Project Spark-Elasticsearch

Marwan MASHRA - Abdurrahman SHAHID - Romain MAGNET

May 6, 2022

1 Introduction

The goal of this project is to retrieve tweets from Twitter and analyze the collected data using Spark and Elasticsearch. However, one must design cautiously the entire pipeline, by taking into account the specificities of each component, and how they can be put together.

With that in mind, we will first see how to get tweets from Twitter using the Twitter API and by using a custom *Listener* object. Subsequently, once the data is ingested, we use Spark (more precisely PySpark) to process the stream of tweets and extract data and metadata of interest for our donwstream tasks. Indeed, we use TextBlob to perform sentiment analysis on the tweets. Then we use the latter results conjointly with the location and tag information to cluster the tweets, using *streaming k-means* from MLLib. Last but not least, we use Elasticsearch to ingest data from Spark and index it. Afterward, we extract interesting information from the data and evaluate a multiterm query. We then end the project by using a highlighter for some regular expression query.

2 Streaming tweets

2.1 Twitter API

Since we only had a developer account with essential access (no elevated access), we had access to the Twitter API v2. There are two ways to retrieve tweets from the API:

1. using Tweepy, which is a python package that offers a wide variety of classes to use the Twitter API, such as *Client*, *StreamingClient*...etc.

2. directly using the REST API of Twitter, which is what we decided to use for this project. We use the package python **requests** to make API calls (send GET and POST requests to the Twitter API).

The first thing to do, in order to get a filtered stream, is to add the rules. The Twitter API is going to look for tweets that match those rules when searching for tweets later. In our case, we wanted to do a sentiment analysis for several famous brands and companies, so we made the following rules:

```
my_rules = [
    {"value": "samsung lang:en", "tag": "samsung"},
    {"value": "apple lang:en", "tag": "apple"},
    {"value": "amazon lang:en", "tag": "amazon"},
    {"value": "facebook lang:en", "tag": "facebook"},
]
```

Then to add those rules, we just send a POST request to the API as following:

```
payload = {"add": my_rules}
response = requests.post(
    "https://api.twitter.com/2/tweets/search/stream/rules",
    auth=self.bearer_oauth,
    json=payload,
)
```

Finally, we send a GET request to get a stream of Tweets. Now we can easily read tweets from this stream, process them to only keep the fields that we want to send (text, tag, location) and send them to our spark client.

2.2 Server

After getting the stream from the Twitter API, we set up a python server using the python package **socket**. To do so, we start by creating a socket, then binding it to a host. Then it starts listening (waiting) for a connection from a client socket (our spark client). After connection, the server starts sending tweets to the client socket. Here is an example of what it looks like:

```python
    new_skt = socket.socket()      # initiate a socket object
    new_skt.bind((host, port))     # Binding host and port
    print(f"Listening on http://{host}:{port}")
    new_skt.listen(5)              #  waiting for client connection
    client_socket, client_addr = new_skt.accept()     # Establish connection with client.
    print("Received request from: " + str(client_addr))
    self.send_tweets(client_socket)
```

We also decided to make the server wait for a random amount of time (up to 2s) between each tweet sent, to simulate the behavior of a real server and so that the spark client is going to receive the tweets at a random rate.

3 Spark

In this part, we're trying to connect to the server we set up previously, get the stream of tweets, process and cluster them.

3.1 Getting and processing the tweets

We start by creating a DStream that connects to our server. Then we create the processing pipeline for the stream. The processing pipeline involves:

- doing a sentiment analysis with textblob to extract the polarity and the subjectivity from the tweets.
- Transforming the location and the tag into a numerical attribute to use them later in the clustering. We do that by using a hashmap (refer to the code for more details about the implementation).
- mapping each tweet into a vector of the python package *MLLib* that contain all the features used for clustering (polarity, subjectivity, location, tag).

Here is wwhat the pipeline looks like :

```python
    raw_tweets = ssc.socketTextStream(HOST, PORT)
    tweets = raw_tweets.map(process_tweets)
    train_data = tweets.map(lambda tweet: Vectors.dense([
            tweet['tag'],
            tweet['location'],
            tweet['polarity'],
            tweet['subjectivity']
        ]))

```

3.2 Clustering

We cluster the data according to the features: polarity, subjectivity, location and tag. To cluster directly from the spark stream, we use the implementation of K-means in *MLLib*.

```
    model = StreamingKMeans(k=4, decayFactor=1.0).setRandomCenters(4, 1.0, 0)
    model.trainOn(train_data)
    result = model.predictOn(train_data)
```

3.3 Evaluating the cluster size in a window

We use a window operation to group tweets from the same cluster together and evaluate the size of the cluster. We group tweets of 30s together, and then move the window by 10s.

```
    pairs = result.map(lambda cluster: (f'cluster-{cluster+1}', 1))
    clusterCounts = pairs.reduceByKeyAndWindow(lambda x, y: x + y, lambda x, y: x - y, 30, 10)
    clusterCounts.pprint()
```

Finally, here is the output:

```
    -------------------------------------------
    Time: 2022-05-06 10:07:10
    -------------------------------------------
    ('cluster-4', 6)
    ('cluster-3', 2)
    -------------------------------------------
    Time: 2022-05-06 10:07:20
    -------------------------------------------
    ('cluster-4', 13)
    ('cluster-3', 4)
    -------------------------------------------
    Time: 2022-05-06 10:07:30
    -------------------------------------------
    ('cluster-4', 16)
    ('cluster-3', 8)
    -------------------------------------------
    Time: 2022-05-06 10:07:40
    -------------------------------------------
    ('cluster-4', 16)
    ('cluster-3', 8)
```

4 Elasticsearch

In this part of the project, our goal is to ingest some tweets from the spark DStream, into Elasticsearch, and index them. This will allow us to run queries on the tweets very efficiently.

We start by running an Elasticsearch server in a docker container. Then we send requests to the server using the python package *requests*. We made a function **elasticsearch_curl** that allows us to send those requests. Now, to insert the tweets, we use foreachRDD that allows us to run each RDD into a function. In that function, we insert the tweets to Elasticsearch in bulk using a PUT request. Here is what the function we use to insert looks like

```

def insert_doc(rdd):
    body = ""
    for tweet in rdd.collect():
        body += '{ "create": { } }' + '\n' + tweet + '\n'
    if body:
        response = elasticsearch_curl(
            f'logs-my_app-default/_bulk?pretty',
            verb='put',
            json_body=body,
            verbose=1
        )
```

Finally, to run queries on the tweets, we send a GET request to Elasticsearch.

For example, to took for tweets that has Apple, Samsung or Amazon, but prioritize Amazon (show them first, or give them a higher matching score), we use the following query:

```
q = {
  "query": {
    "terms": {
      "text": [ "apple", "samsung" ],
      "boost": 1.0
    },
      "terms": {
      "text": [ "amazon" ],
      "boost": 2.0
    }
  }
}
```