**MIS 586 – Assignment 4**

**Name**: Surbhi Chaturvedi **Date**: 4/1/2020

**Answer 1:**

1. Please find the screenshot of the code below:

A screenshot of a social media post

Description automatically generated

Setting the stream time to 600s

A screenshot of a social media post

Description automatically generated

Calculating the hashtag count and sorting it in the descending order below:

A screenshot of a cell phone

Description automatically generated

A screenshot of a social media post

Description automatically generated

A screenshot of a social media post

Description automatically generated

Streaming with the batch interval of 60 secs:

A screenshot of a cell phone

Description automatically generated

Saving the results to BigQuery in form of two tables : hashtags and wordcount.

A screenshot of a cell phone

Description automatically generated

1. Screenshots of the output and the tables created in BigQuery are listed below.

A screenshot of a social media post

Description automatically generated

A screenshot of a social media post

Description automatically generated

A screenshot of a social media post

Description automatically generated

A screenshot of a social media post

Description automatically generatedA screenshot of a social media post

Description automatically generated

**Answer 2:**

**<Source Code>**

"""

This module is the spark streaming analysis process.

Usage:

If used with dataproc:

gcloud dataproc jobs submit pyspark --cluster <Cluster Name> twitterHTTPClient.py

Create a dataset in BigQurey first using

bq mk bigdata\_sparkStreaming

Remember to replace the bucket with your own bucket name

Todo:

1. hashtagCount: calculate accumulated hashtags count

2. wordCount: calculate word count every 60 seconds

the word you should track is listed below.

3. save the result to google BigQuery

"""

import subprocess

import time

from pyspark import SparkConf, SparkContext

from pyspark.sql import SQLContext

from pyspark.streaming import StreamingContext

# global variables

bucket = 'surbhi-mis586-asgnmt4' # TODO: here, replace with your own bucket name

output\_directory\_hashtags = 'gs://{}/hadoop/tmp/bigquery/pyspark\_output/hashtagsCount'.format(

bucket)

output\_directory\_wordcount = 'gs://{}/hadoop/tmp/bigquery/pyspark\_output/wordcount'.format(

bucket)

# output table and columns name

output\_dataset = 'dataset1' #TODO: the name of your dataset in BigQuery

# (Create a BigQuery dataset first using bq mk <your dataset name> in Google Cloud SDK Shell or any alternative approaches.)

output\_table\_hashtags = 'hashtags'

columns\_name\_hashtags = ['hashtags', 'count']

output\_table\_wordcount = 'wordcount'

columns\_name\_wordcount = ['word', 'count', 'time']

# parameter

IP = 'localhost' # ip port

PORT = 9001 # port

STREAMTIME = 600 # time that the streaming process runs

#STREAMTIME = 20 # for test

WORD = ['data', 'spark', 'ai', 'movie', 'coronavirus'] # the words you should filter and do word count

# Helper functions

def saveToStorage(rdd, output\_directory, columns\_name, mode):

"""

Save each RDD in this DStream to google storage

Args:

rdd: input rdd

output\_directory: output directory in google storage

columns\_name: columns name of dataframe

mode: mode = "overwirte", overwirte the file

mode = "append", append data to the end of file

"""

if not rdd.isEmpty():

(rdd.toDF(columns\_name)

.write.save(output\_directory, format="json", mode=mode))

def saveToBigQuery(sc, output\_dataset, output\_table, directory):

"""

Put temp streaming json files in google storage to google BigQuery

and clean the output files in google storage

"""

files = directory + '/part-\*'

subprocess.check\_call(

'bq load --source\_format NEWLINE\_DELIMITED\_JSON '

'--replace '

'--autodetect '

'{dataset}.{table} {files}'.format(

dataset=output\_dataset, table=output\_table, files=files

).split())

output\_path = sc.\_jvm.org.apache.hadoop.fs.Path(directory)

output\_path.getFileSystem(sc.\_jsc.hadoopConfiguration()).delete(

output\_path, True)

def hashtagCount(words):

"""

Calculate the accumulated hashtags count sum from the beginning of the stream

and sort it by descending order of the count.

Ignore case sensitivity when counting the hashtags:

"#Ab" and "#ab" are considered to be a same hashtag

You have to:

1. Filter out the word that is hashtags.

Hashtag usually start with "#" and followed by a serious of alphanumeric

2. map (hashtag) to (hashtag, 1)

3. sum the count of current DStream state and previous state

4. transform unordered DStream to a ordered Dstream

Hints:

You may use regular expression to filter the words

You can take a look at updateStateByKey and transform transformations

Args:

dstream(DStream): stream of real time tweets

Returns:

DStream Object with inner structure (hashtag, count)

"""

# TODO: insert your code here

# 0. Define a function called "updateFunc" compting running sum of hashtag counts:

def updateFunc(new\_values, last\_sum):

return sum(new\_values) + (last\_sum or 0)

# 1. Define hashtag using the following techniques:

# 1.1. "#Ab" and "#ab" are the same: hashtag = words.map(lambda x: x.lower())

# 1.2. Filter out hashtags: .filter(lambda x: len(x) > 2 and x[0] == "#")

# 1.3. map (hashtag) to (hashtag, 1): .map(lambda x: (x, 1))

hashtag = words.map(lambda x: x.lower()).filter(lambda x: len(x) > 2 and x[0] == "#").map(lambda x: (x, 1))

# 2. Define hashtag count using reduceByKey

hashtag\_cnt = hashtag.reduceByKey(lambda cnt1, cnt2: cnt1 + cnt2)

# 3. Update total hashtag count using updateStateByKey and the updateFunc defined above:

hashtag\_cnt\_total = hashtag\_cnt.updateStateByKey(updateFunc)

# 4. Sort the hashtag counts using transform transformation:

# .transform(lambda rdd: rdd.sortBy(lambda x: x[1], ascending=False))

hashtag\_cnt\_total = hashtag\_cnt.updateStateByKey(updateFunc).transform(

lambda rdd: rdd.sortBy(lambda x: x[1], ascending=False))

# 5. Return total hashtag count:

return hashtag\_cnt\_total

def wordCount(words):

"""

Calculte the count of 5 sepcial words for every 60 seconds (window no overlap)

You can choose your own words.

Your should:

1. filter the words

2. count the word during a special window size

3. add a time related mark to the output of each window, ex: a datetime type

Hints:

You can take a look at reduceByKeyAndWindow transformation

Dstream is a serious of rdd, each RDD in a DStream contains data from a certain interval

You may want to take a look of transform transformation of DStream when trying to add a time

Args:

dstream(DStream): stream of real time tweets

Returns:

DStream Object with inner structure (word, count, time)

"""

# TODO: insert your code here

# 1. Define word\_cnt using the following techniques:

# 1.1. "#Ab" and "#ab" are the same: word\_cnt = words.map(lambda x: x.lower())

# 1.2. Filter the words we want: .filter(lambda word: word in WORD)

# 1.3. map (word) to (word, 1): .map(lambda x: (x, 1))

# 1.4. Use reduceByKeyAndWindow transformation to count the word during a special window size:

# .reduceByKeyAndWindow(lambda a, b: a + b, lambda a, b: a - b, 60, 60)

word\_cnt = words.map(lambda x: x.lower()).filter(lambda word: word in WORD).map(

lambda x: (x, 1)).reduceByKeyAndWindow(lambda a, b: a + b, lambda a, b: a - b, 60, 60)

# 2. Use transform and map transformations to update total word count by adding

# date and time related mark to the output:

word\_cnt\_total = word\_cnt.transform(lambda time, rdd: rdd.map(

lambda x: (x[0], x[1], time.strftime("%Y-%m-%d %H:%M:%S"))))

return word\_cnt\_total

if \_\_name\_\_ == '\_\_main\_\_':

# Spark settings

conf = SparkConf()

conf.setMaster('local[2]')

conf.setAppName("TwitterStreamApp")

# create spark context with the above configuration

sc = SparkContext(conf=conf)

sc.setLogLevel("ERROR")

# create sql context, used for saving rdd

sql\_context = SQLContext(sc)

# create the Streaming Context from the above spark context with batch interval size 60 seconds

ssc = StreamingContext(sc, 60)

#ssc = StreamingContext(sc, 5)

# setting a checkpoint to allow RDD recovery

ssc.checkpoint("~/checkpoint\_TwitterApp")

# read data from port 9001

dataStream = ssc.socketTextStream(IP, PORT)

dataStream.pprint()

words = dataStream.flatMap(lambda line: line.split(" "))

# calculate the accumulated hashtags count sum from the beginning of the stream

topTags = hashtagCount(words)

topTags.pprint()

# calculte the word count during each time period 6s

wordCount = wordCount(words)

wordCount.pprint()

"""

save hashtags count and word count to google storage

used to save to google BigQuery

You should:

1. topTags: only save the lastest rdd in DStream

2. wordCount: save each rdd in DStream

Hints:

1. You can take a look at foreachRDD transformation

2. You may want to use helper function saveToStorage

3. You should use save output to output\_directory\_hashtags, output\_directory\_wordcount,

and have output columns name columns\_name\_hashtags and columns\_name\_wordcount.

"""

# TODO: insert your code here

# 0. Use foreachRDD transformation and saveToStorage helper function to

# save hashtags count to google storage:

topTags.foreachRDD(lambda rdd: saveToStorage(rdd, output\_directory\_hashtags,

columns\_name\_hashtags, mode="overwrite"))

# 1. Use foreachRDD transformation and saveToStorage helper function to

# save word count to google storage:

wordCount.foreachRDD(lambda rdd: saveToStorage(rdd, output\_directory\_wordcount,

columns\_name\_wordcount, mode="append"))

# start streaming process, wait for 600s and then stop.

ssc.start()

time.sleep(STREAMTIME)

ssc.stop(stopSparkContext=False, stopGraceFully=True)

print("Running Finished.")

# put the temp result in google storage to google BigQuery

saveToBigQuery(sc, output\_dataset, output\_table\_hashtags, output\_directory\_hashtags)

saveToBigQuery(sc, output\_dataset, output\_table\_wordcount, output\_directory\_wordcount)

print("Saved To BigQuery Finished.")