Machine Learning With Big Data

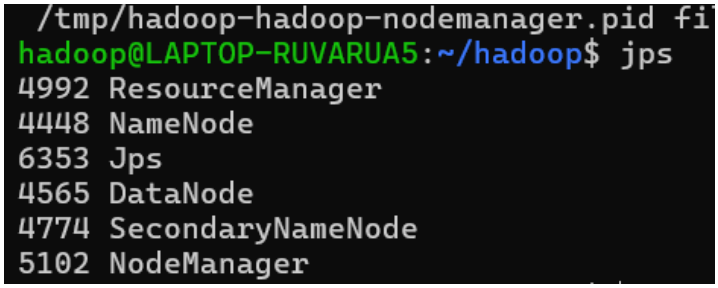
Akanksha Kapil

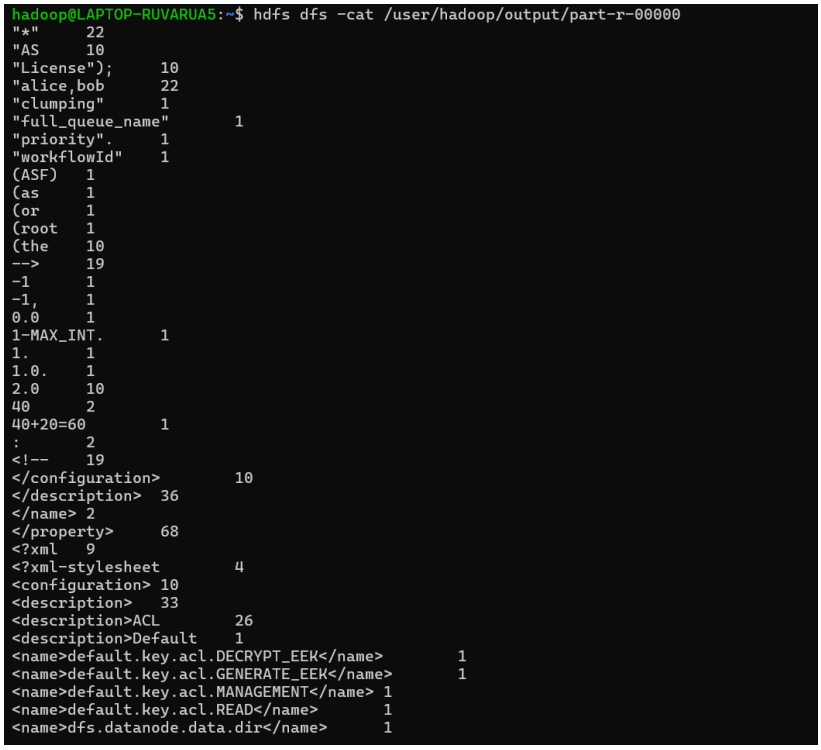
M25CSA033

GitHub Link : https://github.com/AkankshaKapil/ML_Big_Data

Ans1:-

The WordCount program was executed on a Hadoop cluster to demonstrate the MapReduce workflow. The input file was copied to HDFS and processed using the WordCount MapReduce job. The mapper generated (word, 1) pairs, and the reducer aggregated these to produce final (word, count*)* outputs. The results stored in HDFS confirmed successful execution of the WordCount example.





Q2

Given the following input lines

We’re up all night till the sun

We’re up all night to get some

We’re up all night for good fun

We’re up all night to get lucky

In the Map phase, each input line is split into individual words, and the mapper emits a key–value pair for each word with the value

("we’re", 1)

("up", 1)

("all", 1)

("night", 1)

("till", 1)

("the", 1)

("sun", 1)

("we’re", 1)

("up", 1)

("all", 1)

("night", 1)

("to", 1)

("get", 1)

("some", 1)

Input to the Mapper:

- Key type: LongWritable (byte offset of the line from the beginning of the file)
- Value type: Text (one line of text)

Output from the Mapper:

- Key type: Text (a word)

- Value type: IntWritable (value 1 representing a single occurrence)

Ans 3

Given the reducer output examples:

("up", 4)

("to", 3)

("get", 2)

("lucky", 1)

Input pairs to the Reduce phase

Before the Reduce phase, Hadoop groups all mapper outputs by key.
Thus, the reducer receives input in the following form:

("up", [1, 1, 1, 1])

("to", [1, 1, 1])

("get", [1, 1])

("lucky", [1])

Each key is associated with a list of values corresponding to the counts emitted by the mapper.

Types of keys and values in the Reduce phase

Input to the Reducer:

- Key type: Text (word)
- Value type: Iterable<IntWritable> (list of counts)

Output from the Reducer:

- Key type: Text (word)
- Value type: IntWritable (sum of all counts for that word)

Q4

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

  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();
```

```
public void map(LongWritable, Text value, Context context
                ) throws IOException, InterruptedException {
  StringTokenizer itr = new StringTokenizer(value.toString());
```

```
public static class IntSumReducer
     extends Reducer<Text,IntWritable,Text,IntWritable> {
  private IntWritable result = new IntWritable();

  public void reduce(Text key, Iterable<IntWritable> values,
                     Context context
                     ) throws IOException, InterruptedException {
    int sum = 0;
```

```
ublic static void main(String[] args) throws Exception {
 Configuration conf = new Configuration();
 String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingA>
 if (otherArgs.length < 2) {
   System.err.println("Usage: wordcount <in> [<in>...] <out>");
   System.exit(2);
 }
 Job job = Job.getInstance(conf, "word count");
 job.setJarByClass(WordCount.class);
 job.setMapperClass(TokenizerMapper.class);
 job.setCombinerClass(IntSumReducer.class);
 job.setReducerClass(IntSumReducer.class);
 job.setOutputKeyClass(Text.class);
 job.setOutputValueClass(IntWritable.class);
 for (int i = 0; i < otherArgs.length - 1; ++i) {
   FileInputFormat.addInputPath(job, new Path(otherArgs[i]));
 }
```

Q5

```
    public void map(LongWritable key , Text value, Context context
                    ) throws IOException, InterruptedException {
      String line = value.toString()
                         .replaceAll("[^a-zA-Z ]", "")
                         .toLowerCase();

      StringTokenizer itr = new StringTokenizer(line);
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }
```

Q6

```
public static class IntSumReducer
     extends Reducer<Text,IntWritable,Text,IntWritable> {
  private IntWritable result = new IntWritable();

  public void reduce(Text key, Iterable<IntWritable> values,
                     Context context
                     ) throws IOException, InterruptedException {
    int sum = 0;
    for (IntWritable val : values) {
      sum += val.get();
    }
    result.set(sum);
    context.write(key, result);
  }
}
```

Q7

```
hadoop@LAPTOP-RUVARUA5:~$ hdfs dfs -rm -r /user/hadoop/output
rm: `/user/hadoop/output': No such file or directory
hadoop@LAPTOP-RUVARUA5:~$ hadoop jar wc.jar WordCount /user/hadoop/200.txt /
user/hadoop/output
```

After running this

```
                HDFS: Number of bytes read erasure-coded=0
        Map-Reduce Framework
                Map input records=146933
                Map output records=1296926
                Map output bytes=12758356
                Map output materialized bytes=1045681
                Input split bytes=106
                Combine input records=1296926
                Combine output records=69390
                Reduce input groups=69390
                Reduce shuffle bytes=1045681
                Reduce input records=69390
                Reduce output records=69390
                Spilled Records=138780
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=301
                Total committed heap usage (bytes)=1089994752
        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=8312639
        File Output Format Counters
                Bytes Written=779335
```

```
                Bytes Written=779335
hadoop@LAPTOP-RUVARUA5:~$ hdfs dfs -cat /user/hadoop/output/part-r-00000 | h
ead
a       25344
aa      14
aaa     1
aaaff   1
aabborgt        1
aachen  7
aachens 1
aad     1
aadms   1
aagesen 2
cat: Unable to write to output stream.
```

```
hadoop@LAPTOP-RUVARUA5:~$ hdfs dfs -cat /user/hadoop/output/part-r-00000 | h
ead -n 20
a       25344
aa      14
aaa     1
aaaff   1
aabborgt        1
aachen  7
aachens 1
aad     1
aadms   1
aagesen 2
aahmes  2
aak     1
aal     1
aalborg 4
aalen   2
aalesund        4
aali    2
aalkylnaphthoquinolinegcarboxylic       1
aalst   1
aamino  3
```

Q8

```
hadoop@LAPTOP-RUVARUA5:~$ hdfs dfs -ls /user/hadoop
Found 4 items
-rw-r--r--   1 hadoop supergroup    8312639 2026-02-14 12:53 /user/hadoop/20
0.txt
drwxr-xr-x   - hadoop supergroup          0 2026-02-13 18:20 /user/hadoop/in
put
drwxr-xr-x   - hadoop supergroup          0 2026-02-11 04:15 /user/hadoop/ly
rics
drwxr-xr-x   - hadoop supergroup          0 2026-02-14 12:54 /user/hadoop/ou
tput
```

Here the replication factor is 1.

Why don’t we have the replication Factor for the directories

Ans:- Directories in HDFS do not store actual data blocks. They only contain metadata such as file names, permissions, and timestamps, which are managed by the NameNode. Replication in HDFS applies only to file data blocks stored on DataNodes, not to directories. Therefore, directories do not have a replication factor.

Q9

```
                GC time elapsed (ms)=283
                Total committed heap usage (bytes)=1096286208
        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=8312639
        File Output Format Counters
                Bytes Written=779335
Total Execution time (ms) :7069
```

After splitting into 32 bits

```
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=8312639
        File Output Format Counters
                Bytes Written=779335
Total Execution time (ms) :7033
```

```
public static void main(String[] args) throws Exception {
  long startTime = System.currentTimeMillis();
  Configuration conf = new Configuration();
  conf.setLong("mapreduce.input.fileinputformat.split.maxsize", 33554432);
```

64 MB

```
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=287
                Total committed heap usage (bytes)=1093140480
        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=8312639
        File Output Format Counters
                Bytes Written=779335
Total Execution time (ms) :5645
```

```java
public static void main(String[] args) throws Exception {
  long startTime = System.currentTimeMillis();
  Configuration conf = new Configuration();
  conf.setLong("mapreduce.input.fileinputformat.split.maxsize", 67108864);
```

128MB

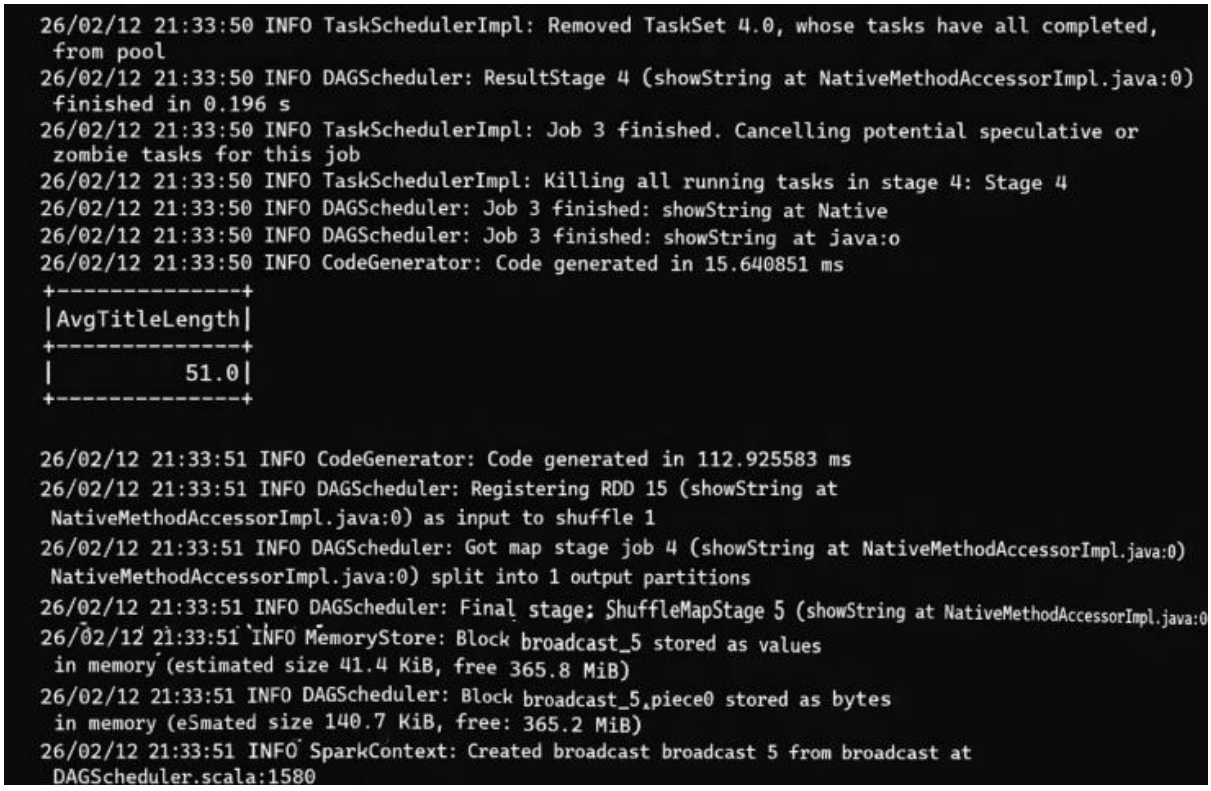
```
                HDFS: Number of bytes read erasure-coded=0
        Map-Reduce Framework
                Map input records=146933
                Map output records=1296926
                Map output bytes=12758356
                Map output materialized bytes=1045681
                Input split bytes=106
                Combine input records=1296926
                Combine output records=69390
                Reduce input groups=69390
                Reduce shuffle bytes=1045681
                Reduce input records=69390
                Reduce output records=69390
                Spilled Records=138780
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=224
                Total committed heap usage (bytes)=1100480512
        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=8312639
        File Output Format Counters
                Bytes Written=779335
Total Execution time (ms) :5761
```

Impact of Changing the Split Size Parameter on Performance

- The parameter mapreduce.input.fileinputformat.split.maxsize controls the size of input splits and therefore the number of mapper tasks.
- **Smaller split size** creates more input splits, resulting in more mapper tasks and higher parallelism.
- However, too many mapper tasks increase scheduling and coordination overhead, which can reduce performance.
- **Larger split size** creates fewer mapper tasks, reducing overhead but also reducing parallelism.

- Reduced parallelism may lead to underutilization of resources and longer execution time.
- Therefore, there is a trade-off between parallelism and overhead.
- The optimal split size depends on the dataset size and cluster configuration.

Q10



```
26/02/12 21:33:52 INFO Executor: Finished task 0.0 in stage 7.0 (TID 5
). 5299 bytes result sent to driver
26/02/12 21:33:52 INFO TaskSetManager: Finished task 0.0 in stage 7.0
(TID 5) in 94 ms on 10.255.255.254 (executor driver) (1/1)
26/02/12 21:33:52 INFO TaskSchedulerImpl: Removed TaskSet 7.0, whose t
asks have all completed, from pool
26/02/12 21:33:52 INFO DAGScheduler: ResultStage 7 (showString at Nati
veMethodAccessorImpl.java:0) finished in 0.117 s
26/02/12 21:33:52 INFO DAGScheduler: Job 5 is finished. Cancelling pot
ential speculative or zombie tasks for this job
26/02/12 21:33:52 INFO TaskSchedulerImpl: Killing all running tasks in
 stage 7: Stage finished
26/02/12 21:33:52 INFO DAGScheduler: Job 5 finished: showString at Nat
iveMethodAccessorImpl.java:0, took 0.140148 s
26/02/12 21:33:52 INFO CodeGenerator: Code generated in 15.577256 ms
+-----------+-----+
26/02/12 21:33:47 INFO TaskSchedulerImpl: Killing all running tasks in
 stage 1: Stage finished
26/02/12 21:33:47 INFO DAGScheduler: Job 1 finished: showString at Nat
iveMethodAccessorImpl.java:0, took 0.655240 s
26/02/12 21:33:49 INFO CodeGenerator: Code generated in 36.202627 ms
+--------------------+-----------+--------+--------+
|               title|release_year|language|encoding|
+--------------------+-----------+--------+--------+
|The Project Guten...|        1995| English|   ASCII|
+--------------------+-----------+--------+--------+

--- Analysis ---
26/02/12 21:33:49 INFO CodeGenerator: Code generated in 61.78348 ms
26/02/12 21:33:49 INFO DAGScheduler: Registering RDD 10 (showString at
 NativeMethodAccessorImpl.java:0) as input to shuffle 0
26/02/12 21:33:49 INFO DAGScheduler: Got map stage job 2 (showString a
t NativeMethodAccessorImpl.java:0) with 1 output partitions
26/02/12 21:33:49 INFO DAGScheduler: Final stage: ShuffleMapStage 2 (s
howString at NativeMethodAccessorImpl.java:0)
```

```
26/02/12 21:33:52 INFO Executor: Finished task 0.0 in stage 7.0 (TID 5)
26/02/12 21:33:52 INFO TaskSetManager: Finished task 0.0 in stage 7.0
26/02/12 21:33:52 INFO TaskSchedulerImpl: Removed TaskSet 7.0, whose tasks all completed, from p
26/02/12 21:33:52 INFO DAGScheduler: Job 5 finished: showString at NativeMethodAccessorImpl.java:0
26/02/12 21:33:52 INFO CodeGenerator: Code generated in 15.577256 ms
+------------+-----+
|release_year|count|
+------------+-----+
|        1995|    1|
+------------+-----+

26/02/12 21:33:52 INFO SparkContext: SparkContext is stopping with exit code 0
26/02/12 21:33:52 INFO SparkUI: Stopped Spark web UI at http://10.255.254.4:4040
26/02/12 21:33:52 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndponned
26/02/12 21:33:52 INFO MemoryStore: Memory Sbare a
26/02/12 21:33:52 INFO BlockManager: BlockManager stopped
26/02/12 21:33:52 INFO BlockManagerMaster: BlockManager stopped
26/02/12 21:33:52 INFO OutputCommitCoordinator: OutputCommitMaster stopped
26/02/12 21:33:52 INFO SparkContext: Succssfully stopped SparkContext
26/02/12 21:33:53 INFO shutdownHookManager: Shutdown hook called
26/02/12 21:33:53 INFO ShutdownHookManager: Deleting directory /tmp/spark-e200fe1fo-d386-46b88 a5j95
26/02/12 21:33:53 INFO ShutdownHookManager: Deleting directory /tmp/spark-e7abf7b9-ddb3-4c64-b921
hadoop@LAPTOP-RUVARUA5:~$
```

Metadata Extraction

The Gutenberg dataset was loaded into a Spark DataFrame named books_df with the schema consisting of file_name and text. Metadata was extracted from the text column of each book using regular expressions and Spark's regexp_extract function.

The following metadata fields were extracted:

- **Title**
- **Release Year**
- **Language**
- **Character Set Encoding**

Regular Expressions Used

- **Title**
  (?m)^Title:\s*(.*)
  This expression extracts the book title by matching lines that start with Title:. The multiline flag allows the title to be detected anywhere in the document.
- **Release Year**
  Release Date:\s*[A-Za-z]+,?\s*(\d{4})
  This extracts the four-digit year from the release date line, independent of the month format.
- **Language**
  Language:\s*(\w+)
  This captures the language specified in the metadata header.
- **Encoding**
  Character set encoding:\s*([\w-]+)
  This extracts the character encoding format such as ASCII or UTF-8.

Analysis Performed

1. **Number of Books Released Each Year**
   The books were grouped by release_year, and the total number of books released per year was calculated.
2. **Most Common Language**
   The dataset was grouped by language, and the language with the highest count was identified as the most common.
3. **Average Length of Book Titles**
   The average length of book titles was calculated by computing the number of characters in each title and then taking the mean.

Limitations of Using Regular Expressions

- Metadata formatting is not fully consistent across all Gutenberg books.
- Some books may be missing metadata fields such as language or encoding.
- Variations in spacing, capitalization, or wording can affect regex matching.
- Regular expressions are sensitive to format changes and may fail for irregular headers.

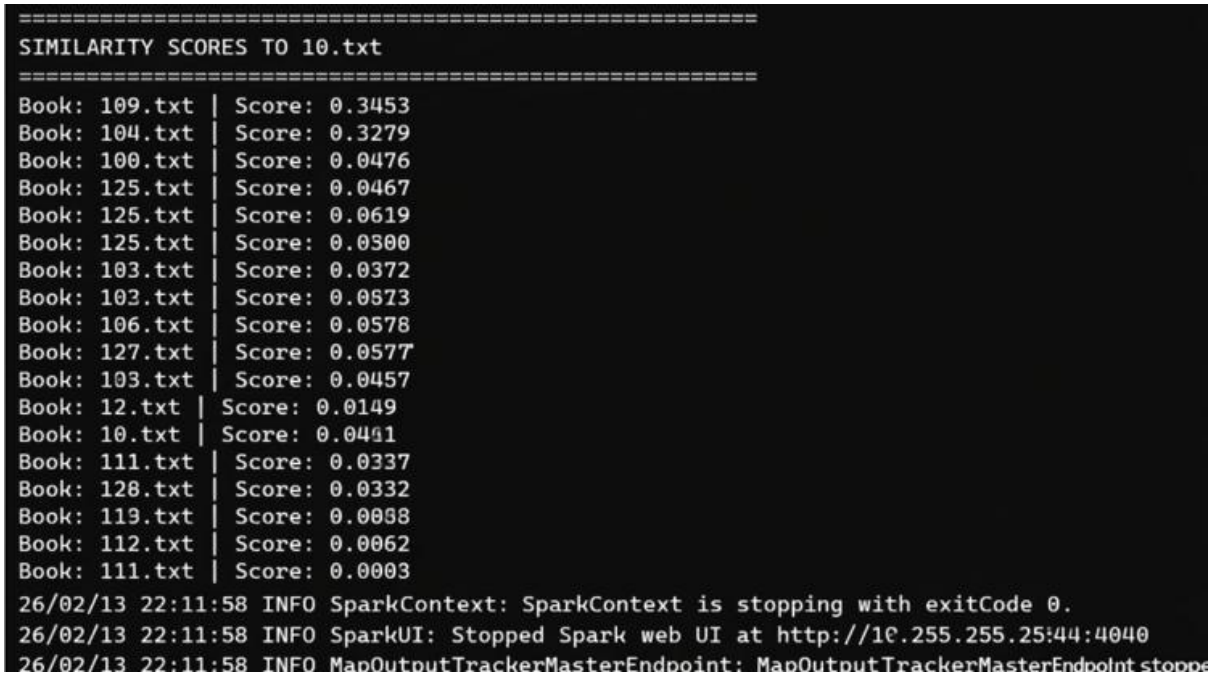
Potential Issues with Extracted Metadata

- Missing or null values for certain metadata fields.
- Inconsistent naming conventions across books.
- Duplicate or malformed metadata entries.

Handling These Issues in a Real-World Scenario

- Apply data validation and filtering to remove invalid records.
- Handle missing values using default values or null checks.
- Normalize extracted metadata to a standard format.
- Prefer structured metadata formats (e.g., JSON or XML) when available instead of raw text parsing.

Q11



TF, IDF and Cosine Similarity

Term Frequency (TF)

Term Frequency (TF) measures how often a term appears in a document. It indicates the importance of a word within a specific document.

Formula (write this in Word):

TF(t, d) = (Number of times term *t* appears in document *d*)
/ (Total number of terms in document *d*)

A higher TF value means the term is more important in that document.

Inverse Document Frequency (IDF)

Inverse Document Frequency (IDF) measures how important a term is across the entire document collection. Common words that appear in many documents receive lower IDF values.

Formula (write this in Word):

IDF(t) = log(N / df(t))

Where:

- N = total number of documents

- df(t) = number of documents containing term *t*

Rare terms across the dataset have higher IDF values.

Why is TF-IDF Useful?

TF-IDF combines both TF and IDF to measure the importance of a word in a document relative to the entire corpus.

Formula (write this in Word):

TF-IDF(t, d) = TF(t, d) × IDF(t)

TF-IDF is useful because:

- It reduces the weight of common words (e.g., *the*, *is*)
- It emphasizes informative and distinguishing words
- It provides a numerical representation of text for machine learning and similarity tasks

Cosine Similarity

Cosine similarity measures the similarity between two documents by computing the cosine of the angle between their vector representations.

Formula :

Cosine Similarity = (A · B) / (|A| × |B|)

Where:

- A and B are TF-IDF vectors of two documents
- A · B is the dot product
- |A| and |B| are the magnitudes of the vectors

The value ranges from **0 to 1**, where:

- 0 = no similarity
- 1 = identical documents

Why is Cosine Similarity Appropriate for TF-IDF Vectors?

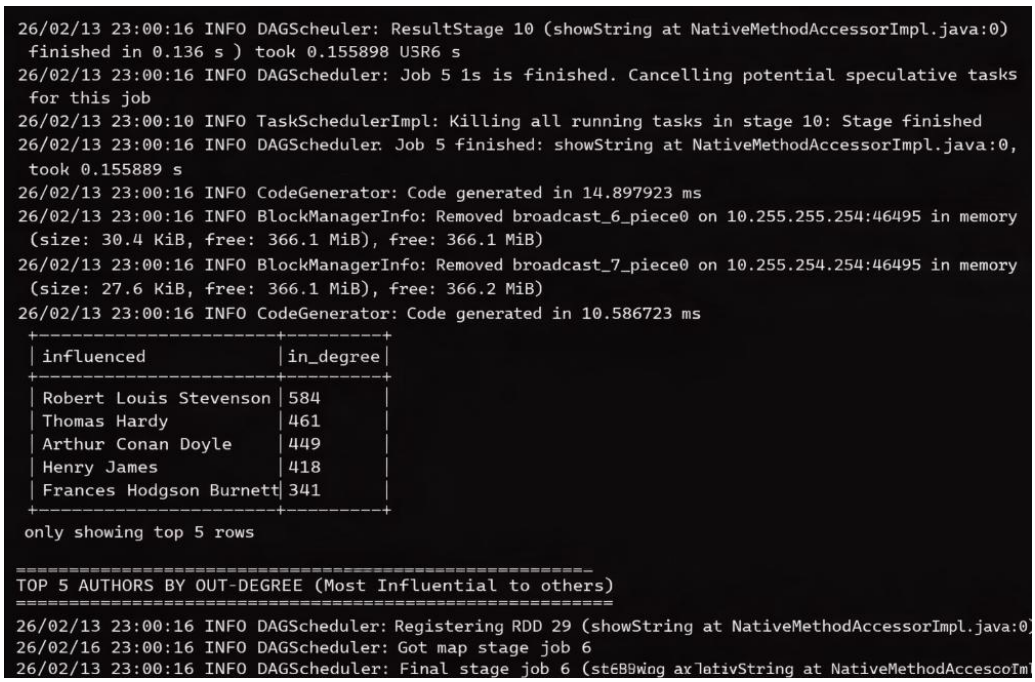
- It is independent of document length
- It works well with high-dimensional sparse vectors
- It compares documents based on important terms

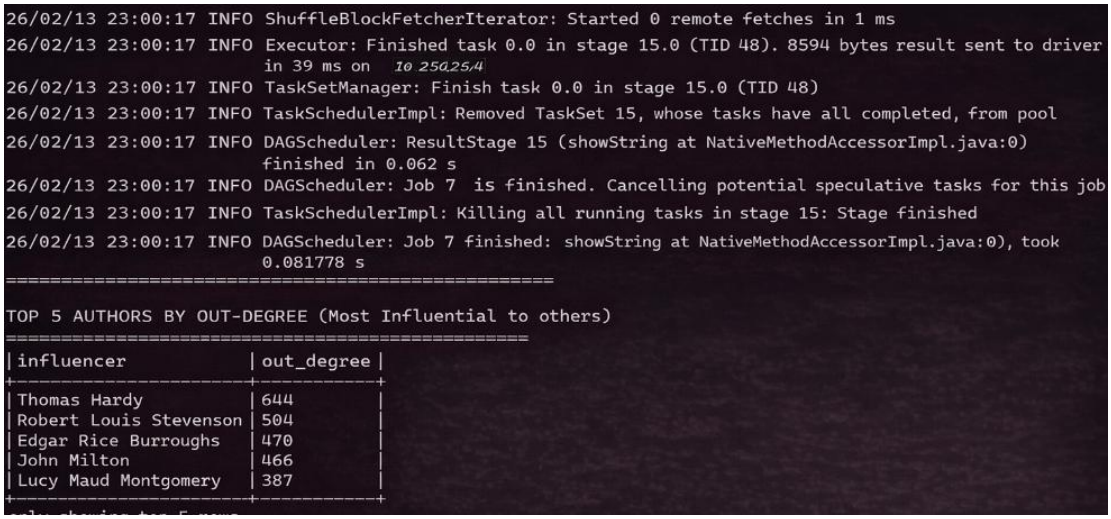
- It is computationally efficient and widely used in text analysis

Calculating pairwise cosine similarity does not scale well because the number of document comparisons grows quadratically with the dataset size. This leads to high computational cost, large memory usage, and long execution times for large document collections.

Apache Spark addresses these challenges by distributing computations across multiple nodes, processing data in memory, and enabling parallel similarity calculations. Spark also allows optimization techniques such as limiting comparisons to top-K similar documents and applying similarity thresholds, making large-scale document similarity computation feasible.

Q12





Representation of the Influence Network

- The influence network was represented using a **Spark DataFrame**, where each row represents an edge between two authors.
- This representation was chosen because DataFrames are structured, easy to query, and optimized by Spark's execution engine.

Advantages:

- Efficient aggregation for computing in-degree and out-degree.
- Optimized execution using Spark SQL and Catalyst optimizer.
- Easy integration with other Spark transformations.

Disadvantages:

- Less flexible than RDDs for custom graph algorithms.
- Slight overhead due to schema enforcement.

Effect of Time Window (X) on Network Structure

- A **small time window (X)** results in a sparse influence network with fewer edges.
- A **large time window (X)** produces a denser network with more connections.
- Increasing X may introduce weaker or less meaningful influence relationships.

Limitations of the Simplified Influence Definition

- Temporal proximity does not guarantee actual influence.

- Ignores genre, citations, and historical relationships between authors.
- Assumes influence is purely time-based, which is an oversimplification.

Scalability and Optimizations for Large Datasets

- Pairwise comparisons become expensive for millions of books and authors.
- Memory and computation costs increase significantly.

Possible optimizations:

- Partition data by time ranges to reduce comparisons.
- Apply filtering or thresholds to limit edges.
- Use broadcast joins where applicable.
- Spark graph libraries such as **GraphFrames** for scalable graph processing.