_hw5.jar

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. 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 your program cannot run with the required Scala/Python/Spark versions, there will be 20% penalty.

5. If the outputs of your program are unsorted or partially sorted, there will be 50% penalty.
6. 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.
7. There will be 20% penalty for late submission within a week and no point after a week.
8. 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