Homework #3 - Spark

Due:11/30/23

Content Covered
Spark

Homework Overview

For this homework you will get a bit of experience writing and running Spark code, and performing some follow-up analysis. In general, you will install and set up Spark, write a basic word count program, and then answer some questions about data flow in Spark applications.

This is an Extra Bonus (5%) assignment

General Homework Requirement

1. **Work Environment:** This homework can be written in Pyspark or Scala .
2. **Programming:** You can use Jupyter Notebook ,Jupyter Lab. Or Google collab
3. **Academic Integrity: You will get an automatic F for the course if you violate the academic integrity policy.**
4. **Teams:** This homework is **an individual assignment**. You are not permitted to work with anyone else on this assignment. All work submitted must be yours and yours alone.

Submission Format

Your submission should be a single zip file that contains a src/ directory for your code, and a single PDF report answering the following questions. Submissions must be made on Brightspace by 11/30/23 @ 11:59PM. **No late submission will be accepted**, If you are late 1 min we will give you 0. Try to upload your HW-3 at least a few hours ago of the Due date and time.

- Your src/ directory should contain your code files and your input dataset (2 txt files from project gutenberg), and your list of stopwords.
- Your report should contain the answers to each of the questions below, including any required screenshots of your application output.

Setup

To prepare your development environment for this homework you must first install and set up PySpark. To install PySpark, follow the instructions here:
https://spark.apache.org/docs/latest/api/python/getting_started/install.html

Once you have Spark setup, you must also get the data you will be using for the homework. Go to Project Gutenberg (https://www.gutenberg.org/ebooks/) and download 2 different books as plain text files. **You may use the same books you used from HW #2.**

Implement and Analyze Word Count [100 points]

Basic Word Count

[Implementation of basic Word Count 50 points]

You must write a basic word count program that reads in a single text file, counts the number of occurrences of each word in that text file, and outputs the result back to a text file as a list of key-value pairs, where the key is the word, and the value is the number of times the word occurred in the text file.

[Implementation of Extending Word Count 5X4=20 points]

In addition to basic word counting extend your code , which must also do the following:

1. It must be case insensitive (see **`lower()`** in Python)
2. It must ignore all punctuation (see, for example, **`translate()`** in Python)
3. It must ignore stop words (see **`filter()`** in Spark)
4. The output must be sorted by count in descending order (see **`sortBy()`** in Spark)

To accomplish this you will use a combination of basic Python, and RDD operations in PySpark.

The following programming guide goes over basics of getting started with Spark and should contain everything you need to complete this part of the homework:
https://spark.apache.org/docs/latest/rdd-programming-guide.html

Analysis [10X3 =30 points]

Answer the following questions based on your WordCount application.

1. In the PySpark REPL/ Jupyter notebook , run your basic word count program on a single text file.
    a. What are the 25 most common words? Include a screenshot of program output to back-up your claim.
    b. How many stages is execution broken up into? Explain why. Include a screenshot of the DAG visualization from Spark's WebUI to back-up your claim.

2. In the PySpark REPL/Jupyter notebook, run your word count extended program on all 2 text files.
    a. What are the 25 most common words? Include a screenshot of program output to back-up your claim.
    b. How many stages is execution broken up into? Explain why. Include a screenshot of the DAG visualization from Spark's WebUI to back-up your claim.

Answer the following based on your knowledge of both MapReduce and Spark:
:

```
 1  lines = sc.textFile(file)
 2  links = lines.map(lambda urls: parseNeighbors(urls)) \
 3               .groupByKey()
 4               .cache()
 5  N = links.count()
 6  ranks = links.map(lambda u: (u[0], 1.0/N))
 7
 8  for i in range(iters):
 9    contribs = links.join(ranks) \
10         .flatMap(lambda u: computeContribs(u[1][0], u[1][1]))
11
12    ranks = contribs.reduceByKey(lambda a,b: a+b) \
13         .mapValues(lambda rank: rank * 0.85 + 0.15*(1.0/N))
14  return ranks
```

3. Given the above spark application, draw the lineage graph DAG for the RDD **`ranks`** on line 12 when the iteration variable **`i`** has a value of 2. Include nodes for all intermediate RDDs, even if they are unnamed.