**Assignment 2**

**Code**

#include <iostream>

#include <fstream>

#include <sstream>

#include <unordered\_set>

#include <vector>

#include <string>

#include <algorithm>

using namespace std;

// Function to calculate the Dice Coefficient

double diceCoefficient(const unordered\_set<string>& setA, const unordered\_set<string>& setB) {

    if (setA.empty() || setB.empty()) return 0.0;

    int intersectionCount = 0;

    // Count common words

    for (const string& word : setA) {

        if (setB.find(word) != setB.end()) {

            intersectionCount++;

        }

    }

    return (2.0 \* intersectionCount) / (setA.size() + setB.size());

}

// Function to read a document and return a set of unique words

unordered\_set<string> readDocument(const string& filename) {

    ifstream file(filename);

    unordered\_set<string> words;

    string word;

    if (file.is\_open()) {

        while (file >> word) {

            // Convert to lowercase and remove punctuation

            transform(word.begin(), word.end(), word.begin(), ::tolower);

            word.erase(remove\_if(word.begin(), word.end(), ::ispunct), word.end());

            if (!word.empty()) {

                words.insert(word);

            }

        }

        file.close();

    } else {

        cerr << "Error: Unable to open file " << filename << endl;

    }

    return words;

}

// Function to perform single-pass clustering

void singlePassClustering(const vector<string>& documents, double threshold) {

    vector<unordered\_set<string>> clusters;

    for (const string& doc : documents) {

        unordered\_set<string> words = readDocument(doc);

        bool foundCluster = false;

        for (auto& cluster : clusters) {

            // Calculate similarity

            if (diceCoefficient(words, cluster) >= threshold) {

                // Add words to the existing cluster

                cluster.insert(words.begin(), words.end());

                foundCluster = true;

                break;

            }

        }

        // Create a new cluster if no similar cluster is found

        if (!foundCluster) {

            clusters.push\_back(words);

        }

    }

    // Print the clusters

    cout << "Clusters formed based on the documents:" << endl;

    for (size\_t i = 0; i < clusters.size(); i++) {

        cout << "Cluster " << i + 1 << ": ";

        for (const string& word : clusters[i]) {

            cout << word << " ";

        }

        cout << endl;

    }

}

int main() {

    // List of document filenames

    vector<string> documents = {

        "2a.txt",

        "2b.txt",

        "2c.txt",

        "2d.txt",

        "2e.txt"

    };

    double threshold = 0.5; // Similarity threshold for clustering

    singlePassClustering(documents, threshold);

    return 0;

}

**OUTPUT**

Clusters formed based on the documents:

Cluster 1: field voice nlp between enables chatbots processing language as a intelligence recognition artificial data of development such focuses it on through systems is image that learning humans subset algorithms from machine allow interaction used make to natural based learn computers and predictions the in various applications

Cluster 2: more field encompasses their various including applications learning that a improve techniques finance broad machine from include intelligence artificial data time enables is systems learn to healthcare and performance over

Cluster 3: tasks speech and recognition uses a machine area of deep data specialized neural is to analyze various in factors large found such it particularly effective processing networks datasets as those that learning image

Cluster 4: processes decision making in plays across crucial unstructured a from data science methods role combines it statistical machine algorithms learning structured to industries and extract insights knowledge