PUVIKARAN Sajeevan

DACI Yanis

Stream Processing and messaging using Apache SPARK

The objective of this practical work is to create a Spark Streaming Application that plots out the popularity of hashtags on Twitter. To achieve this work, we used the python library called Tweepy for accessing the Twitter API.

We also created two Jupyter notebooks called:

- Twitter_python_code which allows us to connect to the Twitter API, collect the tweets and send them to our Spark application. It is our Twitter App Client.
- Lab-3, our Spark application. It permits to process our tweets by taking the information that we are interested in, the hashtags, and plot them.

I) Twitter App Client

The goal of this notebook is to manage the Tweepy library to collect tweets. It will connect as localhost to the port 9999 (chosen randomly by us) and receive the tweets given by the Twitter API. Our Twitter App Client will also do a first data processing. Indeed, when the data come, it will pick a lot of details such as the user who send the tweet and the description. We are not interested by the metadata, so we filter them by only taking the content of a tweet.

Then, to start the raising of data, it will be wait for client connection. Establishing a connection, this notebook will send a request to the Twitter API for receiving the tweets which are in line with the selected topic. The topic is defined in the notebook's sendData function. Finally, once the data are collected, they will be shared with the Spark Application.

```python
# Listen and send tweets Python Class
class TweetsListener(StreamListener):
    def __init__(self, csocket):
        self.client_socket = csocket

    def on_data(self, data):
        try:
            data = json.loads(data)

            if data["user"]["description"] != None:
                print(data["user"]["description"])
                print("\n")
                self.client_socket.send(data["user"]["description"].encode('utf-8'))
            return True
        except BaseException as e:
            print("Error: %s" % str(e))
        return True
```

We created a class called « TweetListener » which contain all functions needed by our App Client. The function *on_data* is used to load the data and realize the first processing by taking only the user and the tweet description.

```python
def sendTweets(c_socket):
    # Authentication
    auth = OAuthHandler(consumer_key, consumer_secret)
    auth.set_access_token(access_token, access_secret)

    # Streaming and filtering by topic
    twitter_stream = Stream(auth, TweetsListener(c_socket))
    twitter_stream.filter(track=['WorldKindnessDay'])
```

This function allows the Twitter App Client to connect to the Twitter API thanks to the Tweepy library. Then, we retrieve only the tweets which are in line with our topic which is « **WorldKindnessDay** ».

```python
if __name__ == "__main__":
    s = socket.socket()     # Create a socket object
    host = "localhost"      # Indicate the host
    port = 9999             # Reserve a port in order to send tweets.
    s.bind((host, port))    # Bind to the port

    print("Listening on port: %s" % str(port))

    s.listen(5)                 # Waiting for client connection.
    c, addr = s.accept()        # Establish connection with client.

    print("Received request from: %s" % str(addr))

    sendTweets(c)
```

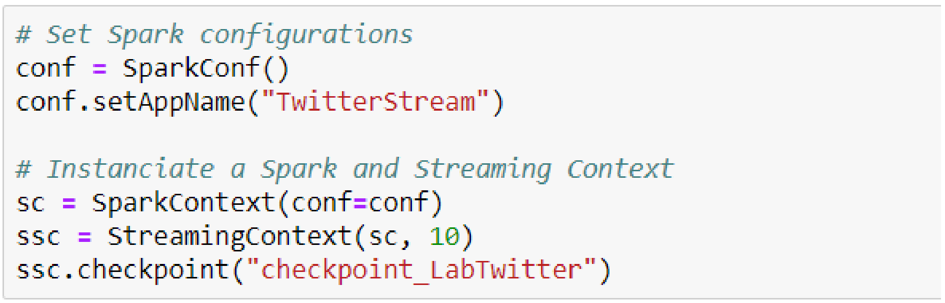
The notebook is binding to the port 9999 and waiting for client connection to send tweets.

<u>II) Spark Application</u>

At first, our Spark Application will create a Spark configuration with the app title name « TwitterStream ». Then, it will create a Spark Context using the previous Spark configuration and create a check point called « checkpoint_LabTwitter » in order to reiterate the operation. After that, a DataStream object is instantiated and will connect to the port 9999. It will take the information sent by the App Client. Once the data received, the Spark Application will read the DataStream and retrieve only hashtags located in the tweet description.

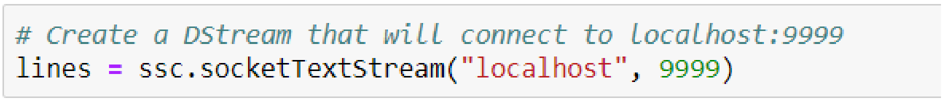
For each hashtag collected, we will create a couple containing the hashtag and the value 1. In this way, we will be able to know which tags are the most popular by adding one to its counter each time it is used in a tweet.

Then, the data frame is converted into a table and the application do a query searching the 10 top hashtags using Spark SQL. Finally, the application will run until the process is stopped.

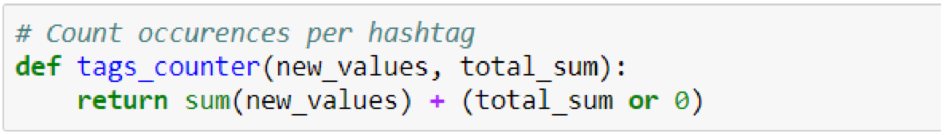


```python
# Set Spark configurations
conf = SparkConf()
conf.setAppName("TwitterStream")

# Instanciate a Spark and Streaming Context
sc = SparkContext(conf=conf)
ssc = StreamingContext(sc, 10)
ssc.checkpoint("checkpoint_LabTwitter")
```



```python
# Create a DStream that will connect to localhost:9999
lines = ssc.socketTextStream("localhost", 9999)
```



```python
# Count occurences per hashtag
def tags_counter(new_values, total_sum):
    return sum(new_values) + (total_sum or 0)
```

In the code above, we create a spark configuration, a check point and the DataStream object which will connect to localhost:9999 (like our Twitter App client). The function tags_counter will be used to count the occurrence per hashtag and return the top 10 hashtags.

```python
# Intialise a Spark SQL Context if not done
def get_sql_context_instance(spark_context):
    if ('sqlContextSingletonInstance' not in globals()):
        globals()['sqlContextSingletonInstance'] = SQLContext(spark_context)
    return globals()['sqlContextSingletonInstance']
```

This function creates a Spark SQL Context from the Spark Context if it does not exist yet. This context helps us to do one of the final steps of this project: Querying from table.

```python
def tag_process(time, rdd):
    print("----------- %s -----------" % str(time))

    try:
        # Use current context to create an SQL context
        sql_context = get_sql_context_instance(rdd.context)

        # RDD to Row RDD Conversion
        row_rdd = rdd.map(lambda w: Row(hashtag=w[0], hashtag_count=w[1]))

        # Transformation in Dataframe
        hashtags_df = sql_context.createDataFrame(row_rdd)

        # Register the dataframe as table
        hashtags_df.registerTempTable("hashtags")

        # Selection of the 10 top hashtags
        hashtag_counts_df = sql_context.sql(
            "select hashtag, hashtag_count from hashtags order by hashtag_count desc limit 10")
        hashtag_counts_df.show()

        # Transformation in Pandas DF and plot results
        histogram = hashtag_counts_df.toPandas()
        histogram.plot(kind='bar',x='hashtag',y='hashtag_count')
    except:
        e = sys.exc_info()[0]
        print("Error: %s" % e)
```

This function will do a data processing by passing from RDD to Row RDD, Spark Data Frame, table and finally Pandas data frame. The query does a search of the 10 most popular Twitter tags currently.

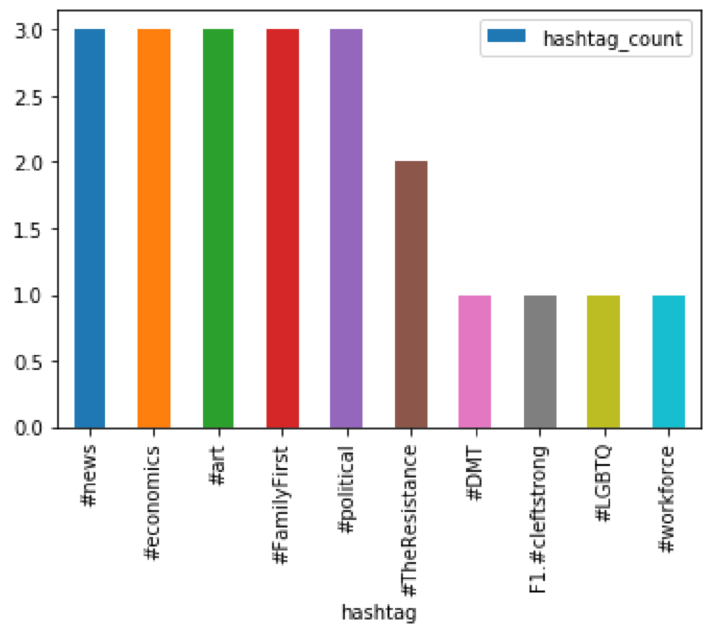
```python
words = lines.flatMap(lambda line: line.split(" "))
hashtag = words.filter(lambda w: '#' in w).map(lambda x: (x, 1))
all_tags = hashtag.updateStateByKey(tags_counter)
all_tags.foreachRDD(tag_process)

ssc.start()
ssc.awaitTermination() # Wait for the process to terminate
```

This code use all functions described above and then start streaming.

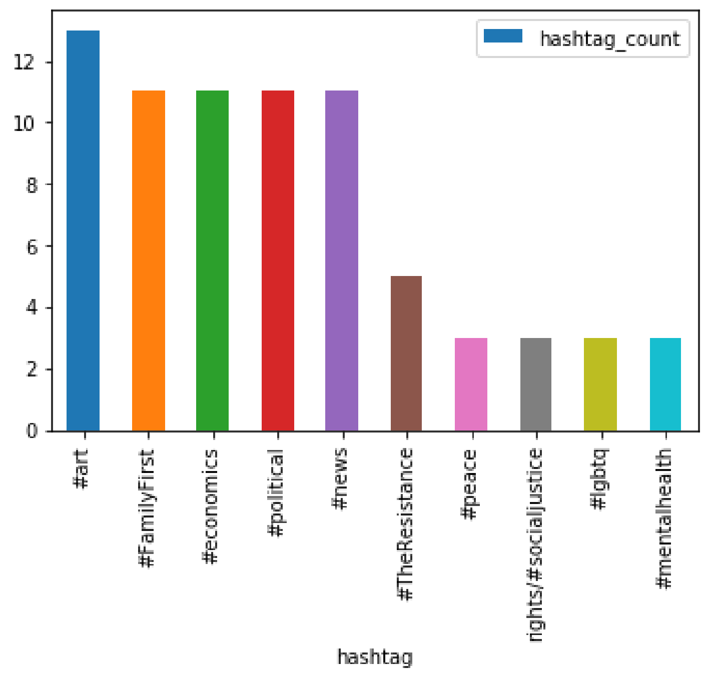
III) Plot of results for the topic WorldKindnessDay

OUTPUT



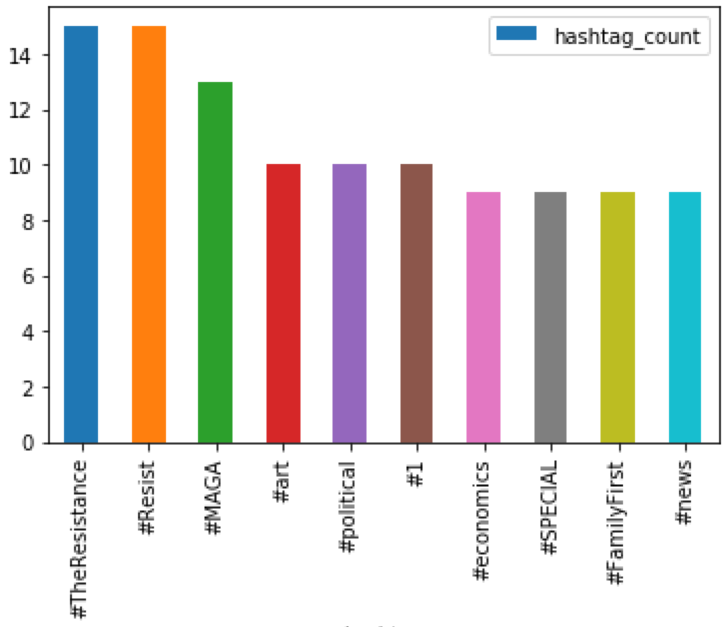
```
----------- 2018-11-13 22:06:30 ----------
+-----------------+------------+
|          hashtag|hashtag_count|
+-----------------+------------+
|       #economics|           3|
|             #art|           3|
|       #political|           3|
|     #FamilyFirst|           3|
|            #news|           3|
|   #TheResistance|           2|
|   #humancapital,|           1|
|💛#tbiptsdsurvivor|          1|
| #StrongerTogether|           1|
|          #Diver.|           1|
+-----------------+------------+
```

Figure 1: Results after 30 seconds



```
---------- 2018-11-13 22:03:00 ----------
-------------+------------+
      hashtag|hashtag_count|
-------------+------------+
         #art|          13|
  #FamilyFirst|          11|
   #economics|          11|
        #news|          11|
   #political|          11|
#TheResistance|           5|
   #vote4kids|           3|
     #ProLife|           3|
 #mentalhealth|           3|
       #lgbtq|           3|
-------------+------------+
```

Figure 2: Results after 2 minutes



```
----------- 2018-11-13 22:18:30 -----------
+--------------+-------------+
|       hashtag|hashtag_count|
+--------------+-------------+
|       #Resist|           15|
|#TheResistance|           15|
|         #MAGA|           13|
|    #political|           10|
|            #1|           10|
|          #art|           10|
|  #FamilyFirst|            9|
|      #SPECIAL|            9|
|    #economics|            9|
|         #news|            9|
+--------------+-------------+
```

Figure 3: Results after 10 minutes

IV) Source code

- Twitter App Client:

```
-   import os
-   import socket
-   import json
-   import tweepy
-   from tweepy import OAuthHandler
-   from tweepy import Stream
-   from tweepy.streaming import StreamListener
-
-   consumer_key    = os.environ['TWITTER_CONSUMER_KEY']
-   consumer_secret = os.environ['TWITTER_CONSUMER_SECRET']
-   access_token    = os.environ['TWITTER_ACCESS_TOKEN']
-   access_secret   = os.environ['TWITTER_ACCESS_SECRET']
-
-   # Listen and send tweets Python Class
-   class TweetsListener(StreamListener):
-       def __init__(self, csocket):
-           self.client_socket = csocket
-
-       def on_data(self, data):
-           try:
-               data = json.loads(data)
-
-               if data["user"]["description"] != None:
-                   print(data["user"]["description"])
-                   print("\n")
-
   self.client_socket.send(data["user"]["description"].encode('utf-8'))
-               return True
-           except BaseException as e:
-               print("Error: %s" % str(e))
-           return True
-
-       def on_status(self, status):
-           print(status.text)
-
-   def sendTweets(c_socket):
-       # Authentication
-       auth = OAuthHandler(consumer_key, consumer_secret)
-       auth.set_access_token(access_token, access_secret)
-
-       # Streaming and filtering by topic
-       twitter_stream = Stream(auth, TweetsListener(c_socket))
-       twitter_stream.filter(track=['WorldKindnessDay'])
-
-
-   if __name__ == "__main__":
-       s = socket.socket()     # Create a socket object
-       host = "localhost"      # Indicate the host
-       port = 9999             # Reserve a port in order to send tweets.
-       s.bind((host, port))    # Bind to the port
-
-       print("Listening on port: %s" % str(port))
-
-       s.listen(5)                 # Waiting for client connection.
-       c, addr = s.accept()        # Establish connection with client.
-
-       print("Received request from: %s" % str(addr))
-
-       sendTweets(c)
-
```

- Spark Application:

```
import json
import sys
import time
import findspark
findspark.init('/home/sajeevan/Téléchargements/spark-2.1.0-bin-hadoop2.7')
```

```python
import pyspark
from pyspark import SparkContext, SparkConf
from pyspark.streaming import StreamingContext
from pyspark.sql import Row, SQLContext
from pyspark_dist_explore import hist

import matplotlib.pyplot as plt

# Set Spark configurations
conf = SparkConf()
conf.setAppName("TwitterStream")

# Instanciate a Spark and Streaming Context
sc = SparkContext(conf=conf)
ssc = StreamingContext(sc, 10)
ssc.checkpoint("checkpoint_LabTwitter")

# Create a DStream that will connect to localhost:9999
lines = ssc.socketTextStream("localhost", 9999)

# Count occurences per hashtag
def tags_counter(new_values, total_sum):
    return sum(new_values) + (total_sum or 0)

# Intialise a Spark SQL Context if not done
def get_sql_context_instance(spark_context):
    if ('sqlContextSingletonInstance' not in globals()):
        globals()['sqlContextSingletonInstance'] = SQLContext(spark_context)
    return globals()['sqlContextSingletonInstance']


def tag_process(time, rdd):
    print("----------- %s -----------" % str(time))

    try:
        # Use current context to create an SQL context
        sql_context = get_sql_context_instance(rdd.context)

        # RDD to Row RDD Conversion
        row_rdd = rdd.map(lambda w: Row(hashtag=w[0], hashtag_count=w[1]))

        # Transformation in Dataframe
        hashtags_df = sql_context.createDataFrame(row_rdd)

        # Register the dataframe as table
        hashtags_df.registerTempTable("hashtags")

        # Selection of the 10 top hashtags
        hashtag_counts_df = sql_context.sql(
            "select hashtag, hashtag_count from hashtags order by
hashtag_count desc limit 10")
        hashtag_counts_df.show()

        # Transformation in Pandas DF and plot results
        histogram = hashtag_counts_df.toPandas()
        histogram.plot(kind='bar',x='hashtag',y='hashtag_count')
    except:
        e = sys.exc_info()[0]
        print("Error: %s" % e)

words = lines.flatMap(lambda line: line.split(" "))
hashtag = words.filter(lambda w: '#' in w).map(lambda x: (x, 1))
all_tags = hashtag.updateStateByKey(tags_counter)
all_tags.foreachRDD(tag_process)

ssc.start()
ssc.awaitTermination() # Wait for the process to terminate

ssc.stop()
```