

VNUHCM - UNIVERSITY OF SCIENCE  
FACULTY OF INFORMATION TECHNOLOGY



---

## Report Lab 02

### Document Clustering with Hadoop MapReduce

Course name: Introduction to Big Data

---

*Student:*

21127063 - Nguyen Van Dang Huynh  
21127528 - Nguyen Thi Minh Minh  
21127183 - Pham Phu Toan  
21127089 - Nguyen Tuan Kiet

*Instructor:*

Teacher: Nguyen Ngoc Thao  
Teacher: Bui Huynh Trung Nam  
Teacher: Do Trong Le

31st March 2024

# Contents

## Tasks on Lab 02

No.	Task title	Percent of completion
1	1.1 Text Cleaning and Term Frequency	100%
2	1.2 Low-Frequency Term Elimination	100%
3	1.3 Top 10 Most Frequent Words	100%
4	1.4 TF-IDF	100%
5	1.5 Highest average tfidf	100%
6	2.1 K-Means on 2D Data	100%
7	2.2 K-Means on Preprocessed Data	50%
8	2.3 Scalable K-Means++ Initialization	10%

# 1 Data Preprocessing:

## 1.1 Text Cleaning and Term Frequency:

### 1.1.1 Data description:

**Input:** the dataset from BBC which is included in the questionnaire. (divided into 5 different classes - each class's document is inside the folder with the same name of class, each class contains multiple docs). **Output:** part-r-\* and an **output.mtx** file (inside MTX folder in the output directory) with the structure:

- The first row is Matrix Header.
- The second row is the summarize rows (the number of terms, the number of docs, the numbers of data rows in files).
- The entire remaining rows are the data rows with the format likes: `<termid, docid, frequency>`.

### 1.1.2 MapReduce Job Implementation:

**Number of mapper-reducer:** 1 only.

**The flow of code:**

---

**Algorithm 1** Text Cleaning and Term Frequency

---

1: Firstly, we create the stopwords, the termid list, the docid list by reading the bbc preprocess files (just reading like a normal java program - the path is encoded in file).

2: **In mapper - output:** `<termid + docid, 1>`:

- Filter the term if it is the stopwords and being in defined termid list (each word being read from doc will be processed to remove the dumping character, we also considered the situation the plural form of the term).
- Mapping term and doc to specific termid and docid.

3: **In reducer - output:** `<termid + docid, freq>`:

- Sum all the occurrences of the term inside a doc.

4: **Output function:**

- This is the function to comprise the part-r-files into the output.mtx file. It will be run after the mapreduce have done its job.
-

### 1.1.3 Results:

```
te > Lab2 > task1_1 > output.mtx
1  %%MatrixMarket matrix coordinate real general
2  9635 2225 160726
3  1 1 1
4  1 11 1
5  1 1199 1
6  1 1228 2
7  1 1242 2
8  1 1905 1
9  1 1913 1
10 1 1920 1
11 1 1948 2
12 1 1962 1
13 1 1973 1
14 1 1986 1
15 1 2030 4
16 1 2033 1
17 1 2091 1
18 1 2134 1
19 1 2188 1
20 1 2201 1
21 1 2217 1
22 1 35 1
23 1 443 1
24 1 551 1
25 1 683 1
```

Figure 1: The output of the first task

**Challenge:** We could hardly read all the forms of term inside the docs (For instance: ‘jump’ following with ‘ed’ at the end will not be considered as a right term although the term ‘jump’ is in the term list). We had to extract the word with regex, also considered the plural form of term but the number of terms across the documents we could recognized is not as enough as in BBC final results. The format mtx file was new to us and we spent a lot of time to find way to adapt to this required format.

## 1.2 Low-Frequency Term Elimination:

### 1.2.1 Data description:

**Input:** the output.mtx file from task 1.1. **Output:** part-r-\* files and an output.mtx file (inside MTX folder inside the output path) with the structure:

- The first row is Matrix header.

- The second row is the summarize rows (the number of terms, the number of docs, the numbers of data rows in files).
- The entire remaining rows are data rows with format likes: `<termid, docid, frequency>`.

### 1.2.2 MapReduce Job Implementation:

**Number of mapper-reducer:** 1 only.

**The flow of code:**

---

**Algorithm 2** Low-Frequency Term Elimination

---

- 1: Firstly, we create the termid list, the docid list by reading the bbc preprocess files (just reading like a normal java program - the path is encoded in file).
  - 2: **In mapper - output:** `<termid, docid + freq>`:
    - We read the `output.mtx` file and group the data by term.
  - 3: **In reducer - output:** `<termid, docid + freq>`:
    - Sum all the occurrences (freq) of the term across the docs.
    - Filter the term where the sum is less than 3.
    - Write the filtered term the output with the same format as task 1.1.
  - 4: **Output function:**
    - This is the function to comprise the part-r-files into the `output.mtx` file. It will be run after the mapreduce have done its job.
-

### 1.2.3 Results:

```
1 %%MatrixMarket matrix coordinate real general
2 9635 2225 159987
3 1 1 1
4 1 11 1
5 1 1199 1
6 1 1228 2
7 1 1242 2
8 1 1905 1
9 1 1913 1
10 1 1920 1
11 1 1948 2
12 1 1962 1
13 1 1973 1
14 1 1986 1
15 1 2030 4
16 1 2033 1
17 1 2091 1
18 1 2134 1
19 1 2188 1
20 1 2201 1
21 1 2217 1
22 1 35 1
23 1 443 1
24 1 551 1
```

Figure 2: The output of the second task

**Challenge:** The instruction for this task is clearly written in the lab paper so we have misunderstood at the first time.

## 1.3 Top 10 Most Frequent Words:

### 1.3.1 Data description:

**Input:** The textbfoutput.mtx file from Task 1.2. Exclude the first 2 rows, the rest of the file has the format of `<termId, docId, frequency>`

**Output:** A `part-r-*` file and a text file (`task_1_3.txt`) that contain a list of the top 10 most frequently occurring terms (across all documents) with the format `<termid, frequency>`.

### 1.3.2 MapReduce Job Implementation:

**Number of mapper-reducer:** 1 only.

**The flow of code:**

---

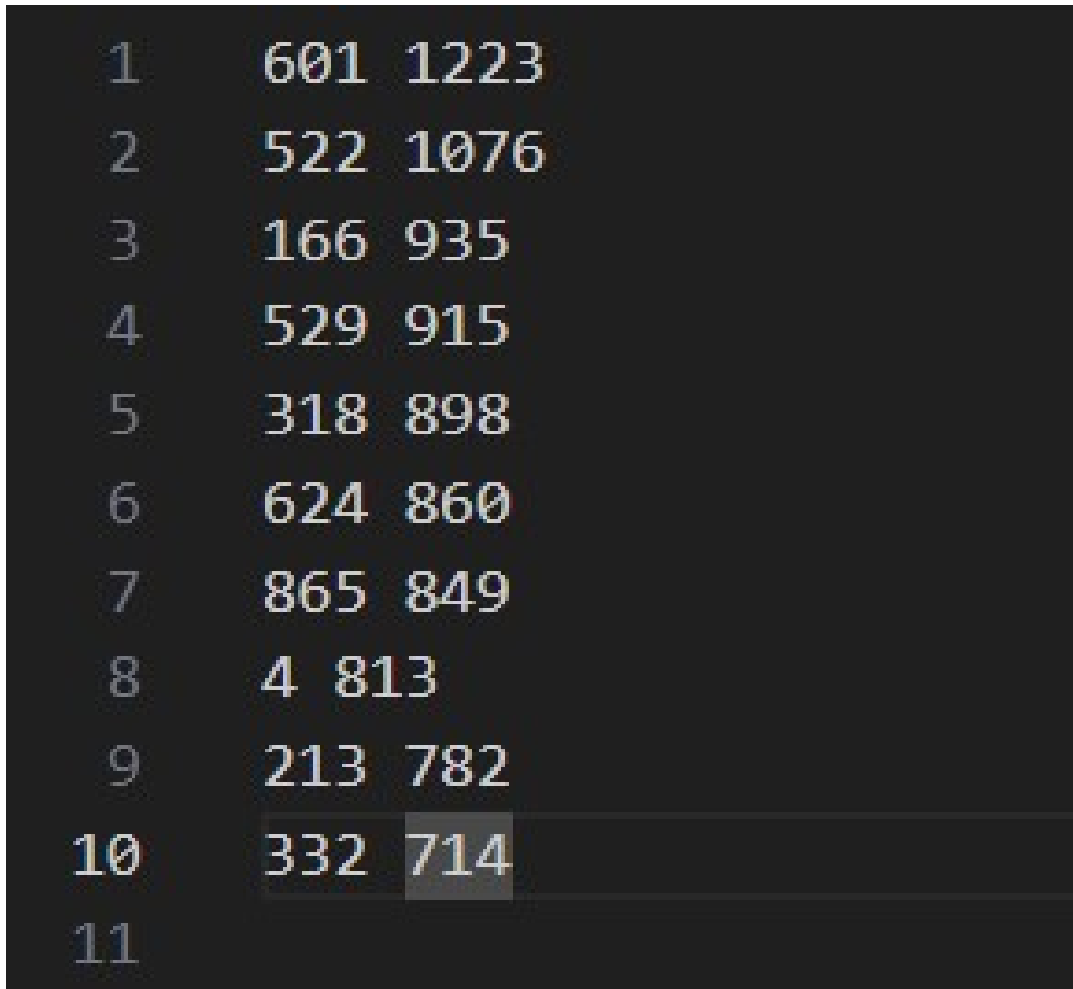
**Algorithm 3** Top 10 Most Frequent Words

---

- 1: Firstly, we create the termId list, the docId list by reading the bbc preprocess files (just reading like a normal java program - the path is encoded in file).
  - 2: **In mapper - output:** <termId, freq>:
    - We read the output.mtx file and group the data with term as key and frequency as value.
  - 3: **In combiner - output:** <'sameKey', termId + sumFreq>:
    - Sum up all the frequencies of a term across all the docs.
    - Put any value in the key as we need the it to be the same among the records.
  - 4: **In reducer - output:** <termId, sumFreq>
    - Split each record in the tuple and put them in a list of hashmaps where the key is termId and the value is sumFreq.
    - Sort the hashmap list decreasingly by the sumFreq.
    - Loop through the list, get the first 10 items from the list and write them to the output.
-



### 1.3.3 Results:



1	601	1223
2	522	1076
3	166	935
4	529	915
5	318	898
6	624	860
7	865	849
8	4	813
9	213	782
10	332	714
11		

Figure 3: The output of the third task

## 1.4 TF-IDF:

### 1.4.1 Data description:

The first mapreduce:

**Input:** the output.mtx file from task 1.2.

**Output:** part-r-\* files and an output.mtx file (inside TF/MTX folder inside the output path) with the structure:

- The first row is Matrix header.

- The second row is the summarize rows (the number of terms, the number of docs, the numbers of data rows in files).
- The entire remaining rows are data rows with format likes: `<termid, docid, tf>`.

### The second mapreduce:

**Input:** the `output.mtx` file from the first map reduce.

**Output:** `part-r-*` files and an `output.mtx` file (inside `TF_IDF/MTX` folder inside the output path) with the structure:

- The first row is Matrix header.
- The second row is the summarize rows (the number of terms, the number of docs, the numbers of data rows in files)
- The entire remaining rows are data rows with format likes: `<termid, docid, tf_idf>`.

### The postprocess mapreduce for task 2.2:

**Input:** the `output.mtx` file from task 1.4.

**Output:** `part-r-*` files and an `output.txt` file (inside `postprocess/TXT` folder inside the output path) with the structure of each row: `<docidtermid1:tf_idf,termid2:tf_idf,...>`

The example of output:

```

1 1000|4386:0.05993663008884753,2804:0.07201529148674757,3871:0.09708232774761996,6234:0.08581818993891022,481:0.036794978494236574,
2 1001|990:0.019869603857211114,3877:0.0406893440247448,948:0.04316676001810469,8009:0.08158025463328501,4223:0.07528486768106468,73
3 1002|468:0.029938634334795187,470:0.03340468340047929,4138:0.04350503509209909,422:0.03340468340047929,856:0.035475168423692055,47
4 1003|1585:0.030887462683417667,125:0.02200467239872842,863:0.03806182972695826,1832:0.09510261185885685,607:0.06417541694975476,13
5 1004|3052:0.10131967372921206,962:0.4988513528346176,2626:0.123935005297693,635:0.08768959237303167,1344:0.03648143055578659,1052:
6 1005|972:0.05212815810431241,23:0.05575976864046819,216:0.047765751681222164,5346:0.12233958942805732,67:0.22395568724903128,3634:
7 1006|1080:0.08038626502449583,4808:0.06366287591558932,4495:0.06862952577671509,980:0.07708254017378759,805:0.016906028794145007,3
8 1007|3062:0.01770758340035135,3096:0.014986845454989817,245:0.024323876863191415,3877:0.020598980412527057,898:0.02059898041252705
9 1008|1192:0.07675283643313487,1012:0.07675283643313487,4225:0.15698434004374445,1242:0.08549831191538455,332:0.018310204811135163,
10 1009|365:0.06553316778331943,4962:0.07829967220104753,3964:0.051106674269799444,2745:0.0426730286835716,4392:0.06974180225177416,8
11 1010|2829:0.06348835824088238,283:0.031995704807643834,1320:0.052579267485115,2996:0.08811560594876235,213:0.0,43:0.074265769345708
12 1011|191:0.05231487088895761,1595:0.07566794834161775,1843:0.1934321572215864,7820:0.13117591277685658,332:0.026157435444478804,14
13 1012|1119:0.08528092937014985,216:0.020344672012372483,1000:0.019254888348887366,302:0.02773826179216658,123:0.038932339068434865,
14 1013|3049:0.057230120094340954,549:0.03427347181318518,3695:0.05535947892261449,2298:0.04072070410297597,6333:0.0730859884894888,2
15 1014|97:0.7988619396143976,2468:0.13551276182591646,198:0.04882721282969377,640:0.05116855762208991,255:0.08648489551356948,4796:0
16 1015|821:0.022668139611747892,1415:0.126580236910732,125:0.019525272691829444,867:0.06800441883524368,1236:0.04642023754935675,340
17 1016|2212:0.06525444781347906,5622:0.07034526392113297,1271:0.020117973905426256,752:0.036805487239580506,957:0.05328349846301644,
18 1017|7571:0.32891870666435685,150:0.04804292275922687,1028:0.07264976791724749,1315:0.04973439567425677,2090:0.044712421993343146,
19 1018|4805:0.05005586451655911,363:0.023165597012563252,1475:0.043915231596594474,406:0.059164444042571436,1843:0.161193464351322,8
20 1019|719:0.026510905075699035,7672:0.02979907814147322,4225:0.08335451683738644,1289:0.023354489642612906,2047:0.02396504602745318
21 1020|805:0.04780325383172036,2608:0.12805420919670027,106:0.07939948596531193,1280:0.18676208283463516,947:0.13126422378518343,200
22 1021|369:0.04409418938175617,53:0.015049483406412462,70:0.02665630341171662,270:0.06569576089858549,103:0.050870850228826134,835:0.
23 1022|8005:0.13594259783076482,948:0.040189742085821614,1855:0.03870454976995947,8006:0.14530501825863407,296:0.018499286349817245,
24 1023|195:0.04600685904947563,2286:0.05090775840658566,216:0.03433163402087843,2265:0.08156805976684882,1493:0.5465700924429501,204

```

Figure 4: The output format for task 2.2

### 1.4.2 MapReduce Job Implementation:

**Number of mapper-reducer:** 2 - the preprocess mapreduce is in a specific file, just to tailor the output from the two mapreduce.

The first mapreduce:

**The flow of code:**

---

#### Algorithm 4 TF-IDF

---

- 1: Firstly, we create the stopwords, the termid list, the docid list by reading the bbc preprocess files (just reading like a normal java program - the path is encoded in file).
  - 2: **In mapper - output:** <docid, termid + freq>:
    - Group termid and its frequency to docid for calculating the tf in reduce.
  - 3: **In reducer - output:** <termid + docid, freq + tf >
    - Calculate the tf of each termid in a doc by divided its frequency to the number of terms in that docs (the number of values pair)
  - 4: **Output function:**
    - This is the function to comprise the part-r-files into the output.mtx file. It will be run after the mapreduce have done its job.
- 

The second mapreduce:

**The flow of code:**

---

**Algorithm 5** TF-IDF

---

- 1: **In mapper - output:** <termid, docid + freq + tf>:
    - Group termid with docid for later calculations.
  - 2: **In reducer - output:** <termid, docid + tf\_idf>
    - Calculate the idf of each termid in a specific doc by get the  $\log(\text{total number of docs} - \text{the size of docid list which is created when the program start} / \text{number of docs where the terms appears} - \text{the number of value pairs})$ .
    - Calculate the tf\_idf of each termid in a specific doc by  $\text{tf} * \text{idf}$
  - 3: **Output function:**
    - This is the function to comprise the part-r-files into the output.mtx file. It will be run after the mapreduce have done its job.
- 

The postprocess mapreduce for task 2.2:

The flow of code:

---

**Algorithm 6** TF-IDF

---

- 1: **In mapper - output:** <docid, termid + tf>:
    - Group termid with tf by docid.
  - 2: **In reducer - output:** <docid, result>
    - Traverse through all key value and add them to the result variable with format: <termid1: tf, termid2: tf,...>.
    - Just write the result to the part-r files.
  - 3: **Output function:**
    - This is the function to comprise the part-r-files into the output.txt file. It will be run after the mapreduce have done its job.
-

### 1.4.3 Results:

```

1  %%MatrixMarket matrix coordinate real general
2  9635 2225 159987
3  1 1 1 0.012345679012345678
4  1 11 1 0.02777777777777776
5  1 1199 1 0.01020408163265306
6  1 1228 2 0.030303030303030304
7  1 1242 2 0.02777777777777776
8  1 1905 1 0.015151515151515152
9  1 1913 1 0.012987012987012988
10 1 1920 1 0.015625
11 1 1948 2 0.02777777777777776
12 1 1962 1 0.009900990099009901
13 1 1973 1 0.017241379310344827
14 1 1986 1 0.017241379310344827
15 1 2030 4 0.03636363636363636
16 1 2033 1 0.0196078431372549
17 1 2091 1 0.00980392156862745
18 1 2134 1 0.012987012987012988
19 1 2188 1 0.006369426751592357
20 1 2201 1 0.015625
21 1 2217 1 0.015151515151515152
22 1 35 1 0.02040816326530612
23 1 443 1 0.020833333333333332
24 1 551 1 0.021739130434782608

```

Figure 5: The output of the first mapreduce

```

1  %%MatrixMarket matrix coordinate real general
2  9635 2225 159987
3  1 1 0.055824550333938766
4  1 11 0.12560523825136222
5  1 1199 0.04614069976580653
6  1 1228 0.13702389627421335
7  1 1242 0.12560523825136222
8  1 1905 0.06851194813710668
9  1 1913 0.05872452697466287
10 1 1920 0.07065294651639126
11 1 1948 0.12560523825136222
12 1 1962 0.04477018393117862
13 1 1973 0.07796187201808691
14 1 1986 0.07796187201808691
15 1 2030 0.16442867552905602
16 1 2033 0.08866252111860863
17 1 2091 0.044331260559304315
18 1 2134 0.05872452697466287
19 1 2188 0.028801201127700897
20 1 2201 0.07065294651639126
21 1 2217 0.06851194813710668
22 1 35 0.09228139953161306
23 1 443 0.09420392868852168
24 1 551 0.09829975167497913

```

Figure 6: The output of the second mapreduce

```

1 1000|4386:0.05993663008884753,2804:0.07201529148674757,3871:0.09708232774761996,6234:0.08581818993891022,481:0.036794978494236574,
2 1001|990:0.019869603857211114,3877:0.0406893440247448,948:0.04316676001810469,8009:0.08158025463328501,4223:0.07528486768106468,73
3 1002|468:0.029938634334795187,470:0.03340468340047929,4138:0.04350503509209909,422:0.03340468340047929,856:0.035475168423692055,47
4 1003|1585:0.030887462683417667,125:0.02200467239872842,863:0.03806182972695826,1832:0.09510261185885685,607:0.06417541694975476,13
5 1004|3052:0.10131967372921206,962:0.4988513528346176,2626:0.123935005297693,635:0.08768959237303167,1344:0.03648143055578659,1052:
6 1005|972:0.05212815810431241,23:0.05575976864046819,216:0.047765751681222164,5346:0.12233958942805732,67:0.22395568724903128,3634:
7 1006|1080:0.08038626502449583,4808:0.06366287591558932,4495:0.06862952577671509,980:0.07708254017378759,805:0.016906028794145007,3
8 1007|3062:0.01770758340035135,3096:0.014986845454989817,245:0.024323876863191415,3877:0.020598980412527057,898:0.02059898041252705
9 1008|1192:0.07675283643313487,1012:0.07675283643313487,4225:0.15698434004374445,1242:0.08549831191538455,332:0.018310204811135163,
10 1009|365:0.06553316778331943,4962:0.07829967220104753,3964:0.051106674269799444,2745:0.0426730286835716,4392:0.06974180225177416,8
11 1010|2829:0.06348835824088238,283:0.031995704807643834,1320:0.052579267485115,2996:0.08811560594876235,213:0.0.43:0.074265769345708
12 1011|191:0.05231487088895761,1595:0.07566794834161775,1843:0.1934321572215864,7820:0.13117591277685658,332:0.026157435444478804,14
13 1012|1119:0.08528092937014985,216:0.020344672012372403,1800:0.019254088348887366,302:0.02773826179216658,123:0.038932339068434865,
14 1013|3049:0.057230120094340954,549:0.03427347181318518,3695:0.05535947892261449,2298:0.04072070410297597,6333:0.0730859884894888,2
15 1014|97:0.7988619396143976,2468:0.13551276182591646,198:0.04882721282969377,640:0.05116855762208991,255:0.08648489551356948,4796:0
16 1015|821:0.022668139611747892,1415:0.126580236910732,125:0.019525272691829444,867:0.06800441883524368,1236:0.04642023754935675,340
17 1016|2212:0.06525444781347906,5622:0.07034526392113297,1271:0.020117973905426256,752:0.036805487239580506,957:0.05328349846301644,
18 1017|7571:0.32891870666435685,150:0.04804292275922687,1028:0.07264976791724749,1315:0.04973439567425677,2090:0.04471242199343146,
19 1018|4805:0.05005586451655911,363:0.023165597012563252,1475:0.043915231596594474,406:0.059164444042571436,1843:0.161193464351322,8
20 1019|1719:0.026510905075699035,7672:0.02979907814147322,4225:0.08335451683738644,1289:0.023354489642612906,2047:0.02396504602745318
21 1020|805:0.04780325383172036,2608:0.12805420919670027,106:0.07939948596531193,1280:0.18676208283463516,947:0.13126422378518343,200
22 1021|369:0.04409418938175617,53:0.015049483406412462,70:0.02665630341171662,270:0.06569576089858549,103:0.050870850228826134,835:0.
23 1022|8005:0.13594259783076482,948:0.040189742085821614,1855:0.03870454976995947,8006:0.14530501825863407,296:0.018499286349817245,
24 1023|195:0.04600685904947563,2286:0.05090775840658566,216:0.03433163402087843,2265:0.08156805976684882,1493:0.5465700924429501,204

```

Figure 7: The output of the postprocess mapreduce for task 2.2

**Challenge:** The challenge of this task is to calculate the tf and idf of each term in the document. We have to calculate the tf of each term in a document by dividing the frequency of the term by the total number of terms in that document. Moreover, We have to find the way to run two job in hadoop.

## 1.5 Highest average TF-IDF:

### 1.5.1 Data description:

**Input:** The output.mtx file from Task 1.4. Exclude the first 2 rows, the rest of the file <termId, docId, tfidf>.

**Output:** A part-r-\* file and a text file (task\_1\_5.txt) that contain 5 terms with the highest average tfidf for the every class, each class is on a separate row.

### 1.5.2 MapReduce Job Implementation:

Number of mapper-reducer: 1 only.

The flow of code:

**Algorithm 7** Highest average TF-IDF

- 1: Before running the map tasks, I prepare a `classIdOfDoc` hashmap where it hashes a `docId` to a `classId`, this helps indicate which class this doc belongs to.
- 2: I also prepare a `termNameOfId` hashmap where it hashes `termId` to `termName`. This way, we'll be able to print out the term names along with their avg tfidf value.
- 3: **In mapper - output:** `<classId + term, tfidf>`:
  - We read the `output.mtx` file and use the `classIdOfDoc` hashmap to convert `docId` to `classId`.
  - Put `classId` and `term` as key so that we can get a tuple of all the `tfidf` value from all the docs in a class for that term when we go to the combiner.
- 4: **In combiner - output:** `<classId, termId + avg_tfidf>`
  - Sum up all the `tfidf` value of a term from the tuple and divide it with the size of the tuple to get the average `tfidf` value.
  - Put `classId` as key so later we can have all the terms along with their average `tfidf` value in a specific class.
- 5: **In reducer - output:**
  - `<className, <termName_1 : avg_tfidf_1, termName_2 : avg_tfidf_2, ...>>`
  - Loop through the tuple of values and create a list of hash maps where the key is `termId` and the value is `avg_tfidf`.
  - Sort the hashmap list decreasingly by the `avg_tfidf`.
  - Loop through the list, get the first 5 items from the list and write them to the output.
  - Use the `termNameOfId` to convert `termId` to `termName`.

**1.5.3 Results:**

```

1 Business  nortel:1.156400109426815, gun:1.1213167145981746, fiat:0.913367850051693, fsa:0.8743920019331186, mexican:0.82270025341
2 Entertainment  carson:1.1661177574051917, aguilara:0.8619131250386199, austria:0.7259612240674095, mar:0.6744149495010123, hendrix
3 Politics    regiment:0.7792707290108023, mock:0.758084516179801, foster:0.7559491048241154, swansea:0.739187875047837, archer:0.71
4 Sport      mutu:0.8644811530145633, mask:0.8541578776014301, michel:0.7036239556345663, hamm:0.6536946855131612, alli:0.64229268288336
5 Tech       printer:1.1681356296629966, p2p:1.128195228709088, poster:0.9411656904277067, rfid:0.778551853607397, cabir:0.7393941956498
6

```

Figure 8: The output of the fifth task

## 2 K-Means Algorithm:

### 2.1 K-Means on 2D Data:

#### 2.1.1 Data description:

**Input:** contains all the 2D Points get from `2DPoints.csv` and converted into txt file. Each line contains coordination of a point (x and y) separated by whitespace.

**Output:** includes two txt files. The first txt file is `task_2_1.clusters.txt` including all the centroids for each cluster (x and y are separated by whitespace). The second txt file is `task_2_1.classes.txt` including all the points with a class assignment for each point (class, x and y are separated by whitespace).

#### 2.1.2 MapReduce Job Implementation:

For this task, we only use 1 Mapper and 3 Reducer for each Job. After we read the txt file, we init k centroids and save them to configuration. When it comes to MapReduce job, the Mapper will have a function `setup()` to read all the centroids and assign each point to the closest centroid. The Reducer then will calculate the average of all points and save new centroids to part-r files. The reason why we choose to have 3 Reducer is because the task assigns  $k = 3$  which is the number of centroids, so each Reducer will calculate 1 centroid.

**Mapper:** There are 2 functions in Mapper class. The first function `setup()` used to read all the centroids from configuration that had been initialized or written after each MapReduce job. The second function `map()` used to read the input `2DPoints.txt` file from HDFS then for each point, it will calculate the closest centroid and assign the centroid id to the point. The output `<key, value>` will be `<centroidId, Point>`.

**Reducer:** After the Shuffle & Sort step, the input `<key, value>` of Reducer will be `<centroidId, [Point1, Point2,...]>` and each Reducer will handle 1 centroid. Then we will loop for the array and calculate the mean coordinate of x and y. This will be our new centroid and the output `<key, value>` of Reducer will be `<centroidId, centroidValue>` where `centroidValue` is the Point created from mean x and mean y.



### 2.1.3 Results:

```
root21127063@DESKTOP-J1VUCON:~/hadoop-3.3.6# hadoop fs -cat /user/huynh/kmeans/output/task_2_1.clusters.txt
1.013864098797468 0.25290389850632916
0.7695291514207927 0.628704592752475
0.421850850150685 0.7361129021552513
```

Figure 9: The output of task\_2\_1.clusters.txt

```
root21127063@DESKTOP-J1VUCON:~/hadoop-3.3.6# hadoop fs -cat /user/huynh/kmeans/output/task_2_1.classes.txt
1 1.093994824 0.190881078
2 0.814768277 0.174076767
1 0.996066461 0.155275699
2 0.804844487 0.220578276
2 0.820221986 0.253063104
2 0.69128398 0.167951533
2 0.886469975 0.252416341
1 1.090874041 0.315280922
1 1.018077359 0.280267589
2 0.803058851 0.165573662
1 1.078103093 0.27661671
1 1.171866625 0.283167643
2 0.837514604 0.197264885
2 0.787866896 0.27248812
1 0.986998961 0.225950154
```

Figure 10: The output of task\_2\_1.classes.txt

**Challenge:** This is the first time we have ever implemented a machine learning algorithm using MapReduce so it is quite hard. At first we did not know how to implement MapReduce for which part of the k-means algorithm and how to run it several times. After we did some researches on configuration of hadoop, we succeeded to find out a way to setup for Mapper. And for Reducer, the problem is when the output directory is already existed, MapReduce will be failed so we had to find a new way to remove the output directory after each MapReduce.

## 2.2 K-Means on Preprocessed Data

### 2.2.1 Data description:

**Input:** The output.mtx file from Task 1.4. Exclude the first 2 rows, the rest of the file <termId, docId, tfidf>.

**Output:** A part-r-\* file and a text file (task\_2\_2.clusters.txt) that contain the centroids for each cluster and a text file (task\_2\_2.classes.txt) that contain all the points with a class assignment for each point (a point is represented by a docId).

### 2.2.2 MapReduce Job Implementation:

**Number of mapper-reducer:** 1 only.

**The flow of code:**

---

**Algorithm 8** K-Means on Preprocessed Data

---

- 1: Before running the map tasks, I prepare a sample of data points to be the initial centroids. The number of centroids is 5. The sample is taken randomly from the data points. When we run the map tasks, we will have the initial centroids.
  - 2: In the mapper, we calculate the distance between each data point and each centroid. We assign the data point to the closest centroid. The output of the mapper is `<centroid, point>`. A point is represented by a docId with the array of tfidf values.
  - 3: In the reducer, we calculate the new centroids by averaging all the data points that are assigned to the same centroid. The output of the reducer is `<centroid, new_centroid>`.
  - 4: We run the map-reduce job until the centroids do not change anymore or reach its max iteration - 10. We will have the final centroids.
- 

**Challenge:** The challenge of this task is to understand the k-means algorithm and how to implement it using MapReduce. We have to figure out how to assign the data points to the closest centroid and how to calculate the new centroids. We also have to figure out how to run the map-reduce job multiple times until the centroids do not change anymore. Importantly, we met a problem that we still not figure out the how to fix. The problem is that our mapper is working fine with 100% successfully but the reducer is not working as expected. We have tried to debug the reducer but we still not figure out the problem. We will try to fix it in the future.

## 2.3 Scalable K-Means++ Initialization

**Idea on the algorithm Scalable K-Means++ Initialization:**

Most of the flow of the code is similar to the K-Means on Preprocessed Data task. The difference

is in the initialization of the centroids. In this task, we use the K-Means++ algorithm to initialize the centroids. The K-Means++ algorithm is an algorithm to choose the initial centroids for the K-Means algorithm. The algorithm is as follows:

---

**Algorithm 9** K-Means++ Initialization

---

- 1: Choose the first centroid randomly from the data points and add it to the set named M.
  - 2: For each data point, calculate the distance between the data point and the nearest centroid in M.
  - 3: Choose the next centroid from the data points with the probability of  $D(x)^2$  where  $D(x)$  is the distance between the data point and the nearest centroid in M.
  - 4: Add the new centroid to the set M.
  - 5: Repeat steps 2-4 until we have k centroids.
  - 6: Apply the K-Means algorithm with the initial centroids from step 1-5.
- 

**Challenge:** Due to the fact that in the previous task, we have not yet figured out how to fix the reducer, we have not yet implemented this task. We will try to fix the reducer in the future and then implement this task. There is also a challenge in the fact that we are under so much pressure from other tasks (coming from different course) that we have not yet had time to implement this task. We will try to implement this task as soon as possible.

## References

[https://stanford.edu/~rezab/classes/cme323/S16/projects\\_reports/bodoia.pdf](https://stanford.edu/~rezab/classes/cme323/S16/projects_reports/bodoia.pdf)

<https://github.com/seraogianluca/k-means-mapreduce>

<https://github.com/SatishUC15/TFIDF-HadoopMapReduce/blob/master/README.md>