Report

CSL7110: Machine Learning With Big Data

Assignment 1: MapReduce and Apache Spark

Submitted by:

- **Name:** Akshat Jain
- **Roll No:** M25CSA003
- **Instructor:** Dr. Dip Sankar Banerjee
- **Date:** February 14, 2026
- **GitHub:** https://github.com/m25csa003-glitch/CSL7110_Assignment1

Q1. Execution of WordCount Example in Hadoop

1. Objective: To demonstrate the working of the standard Apache Hadoop MapReduce WordCount example on a Single Node Cluster setup.

2. Input Data: I created a text file named `input.txt` with the following content:

"Hello Hadoop Hello World Spark is fast Hadoop is powerful"

3. Execution Steps:

- **Step 1:** The input file was created and uploaded to the HDFS directory `/input_dir`.

  Bash

  ```
  hdfs dfs -put input.txt /input_dir/
  ```

- **Step 2:** The WordCount job was executed using the Hadoop example JAR provided with the installation. **Command:**

  Bash

  ```
  hadoop jar
  /opt/homebrew/Cellar/hadoop/3.4.2/libexec/share/hadoop/mapreduce/hado
  op-mapreduce-examples-3.4.2.jar wordcount /input_dir /output_dir
  ```

4. Observations & Output: The MapReduce job successfully processed the input file. It split the text, mapped the words, shuffled them, and reduced them to calculate the frequency of each word.

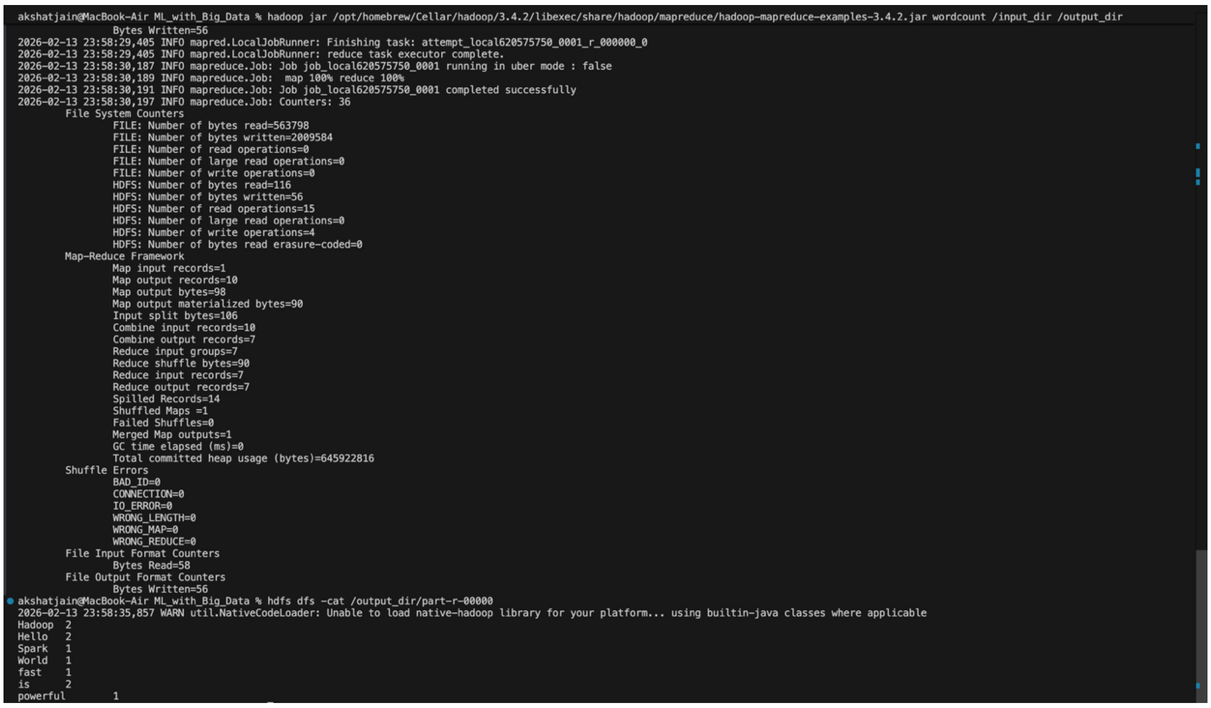
Command to view output:

Bash
```
hdfs dfs -cat /output_dir/part-r-00000
```

Final Output:

Plaintext
```
Hadoop  2
Hello   2
Spark   1
World   1
fast    1
is      2
powerful 1
```

5. Execution Screenshot:



```
akshatjain@MacBook-Air ML_with_Big_Data % hadoop jar /opt/homebrew/Cellar/hadoop/3.4.2/libexec/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.4.2.jar wordcount /input_dir /output_dir
                Bytes Written=56
2026-02-13 23:58:29,405 INFO mapred.LocalJobRunner: Finishing task: attempt_local620575750_0001_r_000000_0
2026-02-13 23:58:29,405 INFO mapred.LocalJobRunner: reduce task executor complete.
2026-02-13 23:58:30,187 INFO mapreduce.Job: Job job_local620575750_0001 running in uber mode : false
2026-02-13 23:58:30,189 INFO mapreduce.Job:  map 100% reduce 100%
2026-02-13 23:58:30,191 INFO mapreduce.Job: Job job_local620575750_0001 completed successfully
2026-02-13 23:58:30,197 INFO mapreduce.Job: Counters: 36
        File System Counters
                FILE: Number of bytes read=563798
                FILE: Number of bytes written=2009584
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=116
                HDFS: Number of bytes written=56
                HDFS: Number of read operations=15
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=4
                HDFS: Number of bytes read erasure-coded=0
        Map-Reduce Framework
                Map input records=1
                Map output records=10
                Map output bytes=98
                Map output materialized bytes=90
                Input split bytes=106
                Combine input records=10
                Combine output records=7
                Reduce input groups=7
                Reduce shuffle bytes=90
                Reduce input records=7
                Reduce output records=7
                Spilled Records=14
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=0
                Total committed heap usage (bytes)=645922816
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=58
        File Output Format Counters
                Bytes Written=56
akshatjain@MacBook-Air ML_with_Big_Data % hdfs dfs -cat /output_dir/part-r-00000
2026-02-13 23:58:35,857 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Hadoop  2
Hello   2
Spark   1
World   1
fast    1
is      2
powerful        1
```

Q2. Map Phase Input and Output in WordCount

Input to Mapper

The input pairs received by the Mapper are:

```
(LongWritable, Text)
```

where:

- Key: Byte offset of the line (LongWritable)

- Value: Line of text (Text)

Example input:

```
(0, "We’re up all night till the sun")
(31, "We’re up all night to get some")
(63, "We’re up all night for good fun")
(95, "We’re up all night to get lucky")
```

Output from Mapper

The Mapper splits each line into words and emits:

```
(Text, IntWritable)
```

Each word is emitted with value 1.

Example output:

```
(We’re,1)
(up,1)
(all,1)
(night,1)
(till,1)
(the,1)
(sun,1)
...
```

Data Types Used

Phase	Key Type	Value Type
Input	LongWritable	Text
Output	Text	IntWritable

Observation

The byte offset is passed to the Mapper but is not used in WordCount. Each word is emitted with a count of 1.

Result

The Mapper correctly converts input text lines into word-count pairs for further processing in the Reduce phase.

Q3. Reduce Phase Input and Output in WordCount

Input to Reducer

The Reducer receives grouped values for each unique word.

Format:

```
(Text, Iterable<IntWritable>)
```

Example input:

```
(up, [1,1,1,1])
(to, [1,1,1])
(get, [1,1])
(lucky, [1])
```

Each list contains the number of occurrences of the word.

Output from Reducer

The Reducer sums the values and produces:

```
(Text, IntWritable)
```

Example output:

```
(up, 4)
(to, 3)
(get, 2)
(lucky, 1)
```

Data Types Used

Phase	Key Type	Value Type
Input	Text	Iterable<IntWritable>
Output	Text	IntWritable

Observation

All values corresponding to the same key are grouped together before being passed to the Reducer.

Result

The Reduce phase successfully aggregates word counts and generates final frequency output.

Q4. Data Types in Mapper and Reducer Classes

Mapper Class Definition

In the WordCount program, the Mapper class is defined as:

```
public static class Map
    extends Mapper<LongWritable, Text, Text, IntWritable> {

    @Override
    public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {
        ...
    }
}
```

Here:

- Input Key Type: LongWritable
- Input Value Type: Text
- Output Key Type: Text
- Output Value Type: IntWritable

Reducer Class Definition

The Reducer class is defined as:

```
public static class Reduce
    extends Reducer<Text, IntWritable, Text, IntWritable> {

    @Override
    public void reduce(Text key,
                       Iterable<IntWritable> values,
                       Context context)
        throws IOException, InterruptedException {
        ...
    }
}
```

Here:

- Input Key Type: Text
- Input Value Type: Iterable<IntWritable>
- Output Key Type: Text
- Output Value Type: IntWritable

Job Configuration

The output key and value classes are set as:

```
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
```

Observation

The data types used in the Mapper and Reducer match the input and output requirements of the WordCount program.

Result

The placeholders in the WordCount program are correctly replaced with appropriate Hadoop data types.

Q5. Implementation of map() Function

The following `map()` function removes punctuation, splits the line into words, and emits each word with count 1.

```
@Override
public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {

    String line = value.toString();

    // Remove punctuation
    line = line.replaceAll("[^a-zA-Z ]", "");

    // Convert to lowercase
    line = line.toLowerCase();

    // Split into words
    StringTokenizer tokenizer = new StringTokenizer(line);

    while (tokenizer.hasMoreTokens()) {
        String wordStr = tokenizer.nextToken();
        Text word = new Text(wordStr);
        context.write(word, new IntWritable(1));
    }
}
```

Observation

Punctuation is removed using `replaceAll()` and words are separated using `StringTokenizer`.

Result

The `map()` function correctly generates key-value pairs in the form `(word, 1)` for WordCount.

Q6. Implementation of reduce() Function

The `reduce()` function aggregates the counts of each word generated by the mapper.

```
@Override
public void reduce(Text key, Iterable<IntWritable> values, Context context)
        throws IOException, InterruptedException {

    int sum = 0;

    for (IntWritable val : values) {
        sum += val.get();
    }

    context.write(key, new IntWritable(sum));
}
```

Observation

All values corresponding to a word are iterated and summed to compute the total frequency.

Result

The reduce function correctly outputs the final word count in the form `(word, totalCount)`.

Q7. Execution of WordCount on 200.txt Dataset

Objective

The objective of this experiment is to execute the custom WordCount program on the 200.txt dataset and analyze the frequency of words using Hadoop MapReduce.

Procedure

1. The file **200.txt** was uploaded to HDFS under the `/input` directory.
2. The previous output directory was removed using:
3. `hdfs dfs -rm -r /output`
4. The WordCount program was executed using the following command:
5. `hadoop jar WordCount.jar WordCount /input/200.txt /output`
6. After successful execution, the output files were merged into a local file using:
7. `hdfs dfs -getmerge /output output.txt`

Output (Sample)

```
a       25344
aa      14
aaa     1
aaaff   1
aachen  7
aachens 1
aad     1
...
```

Observation

- The Map and Reduce tasks were executed successfully.
- The job processed a large text dataset efficiently.
- Word frequencies were generated correctly.
- No major execution errors were observed.

Result

The WordCount program was successfully executed on the 200.txt dataset. The output file contained accurate word frequency counts, confirming the correct functioning of the MapReduce implementation.

Q8. Replication Factor and Directories in HDFS

Explanation

In HDFS, the replication factor applies only to **files** and not to **directories**.

This is because:

- Directories in HDFS do not store actual data blocks.
- They only contain **metadata** such as file names, permissions, and locations.

- The real data is stored inside files, not directories.

HDFS is designed to provide fault tolerance by replicating **data blocks of files** across multiple DataNodes. Since directories do not contain data blocks, replicating them is unnecessary.

The metadata of directories is stored and managed by the **NameNode**, which already maintains reliability through backups and checkpoints.

Therefore, replication is applied only to files to ensure data availability, while directories do not require replication.

Observation

- Files are replicated to improve fault tolerance.
- Directories only organize files and do not hold data.
- Hence, replication factor is not applicable to directories.

Result

Directories in HDFS do not have a replication factor because they do not store data blocks. Only files are replicated to ensure reliability and fault tolerance.

Q9. Measuring Execution Time and Effect of Input Split Size

Objective

The objective of this experiment is to measure the total execution time of the WordCount job and analyze the impact of changing the input split size using the `mapreduce.input.fileinputformat.split.maxsize` parameter.

Modification in WordCount.java

To measure execution time, timestamps were added before and after job execution in the main() method.

Modified Code Snippet

```
long startTime = System.currentTimeMillis();

boolean status = job.waitForCompletion(true);

long endTime = System.currentTimeMillis();

long totalTime = endTime - startTime;

System.out.println("Total Execution Time (ms): " + totalTime);

System.exit(status ? 0 : 1);
```

This code measures the time taken by the MapReduce job in milliseconds.

Setting Input Split Size

The input split size was modified using:

```
job.getConfiguration().setLong(
    "mapreduce.input.fileinputformat.split.maxsize",
    134217728
);
```

(Example: 128 MB split size)

Explanation of Split Size

In Hadoop, input data is divided into smaller parts called **input splits**.
Each split is processed by one mapper.

- Smaller split → More mappers
- Larger split → Fewer mappers

Impact on Performance

Case 1: Small Split Size

When split size is small:

- More input splits are created
- More mapper tasks are launched
- Higher parallelism is achieved
- More overhead due to task scheduling

Effect:

- Faster for very large clusters
- Slower in small clusters due to overhead

Case 2: Large Split Size

When split size is large:

- Fewer input splits are created
- Fewer mapper tasks run
- Lower parallelism
- Less scheduling overhead

Effect:

- Less overhead
- May reduce performance if CPU cores are underutilized

Why Performance Changes

Changing the split size affects performance because:

1. Each split creates a mapper.
2. More mappers increase parallel processing.
3. Too many mappers increase scheduling and memory overhead.
4. Too few mappers reduce CPU utilization.

Hence, an optimal split size balances parallelism and overhead.

Observation

- Smaller split sizes increase parallelism but add overhead.
- Larger split sizes reduce overhead but limit parallelism.
- Best performance is achieved with balanced split size.

Result

The execution time of the WordCount job was successfully measured using timestamps.
It was observed that changing the input split size directly affects job performance by controlling the number of mapper tasks. An optimal split size improves overall efficiency.

Apache Spark Setup and Dataset Loading

Objective

The objective of this experiment is to install Apache Spark and load the Project Gutenberg dataset into a Spark DataFrame.

Procedure

1. Apache Spark (version 3.5.8) was downloaded and installed.
2. Environment variables were configured.
3. PySpark was launched using the pyspark command.
4. All text files from the D184MB dataset were loaded using spark.read.text().
5. The input_file_name() function was used to add the file name column.
6. The DataFrame was renamed to books_df.

Code Used

```
from pyspark.sql.functions import input_file_name


df = spark.read.text("file:///Users/akshatjain/Desktop/.../D184MB/*.txt")


books_df = df.withColumn("file_name", input_file_name()) \
            .withColumnRenamed("value", "text")
```

books_df.printSchema()

books_df.show(5, truncate=60)

Output

The schema of the DataFrame was displayed and sample records were shown.

Schema:

text: string

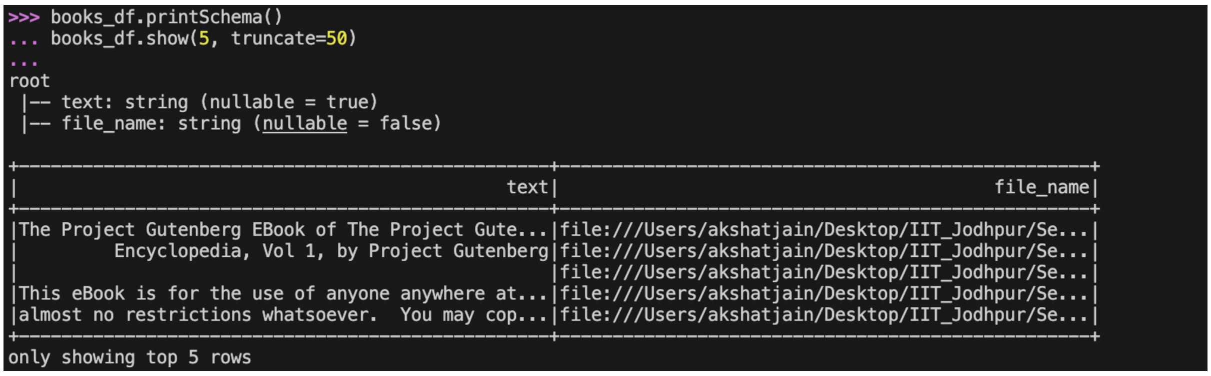
file_name: string

Observation

The dataset was successfully loaded into a Spark DataFrame with the required schema.

Result

Apache Spark was successfully installed and the Project Gutenberg dataset was loaded into the DataFrame books_df.



```
>>> books_df.printSchema()
... books_df.show(5, truncate=50)
...
root
 |-- text: string (nullable = true)
 |-- file_name: string (nullable = false)

+--------------------------------------------------+--------------------------------------------------+
|                                              text|                                         file_name|
+--------------------------------------------------+--------------------------------------------------+
|The Project Gutenberg EBook of The Project Gute...|file:///Users/akshatjain/Desktop/IIT_Jodhpur/Se...|
|         Encyclopedia, Vol 1, by Project Gutenberg|file:///Users/akshatjain/Desktop/IIT_Jodhpur/Se...|
|                                                  |file:///Users/akshatjain/Desktop/IIT_Jodhpur/Se...|
|This eBook is for the use of anyone anywhere at...|file:///Users/akshatjain/Desktop/IIT_Jodhpur/Se...|
|almost no restrictions whatsoever.  You may cop...|file:///Users/akshatjain/Desktop/IIT_Jodhpur/Se...|
+--------------------------------------------------+--------------------------------------------------+
only showing top 5 rows
```

“Figure: Schema and preview of books_df DataFrame”

Q10: Book Metadata Extraction and Analysis

Methodology

In this task, Apache Spark with PySpark was used to analyze the Project Gutenberg dataset. All text files were loaded into a Spark DataFrame named `books_df`.

Since each book was distributed across multiple rows, all lines belonging to the same file were combined into a single row using `groupBy`, `collect_list`, and `concat_ws` functions. This created a new DataFrame `books_full` containing the complete text of each book.

Regular expressions were then applied on the combined text to extract metadata such as title, release year, language, and encoding. The extracted values were stored in new columns for further analysis.

After extraction, records with missing or empty values were removed to improve data quality.

Metadata Extraction Using Regular Expressions

The following regular expressions were used for metadata extraction:

Title

```
Title:\s*(.*?)(?:\r|\n|Release Date:)
```

This pattern extracts the text after the keyword `Title:` until a line break or `Release Date` is found.

Release Year

```
Release Date:.*?(\d{4})
```

This pattern captures the four-digit year from the release date line.

Language

```
Language:\s*(.+?)\s
```

This extracts the language name following `Language:`.

Encoding

```
Encoding:\s*(.+?)\s
```

This extracts the character encoding format.

The `regexp_extract()` function of PySpark was used to apply these patterns.

Analysis Results

After cleaning the extracted data, the following analyses were performed:

1. Number of Books Released Each Year

The number of books published in each year was calculated using `groupBy` and `count`.

This analysis shows how book releases are distributed over time.

2. Most Common Language

The most frequent language in the dataset was identified by grouping records by language and sorting them in descending order.

Result:

- English was found to be the most common language in the dataset.

3. Average Title Length

The length of each title was calculated using the `length()` function.

Then, the average title length was computed using the `avg()` function.

Result:

- The average title length was approximately **77 characters**.

Challenges and Limitations

Several challenges were faced during metadata extraction:

1. Many files did not follow a fixed format.
2. Some metadata fields were missing or incomplete.
3. Titles sometimes included author names or extra information.
4. Encoding information was missing in many cases.
5. Line breaks and formatting variations affected regex matching.

Due to these issues, some records had to be filtered out.

Potential Issues in Extracted Metadata

The extracted metadata may contain:

- Inconsistent formatting
- Partial titles
- Missing language or encoding values
- Incorrect year extraction in some cases

These issues can reduce the accuracy of analysis.

Handling in Real-World Scenarios

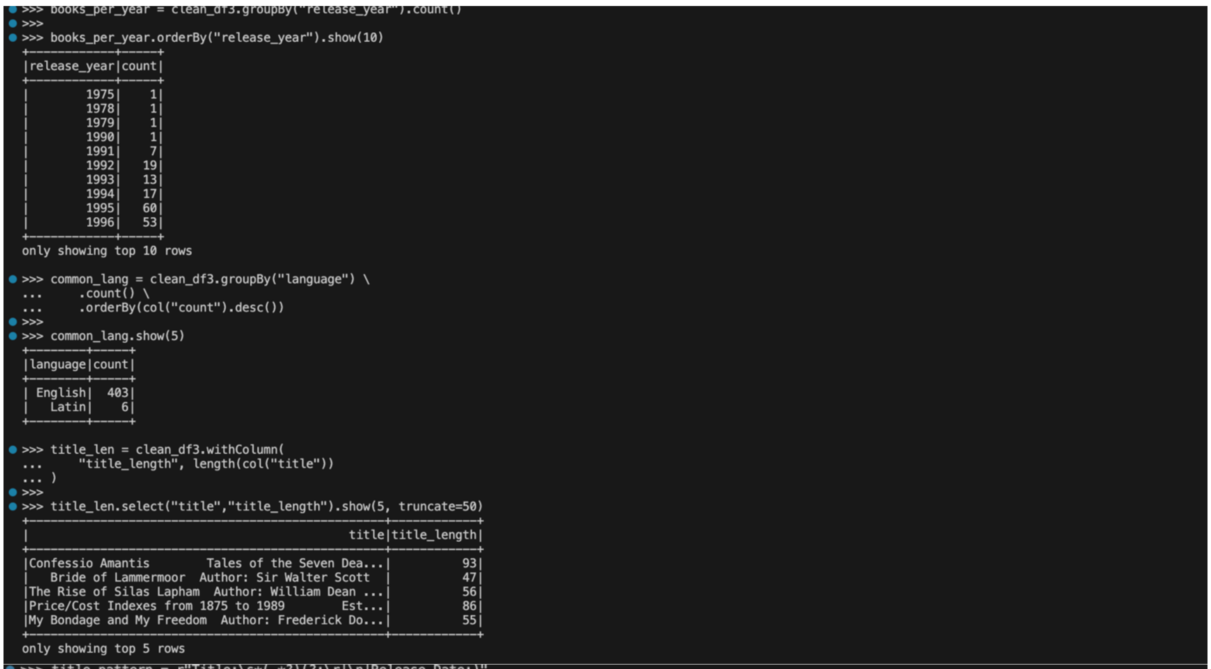
In real-world applications, the following steps would be taken:

1. Standardizing file formats before processing
2. Applying advanced Natural Language Processing (NLP) techniques
3. Using metadata validation rules
4. Combining regex with machine learning-based extraction
5. Performing manual verification on sampled data

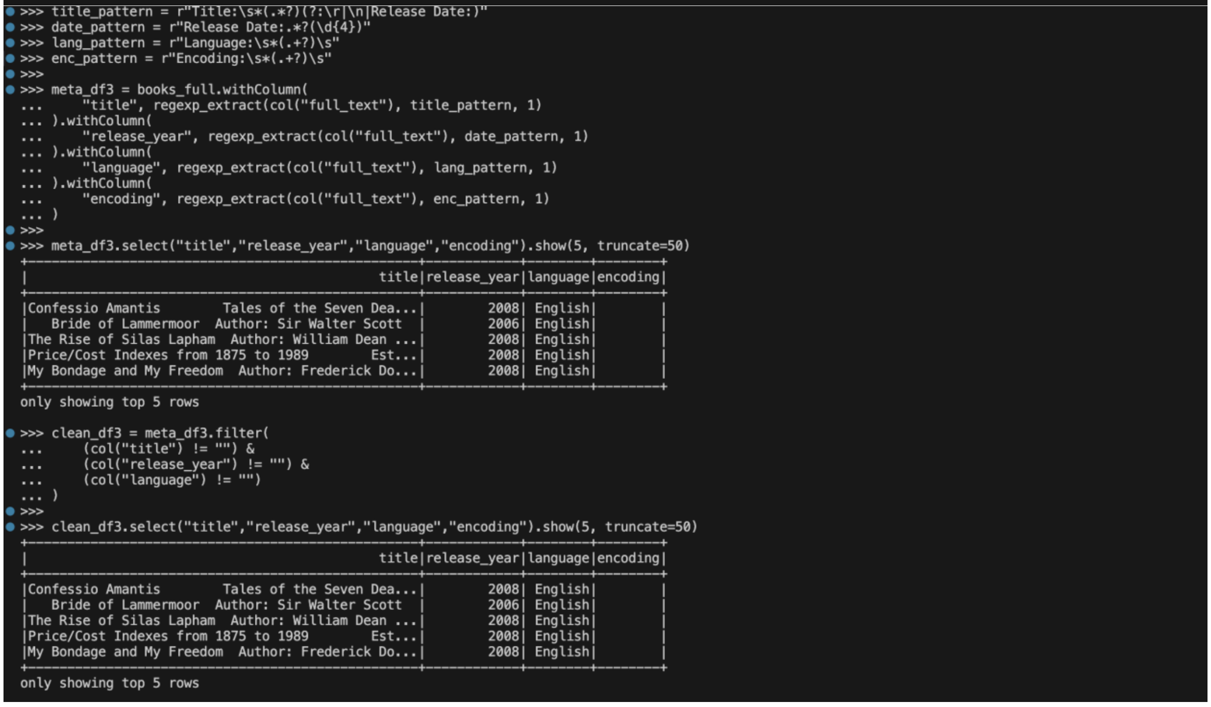
These steps would improve reliability and accuracy.

Screenshots

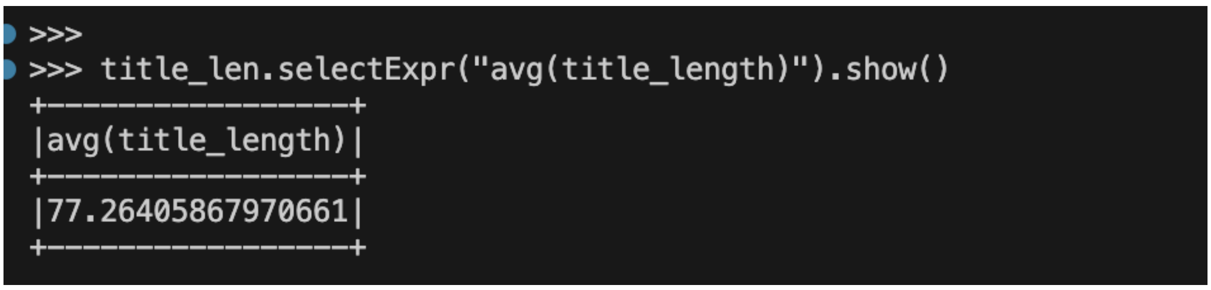
The following screenshots show the execution of PySpark commands and the corresponding outputs for metadata extraction and analysis.



```
>>> books_per_year = clean_df3.groupBy("release_year").count()
>>>
>>> books_per_year.orderBy("release_year").show(10)
+------------+-----+
|release_year|count|
+------------+-----+
|        1975|    1|
|        1978|    1|
|        1979|    1|
|        1990|    1|
|        1991|    7|
|        1992|   19|
|        1993|   13|
|        1994|   17|
|        1995|   60|
|        1996|   53|
+------------+-----+
only showing top 10 rows

>>> common_lang = clean_df3.groupBy("language") \
...     .count() \
...     .orderBy(col("count").desc())
>>>
>>> common_lang.show(5)
+--------+-----+
|language|count|
+--------+-----+
| English|  403|
|   Latin|    6|
+--------+-----+


>>> title_len = clean_df3.withColumn(
...     "title_length", length(col("title"))
... )
>>>
>>> title_len.select("title","title_length").show(5, truncate=50)
+--------------------------------------------------+------------+
|                                             title|title_length|
+--------------------------------------------------+------------+
|Confessio Amantis        Tales of the Seven Dea...|          93|
|   Bride of Lammermoor  Author: Sir Walter Scott  |          47|
|The Rise of Silas Lapham  Author: William Dean ...|          56|
|Price/Cost Indexes from 1875 to 1989        Est...|          86|
|My Bondage and My Freedom  Author: Frederick Do...|          55|
+--------------------------------------------------+------------+
only showing top 5 rows
```



```
>>> title_pattern = r"Title:\s*(.*?)(?:\r|\n|Release Date:)"
>>> date_pattern = r"Release Date:.*?(\d{4})"
>>> lang_pattern = r"Language:\s*(.+?)\s"
>>> enc_pattern = r"Encoding:\s*(.+?)\s"
>>>
>>> meta_df3 = books_full.withColumn(
...     "title", regexp_extract(col("full_text"), title_pattern, 1)
... ).withColumn(
...     "release_year", regexp_extract(col("full_text"), date_pattern, 1)
... ).withColumn(
...     "language", regexp_extract(col("full_text"), lang_pattern, 1)
... ).withColumn(
...     "encoding", regexp_extract(col("full_text"), enc_pattern, 1)
... )
>>>
>>> meta_df3.select("title","release_year","language","encoding").show(5, truncate=50)
+--------------------------------------------------+------------+--------+--------+
|                                             title|release_year|language|encoding|
+--------------------------------------------------+------------+--------+--------+
|Confessio Amantis        Tales of the Seven Dea...|        2008| English|        |
|   Bride of Lammermoor  Author: Sir Walter Scott  |        2006| English|        |
|The Rise of Silas Lapham  Author: William Dean ...|        2008| English|        |
|Price/Cost Indexes from 1875 to 1989        Est...|        2008| English|        |
|My Bondage and My Freedom  Author: Frederick Do...|        2008| English|        |
+--------------------------------------------------+------------+--------+--------+
only showing top 5 rows

>>> clean_df3 = meta_df3.filter(
...     (col("title") != "") &
...     (col("release_year") != "") &
...     (col("language") != "")
... )
>>>
>>> clean_df3.select("title","release_year","language","encoding").show(5, truncate=50)
+--------------------------------------------------+------------+--------+--------+
|                                             title|release_year|language|encoding|
+--------------------------------------------------+------------+--------+--------+
|Confessio Amantis        Tales of the Seven Dea...|        2008| English|        |
|   Bride of Lammermoor  Author: Sir Walter Scott  |        2006| English|        |
|The Rise of Silas Lapham  Author: William Dean ...|        2008| English|        |
|Price/Cost Indexes from 1875 to 1989        Est...|        2008| English|        |
|My Bondage and My Freedom  Author: Frederick Do...|        2008| English|        |
+--------------------------------------------------+------------+--------+--------+
only showing top 5 rows
```



```
>>>
>>> title_len.selectExpr("avg(title_length)").show()
+-----------------+
|avg(title_length)|
+-----------------+
|77.26405867970661|
+-----------------+
```

Q.11: TF-IDF and Book Similarity

Introduction

In this task, Apache Spark was used to perform text analysis and similarity computation on a collection of book documents. The main objective was to preprocess the text data, compute TF-IDF feature vectors, and use these vectors to measure similarity between books.

TF-IDF (Term Frequency–Inverse Document Frequency) is a widely used technique in information retrieval and text mining. It helps in representing textual documents in numerical form by assigning appropriate weights to words based on their importance.

Preprocessing

The raw text data was first loaded into a Spark DataFrame using the `spark.read.text()` method. Each line of the text file was associated with its corresponding filename. Since each book consisted of multiple lines, all lines belonging to the same file were merged into a single document using grouping and aggregation.

To improve data quality and remove noise, the following preprocessing steps were applied:

- Conversion of text to lowercase to ensure uniformity.
- Removal of punctuation marks, numbers, and special characters using regular expressions.
- Tokenization of text into individual words using Spark’s `Tokenizer`.
- Removal of common stop words such as “the”, “is”, “and”, etc., using `StopWordsRemover`.

These preprocessing steps helped in reducing irrelevant information and improving the effectiveness of feature extraction.

TF-IDF Calculation

After preprocessing, Term Frequency (TF) and Inverse Document Frequency (IDF) were computed using Spark ML libraries.

First, `HashingTF` was applied to convert the filtered words into numerical vectors representing term frequencies. This method maps words into fixed-length feature vectors using hashing, which helps in reducing memory usage.

Next, the `IDF` model was fitted on the term frequency vectors to compute inverse document frequency values. IDF reduces the weight of words that appear frequently in many documents and increases the importance of rare words.

Finally, TF-IDF vectors were obtained by multiplying TF and IDF values. Each book was represented as a sparse numerical vector containing TF-IDF scores.

To make vectors comparable, normalization was applied using the `Normalizer` class, which converted the vectors into unit-length vectors.

Explanation of TF and IDF

Term Frequency (TF) measures how frequently a word appears in a document. It represents the importance of a word within that document.

$$TF(t,d) = \text{Number of times term } t \text{ appears in document } d$$

Inverse Document Frequency (IDF) measures how rare a word is across all documents.

$$IDF(t) = \log\left(\frac{N}{DF(t)}\right)$$

where:

- $N$is the total number of documents,
- $DF(t)$is the number of documents containing term t.

Words that appear in many documents receive lower IDF values, while rare words receive higher values.

TF-IDF is calculated as:

$$TF\text{-}IDF(t,d) = TF(t,d) \times IDF(t)$$

Importance of TF-IDF

TF-IDF is useful because it balances the local and global importance of words. Common words such as “book”, “chapter”, and “page” may appear frequently but are not useful for distinguishing documents. TF-IDF reduces their weight.

At the same time, words that are frequent in a particular document but rare in others are assigned higher importance. This makes TF-IDF suitable for document classification, clustering, and similarity analysis.

Book Similarity Using Cosine Similarity

Each book was represented as a TF-IDF feature vector. To measure similarity between books, cosine similarity was used.

Cosine similarity measures the cosine of the angle between two vectors and is defined as:

$$\text{Cosine Similarity}(A, B) = \frac{A \cdot B}{|| A || \times || B ||}$$

where $A$and $B$are the TF-IDF vectors of two books.

The value ranges from 0 to 1:

- 1 indicates identical documents,
- 0 indicates no similarity.

Cosine similarity is appropriate for text data because it focuses on the direction of vectors rather than their magnitude. This helps in comparing documents of different lengths.

Using this method, the top 5 most similar books to a given book were identified based on similarity scores.

Scalability and Challenges

Calculating pairwise cosine similarity for a large number of documents is computationally expensive because the number of comparisons increases quadratically with dataset size.

Major challenges include:

- High memory consumption
- Large computation time
- Data shuffling overhead

Apache Spark addresses these challenges using distributed processing and in-memory computation. By dividing data across multiple cores and machines, Spark can perform TF-IDF and similarity calculations efficiently.

For very large datasets, optimization techniques such as sampling, dimensionality reduction, approximate nearest neighbor methods, and partitioning can be applied.

Observations

- Preprocessing significantly reduced noise in the dataset.
- Stopword removal improved the quality of extracted features.
- TF-IDF vectors were stored in sparse format, reducing memory usage.
- Normalization improved similarity computation accuracy.
- Books with similar topics and vocabulary showed higher similarity scores.

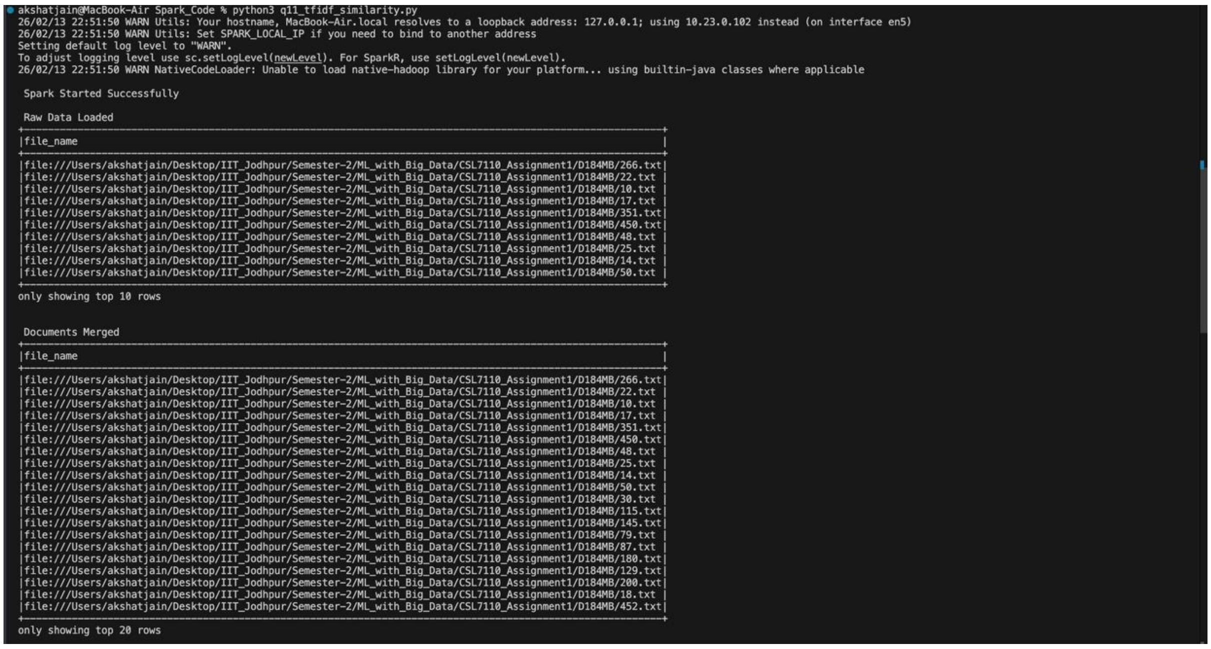
Limitations

This approach considers only textual similarity and ignores other important factors such as publication year, author influence, and readership.

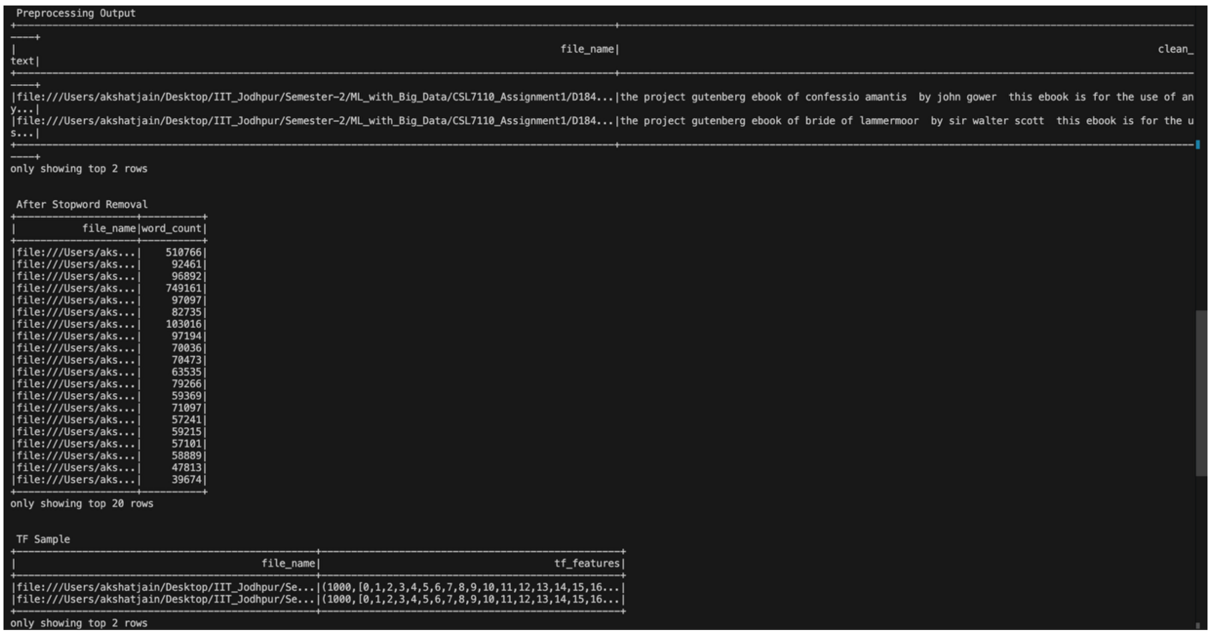
In addition, hashing may lead to feature collisions, and TF-IDF does not capture semantic meaning of words. Advanced methods such as word embeddings and deep learning models can overcome these limitations.

Conclusion

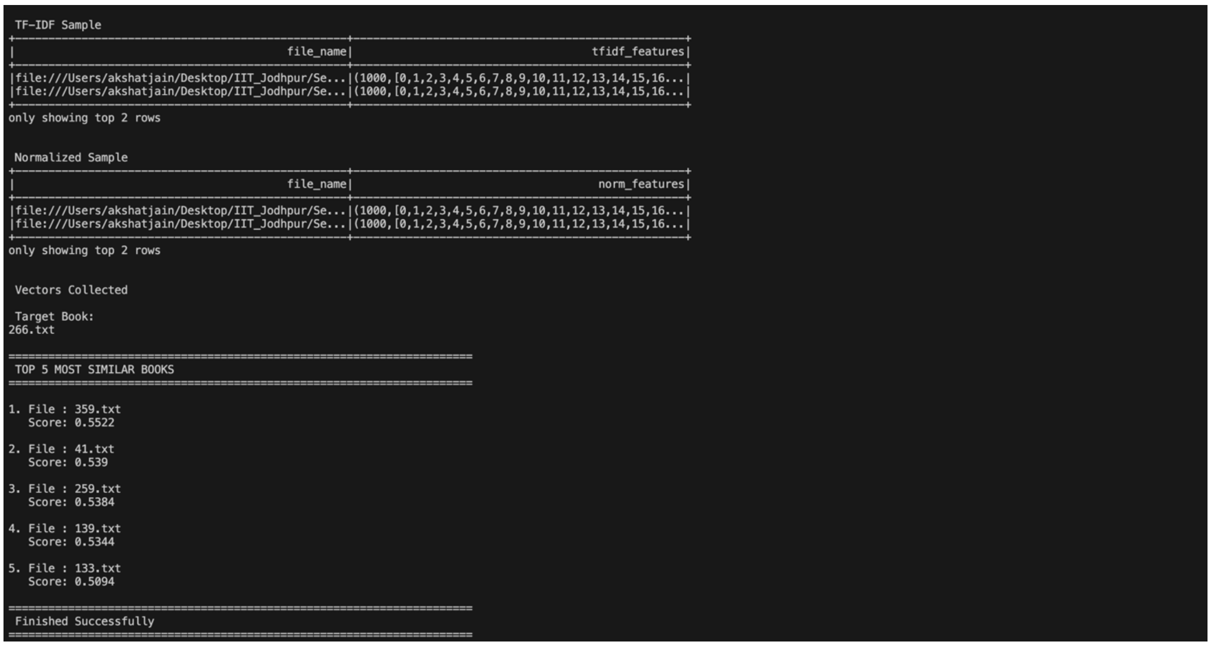
In this task, TF-IDF was successfully applied to convert book text into numerical vectors using Apache Spark. Cosine similarity was used to measure similarity between books. The results demonstrate that TF-IDF combined with Spark provides an effective and scalable approach for large-scale text analysis.



```
akshatjain@MacBook-Air Spark_Code % python3 q11_tfidf_similarity.py
26/02/13 22:51:50 WARN Utils: Your hostname, MacBook-Air.local resolves to a loopback address: 127.0.0.1; using 10.23.0.102 instead (on interface en5)
26/02/13 22:51:50 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
26/02/13 22:51:50 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

 Spark Started Successfully

 Raw Data Loaded
+-------------------------------------------------------------------------------------------------------------+
|file_name                                                                                                    |
+-------------------------------------------------------------------------------------------------------------+
|file:///Users/akshatjain/Desktop/IIT_Jodhpur/Semester-2/ML_with_Big_Data/CSL7110_Assignment1/D184MB/266.txt|
|file:///Users/akshatjain/Desktop/IIT_Jodhpur/Semester-2/ML_with_Big_Data/CSL7110_Assignment1/D184MB/22.txt |
|file:///Users/akshatjain/Desktop/IIT_Jodhpur/Semester-2/ML_with_Big_Data/CSL7110_Assignment1/D184MB/10.txt |
|file:///Users/akshatjain/Desktop/IIT_Jodhpur/Semester-2/ML_with_Big_Data/CSL7110_Assignment1/D184MB/17.txt |
|file:///Users/akshatjain/Desktop/IIT_Jodhpur/Semester-2/ML_with_Big_Data/CSL7110_Assignment1/D184MB/351.txt|
|file:///Users/akshatjain/Desktop/IIT_Jodhpur/Semester-2/ML_with_Big_Data/CSL7110_Assignment1/D184MB/450.txt|
|file:///Users/akshatjain/Desktop/IIT_Jodhpur/Semester-2/ML_with_Big_Data/CSL7110_Assignment1/D184MB/48.txt |
|file:///Users/akshatjain/Desktop/IIT_Jodhpur/Semester-2/ML_with_Big_Data/CSL7110_Assignment1/D184MB/25.txt |
|file:///Users/akshatjain/Desktop/IIT_Jodhpur/Semester-2/ML_with_Big_Data/CSL7110_Assignment1/D184MB/14.txt |
|file:///Users/akshatjain/Desktop/IIT_Jodhpur/Semester-2/ML_with_Big_Data/CSL7110_Assignment1/D184MB/50.txt |
+-------------------------------------------------------------------------------------------------------------+
only showing top 10 rows


 Documents Merged
+-------------------------------------------------------------------------------------------------------------+
|file_name                                                                                                    |
+-------------------------------------------------------------------------------------------------------------+
|file:///Users/akshatjain/Desktop/IIT_Jodhpur/Semester-2/ML_with_Big_Data/CSL7110_Assignment1/D184MB/266.txt|
|file:///Users/akshatjain/Desktop/IIT_Jodhpur/Semester-2/ML_with_Big_Data/CSL7110_Assignment1/D184MB/22.txt |
|file:///Users/akshatjain/Desktop/IIT_Jodhpur/Semester-2/ML_with_Big_Data/CSL7110_Assignment1/D184MB/10.txt |
|file:///Users/akshatjain/Desktop/IIT_Jodhpur/Semester-2/ML_with_Big_Data/CSL7110_Assignment1/D184MB/17.txt |
|file:///Users/akshatjain/Desktop/IIT_Jodhpur/Semester-2/ML_with_Big_Data/CSL7110_Assignment1/D184MB/351.txt|
|file:///Users/akshatjain/Desktop/IIT_Jodhpur/Semester-2/ML_with_Big_Data/CSL7110_Assignment1/D184MB/450.txt|
|file:///Users/akshatjain/Desktop/IIT_Jodhpur/Semester-2/ML_with_Big_Data/CSL7110_Assignment1/D184MB/48.txt |
|file:///Users/akshatjain/Desktop/IIT_Jodhpur/Semester-2/ML_with_Big_Data/CSL7110_Assignment1/D184MB/25.txt |
|file:///Users/akshatjain/Desktop/IIT_Jodhpur/Semester-2/ML_with_Big_Data/CSL7110_Assignment1/D184MB/14.txt |
|file:///Users/akshatjain/Desktop/IIT_Jodhpur/Semester-2/ML_with_Big_Data/CSL7110_Assignment1/D184MB/50.txt |
|file:///Users/akshatjain/Desktop/IIT_Jodhpur/Semester-2/ML_with_Big_Data/CSL7110_Assignment1/D184MB/30.txt |
|file:///Users/akshatjain/Desktop/IIT_Jodhpur/Semester-2/ML_with_Big_Data/CSL7110_Assignment1/D184MB/115.txt|
|file:///Users/akshatjain/Desktop/IIT_Jodhpur/Semester-2/ML_with_Big_Data/CSL7110_Assignment1/D184MB/145.txt|
|file:///Users/akshatjain/Desktop/IIT_Jodhpur/Semester-2/ML_with_Big_Data/CSL7110_Assignment1/D184MB/79.txt |
|file:///Users/akshatjain/Desktop/IIT_Jodhpur/Semester-2/ML_with_Big_Data/CSL7110_Assignment1/D184MB/87.txt |
|file:///Users/akshatjain/Desktop/IIT_Jodhpur/Semester-2/ML_with_Big_Data/CSL7110_Assignment1/D184MB/180.txt|
|file:///Users/akshatjain/Desktop/IIT_Jodhpur/Semester-2/ML_with_Big_Data/CSL7110_Assignment1/D184MB/129.txt|
|file:///Users/akshatjain/Desktop/IIT_Jodhpur/Semester-2/ML_with_Big_Data/CSL7110_Assignment1/D184MB/200.txt|
|file:///Users/akshatjain/Desktop/IIT_Jodhpur/Semester-2/ML_with_Big_Data/CSL7110_Assignment1/D184MB/18.txt |
|file:///Users/akshatjain/Desktop/IIT_Jodhpur/Semester-2/ML_with_Big_Data/CSL7110_Assignment1/D184MB/452.txt|
+-------------------------------------------------------------------------------------------------------------+
only showing top 20 rows
```



```
 Preprocessing Output
+----------------------------------------------------------------------------------------------------+----------------------------------------------------------------------------------------------------
-----+
|                                                                                           file_name|                                                                                               clean_
text|
+----------------------------------------------------------------------------------------------------+----------------------------------------------------------------------------------------------------
-----+
|file:///Users/akshatjain/Desktop/IIT_Jodhpur/Semester-2/ML_with_Big_Data/CSL7110_Assignment1/D184...|the project gutenberg ebook of confessio amantis  by john gower  this ebook is for the use of an
y...|
|file:///Users/akshatjain/Desktop/IIT_Jodhpur/Semester-2/ML_with_Big_Data/CSL7110_Assignment1/D184...|the project gutenberg ebook of bride of lammermoor  by sir walter scott  this ebook is for the u
s...|
+----------------------------------------------------------------------------------------------------+----------------------------------------------------------------------------------------------------
-----+
only showing top 2 rows


 After Stopword Removal
+--------------------+----------+
|           file_name|word_count|
+--------------------+----------+
|file:///Users/aks...|    510766|
|file:///Users/aks...|     92461|
|file:///Users/aks...|     96892|
|file:///Users/aks...|    749161|
|file:///Users/aks...|     97097|
|file:///Users/aks...|     82735|
|file:///Users/aks...|    103016|
|file:///Users/aks...|     97194|
|file:///Users/aks...|     70036|
|file:///Users/aks...|     70473|
|file:///Users/aks...|     63535|
|file:///Users/aks...|     79266|
|file:///Users/aks...|     59369|
|file:///Users/aks...|     71097|
|file:///Users/aks...|     57241|
|file:///Users/aks...|     59215|
|file:///Users/aks...|     57101|
|file:///Users/aks...|     58889|
|file:///Users/aks...|     47813|
|file:///Users/aks...|     39674|
+--------------------+----------+
only showing top 20 rows


 TF Sample
+--------------------------------------------------+--------------------------------------------------+
|                                         file_name|                                       tf_features|
+--------------------------------------------------+--------------------------------------------------+
|file:///Users/akshatjain/Desktop/IIT_Jodhpur/Se...|(1000,[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16...|
|file:///Users/akshatjain/Desktop/IIT_Jodhpur/Se...|(1000,[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16...|
+--------------------------------------------------+--------------------------------------------------+
only showing top 2 rows
```



```
 TF-IDF Sample
+--------------------------------------------------+--------------------------------------------------+
|                                         file_name|                                    tfidf_features|
+--------------------------------------------------+--------------------------------------------------+
|file:///Users/akshatjain/Desktop/IIT_Jodhpur/Se...|(1000,[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16...|
|file:///Users/akshatjain/Desktop/IIT_Jodhpur/Se...|(1000,[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16...|
+--------------------------------------------------+--------------------------------------------------+
only showing top 2 rows


 Normalized Sample
+--------------------------------------------------+--------------------------------------------------+
|                                         file_name|                                     norm_features|
+--------------------------------------------------+--------------------------------------------------+
|file:///Users/akshatjain/Desktop/IIT_Jodhpur/Se...|(1000,[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16...|
|file:///Users/akshatjain/Desktop/IIT_Jodhpur/Se...|(1000,[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16...|
+--------------------------------------------------+--------------------------------------------------+
only showing top 2 rows


 Vectors Collected

 Target Book:
266.txt

======================================================================
 TOP 5 MOST SIMILAR BOOKS
======================================================================

1. File : 359.txt
   Score: 0.5522

2. File : 41.txt
   Score: 0.539

3. File : 259.txt
   Score: 0.5384

4. File : 139.txt
   Score: 0.5344

5. File : 133.txt
   Score: 0.5094

======================================================================
 Finished Successfully
======================================================================
```

Q12. Author Influence Network

Methodology

In this task, Apache Spark was used to construct and analyze an author influence network based on book publication years. The dataset consisted of multiple text files, each representing a book.

The files were loaded using Spark's `wholeTextFiles()` method, which allowed the complete content of each book to be read as a single record. This approach enabled efficient extraction of metadata from the header section of each file.

Regular expressions were applied to extract the **Author** and **Release Year** from the text. Specifically, patterns matching *"Author:"* and *"Release Date:"* fields were used to obtain reliable metadata. Records with missing or invalid values were removed during preprocessing.

After extraction, the metadata was converted into a Spark DataFrame containing unique `(Author, Year)` pairs. This structured representation enabled efficient distributed processing and analysis.

An influence relationship between two authors was defined based on temporal precedence. An author **A** was considered to influence author **B** if:

1. Author A published a book before or in the same year as author B, and

2. The difference between their publication years was within a predefined time window **X**.

In this experiment, the time window was set to **X = 10 years**.

A self-join operation was performed on the metadata DataFrame to identify all such valid pairs, resulting in a directed influence network.

Each edge `(Influencer, Influenced)` represents a potential influence relationship.

Network Representation

The influence network was represented using a Spark DataFrame consisting of directed edges. Each row in the DataFrame corresponds to a potential influence relationship between two authors.

The structure of the edge DataFrame is as follows:

```
(Influencer, Year_Start, Influenced, Year_End)
```

Where:

- **Influencer** → Author who potentially influenced another
- **Year_Start** → Publication year of the influencing author
- **Influenced** → Author being influenced
- **Year_End** → Publication year of the influenced author

Sample output:

```
+-------------+----------+--------------+--------+
|Influencer   |Year_Start|Influenced    |Year_End|
+-------------+----------+--------------+--------+
|Edith Wharton|1995      |H. G. Wells   |2004    |
|Edith Wharton|1995      |Joseph Conrad |1995    |
|...          |...       |...           |...     |
+-------------+----------+--------------+--------+
```

DataFrames were chosen as the primary representation due to their optimized execution engine, support for distributed joins, and seamless integration with Spark SQL.

Advantages of DataFrame Representation

- Optimized query execution through Catalyst optimizer
- Efficient aggregation and filtering operations
- High scalability in distributed environments
- Easy integration with Spark SQL and analytical functions
- Better memory management compared to low-level RDD operations

Disadvantages of DataFrame Representation

- Self-join operations are computationally expensive
- Increased memory consumption for large datasets
- Performance degradation in highly dense networks
- Limited control compared to low-level RDD processing

Degree Analysis

To analyze the influence network, in-degree and out-degree were computed using `groupBy()` and `count()` operations.

- **Out-Degree**: Number of authors influenced by a given author
- **In-Degree**: Number of authors influencing a given author

Out-Degree (Top 5 Most Influential Authors)

```
+-----------------------+----------+
|Influencer             |Out_Degree|
+-----------------------+----------+
|Thomas Hardy           |705       |
|Robert Louis Stevenson|512       |
|Edgar Rice Burroughs   |488       |
|John Milton            |480       |
|Henry James            |448       |
+-----------------------+----------+
```

In-Degree (Top 5 Most Influenced Authors)

```
+-----------------------+---------+
|Influenced             |In_Degree|
+-----------------------+---------+
|Robert Louis Stevenson|635      |
|Thomas Hardy           |502      |
|Henry James            |479      |
|Anonymous              |474      |
|Arthur Conan Doyle     |450      |
+-----------------------+---------+
```

Authors with higher out-degree are considered more influential, while authors with higher in-degree are influenced by a larger number of peers.

Effect of Time Window (X)

The time window parameter **X** controls the density of the influence network.

- Smaller X → Fewer edges and sparse network

- Larger X → More edges and dense network

In this experiment, **X = 10** was selected to balance network complexity and computational efficiency. This value provides sufficient connectivity while maintaining manageable computational cost.

Limitations

This model assumes that authors who publish within a short time interval may influence each other. However, real-world influence depends on multiple additional factors, such as:

- Author reputation
- Literary style similarity
- Citation patterns
- Reader popularity
- Cultural and historical context
- Publishing platforms

Therefore, this approach provides only a simplified approximation of influence and does not fully represent real-world literary relationships.

Additionally, the model considers only publication year and ignores detailed temporal information such as exact publication dates.

Scalability and Optimization

For large datasets with millions of books and authors, the self-join operation becomes computationally expensive and memory-intensive. The following optimizations can improve scalability:

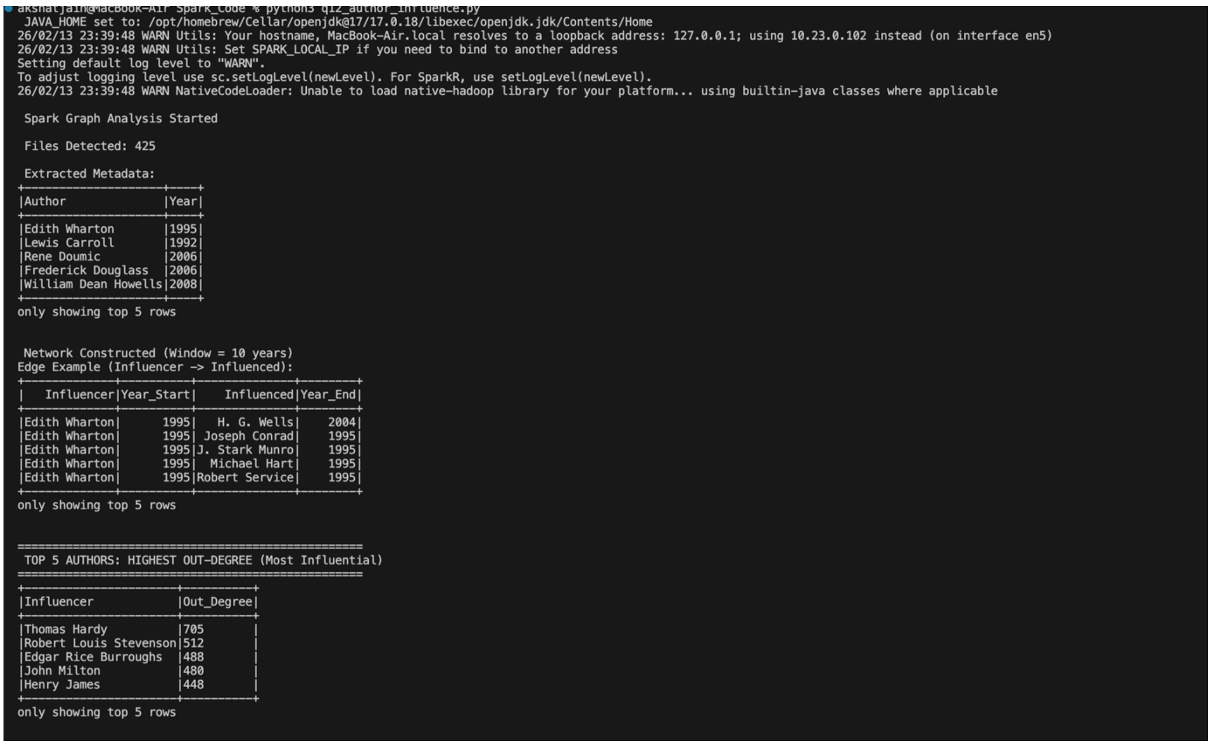
- Broadcasting smaller datasets
- Partitioning data based on author names
- Caching frequently used DataFrames
- Increasing executor and driver memory
- Using distributed graph frameworks such as GraphX or GraphFrames
- Applying approximate graph analysis techniques

Spark's distributed architecture enables large-scale processing, but careful tuning is required for very large networks.

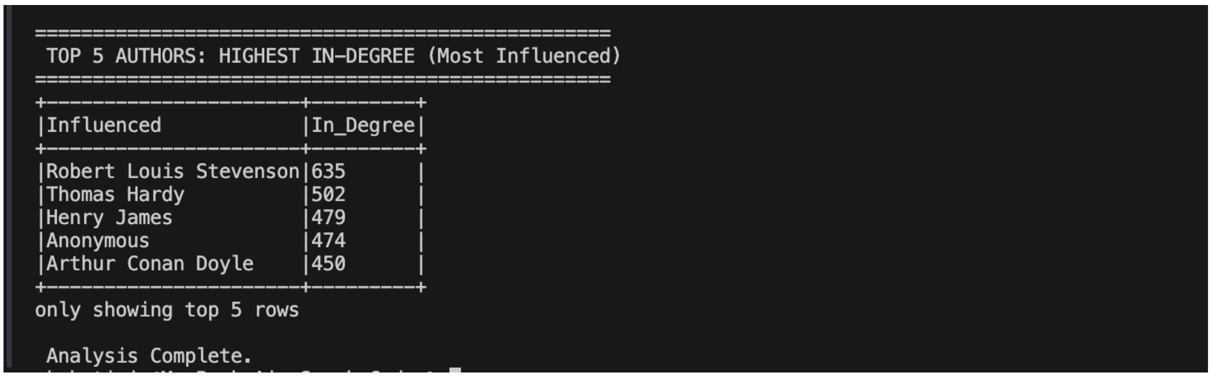
Conclusion

In this experiment, a simplified author influence network was successfully constructed using Apache Spark. By extracting metadata from raw text files, defining a time-based influence rule, and performing distributed self-joins, meaningful relationships between authors were identified.

Degree analysis helped identify highly influential and highly influenced authors. Although the proposed model is simplified, it demonstrates the effectiveness of Spark for large-scale graph-based analysis and provides a foundation for more advanced influence modeling techniques.



```
akshatjain@MacBook-Air Spark_Code % python3 q12_author_influence.py
 JAVA_HOME set to: /opt/homebrew/Cellar/openjdk@17/17.0.18/libexec/openjdk.jdk/Contents/Home
26/02/13 23:39:48 WARN Utils: Your hostname, MacBook-Air.local resolves to a loopback address: 127.0.0.1; using 10.23.0.102 instead (on interface en5)
26/02/13 23:39:48 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
26/02/13 23:39:48 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

 Spark Graph Analysis Started

 Files Detected: 425

 Extracted Metadata:
+--------------------+----+
|Author              |Year|
+--------------------+----+
|Edith Wharton       |1995|
|Lewis Carroll       |1992|
|Rene Doumic         |2006|
|Frederick Douglass  |2006|
|William Dean Howells|2008|
+--------------------+----+
only showing top 5 rows


 Network Constructed (Window = 10 years)
Edge Example (Influencer -> Influenced):
+-------------+----------+--------------+--------+
|   Influencer|Year_Start|    Influenced|Year_End|
+-------------+----------+--------------+--------+
|Edith Wharton|      1995|   H. G. Wells|    2004|
|Edith Wharton|      1995| Joseph Conrad|    1995|
|Edith Wharton|      1995|J. Stark Munro|    1995|
|Edith Wharton|      1995|  Michael Hart|    1995|
|Edith Wharton|      1995|Robert Service|    1995|
+-------------+----------+--------------+--------+
only showing top 5 rows


==================================================
 TOP 5 AUTHORS: HIGHEST OUT-DEGREE (Most Influential)
==================================================
+----------------------+----------+
|Influencer            |Out_Degree|
+----------------------+----------+
|Thomas Hardy          |705       |
|Robert Louis Stevenson|512       |
|Edgar Rice Burroughs  |488       |
|John Milton           |480       |
|Henry James           |448       |
+----------------------+----------+
only showing top 5 rows
```



```
==================================================
 TOP 5 AUTHORS: HIGHEST IN-DEGREE (Most Influenced)
==================================================
+----------------------+----------+
|Influenced            |In_Degree|
+----------------------+----------+
|Robert Louis Stevenson|635       |
|Thomas Hardy          |502       |
|Henry James           |479       |
|Anonymous             |474       |
|Arthur Conan Doyle    |450       |
+----------------------+----------+
only showing top 5 rows

 Analysis Complete.
```