**Problem 1B**

**Frequency of words**

**Submitted By –**

**Himanshi Verma**

Process followed to create the Apache Spark (GCPDataproc) cluster (problem1b-cluster) and execute the job (Problem1B-1.0-SNAPSHOT.jar) file on it.

1. Searched for dataproc on search bar after selecting the project in GCP

A computer screen shot of a computer

Description automatically generated

Fig. 1: GCP DataProc create cluster

1. Customised the dataproc cluser , filled in all details and I have selected one node cluster only therefore 1 master ,0 Workers cluster type and click on create

A screenshot of a computer

Description automatically generated

Fig 2: choose cluster properties and create

1. Once the cluster is created it appears like this.

A screenshot of a computer

Description automatically generated

Fig. 3: Dataproc created cluster

1. Opened the cluster and selected VM Instances –

A screenshot of a computer

Description automatically generated

Fig. 4 : Available cluster Details

1. In order to interact with this VM instance of our cluster, open the instance, now we need to SSH in the VM .

A screenshot of a computer

Description automatically generated

Fig. 5 : Selected VM Details

1. After clicking on SSH , an SSH window opens up ,authorize it. We will use this window to upload our jar file and the input.txt file . Here we will run our program.

A screenshot of a computer

Description automatically generated

Fig. 6: SSH Window Authorize

1. After authorizing ,interact with the VM.

A screenshot of a computer

Description automatically generated

Fig. 7 : SSH window after authorization

1. I have uploaded the reut2-009.sgm file which my java spark program will read . I have also selected and uploaded the jar file. Problem1B-1.0-SNAPSHOT.jar

A screenshot of a computer

Description automatically generated

Fig 8 : Upload files to VM

1. Once these files are uploaded , I switched to root user and listed all files in current directory

Commands used - **sudo su**

**ls -lrt**

A screenshot of a computer

Description automatically generated

Fig. 9 : List all files in directory

1. I ran the command :-

**spark-submit --class com.dal.assignment2.Problem1B Problem1B-1.0-SNAPSHOT.jar**

1. Here Problem1B is the class file that contains main method of my application and Problem1B-1.0-SNAPSHOT.jar is the packaged jar file . Output of successful execution of program is displayed -

A screenshot of a computer

Description automatically generated

Fig. 10: Jar ran successfully

**Minimum frequency –** The output of words with minimum frequency i.e 1 is displayed -

A screen shot of a computer program

Description automatically generated

Fig. 11: Minimum frequency words

**Maximum frequency –** The output of words with max frequency i.e “mln” is displayed and its count 1295.

A black screen with white text

Description automatically generated

Fig. 12: Word with maximum frequency

**Flowchart -**

A diagram of a work flow

Description automatically generated

Fig. 13 : Flowchart for complete program

**Algorithm –**

Below is the algorithm for my complete Spark application to analyze word frequency:

1. **Start the Spark Application**:
   * Initialize a SparkSession.
   * Set the application name and specify the execution mode (e.g., local mode).
2. **Read Input Data**:
   * Read text data from the reut2-009.sgm file using the **sparkSession.read().text()** method. Store the data in a DataFrame (**df**).
3. **Text Preprocessing**:
   * Use regular expressions to clean the text data. Replace HTML tags, special characters, and unwanted symbols with spaces or remove them altogether.
   * Update the DataFrame column containing the text data with the cleaned version using **withColumn()** and **regexp\_replace()** functions.
4. **Split Text into Words**:
   * Split the cleaned text data into individual words.
   * Use the **split()** function to separate words based on spaces.
   * Explode the array of words to create a new row for each word using the **explode()** function.
5. **Filter Stop Words**:
   * Define a list of stop words.
   * Filter out stop words from the list of words using the **filter()** function.
6. **Word Frequency Analysis**:
   * Group the words by their occurrences using the **groupBy()** function.
   * Count the frequency of each word using the **count()** function.
   * Store the word counts in a new DataFrame (**wordCounts**).
7. **Identify Words with Maximum Frequency**:
   * Group the word counts DataFrame (**wordCounts**) and find the maximum count using the **max()** function.
   * Join the maximum count with the original word counts DataFrame to get the word(s) with the maximum frequency.
   * Select the word(s) with the maximum frequency and display them.
8. **Identify Words with Minimum Frequency**:
   * Similar to the previous step, find the minimum count of word occurrences.
   * Join the minimum count with the original word counts DataFrame to get the word(s) with the minimum frequency.
   * Select the word(s) with the minimum frequency and display them.
9. **Stop the Spark Application**:
   * Stop the SparkSession to release resources.
10. **End**:
    * End of the algorithm.

**Code Walkthrough–**

The code has two files: Problem1B.java and WordAnalyzer.java . Let's talk about each one:

**Problem1B:**

This one starts the program. It reads information from a file named "reut2-009.sgm". It gets rid of common words and extra symbols from the text. Then, it asks WordAnalyzer to count how often each word appears. Finally, it finishes by closing down the Spark session.

**WordAnalyzer:**

It's all about counting how many times each word shows up. It gets a bunch of words to look at, called a Dataset<String>. It groups the words together by how often they occur. Then, it figures out which word(s) appear the most and shows that. It also finds the word(s) that show up the least and shows that too. At the end, it shows how many times each word appears.

**Apache Spark Functions -**

In the WordAnalyzer class, I have used functionalities provided by Apache Spark for data processing:

* **Dataset API:** I have utilized the Dataset API to represent the collection of words as a Dataset<String>. This API offers a structured interface for working with distributed data in Spark.
* **Grouping and Aggregation:** I have used the groupBy and count functions to group the words by their frequency and count their occurrences. This involves aggregating data based on specific criteria.
* **SQL Functions:** I have made use of functions provided by the org.apache.spark.sql.functions package for data transformation tasks. Specifically, I have utilized the **max and min** functions to find the word(s) with the maximum and minimum frequencies, respectively.
* **DataFrame Operations:** I have utilized DataFrame operations like select to manipulate and select specific columns from the Dataset. This allows for extracting and processing relevant information from the data.

**References -**

1. Collibra. Stop Words [Online]. Available: <https://productresources.collibra.com/docs/collibra/latest/Content/Settings/ServicesConfiguration/co_stop-words.htm>.
2. Draw.IO. Flowchart[Online]. Available: <https://app.diagrams.net/>