Assignment 2 – Word Count

Due date: Friday 4/30/2021 at 11:59PM Pacific Time

Remember – when in doubt, read the documentation first. It's always helpful to search for the class that you're trying to work with, e.g. pyspark.sql.DataFrame.

Resources:

- PySpark API: https://spark.apache.org/docs/latest/api/python/index.html
- Spark DataFrame: https://spark.apache.org/docs/latest/sql-programming-guide.html
- PySpark_SQL_Cheat_Sheet_Python.pdf uploaded on canvas.

NOTE – Always explicitly use `print` function to print the output of a task. Unlike Jupyter notebook, the exported script fails to print an expression statement.

Task 1 – Remove punctuation and print the first 25 rows (2 points)

- (From JupyterLab terminal) Copy data file (BookReviews_1M.txt) from local FS to HDFS
- Start spark session
- Read data from HDFS into Spark DataFrame
- Remove punctuations using the `removePunctuation` function imported from wordcount_utils.py. *Hint* – Pass the "value" column to `removePunctuation`
- Print the first 25 rows of the resulting dataframe

Task 2 – Find and print top 100 words based on count (5 points)

- Split each sentence(row) into words based on the delimited space (" ")
- Put each word in each sentence into its own rows and assign the result to a new dataframe
- Remove all empty lines in the dataframe (due to empty lines and/or words)
- Group rows in the dataframe by unique words and count the rows in each group
- Sort the word count dataframe
- Print the first 100 rows of the dataframe sorted by word count

Task 3 – Record execution time for 1, 2 and 4 cores (2 points)

- Repeat task 1 and 2 for 1, 2 and 4(if available) cores. The following code snippet shows how to change core count:

```
conf = pyspark.SparkConf().setAll([('spark.master', 'local[coreCount]'),
                                   ('spark.app.name', 'Word Count')])
```

- Repeat the experiment 3 times for each case and record execution time - from reading the data into a dataframe to printing the top 100 words. *Hint* – use `time.time()` in python
- Record execution time in the below format and save it to execution_time.csv:
  - #cores, time0, time1, time2, mean, stdev

Task 4 – Save the sorted word counts to HDFS as a CSV file (1 point)

NOTE: Spark uses a distributed memory system, and stores working data in fragments known as "partitions". This is advantageous when a Spark cluster spans multiple machines, as each machine will only require part of the working data to do its own job. By default, Spark will save each of these data partitions into an individual file to avoid I/O collisions. We want only one output file, so we will need to fuse all the data into a single partition first.

1. Coalesce the sorted dataframe(task 2) to one partition. This makes sure that all our results will end up in the same CSV file. Save the 1-partition dataframe to HDFS using the DataFrame.write.csv() method. Use the following code snipped:

   ```
   <<sortedDataFrame>>.coalesce(1).write.csv("hdfs:///wordCountsSorted.csv",
   header=True, mode="overwrite")
   ```

2. Copy the file from HDFS to local file system:

- Run `hadoop fs -ls /` to list the root directory of the HDFS. You should see the CSV file that you have saved. Counterintuitively, this CSV file is a folder, which contains individually saved files from each partition of the saved dataframe
- Run `hadoop fs -ls /wordCountsSorted.csv/` to see what is inside the saved folder. Since we made sure to coalesce our dataframe to just one partition, we should expect to find only one saved partition in this folder, saved also as a CSV. Note the name of this file, it should look something like part-00000-xx.....xx.csv.
- Run the following command to copy the results CSV from HDFS to the current folder on your local file system. Rename it to something simple like results.csv :

  ```
  hadoop fs -copyToLocal /wordCountsSorted.csv/part-00000-*.csv .
  ```

- We want you to submit a CSV containing the first 101 rows of the results file(header row + top 100 words by count). To do this, use the command (You can also do so manually since CSV files are in plain text):
  `head -n 101 results.csv > 101_rows.csv`

Instructions for submission:

1. Save the Python notebook as Python file (.py) – File > Export Notebook as > Executable Script
2. Submit the Python notebook (.ipynb or convert to PDF), 101_rows.csv and execution_time.csv to "PA1 notebook" and Python file(.py) to "PA1" on Gradescope