Introduction to Data Science
CA3 - Report
1403/01/31

Mohammad Yaser Azad | 810800003
Parham Javan | 810800008

The primary objective of this assignment is to acquaint ourselves with PySpark, a robust tool for handling large-scale data. By leveraging its functionalities, we will analyze real-world datasets. Our focus will be on Spotify song data, where we aim to demonstrate our proficiency in data manipulation, aggregation, and insightful analysis.

Summary

In this report, we present the outcomes of our PySpark exploration, which encompasses both warm-up exercises and the main task involving Spotify data. Our journey begins with the installation and setup of PySpark. We then tackle a series of warm-up exercises, analyzing a stocks dataset. Finally, we delve into the main task, extracting valuable statistics and insights from the Spotify dataset.

Dataset that we use:

1. **Stocks Dataset (stocks.csv)**:

This dataset provides information about stock market performance, including daily opening, closing, highest, and lowest prices, along with trading volume.

To begin our analysis, we'll read the CSV file into a PySpark DataFrame using SparkSession's read.csv() method.

Next, we'll explore the schema of the DataFrame using printSchema() to understand the structure and data types of the columns.

2. **Spotify Dataset (spotify.parquet)**:

The Spotify dataset contains details about songs streamed on Spotify, including attributes like album, artist, musical characteristics, and release date. Our main task involves extracting meaningful insights about song characteristics and trends from this dataset.

Now, let's proceed with the warm-up steps:

Step 1: Reading the CSV file We'll load the stocks dataset from the CSV file into a DataFrame. This step is essential for accessing the dataset and initiating our analysis.

Step 2: Understanding the Schema By printing the schema, we'll gain insights into the columns present in the dataset. Understanding the schema helps us work with the data effectively.

Steps 3 and 4: Filtering Data We'll filter the DataFrame to select records based on specific conditions. For instance, we can focus on subsets of data where closing prices are less than 500 or opening prices are greater than 200 and closing prices are less than 200.

Step 5: Extracting Year from Date To analyze trends on an annual basis, we'll create a new column named 'Year' by extracting the year from the 'Date' column using the withColumn() function.

Step 6: Aggregating Data: We group the DataFrame by the 'Year' column and compute the **minimum volume traded** for each year using the groupBy() and min() functions. This aggregation provides insights into the **lowest trading volumes recorded annually**.

Step 7: Further Aggregation: We extend the aggregation to include both the 'Year' and 'Month' columns. For each combination, we calculate the **highest low price**. This additional analysis offers granularity in identifying the lowest prices within specific time periods.

Step 8: Summary Statistics: We compute the **mean** and **standard deviation** of the 'High' price over the entire dataset using the select(), mean(), and stddev() functions. These summary statistics offer insights into the **central tendency and dispersion of high prices**.

Main Task

1. **Checking the Schema**:

We'll begin by creating a SparkSession and reading the provided Spotify dataset in Parquet format into a DataFrame. Printing the schema will allow us to understand the structure and data types of the dataset, providing insights into the available columns and their formats.

2. **Pre-processing Columns**:

In this step, we'll focus on the 'release_date' column.

We'll convert it to the date type using the withColumn() function.

This conversion ensures that the 'release_date' column is in a suitable format for further analysis

3. **Aggregation, Filtering, and Transformation**:

Here's what we'll do:

Aggregate statistics for danceability and energy features by year. This will help us understand the average and variation of these characteristics over time.

Filter the dataset to exclude songs with explicit content. This ensures that our analysis focuses on non-explicit songs only.

Convert the duration from milliseconds to minutes. Additionally, we'll create a binary feature called 'long_song' based on a specified threshold (e.g., 15 minutes). This will provide insights about song length.

Calculate the number of long songs in the dataset, giving us information about the prevalence of longer-duration songs.

4. **Dealing with Array Columns**: We'll handle array columns, such as 'artists':

First, we'll split the string to create an array, separating each artist.

Next, we'll explode the array to obtain each artist in a separate row. This facilitates analysis at the artist level.

Finally, we'll drop the intermediate column to maintain data integrity.

5. **Top-K Records**: To identify the top songs based on a specific feature (e.g., valence), we'll use the orderBy() function to sort the DataFrame in descending order.

Then, we'll use the limit() function to retrieve the top K records.

This analysis will provide insights into the most positively or negatively perceived songs.

Question1

1. **Hadoop**:

Hadoop is designed for distributed storage and processing of large datasets across clusters of computers.

Key components of Hadoop include:

Hadoop Distributed File System (HDFS): It stores data across multiple nodes in a Hadoop cluster. HDFS provides high availability and fault tolerance by replicating data blocks across nodes.

Yet Another Resource Negotiator (YARN): YARN is the resource management layer in Hadoop. It manages resources and schedules tasks across the cluster, allowing different data processing engines to run on the same Hadoop cluster.

Map-Reduce: Map-Reduce is a programming model for processing and generating large datasets in parallel across a Hadoop cluster. It involves two main phases:

Map phase: Processes input data and generates key-value pairs.

Reduce phase: Aggregates and summarizes the intermediate data.

2. **Spark**:

Spark is a fast and versatile cluster computing system that provides in-memory processing capabilities for big data workloads.

Key components of Spark include:

Spark Core: Provides distributed task scheduling, fault recovery, and data sharing among different parallel computations.

Spark SQL: Enables querying data using SQL, DataFrame API, and integration with external data sources.

Spark Streaming: Allows real-time processing of streaming data by dividing data streams into micro-batches and processing them using Spark's batch processing capabilities.

Directed Acyclic Graph (DAG) Execution Engine: Spark uses a DAG execution engine to optimize data processing workflows.

3. **Lazy Evaluation**:

Lazy evaluation is a feature in Spark that defers executing transformations until an action is executed.

For example, you can apply a transformation (e.g., filtering a DataFrame) using df.filter(), but Spark won't actually perform the filtering until you run an action (e.g., displaying the DataFrame using df.show()).

Why use lazy evaluation? By postponing transformations, Spark avoids bringing the entire DataFrame into memory immediately, saving cluster capacity.

When combined with the **catalyst optimizer**, Spark optimizes the execution plan by evaluating all pending transformations together and determining the most efficient way to combine them.
lazy evaluation" refers to a computational model where Instead of executing an operation immediately when it is defined, lazy evaluation defers the computation until the result is required by another operation to optimize performance and resource utilization. It allows these frameworks to build a directed acyclic graph (DAG) of transformations and only execute the computations when an action is triggered.

Question2

Parquet files address challenges associated with large datasets. Unlike CSV files, Parquet files adopt a columnar format, storing values from the same column together. This design enables more efficient compression and encoding techniques on a per-column basis, reducing the amount of data read from disk. Specifically:

Efficient Storage: Parquet files outperform CSV files by compressing individual columns, resulting in better overall compression ratios. This approach contrasts with compressing the entire CSV file as a single unit.

Schema Self-Description: Parquet files include metadata describing the data schema. This self-descriptive feature allows for schema evolution, enabling the addition or modification of columns without rewriting the entire dataset.

Predicate Pushdown: Parquet files support predicate pushdown, allowing query engines to apply filters directly at the storage layer. This optimization minimizes irrelevant data reads from disk. In contrast, CSV files lack native support for predicate pushdown, potentially leading to inefficient memory-based filtering for large datasets.

In summary, Parquet files enhance storage efficiency, schema flexibility, and query performance when dealing with substantial data volumes.

Question3

Identify the intermediate result (RDD or Data Frame) that you want to save as a checkpoint.

Invoke the checkpoint() method on that result. This action will initiate the check pointing process.

Specify a **checkpoint directory** where Spark will store the checkpoint data. Ensure that this directory resides on a reliable storage system accessible to all nodes in your Spark cluster.

By adhering to these guidelines, you can effectively manage checkpoints in your Spark application. Remember that check pointing enhances fault tolerance and facilitates recovery during job execution.

Set Checkpoint Directory in Spark Configuration:

```
val spark = SparkSession.builder()
```

```
.appName("CheckpointExample")

.config("spark.sql.streaming.checkpointLocation", "hdfs://path/to/checkpoint/dir")

.getOrCreate()
```

Trigger Checkpointing:

```
val intermediateRDD = sourceRDD.map(...)

intermediateRDD.checkpoint()

intermediateRDD.count() // Trigger action to save checkpoint
```

Question4

By utilizing Spark Structured Streaming with the Parquet file format, partitioning data by date, and employing efficient filtering through Spark SQL, you can achieve faster and more efficient filtering of streaming data based on specific columns compared to traditional filtering methods.

Question5

Data Size:

Pandas: Well-suited for medium-sized datasets that fit into memory (typically a few gigabytes).

PySpark: Designed for large-scale datasets that exceed single-machine memory capacity, distributing computations across a cluster of machines.

Processing Complexity:

Pandas: Excellent for exploratory data analysis, data cleaning, and manipulation. Provides a rich set of functions for data wrangling, grouping, filtering, and visualization.

PySpark: Ideal for complex tasks like large-scale aggregations, joins, and machine learning on big data.

User Experience:

Pandas: Offers a user-friendly and intuitive interface, akin to working with a spreadsheet. Familiar to users accustomed to tools like Excel.

PySpark: Provides scalability and performance benefits but has a steeper learning curve. Users must grasp distributed computing concepts and Spark's API to fully utilize its capabilities.