Finding Trending Twitter Hashtags using Spark

System set up:

1. **Spark and Python3** (You have them already)
2. **Tweepy**: A Python module for twitter API query
    a. Install:

```
pip install tweepy
```

Part 1: Create the stream of tweets in your local machine

Step 1:

Twitter has its own API that allows programmers/developers to query twitter's datastore. In order to do so, you have to first apply for a developer account from this url: https://developer.twitter.com/en/apply-for-access
Once you apply by filling up necessary information, it may take some time to have your own account.
Once you have your account, you have to create a new Application from this url: https://developer.twitter.com/en/portal/dashboard. You may follow the instructions of creating an app from here: https://smashballoon.com/doc/create-your-own-twitter-app/.

Each application will have its own API Key, API Key secret and Bearer Token. **You will need these codes to proceed with part 1 of this tutorial.**

Step 2

You need to create a client that will get the tweets from Twitter using Python and passes them to the Spark Streaming instance. In java this part is taken care of by **TwitterUtils** class provided by spark. In python, we wrote the client for you.

You need to copy the following code in a file named **receive_tweets.py** and then run the code from command line: **python receive_tweets.py.** Once you run it, you would not start seeing tweets, rather it will wait for a spark-job to connect to the tcp port 5555.

Code:

```
import tweepy
import socket
import json

access_token = 'Enter API Key'
access_secret = 'Enter API Key Secret'
bearer_token = 'Enter Bearer Token'


# Subclass tweepy.StreamingClient to print the tweets
class TweetPrinter(tweepy.StreamingClient):
    def __init__(self, bearer_token, client_socket):
```

```
        tweepy.StreamingClient.__init__(self, bearer_token)
        self.client_socket = client_socket

    def on_tweet(self, tweet):
        try:
            message = f'{tweet.id}: {tweet.text}'
            print(message.encode('utf-8'))
            self.client_socket.send(message.encode('utf-8'))
            print('-'*50)
            return True
        except BaseException as e:
            print('Error on_data: %s' % str(e))
        return True

if __name__ == '__main__':
    new_skt = socket.socket()         # initiate a socket object
    host = '127.0.0.1'                # local machine address
    port = 5555                       # specific port for your service.
    new_skt.bind((host, port))        # Binding host and port

    print('Now listening on port: %s' % str(port))

    new_skt.listen(5)                 #  waiting for client connection.
    client_socket, addr = new_skt.accept()        # Establish connection with client. it returns first a sock

    print('Received request from: ' + str(addr))
    # and after accepting the connection, we will sent the tweets through the socket

    printer = TweetPrinter(bearer_token, client_socket)

    # add new rules
    rule = tweepy.StreamRule(value='football')
    printer.add_rules(rule)

    printer.filter()
```

Explanation

receive_tweets.py has two roles -

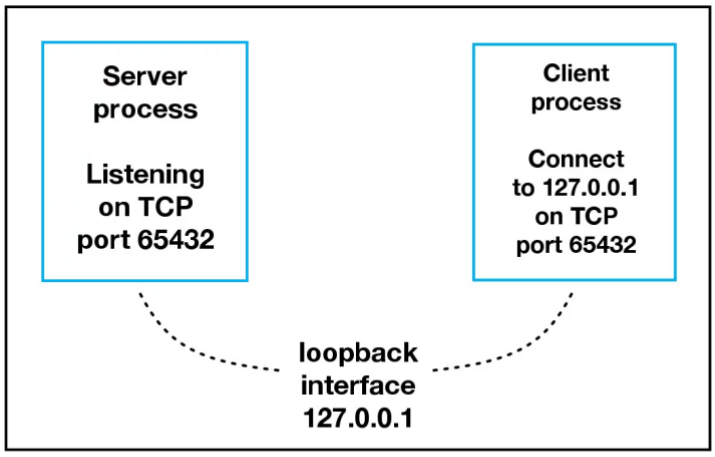
1. Working as a server that serves tweets to **spark-job read_tweets.py** via TCP socket.
2. Fetching tweets

Job 1:

In main, we create a TCP socket and bind it to “localhost” in port 5555.

A TCP *socket* is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the TCP layer in your computer can identify the application that data is destined to be sent to.

Since, both spark-job and **receive_tweets.py** will be running in the same machine (your laptop/host), we use the loopback interface (IPv4 address 127.0.0.1. The loopback mechanism runs a network service on a host without requiring a physical network interface. Data never leaves the host or touches the external network. In the diagram below, the loopback interface is contained inside the host.



This represents the internal nature of the loopback interface and that connections and data that transit it are local to the host.
When spark-job connects to this socket, it uses the TCP connection to send tweets to the spark-job. This brings us to the second job of this program.

Job 2: For sending tweets, it uses the bearer token to validate itself to use the tweeter api. It does so by providing the authentication credentials to TweetPrinter which is an instance of tweepy .StreamingClient. It also associates a filtering rule tweepy.
on_tweet () function of tweepy.StreamingClient class is called whenever tweets are available meeting the filtering criteria. Thus, inside we use the tcp connection to relay the tweet text to spark job.
How the spark job uses the tweet text is up next.

Part 2: Count frequency of tweets in spark

Spark Streaming is an extension of the core Spark API that enables scalable, high-throughput, fault-tolerant stream processing of live data streams. Data can be ingested from many sources like Kafka, Flume, Kinesis, or TCP sockets, and can be processed using complex algorithms expressed with high-level functions like **map**, **reduce**, **join** and **window**.
In this tutorial we use TCP sockets as source for data stream. Recall that, in part 1, we already have a socket open at (localhost, 9015). In this part, we use that socket as source for stream of tweets.
Here is the code that reads tweet text from that socket, preprocess the tweet and perform word-count. Then it updates previously known word-count accordingly. Eventually it stores the current word-count in a temporary table.

Code:

```
import findspark
findspark.init()
import pyspark

# import necessary packages
from pyspark import SparkContext
from pyspark.streaming import StreamingContext
from pyspark.sql import SQLContext
from pyspark.sql.functions import desc

sc = SparkContext()
```

```
sc.setLogLevel("ERROR")
# we initiate the StreamingContext with 10 second batch interval. #next we initiate our sqlcontext
ssc = StreamingContext(sc, 10)
sqlContext = SQLContext(sc)

# initiate streaming text from a TCP (socket) source:
socket_stream = ssc.socketTextStream("127.0.0.1", 5555)
# lines of tweets with socket_stream window of size 60, or 60 #seconds windows of time
lines = socket_stream.window(60)

# just a tuple to assign names
from collections import namedtuple
fields = ("hashtag", "count" )
Tweet = namedtuple( 'Tweet', fields )
# here we apply different operations on the tweets and save them to #a temporary sql table
( lines.flatMap( lambda text: text.split( " " ) ) #Splits to a list
  # Checks for    hashtag calls
  .filter( lambda word: word.lower().startswith("#") )
  .map( lambda word: ( word.lower(), 1 ) ) # Lower cases the word
  .reduceByKey( lambda a, b: a + b )
 # Stores in a Tweet Object
  .map( lambda rec: Tweet( rec[0], rec[1] ) )
 # Sorts Them in a dataframe
  .foreachRDD( lambda rdd: rdd.toDF().sort("count")
 # Registers only top 10 hashtags to a table.
  .limit(10).registerTempTable("tweets") ) )

# start streaming and wait couple of minutes to get enought tweets
ssc.start()

# import libraries to visualize the results
import time
from IPython import display
import matplotlib.pyplot as plt
import seaborn as sns
import pandas

get_ipython().run_line_magic('matplotlib', 'inline')
top_10_tags = sqlContext.sql( 'Select hashtag, count from tweets' )
top_10_df = top_10_tags.toPandas()
display.clear_output(wait=True)
plt.figure( figsize = ( 10, 8 ) )
sns.barplot( x='count', y='hashtag', data=top_10_df)
plt.show()
```

Explanation:

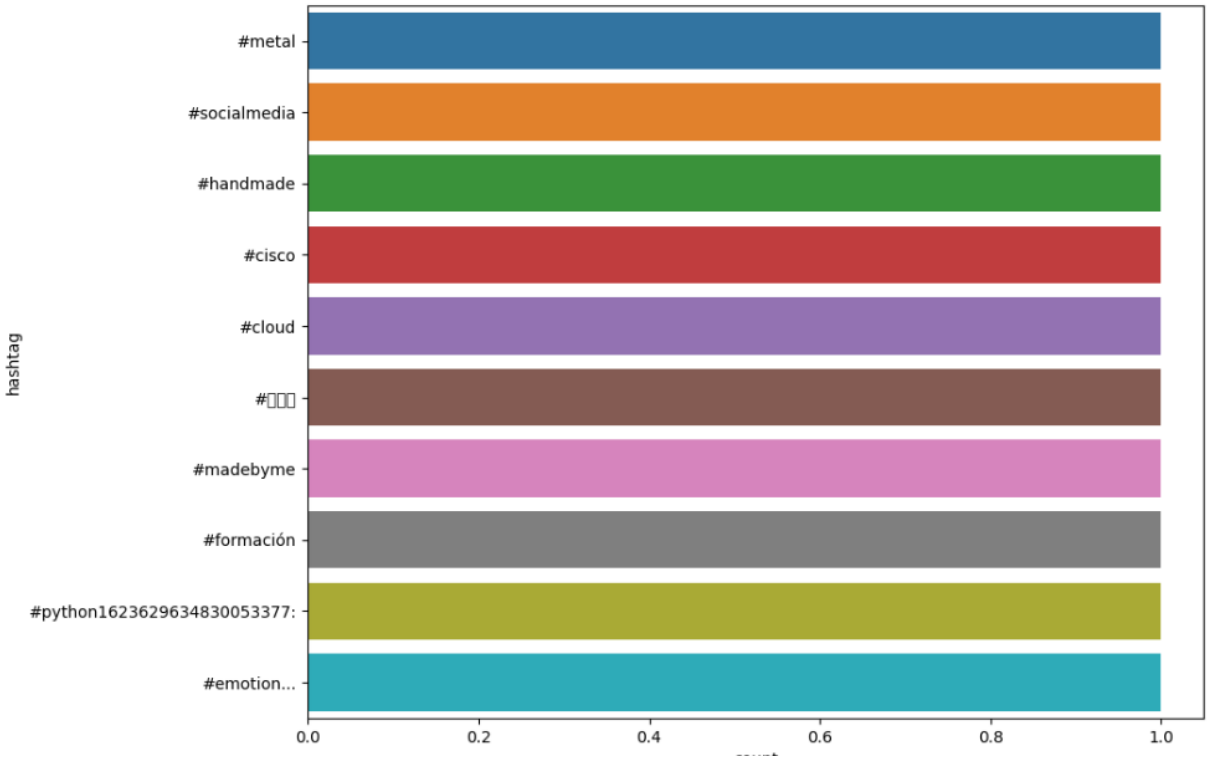
First, we have to create an instance of Spark Context **sc**, then we created the Streaming Context **ssc** from sc with a batch interval 10 seconds that will do the transformation on all streams received every two seconds. Notice we have set the log level to ERROR in order to disable most of the logs that Spark writes.

Then we define our main ssc.socketTextStream socket_stream that will connect to the socket server we created before on port 5555 and read the tweets from that port. Each record in the DStream will be a tweet.

Now, we'll define our transformation logic. First we'll split all the tweets into words. Then we'll convert them to lowercase and filter hashtags from all words.
Then we need to calculate how many times the hashtag has been mentioned. We can do that by using the function reduceByKey. This function will calculate how many times the hashtag has been mentioned.
The, we will store this (hastag, count) tuple as a Tweet object in RDD. Finally, we will convert it to a DataFrame, sort it in descending order and limit results to top 10 values.

This logic is invoked when streaming context ssc is started. Then we do processing in every batch in order to convert it to temporary table using **Spark SQL Context** and then perform a SELECT query to retrieve the top 10 hashtags with their counts and put them into **top_10_df** data frame.

Result:



The job starts plots hashtag and counts. Your output may be different since the tweets are being processed real-time.

Acknowledgement:

1. Spark streaming API guide: https://spark.apache.org/docs/latest/streaming-programming-guide.html

2. HANEE MEDHAT's article on twitter streaming using spark: https://www.toptal.com/apache/apache-spark-streaming-twitter