Đại học Khoa học Tự nhiên – ĐHQG TP.HCM

----------□□----------

Course: Introduction of Big Data

Lab 2: Document Clustering with Hadoop MapReduce

Class: 21KHMT1

Teacher:

Mrs. Nguyễn Ngọc Thảo

Mr. Bùi Huỳnh Trung Nam

Mr. Đỗ Trọng Lễ

Student information:

21127456 – Võ Cao Trí

21127608 – Trần Trung Hiếu

21127668 – Đinh Quang Phong

Assign task:

NO	Name	Task	Completion level
1	Tri, Hieu	Task_1_1	100%
2	Tri, Phong	Task_1_2	100%
3	Tri, Phong	Task_1_3	100%
4	Hieu, Tri	Task_1_4	100%
5	Tri, Hieu	Task_1_5	100%
6	Phong, Hieu	Task_2_1	100%
7	Phong, Tri	Task_2_2	100%
8	Tri, Hieu, Phong	Write report	100%

1. Data Description

- Dataset: BBC
- All rights, including copyright, in the content of the original articles are owned by the BBC.
- Consists of 2225 documents from the BBC news website corresponding to stories in five topical areas from 2004-2005.
- Class Labels: 5 (business, entertainment, politics, sport, tech)
- Stop word list: a file containing common words like “the” “a” “an”.
- 2D data points

2. Data Preprocessing

2.1. Task 1.1: Text Cleaning and Term Frequency

- Input: **bbc folder.**
- Output: **task_1_1.mtx**
- Mapper class
    - It reads a file (stop words) to skip during processing.
    - Processing Each Line: For each input line, the mapper performs the following steps:
        - Converts the line to lowercase (if case-insensitive mode is enabled).
        - Splits the line into individual words using the regular expression \\s*\\b\\s*
        - Filters out empty words.
        - Filters out words that start with non-alphanumeric characters.
        - Filters out words that match the stop words defined in the patternsToSkip set.
        - Constructs a key-value pair where the key is a combination of the word and the filename.
        - Emits the key-value pair.
            - Key: Concatenation of the word and the file name, separated by a space.
            - Value: The value is a constant (one), indicating the occurrence of the word.
- Reducer class
    - It operates on the intermediate key-value pairs generated by the mappers.
    - For each unique word, the reducer aggregates the associated values (counts) to produce a final result.
    - Processing each word: For every unique word encountered, the reducer iterates through the associated counts.
    - It accumulates the counts by adding them together.
    - The goal is to compute the total occurrence count for each word.

    - Emitting Key-Value Pairs:
        - Key: Concatenation of the word and the file name, separated by a space.
        - Value is the frequency(sum of occurrences) for that word.

2.2. Task 1.2: Low-Frequency Term Elimination

- Input: **task_1_1.mtx**
- Output: **task_1_2.mtx**
- The mapper class: For each input line (representing a word, a document and its frequency), it performs the following steps:
    - Splits the input line into parts using whitespace as the delimiter.
    - Checks if the line contains exactly three parts (word, filename, and frequency).
    - Parses the frequency value as an integer.
    - If the frequency is greater than or equal to 3, construct a key by concatenating the word and filename (separated by a space).
    - Emits the key-value pair where the key is the combined word and filename, and the value is the frequency.
- The Reducer class
    - Aggregates the intermediate key-value pairs generated by the mappers.
    - For each unique key (constructed word + document name), it iterates through the associated values (counts).
    - It accumulates the counts by adding them together.
    - Emitting Key-Value Pairs in the IntSumReducer:
    - After processing all the counts for a specific word, the reducer emits a key-value pair.
        - Key (word + " " + filename).
        - Value is the frequency (sum of occurrences).

2.3. Task 1.3: Top 10 Most Frequent Words

- Input: **task_1_2.mtx**
- Output: **task_1_3.txt**
- Map class:
    - The Map function processes input data in the form of key-value pairs.
    - The input text is split into parts using whitespace as the delimiter (using value.toString().split("\\s+")).
    - If the split results in an array of length 3 (indicating that the line contains three parts), the frequency (an integer) is parsed from the third part.
- Reduce Class:
    - The Reduce function processes intermediate key-value pairs generated by the Map function.
    - It receives a key (a Text object representing a term) and an iterable collection of values (a list of IntWritable objects representing frequencies).
    - The Reduce function calculates the total frequency (sum of occurrences) for each term by iterating through the values.

- The term-frequency pairs are stored in a TreeMap called frequencyMap, where the key is the total frequency, and the value is a list of terms with that frequency.
- The cleanup method is called after all intermediate pairs have been processed. It selects the top 10 terms (based on frequency) from the frequencyMap and emits them as the final output.
- Key-Value Pair Format:
  - Key: The key is a concatenation of the term and the file name, separated by a space.
  - Value: The value is the frequency (sum of occurrences) for that term.

2.4. Task 1.4: TF-IDF

2.5.1. Task 1.4.1: calculate the number of words in each docID

- Input file: **task_1_2.mtx**
- Output file: **task_1_4_1.mtx**
- hadoop jar target/task_1_4_1.java-1.0-SNAPSHOT.jar task_1_4.task_1_4_1 /output1/task_1_2.mtx /output_task_1_4/task_1_4_1/
- Map class: transform input text into new key - value set included
  - The input of the map cuts the string of each line of the data file **task_1_2.mtx** and we have the form termID docID frequency in Text form
  - The map's output have
    - key: docID in Text format
    - value: termID frequency in Text format
- Reduce class: calculate the number of words in each docID
  - input of reduce
    - key: Text
    - value: Iterable<Text>
  - output of reduce
    - key: docID in Text format
    - value: number of words in each docID in Text format
- output example of file **task_1_4_1.mtx**

2.5.2. Task 1.4.2: Combine data from two files **task_1_2.mtx**, **task_1_4_1.mtx** to create the output termID docID frequency wordsPerDoc

- Input file: **task_1_2.mtx, task_1_4_1.mtx**
- Output file: **task_1_4_2.mtx**
- hadoop jar target/task_1_4_2.java-1.0-SNAPSHOT.jar task_1_4.task_1_4_2 /output1/task_1_2.mtx /output_task_1_4/task_1_4_2/ -skip /output_task_1_4/task_1_4_1/task_1_4_1.mtx
- Map class:
    - Setup method: Take each line of data from task_1_4_1.mtx and transfer it to the HashTable cache including
        - key: docID in String format
        - value: wordsPerDoc in Integer format
    - Feature: Take each line of data of the file **task_1_2.mtx**, separate it into substrings including termID docID frequency and merge it with the value of the HashTable in the cache of the corresponding docID to find wordsPerDoc
    - Input: Text
    - Output
        - Key: termID docID in Text format
        - Value: frequency wordsPerDoc in Text format
- Reduce class: Perform data writing to file
    - Output
        - Key: termID docID in Text format
        - Value: frequency wordsPerDoc in Text format
- output example of file task_1_4_2.mtx

```
0 business/006.txt 3 21
0 business/023.txt 4 20
0 business/044.txt 4 66
0 business/046.txt 4 46
0 business/063.txt 9 73
0 business/101.txt 3 57
0 business/110.txt 3 24
0 business/153.txt 4 80
0 business/156.txt 3 21
0 business/170.txt 3 50
0 business/262.txt 5 113
0 business/405.txt 3 114
0 business/453.txt 5 51
0 business/462.txt 3 47
0 business/507.txt 5 58
0 politics/130.txt 3 65
0 sport/179.txt 5 125
```

2.5.3. Task 1.4.3: Determine the number of docIDs that contain termID

- Input file: **task_1_2.mtx**
- Output file: **task_1_4_3.mtx**
- hadoop jar target/task_1_4_3.java-1.0-SNAPSHOT.jar task_1_4.task_1_4_3 /output1/task_1_2.mtx /output_task_1_4/task_1_4_3/
- Map class:
    - Take each line of data from file **task_1_2.mtx** and cut it into subStrings of the form termID docID frequency
    - Output:
        - Key: termID in Text format
        - value: 1 in IntWritable format
- Reduce class:
    - Calculate the number of termIDs of the returned map
    - Output:
        - Key: termID in Text format
        - Value: number of docs have termID in IntWritable format
- output example of file task_1_4_3.mtx

5 35
50 5
500 6
500m 1
50m 3
50th 1
53 1
5bn 4
5m 3
6 59
60 1
60m 7
64 2
65 1
6bn 2
7 35
70 1

2.5.4. Task 1.4.4: Combine data from two files **task_1_4_2.mtx, task_1_4_3**.mtx to create data streams of the form termID docID frequency wordsPerDoc docsPerWord

- Input file: **task_1_4_2.mtx, task_1_4_3.mtx**
- Output file: **task_1_4_4.mtx**
- hadoop jar target/task_1_4_4.java-1.0-SNAPSHOT.jar task_1_4.task_1_4_4 /output_task_1_4/task_1_4_2/task_1_4_2.mtx /output_task_1_4/task_1_4_4/ -skip /output_task_1_4/task_1_4_3/task_1_4_3.mtx
- Map class:
    - Setup method: Take each line of data from task_1_4_3.mtx and transfer it to the HashTable cache including
        - Key: termID in String format
        - Value: number of docs have termID - docsPerWord in Integer format
    - Feature: Take each line of data of file **task_1_4_2.mtx**, split into substrings including termID docID frequency wordsPerDoc and combine with the value of the HashTable in cache with each corresponding termID to find filesPerWord
    - Input: Text
    - Output
        - Key: termID docID in Text format
        - Value: frequency wordsPerDoc filesPerWord in Text format
- Reduce class: Perform data writing to file
    - Output
        - Key: termID docID in Text format
        - Value: frequency wordsPerDoc filesPerWord in Text format
- output example of file **task_1_4_4.mtx**

0 business/006.txt 3 21 24
0 business/023.txt 4 20 24
0 business/044.txt 4 66 24
0 business/046.txt 4 46 24
0 business/063.txt 9 73 24
0 business/101.txt 3 57 24
0 business/110.txt 3 24 24
0 business/153.txt 4 80 24
0 business/156.txt 3 21 24
0 business/170.txt 3 50 24
0 business/262.txt 5 113 24
0 business/405.txt 3 114 24
0 business/453.txt 5 51 24
0 business/462.txt 3 47 24
0 business/507.txt 5 58 24
0 politics/130.txt 3 65 24
0 sport/179.txt 5 125 24
0 sport/371.txt 6 417 24
0 sport/488.txt 3 84 24
0 sport/491.txt 7 508 24
0 tech/032.txt 3 198 24
0 tech/138.txt 3 98 24

2.5.5. Task 1.4.5: Determines the number of documents present in the data set

- Input file: **task_1_2.mtx**
- Output file: **task_1_4_5.mtx**
- hadoop jar target/task_1_4_5.java-1.0-SNAPSHOT.jar task_1_4.task_1_4_5 /output1/task_1_2.mtx /output_task_1_4/task_1_4_5/
- Map class: Take each line of data of file task_1_2.mtx and cut it into small subStrings of the form termID docID frequence
    - output:
        - key: docID in Text format
        - value: 1 in IntWritable format
- Reduce class:
    - Calculate the number of files in the returned data by adding docID and Set
    - Cleanup method: used to write data including
        - key: "file" in Text format
        - value: size of Set is the number of documents contained in the dataset
- output example of file **task_1_4_5.mtx**

file 2224

2.5.6. Task 1.4.6: calculate tf*idf

- Input file: **task_1_4_4.mtx, task_1_4_5.mtx**
- Output file: **task_1_4.mtx**
- hadoop jar target/task_1_4_6.java-1.0-SNAPSHOT.jar task_1_4.task_1_4_6 /output_task_1_4/task_1_4_4/task_1_4_4.mtx /output_task_1_4/task_1_4_6/ -skip /output_task_1_4/task_1_4_5/task_1_4_5.mtx
- Map class:

    - Setup method: Take each line of data from task_1_4_5.mtx and transfer it to the HashTable cache including
        - Key: "file" in String format
        - Value: the number of documents contained in the dataset in Integer format
    - Feature: Take each line of data of file **task_1_4_4.mtx**, split into substrings including termID docID frequency wordsPerDoc docsPerWord and use the the number of documents to calculate the tf*idf
    - Output
        - Key: termID docID
        - value: tf*idf
- Reduce class:Perform data writing to file
    - Output
        - Key: termID docID
        - value: tfidf

2.5. Task 1.5: Highest average tfidf

2.5.1. Count files in the folder

- Input: the folder
- Output: the folder name and the number of files the folder contains.

```
business 510
entertainment 386
politics 417
sport 511
tech 401
```

2.5.2. Calculate avg_tfidf

- Input: **task_1_4.mtx** and **numFile.txt** from 2.6.1
- Output: key termid + folder and the value: avg_iftdf.
- Mapper
    - The lineText is split into an array of strings using whitespace as the delimiter.
    - The second part of the split above is further split using the forward slash ("/") as the delimiter.
    - The key is constructed by concatenating the first part of the original line (parts[0]) with the first part of the second split (part1[0]) to get the termid and the folder of this term.
    - The fileNumValue is retrieved from a Hashtable (indexed by part1[0]) to get the number of files in folder part[1]0.
    - The function calculates the result by dividing the tfidf by fileNumValue.
- Reduce Function:
    - The Reduce function processes intermediate key-value pairs generated by the Map function.
    - For each key, it iterates through the values associated with that key.

- The function accumulates the values (which represent partial results) by adding them to the sum.
- The final key-value pair emitted is (key, sum).

- Format of Key-Value Pairs:
  - In the Map function, the emitted key is termid + “ ” + foldername.
  - The value emitted in the Map function is avg_tfidf/ number of files in folder.
  - In the Reduce function, the emitted key remains the same.
  - The value emitted in the Reduce function is the accumulated.

File contents

```
ard sport 5.9546967E-4
asahi business 0.0010938314
asbestos business 7.5726665E-4
asbos politics 5.6E-4
ash tech 3.210324E-4
ashcroft politics 5.2176975E-4
ashdown sport 3.6389433E-4
ashley business 9.375725E-4
asia business 0.0027165941
```

2.5.3. Top 5 high avg_tfidf of each folder.

- Input: **avg_iftdf.txt** from 2.6.2
- Output: top 5 with high avg_iftdf(key: folder name, termid and the value: avg_iftdf) in each folder.
- The Map function processes input data in the form of key-value pairs.
  - The lineText is split into an array of strings (parts) using whitespace as the delimiter.
  - If the length of parts is equal to 3, the function proceeds.
  - The key is parts[1] + " " + parts[0] (folder name and termid)
  - The value is parts[3] (avg_tfidf)
- Reduce Function:
  - The Reduce function processes intermediate key-value pairs generated by the Map function.
  - For each key, it iterates through the values associated with that key.
  - The key is split into parts, and the first part (folder name) is extracted.
  - If the topFrequencies map does not contain an entry for the folder name, a new entry is created with a reverse-ordered TreeMap.
  - The frequencyMap within the topFrequencies map is updated to include the calculated sum and the corresponding key.
  - If the size of frequencyMap exceeds 5, the entry with the highest key (frequency) is removed.
  - The cleanup method is called after all key-value pairs have been processed.
- Format of Key-Value Pairs:
  - In the Map function, the emitted key is folder + “ ” + termid

- The value emitted in the Map function is avg_tfidf.
- In the Reduce function, the emitted key and value remain the same.

3. K-Means Algorithm

3.1. Task 2.1: K-Means on 2D Data

- Input: **2DPoints.csv** each row is in a format of **class,x1,x2**
- Output: Center points result for each cluster in file **task_2_1.clusters** and Cluster assignment for each data point in file **task_2_1.classes.**
- Mapper**:**
  - Purpose: Assigns each data point to its closest cluster based on current centroid positions.
  - Key Steps:
    - Retrieves current centroids: Reads centroid coordinates from configuration data set in the setup phase (Initial centroids are chosen randomly from the beginning of the main function).
    - Processes each input data point:
      - Parse coordinates (class,x1,x2) from the point string.
        - class : Integer
        - x1 : Double
        - x2 : Double
      - Calculates Euclidean distance to each centroid.

        $$\sqrt{(class2 - class1)^2 + (x2' - x2)^2 + (x1' - x1)^2}$$
      - Identifies the closest centroid using the minimum distance.
      - Emits a key-value pair:
        - Key: The index of the closest centroid (IntWritable type, representing a cluster).
        - Value: The coordinates of the data point (Text type format: **class** + **“,”** + **x1** + **“,”** + **x2**).
- Combiner:
  - Purpose: Calculate new centroid vector based from mapper output and write out cluster assignment for each data points to file **task_2_1.classes**
  - Key Steps:
    - For each key-value pairs given by mapper the new centroids component is calculated:
      - New centroid class = average of sum of all values classes
      - New centroid x1 = average of sum of all values x1
      - New centroid x2 = average of sum of all values x2
    - Write to **task_2_1.classes** with key is Cluster index and value is the assigned point.
    - Emits a key-value pair to Reducer stage:
      - Key: The index of the closest centroid (IntWritable type, representing a cluster).

- Value: The coordinates of the new centroid coordination (Text type format: **class + “,” + x1 + “,” + x2**).

- Reducer**:**
  - Purpose: Calculate new centroid vectors based on combiner output.
  - Key Steps:
    - All steps are same as that of Combiner stage except Reducer stage don’t write out cluster assignment for each data point in file **task_2_1.classes** only emitting the final cluster centers to file **task_2_1.clusters.**

➢ Key Points:
  - Only at the first iteration that Mapper would read the randomly selected centroids but the rest iterations each Mapper would read the previous emitted centroids to generate a new one until they remain unchanged, that is when the Program stops or until **Max_iteration** is reached.
  - The Combiner serves as a preprocessing stage for reducers to partially generate new centroids much faster.

3.2. Task 2.2: K-Means on Preprocessed Data

- Input: **task_1_4.mtx** file from task 1.4 each row is in a format: **termid docid tfidf** separated by single space.
- Output: Cluster centers result in file **task_2_2.clusters** and Cluster assignment for each data point in file **task_2_2.classes.**
- Mapper**:**
  - Purpose: Assigns each data point to its closest cluster based on current centroid positions.
  - Key Steps:
    - Retrieves current centroids: Reads centroid coordinates from configuration data set in the setup phase (Initial centroids are chosen randomly from the beginning of the main function).
    - Processes each input data point:
      - Parse vectors (termid,docid,tfidf) from the point string:
        - termid : as String type
        - docid : as String type
        - tfidf : double type

Calculates Euclidean distance to each centroid

$$\sqrt{(termid2 - termid1)^2 + (docid2 - docid1)^2 + (tfidf2 - tfid12)^2}$$

- The subtraction for termid and docid explanation:
  - For termid : Using **Jaccard Similarity** if term1 = term2 then result will return Integer 1 else 0
  - For docid: If the **folder name** of docid1 = folder name of docid2 then return 1 else 0. Ex: folder1/12.txt = folder1/34.txt
- Identifies the closest centroid using the minimum distance.

        - Emits a key-value pair:
            - Key: The index of the closest centroid (IntWritable type, representing a cluster).
            - Value: The coordinates of the data point (Text type format: **termid** + “ ” + **docid** + “ ” + **tfidf**).
- Combiner:
    - Purpose: Calculate new centroid vector based from mapper output and write out cluster assignment for each data points to file **task_2_2.classes**
    - Key Steps:
        - For each key-value pairs given by mapper the new centroids component is calculated:
            - New centroid termid = the most common termid between all values
            - New centroid docid = the foldername is the most common between all values + average sum of fileID of all values
            - New centroid tfidf = average of sum of all values tfidf
        - Write to **task_2_2.classes** with key is Cluster index and value is the assigned TFIDF vectors.
        - Emits a key-value pair to Reducer stage:
            - Key: The index of the closest centroid (IntWritable type, representing a cluster).
            - Value: The coordinates of the new centroid coordination (Text type format:**termid** + “ ” + **docid** + “ ” + **tfidf**).
- Reducer**:**
    - Purpose: Calculate new centroid vectors based on combiner output.
    - Key Steps:
        - All steps are same as that of Combiner stage except Reducer stage don’t write out cluster assignment for each data point in file **task_2_2.classes** only emitting the final cluster centers to file **task_2_2.clusters.**

➢ Key Points:

- Only at the first iteration that Mapper would read the randomly selected centroids but the rest iterations each Mapper would read the previous emitted centroids to generate a new one until they remain unchanged, that is when the Program stops or until **Max_iteration** is reached.
- The Combiner serves as a preprocessing stage for reducers to partially generate new centroids much faster.
- Most steps are same as Task 2.1 only the calculation is different

4. Results

4.1. Task 1.1:

```
050505 politics/395.txt 1
0530 tech/117.txt 1
0530gmt politics/126.txt 1
056 entertainment/168.txt 1
056 tech/093.txt 1
05bn business/051.txt 1
05bn business/349.txt 1
05m sport/006.txt 1
06 business/154.txt 1
06 business/410.txt 1
06 business/450.txt 1
06 entertainment/057.txt 3
06 entertainment/329.txt 1
06 politics/128.txt 1
06 sport/060.txt 2
060 sport/170.txt 1
0619 business/089.txt 1
0630 tech/117.txt 1
069 business/503.txt 1
06bn business/078.txt 1
07 business/262.txt 1
```

- Difficulty:
    - Simultaneously calling multiple input files from the bbc folder and stopword.txt file to remove unnecessary characters
    - Difficulty renaming files and changing the format of output files in map reduce
    - Search for tools to debug when execution fails → use logger

4.2. Task 1.2:

5 sport/450.txt 3
5 sport/451.txt 3
5 sport/468.txt 3
5 sport/480.txt 3
5 sport/510.txt 4
5 tech/129.txt 3
5 tech/175.txt 3
5 tech/196.txt 3
50 business/152.txt 3
50 entertainment/282.txt 14
50 entertainment/285.txt 6
50 entertainment/288.txt 9
50 politics/115.txt 3
500 business/049.txt 3
500 business/103.txt 3
500 politics/365.txt 3
500 tech/141.txt 3
500 tech/173.txt 3
500 tech/268.txt 3
500m politics/181.txt 3
50m sport/112.txt 4
50m sport/118.txt 3
50m sport/121.txt 4
50th sport/115.txt 4
53 business/045.txt 4
5bn business/166.txt 4
5bn business/277.txt 3
5bn business/441.txt 4
5bn politics/366.txt 3

4.3. Task 1.3:

File contents

```
s     9070
i     3906
year 2309
people    2045
one  1892
first  1355
t     1343
time 1322
two  1282
world      1201
```

4.4. Task 1.4:

```
9bn business/441.txt 0.17164799571037292
9m entertainment/312.txt 0.2175769954919815
9m entertainment/313.txt 0.24698099493980408
a350 business/335.txt 0.3042849898338318
aaas sport/058.txt 0.0507659986615181
aaas sport/061.txt 0.10814300179481506
aac sport/068.txt 0.06236699968576431
aadc tech/013.txt 0.10246200114488602
aaliyah entertainment/230.txt 1.3388539552688599
abba entertainment/265.txt 0.41163501143455505
abba entertainment/346.txt 0.2175769954919815
abbas politics/149.txt 0.20768600702285767
abbas politics/209.txt 0.40614598989486694
abbott sport/383.txt 0.0766490027308464
abc entertainment/115.txt 0.358364999294281
abc entertainment/205.txt 0.2769179940223694
aberdeen sport/107.txt 0.133884996175766
abeyie sport/163.txt 0.133884996175766
able politics/275.txt 0.12405399978160858
able politics/378.txt 0.09176500141620636
```

- Difficulty:
    - Call 2 executable files from 2 directories at the same time
    - Determine the input and output data types of map and reduce

4.5. Task 1.5:

- The result of avg_tdifd

File contents

```
aru sport 5.9546967E-4
asahi business 0.0010938314
asbestos business 7.5726665E-4
asbos politics 5.6E-4
ash tech 3.210324E-4
ashcroft politics 5.2176975E-4
ashdown sport 3.6389433E-4
ashley business 9.375725E-4
asia business 0.0027165941
```

- The result of task_1_5

File contents

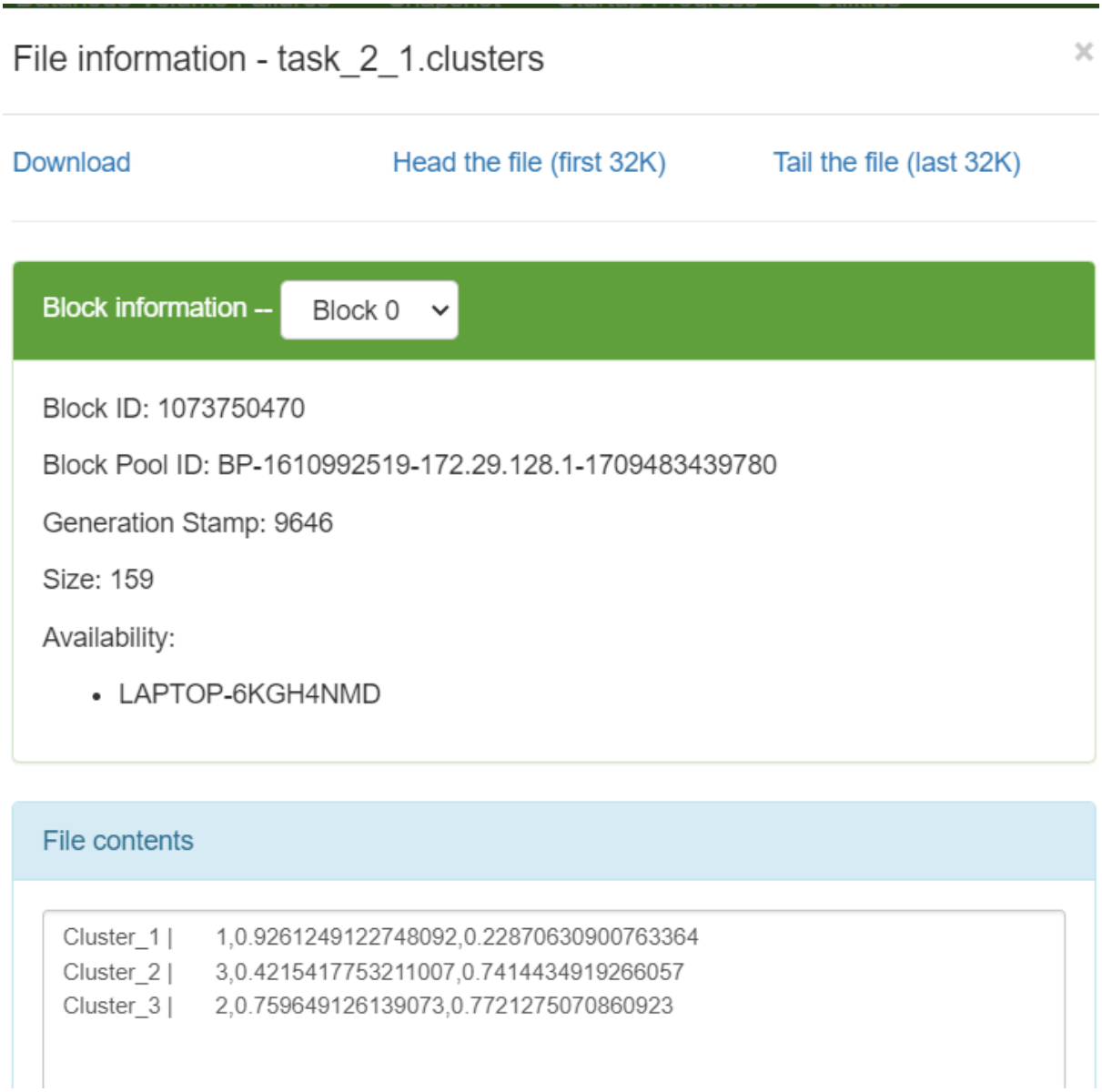
```
tech people 0.02083414
tech mobile 0.018996596
tech software 0.017560793
tech games 0.016836286
tech users 0.015308779
politics labour 0.03257529
politics election 0.029285165
politics blair 0.028275808
politics party 0.026508622
politics brown 0.020817501
business sales 0.01767237
business oil 0.016955897
business economy 0.016709367
business growth 0.016189849
business bank 0.01616632
entertainment film 0.055315044
entertainment best 0.031092886
entertainment show 0.021819033
entertainment band 0.01854118
entertainment festival 0.017690867
sport i 0.057579543
sport 6 0.02189407
sport england 0.019108051
sport o 0.014443213
sport wales 0.01362502
```

4.6. Task 2.1:

- Output sample:

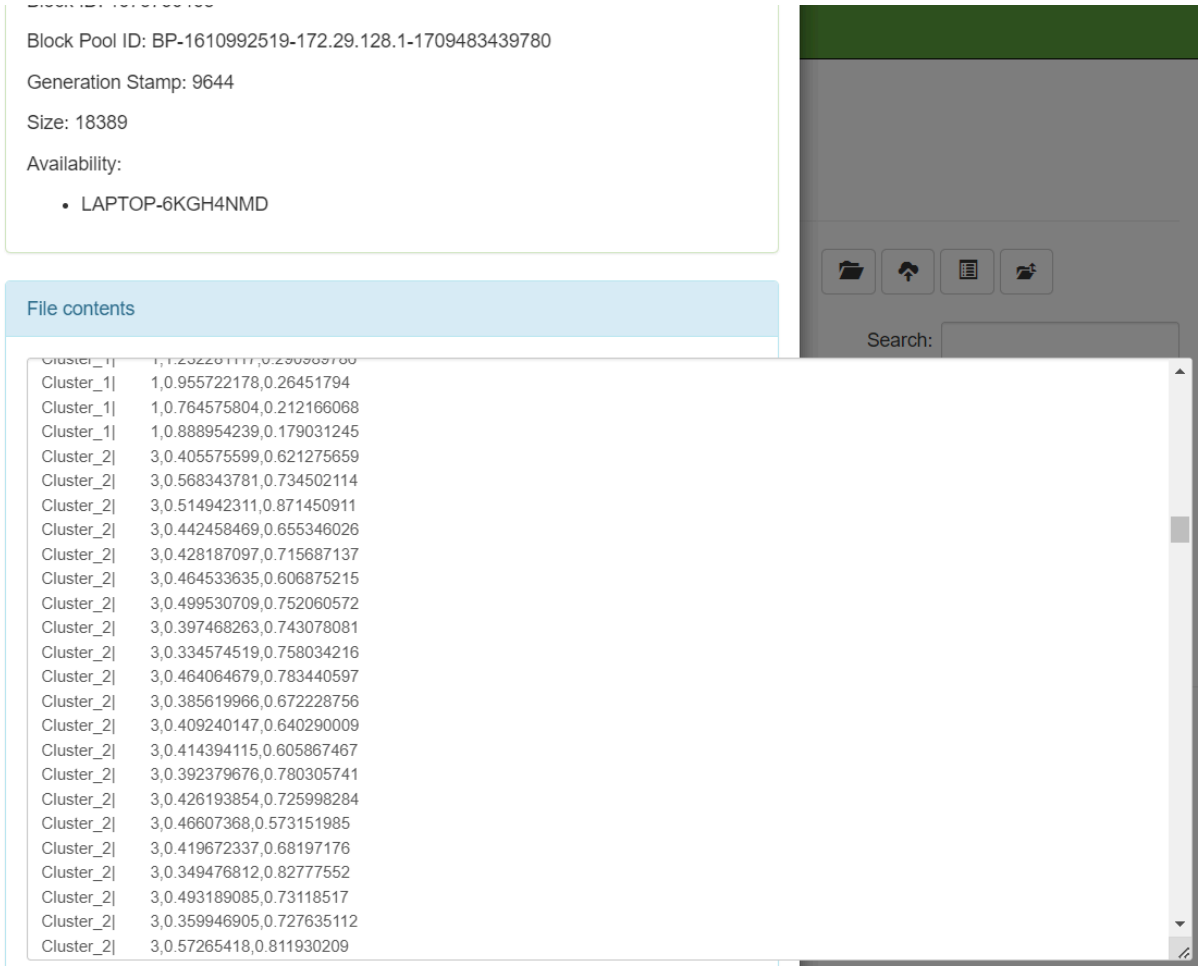
- Content of task_2_1.clusters:

On K = 3



- Content of task_2_1.classes:

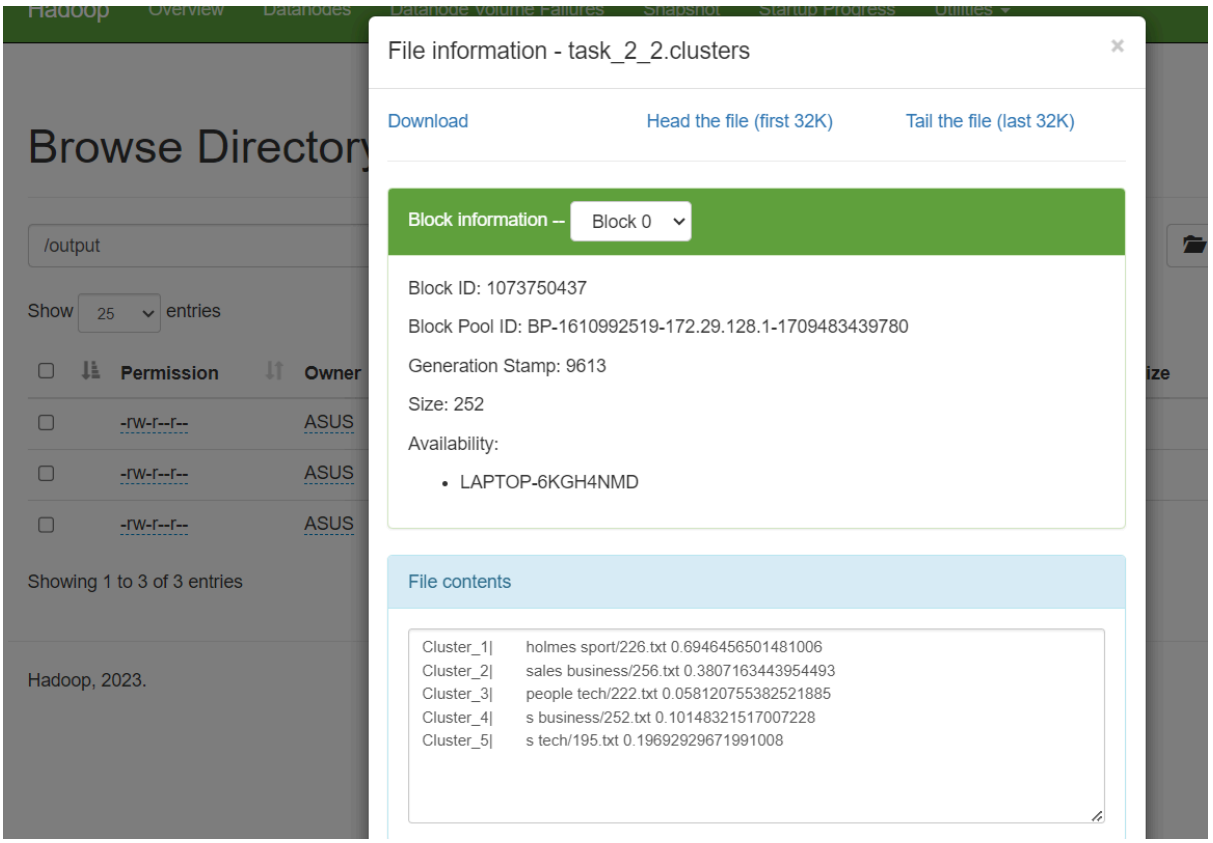
On K = 3



4.7. Task 2.2:

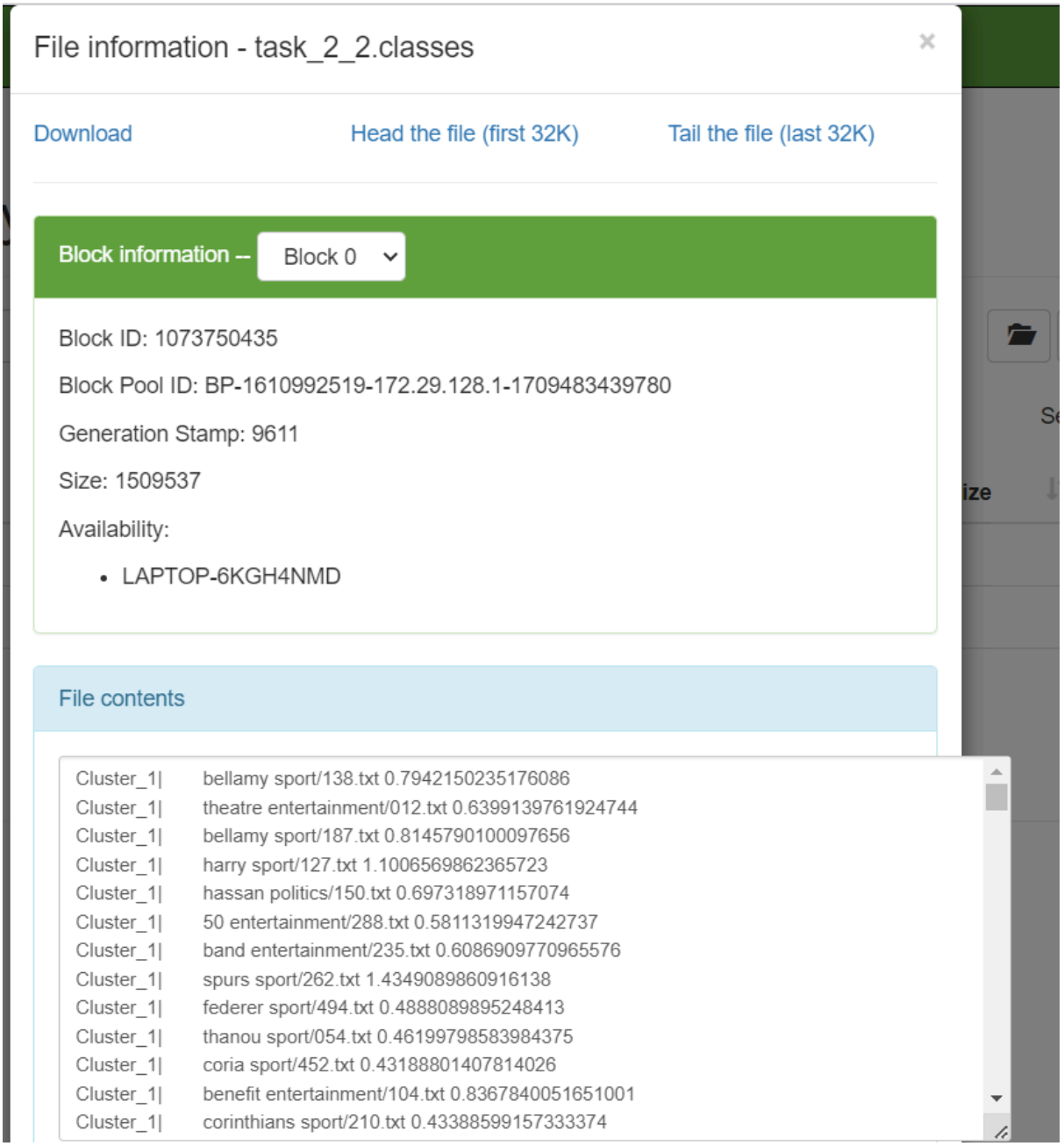
- Output sample:
    - Content of task_2_2.clusters:

On K = 5, Max iteration = 10:



- Content of task_2_2.classes

On K = 5, Max iteration = 10:



- Difficulty: Hard to choose a suitable distance metric for TFIDF that could make them convergence

5. Conclusion

- Data Preprocessing: It involves cleaning, transforming, and organizing raw data to make it suitable for further processing.
- Hadoop MapReduce: MapReduce is commonly used for big data processing, including tasks like word counting, data aggregation, and document clustering.
- Find out how Kmean works and Kmean collects words with the same meaning so you can find words that have the same theme and are easy to recognize, analyze trends through words with the same or close meaning.

6. Reference

[1]https://www.youtube.com/watch?v=knAS0w-jiUk
[2]https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/ClusterSetup.html
[3]https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/SecureMode.html
[4]https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/SingleCluster.html
[5]https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html#Example%3A_WordCount_v1.0
[6]https://research.google/pubs/mapreduce-simplified-data-processing-on-large-clusters/
[7]https://storage.googleapis.com/gweb-research2023-media/pubtools/pdf/16cb30b4b92fd4989b8619a61752a2387c6dd474.pdf
[8]https://www.slideshare.net/oom65/ hadoop-security-architecture.
[9]Word Count
[10]Install Hadoop
[11]https://github.com/seraogianluca/k-means-mapreduce.git
[12]https://www.youtube.com/watch?v=KzJORp8bgqs&t=366s&pp=ygUQS01lYW4gY2x1c3RlcmluZw%3D%3D
[13]https://github.com/arulalanv/WordCount-after-StopWords-Removal-using-MapReduce