Basic Twitter Sentiment Analytics using Apache Spark Streaming APIs and Python

Estimated time: 8 hours

In this project, you will learn about processing live data streams using Spark's streaming APIs and Python. You will be performing a basic sentiment analysis of real-time tweets. In addition, you will also get a basic introduction to Apache Kafka, which is a queuing service for data streams.

Submission Instructions:
Submit a single zipped folder that contains two files: (1) the `twitterStream.py` file containing your code updates and (2) `plot.png`, an image of the plot that your code produces.

Background

In this project, you will be processing streaming data in real time. One of the first requirements is to get access to the streaming data; in this case, real-time tweets. Twitter provides a very convenient API to fetch tweets in a streaming manner. Start by reading the basic documentation found here.

In addition, you will also be using Kafka to buffer the tweets before processing. Kafka provides a distributed queuing service which can be used to store the data when the data creation rate is more than processing rate. It also has several other uses. Read about the basics of Kafka (`Introduction` up to `Quick Start`) from the official documentation.

Finally, as an introduction to stream processing API provided by Spark, read the programming guide from the beginning up to, and including, the section on discretized streams (DStreams). Pay special attention to the "quick example" as it will be fairly similar to this project.

Project Setup

Installing Required Python Libraries

We have provided a text file containing the required python packages: `requirements.txt`. To install all of these at once, simply run (only missing packages will be installed):

```
$ sudo pip install -r requirements.txt
```

Installing and Initializing Kafka

Download and extract the latest binary from https://kafka.apache.org/downloads.html

Start zookeeper service:

```
$ bin/zookeeper-server-start.sh config/zookeeper.properties
```

Start kafka service:

```
$ bin/kafka-server-start.sh config/server.properties
```

Create a topic named `twitterstream` in kafka:

```
$ bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1
--partitions 1 --topic twitterstream
```

Check what topics you have with:

```
$ bin/kafka-topics.sh --list --zookeeper localhost:2181
```

Using the Twitter Streaming API

In order to download the tweets from twitter streaming API and push them to kafka queue, we have provided a python script `twitter_to_kafka.py`. The script will need your twitter authentication tokens (keys). A guide to create an app on twitter to get your authentication tokens can be found here.

Once you have your authentication tokens, create or update the `twitter.txt` file with these credentials. Note, never commit your credentials on code sharing websites, such as Github.

After updating the text file with your twitter keys, you can start downloading tweets from the twitter stream API and push them to the twitterstream topic in Kafka. Do this by running our program as follows:

```
$ python twitter_to_kafka.py
```

Note, this program must be running when you run your portion of the assignment, otherwise you will not get any tweets.

To check if the data is landing in Kafka:

```
$ bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic twitterstream
--from-beginning
```

Running the Stream Analysis Program

```
$ $SPARK_HOME/bin/spark-submit --packages org.apache.spark:spark-streaming-kafka_2.10:1.5.1
twitterStream.py
```

Project Requirements

You will be provided a basic stub for the project in `twitterStream.py`. You job is to complete the missing parts of the file so that it writes the running total count of the number of positive and negative words that have been tweeted. In addition, you will have to plot the total positive and negative word counts for each timestep. See the example outputs below.

The word lists for positive words and negative words are given in the `positive.txt` and `negative.txt` files respectively.

The main function of the program is given and you should not need to change it. From the main function it should be relatively clear how the program is going to operate. However, you will have to provide the code for a few of the functions discussed in more detail below.

`load_wordlist(filename)`
This function is used to load the positive words from `positive.txt` and the negative words from `negative.txt`. This function needs to return the words as a list or set.

`stream(ssc, pwords, nwords, duration)`
This is the main streaming function consisting of several steps that you must complete. There is skeleton code for starting the stream, getting the tweets from kafka, and stopping the stream. You must complete the intermediate steps of taking the streamed tweets and finding the number of positive and negative words.

The `tweets` variable is a DStream object on which you can perform similar transformations that you could to an RDD. Currently, `tweets` consists of rows of strings, with each row representing one tweet. You can view this structure by using the pprint function: `tweets.pprint()`.

The example given in the streaming guide shows how you can aggregate the counts for *all* words. However, we want to combine the positive word counts together and to combine the negative word counts together. Therefore, rather than mapping each word to (word, 1) (as is done in the example), you can map each word to ("positive", 1) or ("negative", 1) depending on which class that word belongs. Then, the counts can be aggregated using the reduceByKey function to get the total counts for "positive" and the total counts for "negative".

If you didn't make anymore changes, the DStream would store the counts at each time step, not the running total. You want both. To get the running total, you must use the updateStateByKey function. For more details, read the section about this operation. It is this DStream (the one with the running totals) that you should output using the pprint function. Your output should look like the console output shown below.

The other DStream (the one that doesn't store the running total) can be used to make the plot that shows the total counts at each time step. However, you first must convert the DStream into

something useable. To do this, use the foreachRDD function. With this function, we can loop over the sequence of RDDs of that DStream object and get the actual values. Example code is given, you will just have to substitute the correct DStream object.

`make_plot(counts)`

In this function, you are to use matplotlib to plot the total word counts at each time step. This information should be stored in the output of your stream function (`counts`). This matplotlib tutorial has plenty of simple examples that should have all the information needed to make this plot. Ultimately, your output should look something like the plot below.

```
-------------------------------------------
Time: 2015-12-10 23:03:00
-------------------------------------------
('positive', 180)
('negative', 102)

-------------------------------------------
Time: 2015-12-10 23:03:10
-------------------------------------------
('positive', 400)
('negative', 198)

-------------------------------------------
Time: 2015-12-10 23:03:20
-------------------------------------------
('positive', 569)
('negative', 300)

-------------------------------------------
Time: 2015-12-10 23:03:30
-------------------------------------------
('positive', 733)
('negative', 394)

-------------------------------------------
Time: 2015-12-10 23:03:40
-------------------------------------------
('positive', 909)
('negative', 504)
```