EECS E6893 Big Data Analytics - Fall 2023
Homework Assignment 2: Streaming Big Data Analytics & Data Analytics Pipeline

Due Friday, October 20th, 2023, by 5:00pm

Streaming Analytics

Abstract: In this section, we will demonstrate how to handle live streaming data, where information is received in real-time. Here, analytical operations need to be performed dynamically as the data arrives, without having a central repository of all the data to refer to. While the ideal choice would have been Twitter Streaming, due to recent organizational changes and the complexity of accessing the data, we will instead use the Finnhub API. By making frequent requests to this API for stock price data, we will simulate streaming analysis and showcase the process of handling real-time data.

Setup:
Step 1: Create your account using your Columbia email ID on Finnhub
Step 2: Login and Save your API Key (You will use your own unique API key in the following exercise)
Step 3: Create your GCP Dataproc Cluster as done for previous assignments.
Step 4: In your Dataproc cluster’s Jupyter notebook run the following to install the Finnhub api

```
In [33]: !pip install finnhub-python
```

Skeleton Code:
To pull data from Finnhub for a particular stock – the syntax is as follows:

```python
import finnhub
finnhub_client = finnhub.Client(api_key="YOUR UNIQUE API KEY")
print(finnhub_client.stock_candles('STOCK NAME', 'DATA RESOLUTION', start_unix_timestamp, end_unix_timestamp))
```

This code needs to be modified as per the requirements of the question. A detailed explanation of the parameters can be found at this link under the “Stock Candles” section.

Tasks (20 points)
Read all 3 parts collectively to decide the approach to be taken.

1. Pull the stock data of “AAPL” at 1-minute level data resolution every 5 minutes. Each such pull made once every 5 minutes should have data from the (current timestamp - one hour) to the current timestamp when you hit the api i.e. one hour's worth of data in each pull. Let this process run for 30 minutes. At the end of your program, your code should have generated ~1.5 hours’ worth of data for the stock and the api should have been called 7 times (once every 5 minutes, over a period of 30 minutes).

2. In each data pull, the data should be incrementally loaded into a Spark data frame where the dataframe schema will be ['Stock Name', “UTC Timestamp”, “c”, “l”, “h”, “o”, “v”] where UTC timestamp will be obtained by converting the UNIX timestamps of each entry the api call will return under the “t” key.
Incremental loading entails fetching data in stages. Initially, data covering an hour, say from timestamp $x$ to $x+60$ minutes, is retrieved and stored in a dataframe. In the subsequent pull, fetch data from $x+5$ minutes to $x+65$ minutes. The data from $x+5$ to $x+60$ minutes, already present in the dataframe, gets replaced by the new pull (though it might remain the same in our exercise, real-world scenarios could witness changes in the same data points pulled at different intervals. HINT: To do this filtering operation explore the concept of antijoins). The data points from $x+60$ to $x+65$ minutes are directly added. This process is repeated for all pulls within the next 30 minutes, resulting in a dataframe containing data from $x$ to $x+1.5$ hours. (It may exactly not be 1.5 hours since the api sometimes may not return the exact timestamp till which you request due to lags and a few datapoints maybe missing here and there which is fine)

Sample flow of incremental update operation:

<table>
<thead>
<tr>
<th>Data Currently in dataframe</th>
<th>Data from latest api pull</th>
<th>Final dataframe state after incremental update</th>
</tr>
</thead>
</table>

(3) Every 5 minutes, after inserting data into the dataframe created in part (2), compute the 30-minute moving averages for each stock’s "c", "l", "h", "o", and "v" values. Store these moving averages incrementally in a separate PySpark dataframe with the schema ["Datetime", "c_MA", "l_MA", "h_MA", "o_MA", "v_MA"], where "Datetime" represents the end timestamp of the moving average window.

Moving Averages for the data called till 2023-10-02 22:42:00

Moving Averages for all the data present after next api call at 2023-10-02 22:46:00

IMPORTANT NOTE: The data should be pulled anytime between 12pm – 4pm between Monday – Friday since that is the only time you will get real time stock data from this API, when the markets are operational. Plan your assignment timeline accordingly. You will most probably not be able to pull data outside these timings accurately and on weekends.
Airflow Data Pipelining:

Task 1 Helloworld (35 pts)

Q1.1 Read through the tutorial slides and install Airflow either on your local laptop or on a VM of GCP. You can also use google cloud composer if you know how to use that. (20 pts)
   1. Provide screenshots of terminals after you successfully start the webserver and scheduler.
   2. Provide screenshots of the web browser after you successfully login and see the DAGs.

Q1.2 Run helloworld with SequentialExecutor (15 pts)
   1. Provide screenshots of Tree, Graph, and Gantt charts after execution. (10 pts)
   2. Explore other features and visualizations you can find in the Airflow UI. Choose two features/visualizations (other than tree, graph, and Gantt), explain their functions and how they help monitor and troubleshoot the pipeline, use helloworld as an example. (5 pts)

Task 2 Build workflows (45 pts)

Q2.1 Implement the DAG below (20 pts)

For each kind of operator, use at least 3 different commands. For example, you can choose sleep, print, count functions for Python operators, and echo, run bash script, run python file for Bash operators.
   1. Provide screenshots of Tree and Graph in airflow. (5 pts)
   2. Manually trigger the DAG and provide screenshots of Gantt. (10 pts)
   3. Schedule the first run immediately and then schedule running the program every 30 minutes. Describe how you decide the start date and schedule interval. Provide screenshots of running history after two repeats (first run + 2 repeats). On your browser, you can find the running history. (5 pts)
Q2.2 Stock price fetching, prediction, and storage every day (25 pts)

(1) Schedule fetching the stock price of [AAPL, GOOGL, META, MSFT, AMZN]. Use Yahoo! Finance data downloader [https://pypi.org/project/yfinance/](https://pypi.org/project/yfinance/).

(2) Preprocess data if you think necessary.

(3) Train/update 5 linear regression models for stock price prediction for these 5 corporates. Each linear model takes the “open price”, “high price”, “low price”, “close price”, “volume” of the corporate in the current day as the feature and predicts the “high price” for the next day.

(4) Calculate the relative errors for the last 5 days, i.e., (prediction made from yesterday’s data for today - actual price today) / actual price today, and update the prediction date and 5 errors into a table, e.g., a csv file.

(5) Provide screenshots of your code, the resultant errors csv, and the Airflow DAG. Describe with screenshots briefly how to build this workflow, e.g., what the DAG is with the various tasks, how you manage the cross tasks communication, how you setup the airflow scheduler...

Pointers for Q2.2: (You have to think how to create the different tasks and the overall Airflow execution DAG based on the question requirements, below are pointers that may give you some ideas)

- Pull historical data for each corporate till the current date and store data for each in a csv. (Set period in history to “max” yf.Ticker(company_tag).history(period=’max’))

- Use this csv to incrementally train the linear regression models for each corporate’s data. The dataset for each model X and y should be created such that each X(d) = [“open price”, “high price”, “low price”, “close price”, “volume”] of date d and the corresponding y = “High price” for date d+1 as stated in the question.

- Assume current date (the date till which you have pulled the data) is d_current.

  For each corporate,
  
  o  Train/Update a model using data from [X,y] till (d_current – i days) and use this model to predict the y for the (d_current – (i-1) day). Repeat the training and prediction for i = 5,4,3,2,1 (make predictions for last 5 days and calculate relative errors for each prediction)

  o  For each value of i keep storing the relative errors in a csv for each model for each corporate on each of the 5 testing days.

  o  The final errors csv should look something like below:

    |   | A   | B          | C          | D          | E          | F          | G          |
    |---|---|---|---|---|---|---|---|
    |   | Date | APPLE | GOOGLE | META | MICROSOFT | AMZN |          |
    | 1 | 0   | 11/26/2022 | -0.00501 | -0.00083 | -0.001 | -0.00167 | 0.00577    |
    | 2 | 1   | 11/27/2022 | -0.00091 | -0.00755 | -0.0097 | -0.00698 | 0.00609    |
    | 3 | 2   | 11/28/2022 | 0.02659  | 0.00703  | 0.00526 | 0.00359  | 0.00829    |
    | 4 | 3   | 11/29/2022 | 0.019557 | 0.00888  | 0.00817 | 0.01145  | -0.0204    |
    | 5 | 4   | 11/30/2022 | 0.00974  | 0.01098  | -0.00189 | 0.00092  | 0.014896   |

To give an example: the value in cell C2 represents the relative error produced by the linear regression model trained on Apple data till 11/25/2022 and the prediction made for 11/26/2022. Similarly, the value in cell C3 represents the relative error produced by the updated linear regression model trained on Apple data till 11/26/2022 and the prediction made for 11/27/2022. The value in cell F6 represents the relative error produced by the linear regression model trained on Microsoft data till 11/29/2022 and the prediction made for 11/30/2022.
Homework Submissions

Streaming Analytics:
- PDF with screenshots of your code, brief explanation of the code workflow and the results i.e. dataframe holding the streamed data and the dataframe holding the moving averages.
- Code file for Streaming Analytics section

Airflow:
- Provide your screenshots, answers and code as mentioned in the individual questions.