

# Lab 02: Document Clustering with Hadoop MapReduce

## Lab Instructors:

**Đỗ Trọng Lễ**  
dtle@selab.hcmus.edu.vn

**Bùi Huỳnh Trung Nam**  
huynhtrungnam2001@gmail.com

## Abstract

This lab work will guide you through applying the **k-means clustering algorithm** on document datasets using Hadoop MapReduce. It consists of two parts: **data preprocessing** and **algorithm implementation**.

*This lab draws inspiration from the University of Washington's CSE547: Machine Learning for Big Data course.*

## Background knowledge

### TF-IDF (Term Frequency-Inverse Document Frequency)

**TF-IDF** is a numerical statistic that reflects the importance of a term within a document relative to a collection of documents, often used in information retrieval and text mining. It includes two components: **Term Frequency (TF)** and **Inverse Document Frequency (IDF)**.

#### Term Frequency (TF):

- Measures how often a term occurs in a document.
- Calculated as the ratio of the number of times a term appears in a document to the total number of terms in that document.
- Emphasizes the significance of a term within a specific document.

#### Inverse Document Frequency (IDF):

- Measures the rarity of a term across all documents in a collection.
- Calculated as the logarithm of the ratio of the total number of documents to the number of documents containing the term.
- Highlights terms that are unique or infrequent in the entire document corpus.

The product of TF and IDF results in a weighted score for each term, providing a measure of its importance in a specific document and across the entire document collection.

### Document Clustering

**Document clustering**, also known as **text clustering** or text categorization, is a natural language processing (NLP) technique that groups similar documents together based on their content. The goal is to discover inherent structures in a collection of documents, facilitating organization, search, and analysis. Clustering enables the identification of relationships and patterns within unstructured textual data.

## Part 1: Data Preprocessing (5 pts)

In this part, we will process textual data and convert each text document into a vector representation suitable for clustering using TF-IDF feature.

### Tasks:

#### 1.1 Text Cleaning and Term Frequency (1 pt):

The dataset we will consider comes from the BBC (<http://mlg.ucd.ie/datasets/bbc.html>).

- **Input:** Path of the directory containing the dataset. Download raw text files from the dataset page.
- **Output:** An **MTX file (Matrix Market format)** with three columns:
  - *termid*: Unique identifier for each term (word) in the vocabulary.
  - *docid*: Unique identifier for each document.
  - *frequency*: Frequency of each term in a specific document.

To understand the intended output file format, please refer to the .mtx file in the preprocessed dataset from the dataset page.

- **Steps:**
  1. Read the text data.
  2. Remove stop words (common words like "the," "a," "an"). (Download the [stop word list](#) here)
  3. Tokenize the text (split into individual words).
  4. Calculate the frequency of each term within each document.
  5. Write the output to ***task\_1\_1.mtx*** with *termid*, *docid*, and *frequency*. Get *termid* from *.terms* file and *docid* from *.docs* file. The id corresponds to the line number in the file.

#### 1.2 Low-Frequency Term Elimination (1 pt):

- **Input:** MTX file generated from Task 1.1.
- **Output:** MTX file with terms having a frequency < 3 removed.
- **Steps:**
  1. Read the MTX file.
  2. Filter terms where the frequency is less than 3 across all documents.
  3. Write the updated MTX file to ***task\_1\_2.mtx*** excluding low-frequency terms.

#### 1.3 Top 10 Most Frequent Words (1 pt):

- **Input:** MTX file from Task 1.2.
- **Output:** List of the top 10 most frequently occurring terms along with their frequencies.
- **Steps:**

1. Read the MTX file.
2. Calculate the total frequency of each term **across all documents**.
3. Sort the terms in descending order based on their total frequency.
4. Output the top 10 most frequent terms and their corresponding frequencies to *task\_1\_3.txt*.

#### 1.4 TF-IDF (1 pt):

**Term Frequency (TF):** Measures how often a term appears in a document. It is calculated as the ratio of the number of occurrences of a term to the total number of terms in the document.  $f(t, d)$  is the number of times term  $t$  appearing in document  $d$ .

$$tf(t, d) = \frac{f(t, d)}{\sum\{f(w, d): (w \in d)\}}$$

**Inverse Document Frequency (IDF):** Measures the importance of a term across a collection of documents. It is calculated as the logarithm of the ratio of the total number of documents to the number of documents containing the term.  $idf(t, D)$  is a measure of how unique or important the term  $t$  is across the entire document collection  $D$ .

$$idf(t, D) = \log \frac{|D|}{|\{d \in D: t \in d\}|}$$

**TF-IDF Score:** Combines both TF and IDF to give a weighted score that highlights terms that are important in a document but not too common across all documents.

$$tfidf(t, d, D) = tf(t, d) \times idf(t, D)$$

- **Input:** MTX file from Task 1.2.
- **Output:** Convert the *term-doc-frequency* matrix into a *term-doc-tfidf* matrix.
- **Steps:**
  1. Read the MTX file.
  2. Calculate the TF-IDF score for each term-document pair.
  3. Write the final MTX file to *task\_1\_4.mtx*.

#### 1.5 Highest average tfidf (1 pt)

For each term  $t$ , take the average tfidf over each class  $C_i = \{\text{documents in class } i\}$

$$avg\_tfidf(t, C_i, D) = \frac{1}{|C_i|} \sum_{d \in C_i} tfidf(t, d, D)$$

For each class  $C_i$ , write the 5 terms with the highest average tfidf for the class to *task\_1\_5.txt* (e.g *Tech: spywar:0.69, aol:0.58, : : : Business: : : :*).

## Part 2: K-Means Algorithm (4 pts)

In this part, you will implement the K-Means algorithm to cluster your preprocessed data.

### Tasks:

#### 2.1 K-Means on 2D Data (1.5 pt):

- **Input:** A sample file containing 2D data points (one point per line, separated by spaces).
- **Output:** Cluster assignments for each data point.
- **Steps:**
  1. Define the desired number of clusters (k).
  2. Implement the K-Means algorithm:
    - Initialize k centroids randomly.
    - Assign each data point to the closest centroid based on Euclidean distance.
    - Recompute centroids as the mean of points assigned to each cluster.
    - Repeat steps 2.2 and 2.3 until convergence (centroids no longer change significantly).
  3. Run the K-Means with K=3 for 20 iterations. Output the clusters centers to *task\_2\_1.clusters* and cluster assignment for each data point to *task\_2\_1.classes* in text format.

#### 2.2 K-Means on Preprocessed Data (1.5 pt):

- **Input:** TF-IDF MTX file from Task 1.4. For convenience, you can convert data to tfidf.txt: Each row is in the form of *docid|termid1:tfidf1,termid2:tfidf2,: : .*
- **Output:** Cluster assignments for each document represented in the MTX file and mean of each clusters.
- **Steps:**
  1. Define the desired number of clusters (k).
  2. Implement the K-Means algorithm (similar to Task 2.1) but calculate distances using a suitable distance metric for TF-IDF vectors (e.g., cosine similarity).
  3. Run k-means with K = 5 for 10 iterations. Output the clusters centers to *task\_2\_2.clusters* and cluster assignment for each data point to *task\_2\_2.classes* in text format. For each iteration, report mean (top 10 words in tfidf) of each clusters to *task\_2\_2.txt* and objective function value to *task\_2\_2.loss* in text format.

#### 2.3 Scalable K-Means++ Initialization (1 pt):

- **Read the article** "[Scalable K-Means++](#)" to understand the K-Means|| algorithm for improved centroid initialization.
- Implement K-Means|| to initialize centroids for your K-Means algorithm.
- Run k-means with K = 5 for 10 iterations. For each iteration, report mean (top 10 words in tfidf) of each clusters to *task\_2\_3.txt* and objective function value to *task\_2\_3.loss* in text format.

## Report (1 pt)

The report for this lab on Document Clustering with Hadoop MapReduce should address the following:

- **Data Description:** Briefly describe the text data used for the exercise.
- **MapReduce Job Implementation:**
  - Describe the overall design of your MapReduce job, including the number of mappers and reducers used.
  - For each task:
    - Explain the logic implemented in the Map function.
    - Explain the logic implemented in the Reduce function.
    - Describe the format of the key-value pairs emitted at each stage.
- **Results:**
  - Include a sample of the output from each MapReduce task.
  - Briefly discuss any challenges faced during the implementation and how you addressed them.
- **Conclusion:**
  - Summarize the key learnings from this lab.

### Additional Considerations:

- Include any relevant code snippets for the Map and Reduce functions (you can use pseudocode if actual code is too lengthy).
- Mention any assumptions made during the implementation.
- Maintain a clear and organized structure in your report, making it easy for the reader to follow the steps involved.

### References.

- Properly acknowledge any reference code utilized in your work within this section; failure to do so may be construed as **academic dishonesty**.