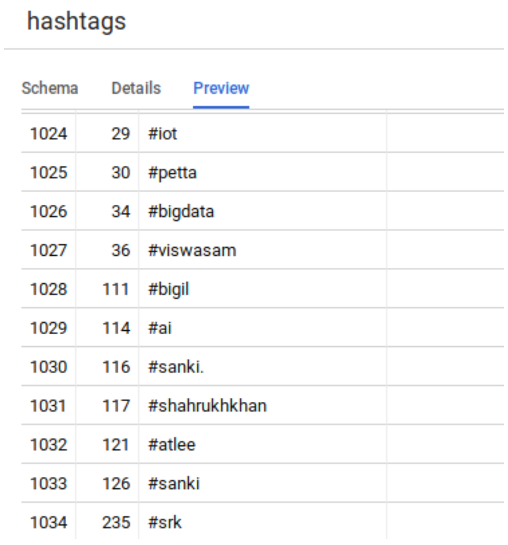
EECS E6893 Big Data Analytic HW3

Chong Hu ch3467

October 31, 2019

Problem 1. `Twitter data analysis with Spark Streaming`

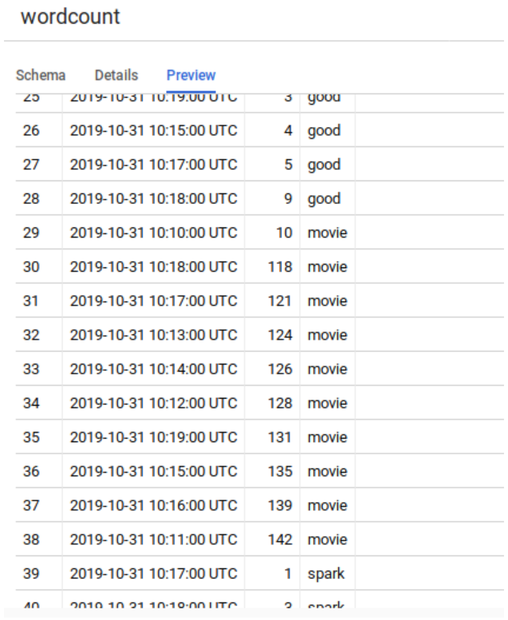
1. Hashtag result



1024	29	#iot
1025	30	#petta
1026	34	#bigdata
1027	36	#viswasam
1028	111	#bigil
1029	114	#ai
1030	116	#sanki.
1031	117	#shahrukhkhan
1032	121	#atlee
1033	126	#sanki
1034	235	#srk

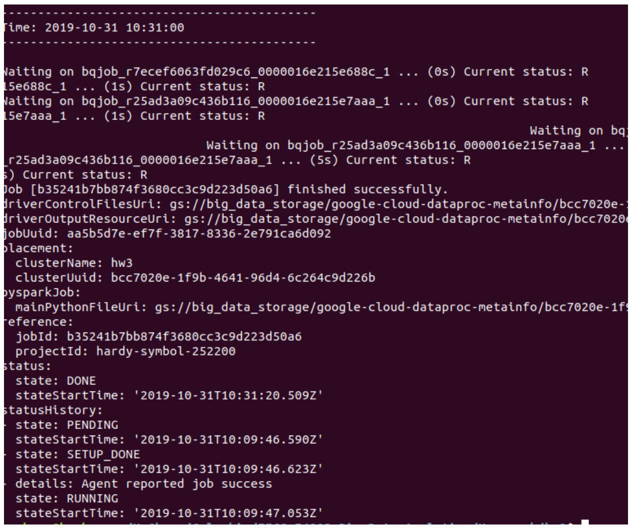
Figure 1: Hashtag result preview in Big Query

2. Word Count result



25	2019-10-31 10:19:00 UTC	3	good
26	2019-10-31 10:15:00 UTC	4	good
27	2019-10-31 10:17:00 UTC	5	good
28	2019-10-31 10:18:00 UTC	9	good
29	2019-10-31 10:10:00 UTC	10	movie
30	2019-10-31 10:18:00 UTC	118	movie
31	2019-10-31 10:17:00 UTC	121	movie
32	2019-10-31 10:13:00 UTC	124	movie
33	2019-10-31 10:14:00 UTC	126	movie
34	2019-10-31 10:12:00 UTC	128	movie
35	2019-10-31 10:19:00 UTC	131	movie
36	2019-10-31 10:15:00 UTC	135	movie
37	2019-10-31 10:16:00 UTC	139	movie
38	2019-10-31 10:11:00 UTC	142	movie
39	2019-10-31 10:17:00 UTC	1	spark
40	2019-10-31 10:18:00 UTC	3	spark

Figure 2: WordCount result preview in Big Query



```
-------------------------------------------
Time: 2019-10-31 10:31:00
-------------------------------------------

Waiting on bqjob_r7ecef6063fd029c6_0000016e215e688c_1 ... (0s) Current status: R
15e688c_1 ... (1s) Current status: R
Waiting on bqjob_r25ad3a09c436b116_0000016e215e7aaa_1 ... (0s) Current status: R
15e7aaa_1 ... (1s) Current status: R
                                                                     Waiting on bq
                               Waiting on bqjob_r25ad3a09c436b116_0000016e215e7aaa_1 ...
_r25ad3a09c436b116_0000016e215e7aaa_1 ... (5s) Current status: R
s) Current status: R
Job [b35241b7bb874f3680cc3c9d223d50a6] finished successfully.
driverControlFilesUri: gs://big_data_storage/google-cloud-dataproc-metainfo/bcc7020e-
driverOutputResourceUri: gs://big_data_storage/google-cloud-dataproc-metainfo/bcc7020
jobUuid: aa5b5d7e-ef7f-3817-8336-2e791ca6d092
placement:
  clusterName: hw3
  clusterUuid: bcc7020e-1f9b-4641-96d4-6c264c9d226b
pysparkJob:
  mainPythonFileUri: gs://big_data_storage/google-cloud-dataproc-metainfo/bcc7020e-1f
reference:
  jobId: b35241b7bb874f3680cc3c9d223d50a6
  projectId: hardy-symbol-252200
status:
  state: DONE
  stateStartTime: '2019-10-31T10:31:20.509Z'
statusHistory:
- state: PENDING
  stateStartTime: '2019-10-31T10:09:46.590Z'
- state: SETUP_DONE
  stateStartTime: '2019-10-31T10:09:46.623Z'
- details: Agent reported job success
  state: RUNNING
  stateStartTime: '2019-10-31T10:09:47.053Z'
```

Figure 3: Finish successfully in terminal

Code: `sparkStreaming.py`

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# Columbia EECS E6893 Big Data Analytics
"""
```

```python
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

    Remeber 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 = "big_data_storage"
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 = 'twitter_analysis'  # 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']

# parameter
IP = 'localhost'  # ip port
PORT = 9001  # port
```

```python

STREAMTIME = 600  # time that the streaming process runs
# STREAMTIME = 20  # for test

WORD = ['data', 'spark', 'ai', 'movie',
        'good']  # 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" 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 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)
    """

    def updateFunc(new_values, last_sum):
        return sum(new_values) + (last_sum or 0)

    hashtag = words.map(lambda x: x.lower()).filter(
        lambda x: len(x) > 2 and x[0] == "#").map(
        lambda x: (x, 1))
    hashtag_cnt = hashtag.reduceByKey(lambda cnt1, cnt2: cnt1 + cnt2)
    hashtag_cnt_total = hashtag_cnt.updateStateByKey(updateFunc)
    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)
    """
    word_cnt = 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, 60, 60)
    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)
    # 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.

    topTags.foreachRDD(lambda rdd: saveToStorage(rdd, output_directory_hashtags,
                                                 columns_name_hashtags,
                                                 mode="overwrite"))
    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)
    # 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)
```

twitterHTTPClient.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# Columbia EECS E6893 Big Data Analytics
"""
This module is used to pull data from twitter API and send data to
Spark Streaming process using socket. It acts like a client of
twitter API and a server of spark streaming. It open a listening TCP
server socket, and listen to any connection from TCP client. After
a connection established, it send streaming data to it.


Usage:
  If used with dataproc:
    gcloud dataproc jobs submit pyspark --cluster <Cluster Name> twitterHTTPClient.py

  Make sure that you run this module before you run spark streaming process.
  Please remember stop the job on dataproc if you no longer want to stream data.

```

```python
Todo:
  1. change the credentials to your own

"""

import json
import socket

from tweepy import OAuthHandler
from tweepy import Stream
from tweepy.streaming import StreamListener

# credentials
# replace with your own credentials
ACCESS_TOKEN = '1186641375900647425-Yk6Mr116gbZixYbOpt2bXGiODWZiZa'  # your access token
ACCESS_SECRET = '8xYHZbwKlF4zMdqHEWuwgO1vOocHuictsQnCn4ElPT1Ny'  # your access token secret
CONSUMER_KEY = 'Wg4xVrzEODU8Ey4sYPQvOntcv'  # your API key
CONSUMER_SECRET = 'FBXUfei2Sp7W7dfuLtK8wbY5BF2OKB4Ck5T5IGiE7AoEdWtQZW'  # your API secret key

# the tags to track
tags = ['#', 'bigdata', 'spark', 'ai', 'movie']


class TweetsListener(StreamListener):
    """
    tweets listener object
    """

    def __init__(self, csocket):
        super(TweetsListener, self).__init__()
        self.client_socket = csocket

    def on_data(self, data):
        try:
            msg = json.loads(data)
            print('TEXT:{}\n'.format(msg['text']))
            self.client_socket.send(msg['text'].encode('utf-8'))
            return True
        except BaseException as e:
            print("Error on_data: %s" % str(e))
            return False
        # return True

    def on_error(self, status):
        print(status)
        return False


def sendData(c_socket, tags):
    """
    send data to socket
    """
    auth = OAuthHandler(CONSUMER_KEY, CONSUMER_SECRET)
    auth.set_access_token(ACCESS_TOKEN, ACCESS_SECRET)
    twitter_stream = Stream(auth, TweetsListener(c_socket))
    twitter_stream.filter(track=tags, languages=['en'])


class twitter_client:
    def __init__(self, TCP_IP, TCP_PORT):
        self.s = s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.s.bind((TCP_IP, TCP_PORT))

    def run_client(self, tags):
        try:
            self.s.listen(1)
            while True:
                print("Waiting for TCP connection...")
                conn, addr = self.s.accept()
                print("Connected... Starting getting tweets.")
                sendData(conn, tags)
                conn.close()
        except KeyboardInterrupt:
            exit


if __name__ == '__main__':
    client = twitter_client("localhost", 9001)
    client.run_client(tags)
```