Large Scale Computing - Final Project

Zhe HUANG, Mathis REYMOND

May 2023

1 Introduction

Streaming data analysis is the process of analyzing data in real-time as it is generated or received from the source. It is a challenging task as the data is continuously coming in, and the analysis must be done without any delay. PySpark provides a robust framework to perform streaming data analysis on big data sets.

The project was aiming to use a streaming API that publishes some textual data (such as Twitter's API) and perform sentiment analysis on the data. However, as we failed at finding a working API that publishes relevant data for the task (API for mobility network does not provide data that allow to perform sentiment analysis), we decided to load a dataset of tweets and create a stream of data ourselves using socket.

2 Data and Task

2.1 Data

We used the MBTI Personality Type Twitter Dataset (available here). The Myers-Briggs Type Indicators (MBTI) is one of the most popular personality model which creates a binary categorization based on four different dimensions and produces 16 possible personality type depending on the combination of these four values : Introversion/Extraversion, Intuition /Sensing, Feeling /Thinking and Perception /Judgement. The dataset provides over 7800 tweets labeled with user's personality types.

2.2 Task

What we wanted to do was to perform sentiment analysis on the tweet to determine how positive they are and then build a dynamical bar plot with the 16 personality types that shows, in average, how positive each personality type is.

3 Methodology

Our code is divided in three parts: Sender, Receiver with Spark, Reveiver without Spark.

3.1 Sender

For our experiment, we use one notebook as the sender. It will send a message to the receiver. The data we will be using is a CSV file containing tweets related to the Myers-Briggs Type Indicator (MBTI). We have loaded the data as a pandas DataFrame and created a new column to store the first 100 characters of the text.

To establish a connection with the receiver, we create a socket object, set the host to the local machine name, and the port number to 8080. We then bind the socket to a public host and a well-known port and listen for incoming connections.

Once a connection is established, we send the first 10 shortened messages to the client, with a 3-second delay between each message. Finally, we close the connection and break the loop.

3.2 Receiver with Spark

The second part is about connecting to the server and receiving the incoming stream of data from the sender.

We once wrote a notebook as a receiver. In theory, we should run the sender notebook before running this notebook. However, we cannot run this receiver with Spark Streaming as we faced issues with setting up the Spark environment on our laptops.

But we can still explain the original idea: We create a Spark context and a local StreamingContext with two working threads and a batch interval of 3 seconds. Then, we create a DStream that connects to the hostname:port, like localhost:8080. We print a message indicating that the connection has been established with the server.

Next, we define a function to perform sentiment analysis on each message. This function uses the TextBlob library to compute the sentiment polarity of each message. If the polarity is greater than 0, the sentiment is positive. If the polarity is less than 0, the sentiment is negative. Otherwise, the sentiment is neutral.

We tag each message with its sentiment and print the tagged messages. Finally, we start the streaming computation and wait for the streaming to finish.

3.3 Receiver without Spark

After giving up the SparkStraming method, we created a new notebook to perform as the receiver. In this notebook we create a socket object and connect it to the server on a specified port number. It then receives data from the server in an infinite loop and performs sentiment analysis on each message using the TextBlob library. The sentiments are added to a list, and the message and its corresponding sentiment are printed.

After all the messages have been received and analyzed, the list of sentiments is converted to a NumPy array and converted to numerical values. K-means clustering is then performed on the sentiments array with three clusters, and the cluster labels are printed.

Finally, the data points and their clusters are plotted using a scatter plot. The main idea is to receive the incoming stream of data, analyze the sentiments of the messages, and use clustering to categorize the sentiments into three clusters.

4 How to run the code

For the code to be run, you need to have installed Spark, PySpark, TextBlob and the basic data-science libraries. Then you have to initialize the server by running the first cells of 'Sender.ipynb' until "Server established, waiting for incoming connections..." is printed. Then, you have to run the first cells of 'Receiver_withoutSpark.ipynb' until you see "Connection established with the server...". From that point, you can execute the last cell of 'Sender.ipynb' to start sending the messages. Finally, go back to 'Receiver_withoutSpark.ipynb' to read and process the incoming data.

5 Other attempt

In the previous approach, we didn't manage to update the barplot in real time, as data comes in. So we tried something else.

After importing the necessary libraries, we create a SparkContext and a StreamingContext, which will be used to process the text data stream. Then we initialize the bar plot, which will display the average polarity of the messages by personality type.

The 'update_plot' function is defined next, which takes an RDD as input. The function first calculates the sentiment polarity of each word in the RDD. It then groups the words by their personality type and calculates the total polarity for each type. Finally, it calculates the average polarity for each personality type and updates the bar chart with the new data.

Then we create a Spark DStream from a text file stream to which we apply transformations to clean and process the messages. These transformations include filtering out empty lines, converting all text to lowercase, splitting the text into words, and calculating the sentiment polarity of each word.

Finally, we group the words by personality type and calculates the total polarity for each type. It then applies the 'update_plot' function to the resulting RDDs to update the bar plot as stream flows in.

If we manage to read the stream of messages in this way, the bar plot never showed up.

6 Challenges Faced

During the process, we encountered a lot of challenges, such as permission errors when writing to a file, issues with both sending and reading tweets at the same time on localhost ports, and problems with updating the plot dynamically. If we managed to resolve most of these challenges, despite all our efforts, we couldn't fix the dynamical plot.