

## Homework3

1.

### (1) code screenshots

```

from pyspark import SparkConf, SparkContext
from pyspark.streaming import StreamingContext
from pyspark.sql import Row, SQLContext
import sys
import requests
import subprocess
import re
from google.cloud import bigquery
import re
from pyspark.sql.functions import lit, unix_timestamp
import time
import datetime

# parameter
IP = 'localhost'      # ip port
PORT = 9001          # port
STREAMTIME = 600      # time that the streaming process runs
windowLength = 60     # window length for wordcount
slideInterval = 60    # slide interval for wordcount
timestamp = datetime.datetime.fromtimestamp(time.time()).strftime("%Y-%m-%d %H:%M:%S")

# global variables
bucket = "big_data_hw" # TODO : 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 = 'bigdata_sparkStreaming' #the name of your dataset in BigQuery
output_table_hashtags = 'hashtags'
columns_name_hashtags = ['hashtags', 'count']
output_table_wordcount = 'wordcount'
columns_name_wordcount = ['word', 'count', 'time']

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

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 saveToStorage_hash(rdd):
    """
    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 = "overwrite", overwrite the file
             mode = "append", append data to the end of file
    """
    if not rdd.isEmpty():
        rdd.toDF( columns_name_hashtags ) \
            .orderBy('count', ascending=False) \
            .write.save(output_directory_hashtags, format="json", mode="overwrite")

```

```

def saveToStorage_word(rdd):
    """
    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 = "overwrite", overwrite the file
            mode = "append", append data to the end of file
    """
    if not rdd.isEmpty():
        rdd.toDF( columns_name_wordcount ) \
            .orderBy(['word', 'time']) \
            .write.save(output_directory_wordcount, format="json", mode="append")

# helper function
def filterFunc(hashtags):
    if re.match("^[0-9a-z]+$", hashtags):
        return True
    else:
        return False

# helper function
def updateFunction(newValues, runningCount):
    if runningCount is None:
        runningCount = 0
    return sum(newValues, runningCount) # add the new values with the previous running count to get the new count

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" is 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 series 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)
    """

    hashtags = words \
        .map(lambda x: x.lower()) \
        .filter(filterFunc) \
        .map(lambda x: (x, 1)) \
        .updateStateByKey(updateFunction)

    return hashtags

```

```

def wordCount(words):
    """
    Calculate the count of 5 special words in 60 seconds 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 series 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)
    """

    res = words \
        .map(lambda x: x.lower()) \
        .filter(lambda x: x in WORD) \
        .map(lambda x: (x, 1)) \
        .reduceByKeyAndWindow(lambda x, y: x+y, lambda x, y: x-y, windowLength, slideInterval) \
        .transform(lambda timestamp, rdd: rdd.map(lambda x: (x[0], x[1], timestamp)))

    return res

```

```

if __name__ == '__main__':
    # Spark settings
    conf = SparkConf()
    conf.setMaster('local[2]')
    conf.setAppName("TwitterStreamApp")

    # create spark context with the above configuration
    sc = SparkContext.getOrCreate(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 5 seconds
    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)
    words = dataStream.flatMap(lambda line: line.split(" "))

    # calculate the accumulated hashtags count sum from the beginning of the stream
    topTags = hashtagCount(words)

    # Calculate the word count during each time period 60s
    wordCount = wordCount(words)

    topTags.foreachRDD(saveToStorage_hash)
    wordCount.foreachRDD(saveToStorage_word)

    # start streaming process, wait for 600s and then stop.
    ssc.start()
    time.sleep(STREAMTIME)
    ssc.stop(stopSparkContext=False, stopGraceFully=True)

    saveToBigQuery(sc, output_dataset, output_table_hashtags, output_directory_hashtags)
    saveToBigQuery(sc, output_dataset, output_table_wordcount, output_directory_wordcount)

```

## (2) BigQuery Preview

Hashtags:

test-project-251000

bigdata\_sparkStreaming

hashtags

wordcount

my\_dataset

Run

Save query

Save view

hashtags

Schema

Details

Preview

Row	count	hashtags
1	29	#1dayforsrksbday
2	25	#bigdata
3	24	#wonhocomeback
4	21	#iot
5	19	#artificialintelligence

Time: 2019-10-31 19:21:20

('ai', 144)

('bigil', 47)

('halloween', 35)

('machinelearning', 32)

('1dayforsrksbday', 29)

('bigdata', 25)

('wonhocomeback', 24)

('iot', 21)

('artificialintelligence', 19)

('spinel', 19)

...

## Wordcount:

test-project-251000

bigdata\_sparkStreaming

hashtags

wordcount

my\_dataset

Run

Save query

Save view

wordcount

Schema

Details

Preview

Row	time	count	word
1	2019-10-31 19:12:10 UTC	8	ai
2	2019-10-31 19:12:10 UTC	2	data
3	2019-10-31 19:12:10 UTC	9	good
4	2019-10-31 19:12:10 UTC	164	movie
5	2019-10-31 19:12:10 UTC	8	spark

Time: 2019-10-31 19:14:10

('ai', 14, datetime.datetime(2019, 10, 31, 19, 14, 10))

('movie', 158, datetime.datetime(2019, 10, 31, 19, 14, 10))

('good', 6, datetime.datetime(2019, 10, 31, 19, 14, 10))

('spark', 7, datetime.datetime(2019, 10, 31, 19, 14, 10))

('data', 1, datetime.datetime(2019, 10, 31, 19, 14, 10))