Contents

Big Data Final Project_2024 Fall_Intake

Team Members

Aye Kyi Kyi Cho – Std: 1276026

May Khaing Latt – Std: 1276661

Overview

This program performs sentiment analysis on large-scale tweet data stored in CSV files, specifically designed to handle big data efficiently. It leverages the power of **Apache Spark** for distributed processing, enabling the handling of vast datasets with high performance and scalability. The data is loaded into **MongoDB**, a flexible and scalable NoSQL database, which serves as the backend for storing and managing large volumes of tweet data. The program performs sentiment analysis using the **TextBlob** library, processing data at scale and generating insights from extensive tweet datasets. The results, including sentiment distribution and polarity trends, are visualized using **matplotlib** to provide meaningful and actionable insights. This pipeline automates the entire workflow—data ingestion, processing, analysis, and visualization—enabling efficient analysis of big data and providing valuable insights into public sentiment from massive tweet datasets.

Dataset

The Covid 19 dataset used in this project is downloaded from url (hΣps://www.kaggle.com/code/gauravduΣakiit/covid-19-senΘment-analysis-on-completedata/input).

The program expects two CSV files:

- Corona_NLP_test.csv

GitHub Link

https://github.com/maykhainglatt/BigDataProject

5 Vs of Big Data

The **5 Vs of Big Data** in the context of my provided code. My code involves handling large datasets, processing data using **Apache Spark**, storing and retrieving data from **MongoDB**, and performing **sentiment analysis** on text data (tweets). Here's how each **V** relates to my project:

1. Volume:

- **What it means in my code**: My project processes a **large amount of data** from **CSV files** and stores it in a **MongoDB database**. The **volume** refers to the amount of data I am working with. Given I am using **Apache Spark**, I am likely working with large datasets that cannot be processed efficiently using traditional methods or tools.

- **In my case**: The data I am loading could be large, consisting of thousands or even millions of tweets in a CSV format, which are then inserted into **MongoDB**. Apache Spark is used here because it's designed to scale and handle large volumes of data efficiently across multiple machines.

2. Velocity:

- **What it means in my code**: **Velocity** refers to the speed at which data is generated, processed, and analyzed. In my case, I am working with **tweets** that can be generated in real-time. Processing these tweets as quickly as possible is important, especially when dealing with time-sensitive data such as sentiment analysis on the latest tweets.
- **In my case**: My code processes and analyzes tweets based on the date they were posted (TweetAt). The **Velocity** aspect here might not be about real-time processing (unless I am stream-processing new tweets), but it's still relevant since I am grouping and visualizing sentiment data based on tweet date, which might involve processing data that is continuously being updated.

3. Variety:

- **What it means in my code**: **Variety** refers to the different **types of data** I am dealing with. In my case, the data comes from **tweets** that may be structured (metadata about the tweet) and unstructured (the tweet's content itself).

- **In my case**: The **OriginalTweet** column contains free-text, unstructured data (the tweet text itself). You're using **TextBlob** for sentiment analysis to process this unstructured text. Additionally, the data from MongoDB includes metadata such as UserName, ScreenName, and Location, which are structured data fields. Spark helps handle both structured and unstructured data, making it ideal for my project.

4. Veracity:

- **What it means in my code**: **Veracity** refers to the **accuracy** and **trustworthiness** of the data. In my project, the tweets I process might have inconsistencies, spelling errors, sarcasm, or slang that could affect the sentiment analysis.

- **In my case**: Since I am performing **sentiment analysis** using **TextBlob**, it is essential that the data is accurate and clean. You may need to deal with noisy data (e.g., tweets containing irrelevant text, hashtags, or links) that could affect the veracity of the sentiment results. Ensuring that the **OriginalTweet** text is correctly processed (e.g., removing irrelevant characters) is key to improving the accuracy of my analysis.

5. Value:

- **What it means in my code**: **Value** refers to the **usefulness** of the data. The purpose of collecting and analyzing tweets is to extract meaningful insights, such as understanding **public sentiment** towards a specific topic, in this case, **COVID-19** (likely from my earlier mention of Omicron variant tweets).
- **In my case**: After loading the data into MongoDB and processing it with Spark, I perform sentiment analysis to categorize the tweets as **Positive**, **Negative**, or **Neutral**. The **Value** lies in visualizing this sentiment data over time and understanding trends in public opinion. By performing analysis and visualizing the results, I am extracting **valuable insights** from the vast amounts of data stored in MongoDB.

6. Volatility (optional but increasingly included)

- The script handles dynamically changing values (e.g., sentiment by date) and adapts to missing or unexpected data (e.g., missing or invalid dates). Dynamic Changes and Missing Data: In the analyze_and_visualize method, the script handles dynamic changes in sentiment analysis based on tweet data, including missing or invalid dates (via pd.to_datetime(df['TweetAt'], errors='coerce') which converts invalid dates to NaT).

7. Visualization

- While not a core "V," visualization is critical in big data analytics. The program uses Matplotlib to create:
  - Sentiment distribution bar graphs.
  - Average polarity line plots.
  - Negative sentiment intensity bar graphs.

Summary in relation to the code:

- **Volume**: You are dealing with large datasets (tweets) that are stored in MongoDB and processed using Spark.
- **Velocity**: Although not real-time in the strictest sense, I am processing and analyzing data based on the **date** of the tweets, which could be updated frequently as new data arrives.
- **Variety**: My data consists of both structured (metadata) and unstructured (tweet text) data, and Spark handles this efficiently.
- **Veracity**: The accuracy of sentiment analysis relies on the quality of the tweet data (e.g., cleaning the text, handling sarcasm).
- **Value**: The value is in the sentiment analysis, visualizing trends, and extracting insights that are meaningful, such as tracking the sentiment of tweets over time.
- **Volatility:** The code handles volatility by adapting to dynamic sentiment changes and missing dates, while **Visualizing** sentiment distribution, average polarity, and negative sentiment intensity using stacked bar plots and line charts with Matplotlib.

My code demonstrates the key aspects of working with **Big Data** through **Apache Spark**, **MongoDB**, and **TextBlob**, which together help handle the complexity of large, varied, and valuable datasets while ensuring efficient processing and useful outputs.

MongoDB Functions in My Code:

1. MongoClient Initialization:
   - Establishes a connection to the MongoDB server using the provided URI.
   - Selects the database and collection to work with for further operations.
2. Inserting Data into MongoDB:
   - Loads CSV data into MongoDB by converting it to a dictionary format.
   - Inserts multiple documents (records) into the specified MongoDB collection.
3. Deleting Data in MongoDB:
   - Clears any existing data in the collection by deleting all documents before loading new data.
4. Fetching Data from MongoDB:
   - Retrieves documents from the MongoDB collection to be used for further analysis or display.
   - Converts the retrieved data into a list format for processing.
5. Counting Documents in MongoDB:
   - Counts the number of documents in the MongoDB collection to provide insights about the dataset size.
6. Removing Unnecessary Fields:
   - Removes the _id field (automatically added by MongoDB) from the data when displaying or processing it.
7. MongoDB's Role in the Pipeline:
   - **Storage**: MongoDB is used to store both raw and processed data.
   - **Data Management**: MongoDB functions help insert, query, delete, and count the data efficiently.
   - **Integration**: MongoDB works with PySpark for processing large datasets and with TextBlob for sentiment analysis.
   - **Data Preprocessing**: Before performing any analysis, MongoDB is cleared of old data to ensure that only the new data is processed.

These MongoDB functions help manage data effectively within your sentiment analysis pipeline

Spark Data Frame

1. Creation:
   - A Spark DataFrame is created by reading a CSV file using self.spark.read.csv(), which loads the data into a distributed structure for efficient processing.
2. Conversion to Pandas:
   - The DataFrame is converted to a Pandas DataFrame using .toPandas() for easier data manipulation and interaction with libraries like Pandas and Matplotlib.

3. Handling Duplicates:
   - Duplicates are removed in the Pandas DataFrame using .drop_duplicates() to ensure clean data before processing.
4. MongoDB Integration:
   - Data from the Pandas DataFrame is converted into a dictionary with .to_dict() and inserted into MongoDB using insert_many().
5. Sentiment and Polarity Calculations:
   - Sentiment analysis and polarity scores are computed for the 'OriginalTweet' column using the TextBlob library, and these results are added as new columns in the DataFrame.
6. Date Handling:
   - The TweetAt column is converted to a proper datetime format using pd.to_datetime() for further analysis based on time.
7. Data Grouping and Aggregation:
   - The DataFrame is grouped by date and sentiment, and then aggregated using .groupby() to calculate sentiment counts and polarity averages for visualization.
8. Visualization:
   - Aggregated results (like sentiment distribution and polarity trends) are visualized using Matplotlib after processing the Spark DataFrame.
9. Summary
   - **Spark DataFrames** provide a distributed, scalable structure for processing large datasets efficiently, while conversion to **Pandas DataFrame** allows for simpler operations and visualization.

TextBlob

TextBlob is a Python library used for natural language processing (NLP), and it provides an easy way to perform sentiment analysis:

1. **Sentiment Analysis**:
   - **Polarity** ranges from -1 (negative) to 1 (positive), with 0 representing neutral.
   - The get_sentiment() method analyzes the **'OriginalTweet'** text to determine sentiment:
     - Positive: polarity > 0
     - Neutral: polarity = 0
     - Negative: polarity < 0
2. **Workflow**:
   - TextBlob is used in the method get_sentiment() to evaluate the polarity of each tweet's text.

- The calculated **sentiment** ('Positive', 'Neutral', 'Negative') and **polarity** (numerical value) are added to the DataFrame as new columns.

3. **Usage**:
   - **TextBlob** is fast and simple, making it an ideal choice for basic sentiment analysis tasks.
   - The library also supports additional NLP tasks like part-of-speech tagging, noun phrase extraction, and translation.
4. **Integration**:
   - In your code, TextBlob is applied to the **'OriginalTweet'** column in the DataFrame to perform sentiment analysis, and the results are used for visualization and further analysis.

CSV processing in a Spark Master-Slave Configuration

1. Master-Slave Setup Overview:

1. **Master Node**: Manages job scheduling and overall coordination.
2. **Slave Nodes**: Perform actual data processing tasks (data reading, transformation).

2. Controlling CSV Processing:

1. **Master Reads CSV**:
   - Master node reads CSV files using Spark (self.spark.read.csv()).
   - Files can be from local disk or distributed storage.
2. **Distribute Tasks to Slave Nodes**:
   - Spark distributes reading and processing tasks to slave nodes automatically.
   - Slave nodes process different parts or full CSV files in parallel.
3. **Data Transformation**:
   - Transformations (e.g., filtering, aggregation) are distributed across slave nodes.
   - Spark handles the parallelization of these tasks.
4. **Write Results Back**:
   - After processing, results are written back to the storage system (e.g., MongoDB, HDFS).
   - Master node controls the final writing process.

3. Controlling the Process:

1. **Master Handles CSV Read**: Master manages where CSV files are located and starts reading.
2. **Slave Nodes Perform Work**: Slave nodes execute the transformation and processing tasks.
3. **Automatic Task Distribution**: Spark automatically assigns tasks to slave nodes based on load and data partitioning.
4. **Optimizing with Partitioning**: Split large CSV files into partitions for better load distribution.

5. **MongoDB Integration**: Data processed by slave nodes is inserted into MongoDB by the master node.

4. Key Points:

1. **Master Node**: Initializes tasks, reads CSV files, writes results.
2. **Slave Nodes**: Handle parallel processing of CSV data.
3. **Spark Manages Distribution**: Automatic task assignment and load balancing.