**Spark and Streaming**

**1. Spark Matrix Multiply:**

Reimplement matrix multiply in spark. Your implementation must use a map transformation as well as either reduceByKey or groupByKey. Data will be of the same format as Task 1.C: rdd = sc.parallelize(createSparseMatrix(..., “A...”) + createSparseMatrix(B, “B...”)). Please place this code in a new python file along with II.B, and have your code test on the same test matrices as those already provided for I.C.

**2. Umbler’s Middle-Out Filter:**

You’ve been hired by Umbler, the latest big shot startup company. Umbler tracks mentions of nonfluencies like “um”, “oh”, “sigh” and counts the amount of unique 3-word phrases that appear immediately after these nonfluencies in social media (e.g. “Ohh. SBU CS rocks!” will count as “sbu cs rocks!” occurs for the nonfluency group “oh”). They run this only for posts originating from a large set of locations. Unfortunately, they have recently started having memory issues, not being able to store all of the unique phrases that appear after fluencies. They are so frustrated and worried about memory that, not only do they not want to store the 3-word phrases, but they also do not want to store the long list of locations.

This is where you come in: they heard you were an all stat data scientist because of your SBU degree and hope you can help them figure out how to fix this issue so they can keep tracking all of the phrases that appear after nonfluencies, originating from predefined locations. Specifically, they need a streaming solution for records that arrive in the following form:

(<location>, <post>)

(hint: “middle-out” is simply a reference to the show Silicon Valley and has no real meaning here. Instead, the Bloom filter and Flajolet-Martin algorithms may be helpful.)

There are 4 classes of nonfluencies to track (+ indicates match repeats of last letter).

'MM':[r'mm+'],

'OH': [r'oh+', r'ah+'],

'SIGH': [r'sigh', r'sighed', r'sighing', r'sighs', r'ugh', r'uh'],

'UM': [r'um+', r'hm+', r'huh']

Note: “+” means one or more of the last letter.

All non-fluencies can end in a punctuation (. , ! ?)

The algorithm, per record, generally takes the form: (1) check if contains nonfluency anywhere in post (can use memory for nonfluency; allow punctuation at the end of the nonfluency; can do this before or after 2), (2) check if location is valid for tracking (can’t keep all locations in memory), (3) increment the count of unique 3-word phrases if unique. The list of valid locations is given the file: umbler\_locations.csv (even though it is titled a csv, treat each line as if a compleat string to match; e.g. “New York, NY” should just be one location).

Location matching should be case insensitive.

Definition of word: For the purposes of this assignment a “word” will be anything that results from string.split(‘ ‘). Sample data is available here: umblerData.zip

Spark Restriction: You must utilize the streaming style algorithms within Spark RDDs (not necessarily streaming data). The data (i.e. (location, post) tuples in any form) should remain in RDDs until the final stage after aggregating to distinct phrase counts. Once the rdd begins being processed, you must not store the list of locations or a list of phrases seen thus far (or keep them in a broadcast variable). Rather you should use a smaller structure within broadcast variables or find another way to keep them in RDDs Please use the template code mentioned below. Extra credit may be given for implementing an additional Spark streaming context version, but the intended version just uses regular Spark RDDs (i.e. initialized from stable storage with sc.textfile(...)).