03 PySpark Text Analysis and Word Counting Assignment

PySpark Text Analysis and Word Counting Assignment

Assignment Overview

In this assignment, you will work with text data to perform word frequency analysis using Apache Spark's distributed computing capabilities. You'll implement two different approaches to counting words in a large text file, learning fundamental concepts of text processing and data transformation in distributed systems.

Learning Objectives

By completing this assignment, you will:

- Master fundamental PySpark operations for text processing
- Understand different approaches to word counting in distributed systems
- Learn to use regular expressions for text normalization
- Practice RDD transformations: `flatMap`, `map`, `reduceByKey`, and `sortByKey`
- Compare action vs. transformation operations in Spark
- Work with key-value pairs and sorting operations

Dataset Description

You will be working with a text file containing a complete book or large text document. This could be a classic novel, collection of documents, or any substantial text corpus.

Dataset URL:

1. https://raw.githubusercontent.com/KirkYagami/learn_databricks/main/04_Spark/03_Datasets/book.txt
2. Pride and Prejudice: https://www.gutenberg.org/files/1342/1342-0.txt

Data Format

The dataset is a plain text file (.txt) containing:

- Multiple lines of text
- Various punctuation marks and special characters
- Mixed case letters
- Potentially non-ASCII characters

Assignment Tasks

Task 1: Basic Word Count

Write a PySpark program that counts the frequency of each word in the text file using the simplest approach.

Requirements:

1. Set up a Spark configuration with application name "WordCount"
2. Read the text file into an RDD
3. Split each line into individual words using whitespace as delimiter
4. Count the frequency of each word using `countByValue()`
5. Handle character encoding properly (ASCII encoding with ignore option)
6. Display results showing each word and its count
7. Filter out empty words from the output

Expected Approach:

- Use `flatMap()` to split lines into words
- Use `countByValue()` to count word frequencies
- Handle character encoding issues gracefully

Task 2: Advanced Word Count with Normalization and Sorting

Write an improved PySpark program that normalizes text and sorts results by frequency.

Requirements:

1. Create a text normalization function that:
   - Converts all text to lowercase
   - Removes punctuation and special characters
   - Splits text using word boundaries (not just whitespace)
2. Set up Spark configuration with the same application name
3. Apply the normalization function to process the text
4. Use the map-reduce pattern to count words
5. Sort results by word frequency (ascending order)
6. Display results with proper formatting showing frequency and word

Expected Approach:

- Implement a `normalizeWords()` function using regular expressions
- Use `flatMap()` with the normalization function
- Use `map()` and `reduceByKey()` for counting
- Use key swapping and `sortByKey()` for sorting by frequency
- Handle character encoding properly

Task 3: Comparison and Analysis

Answer the following questions in markdown cells or a separate document:

1. **Method Comparison**:
   - What are the key differences between using `countByValue()` vs. `map()` + `reduceByKey()`?
   - Which approach is more suitable for very large datasets and why?
2. **Text Normalization Impact**:
   - Compare the results from Task 1 and Task 2
   - How does normalization affect the word counts?
   - Give examples of words that are counted differently
3. **Performance Considerations**:
   - Explain why `reduceByKey()` is preferred over `groupByKey()` for this operation
   - What is the difference between transformations and actions in your code?
   - Identify all transformations and actions used in both tasks
4. **Regular Expression Analysis**:
   - Explain what the regex pattern `r'\W+', re.UNICODE` does
   - Why is this better than simple `.split()` for text processing?
5. **Sorting Strategy**:
   - Explain the key swapping technique used in Task 2: `(word, count) → (count, word)`
   - Why is this necessary for sorting by frequency?

Technical Requirements

Environment Setup

- Google Colab environment
- PySpark will be installed directly in the notebook
- Access to the provided text dataset (hosted on GitHub)

Installation Instructions for Google Colab

Add these commands at the beginning of your notebook:

PYTHON

```python
# Install PySpark
!pip install pyspark

# Import required libraries
import re
from pyspark import SparkConf, SparkContext
import os
```

Data Access

Since you're using Google Colab, you'll need to:

1. Download the text file from the GitHub URL directly in your notebook
2. Use the appropriate file path for Colab's file system

Example for loading data in Colab:

```python
# Download dataset (replace with actual GitHub URL)
!wget "YOUR_GITHUB_DATASET_URL" -O book.txt

# Read the file in your Spark program
input = sc.textFile("book.txt")
```

File Structure

Your submission should include:

```
assignment_submission/
├── Word_Count_Analysis.ipynb (Jupyter notebook with both tasks)
├── analysis_answers.md (or include answers in notebook markdown cells)
└── README.md (optional)
```

Google Colab Guidelines

- Use separate code cells for Task 1 and Task 2
- Include markdown cells to explain your approach and findings
- Make sure to properly initialize Spark in each section
- Remember to stop the SparkContext when switching between tasks
- Use clear section headers in your notebook
- Include sample outputs showing the differences between approaches

Code Guidelines

- Use meaningful variable names
- Include comments explaining complex operations
- Use the appropriate file path for Google Colab environment
- Follow Python PEP 8 style guidelines
- Include markdown cells to document your process and findings
- Make sure to properly initialize and stop SparkContext between tasks

Expected Output Format

Sample Output for Task 1 (Basic Word Count):

```
the 1234
and 892
to 654
of 543
...
```

Sample Output for Task 2 (Normalized and Sorted):

```
i:      1
me:     2
us:     3
our:        4
...
the:        1234
```

Submission Instructions

1. Complete both word counting tasks in Google Colab
2. Test your code with the provided text dataset
3. Answer all questions in Task 3 (can be in markdown cells or separate document)
4. Include sample outputs and analysis of the differences
5. Download your completed notebook (.ipynb file)
6. Submit the notebook file and any additional documents

Evaluation Criteria

- **Correctness**: Programs produce accurate word counts and handle text properly
- **Code Quality**: Clean, readable, well-commented code following best practices
- **Text Processing**: Proper implementation of normalization and encoding handling
- **Analysis**: Thoughtful comparison of approaches and understanding of concepts
- **Documentation**: Clear explanations in markdown cells with sample outputs

Hints and Tips

1. Pay attention to character encoding issues - use ASCII encoding with 'ignore' option
2. Test your normalization function with sample text before processing the entire file
3. Remember that `countByValue()` returns results to the driver - suitable for smaller datasets
4. Use `flatMap()` when you need to split each input element into multiple output elements
5. The key swapping technique `(word, count) → (count, word)` enables sorting by frequency
6. In Google Colab, you may need to restart and clear output between major code changes
7. Use `sc.stop()` before creating a new SparkContext if you need to restart
8. Check your outputs for empty strings and filter them appropriately

Common Pitfalls to Avoid

- Forgetting to handle character encoding (ASCII with ignore)
- Not filtering out empty words or whitespace-only results
- Using `groupByKey()` instead of `reduceByKey()` for aggregation
- Not properly normalizing text (case sensitivity, punctuation)
- Creating multiple SparkContexts without properly stopping the previous one
- Assuming all text is in ASCII format without proper encoding handling
- Not understanding the difference between actions and transformations

Extension Challenges (Optional)

If you finish early, try these additional challenges:

1. **Top N Words**: Modify Task 2 to show only the top 20 most frequent words
2. **Word Length Analysis**: Create a program that analyzes the distribution of word lengths
3. **Stop Words Filtering**: Remove common stop words (the, and, or, etc.) from your analysis
4. **Character Count**: Implement a character frequency counter alongside word counting

Resources

- PySpark Documentation
- Python Regular Expressions Guide
- Spark Programming Guide
- Text Processing with Spark