Prajakta Deokule

3330

A1

ASSIGNMENT 5

Implement Information Retrieval using TF/ IDF algorithm. Assume suitable data.

```
import java.io.*;
import java.util.*;
public class TF_IDF {

    public static ArrayList<String> getDistinctWords(ArrayList<ArrayList<String>> allwords){

      ArrayList<String> distinct = new ArrayList<String>();
          int i,j;

          for (i= 0; i < allwords.size(); i++)
             for (j = 0; j < allwords.get(i).size(); j++)
                   if (!distinct.contains(allwords.get(i).get(j)))
                                distinct.add(allwords.get(i).get(j));

             return distinct;
      }


      public static void printWords(ArrayList<ArrayList<String>> allwords) {

             int i,j;
          for (i = 0; i < allwords.size(); i++) {
             for (j = 0; j < allwords.get(i).size(); j++) {
                System.out.print(allwords.get(i).get(j) + " ");
              }
             System.out.println();
          }
      }

      public static void printWeightMatrix(ArrayList<ArrayList<Double>> mat) {

             // print the weight matrix
             int i,j;
          for (i= 0; i<mat.size(); i++){
              for (j=0; j<mat.get(i).size(); j++) {
                System.out.print(mat.get(i).get(j)+" ");
               }
               System.out.println();
           }
      }

      public static double calculateIdf(String term, ArrayList<ArrayList<String>> allwords) {

             // ni= no. of docs containing term i
          int ni = 0;

          for (int i = 0; i < allwords.size(); i++) {
             if (allwords.get(i).contains(term)) {
             ni += 1;
              }
          }

          double idf = (Math.log(4 / ni) / Math.log(2));
```

```
        return idf;
    }
    public static double get_cosine_sim(ArrayList<Double> query, ArrayList<Double> doc) {

          double num = 0,ss_doc = 0;
        double ss_q = 0;

        for (int i = 0; i < query.size(); i++) {
              num += query.get(i) * doc.get(i);
              ss_doc += doc.get(i) * doc.get(i);
              ss_q += query.get(i) * query.get(i);
        }

        double den = Math.sqrt(ss_q) * Math.sqrt(ss_doc);

        return num / den;
    }

    public static void main(String[] args) throws FileNotFoundException {

            List<String> stopwords=Arrays.asList(new String[]
{"and","up","the","to","a","up","of","down","after","before"});

            // matrix of all imp words in each doc

            ArrayList<ArrayList<String>> allwords = new ArrayList<ArrayList<String>>();

            // store imp words from each doc in arraylist

            Scanner myReader;
            Scanner sc = new Scanner(System.in);
            int NUM=4;
            String st;

            for (int i = 0; i < NUM; i++) {


                        if(i!=NUM-1)
                              System.out.println("Enter the name of the document:");


                        else
                              System.out.println("Enter the name of the document to
compare");

                        st=sc.nextLine();
                        File file=new File("C:\\Users\\User\\eclipse-
workspace\\InformationRetrieval\\src\\"+st+".txt");


              myReader = new Scanner(file);

              while (myReader.hasNextLine()) {

                   ArrayList<String> inner = new ArrayList<String>();

                   String[] data = myReader.nextLine().split(" ");

                   for (String word : data) {
                      // add only if this word isnt a stopword
                   if (!stopwords.contains(word))
                         inner.add(word);
                  }

                   allwords.add(inner);
```

```
            }//close while loop

    }//close for(int i=0;i<4;i++)


    // create a list of all distinct words

    ArrayList<String> distinct_words = getDistinctWords(allwords);

    // create a matrix of weights
    ArrayList<ArrayList<Double>> wt_mat = new ArrayList<ArrayList<Double>>();


    for (int i = 0; i < 4; i++) {

           ArrayList<Double> inner = new ArrayList<Double>();

           for (int j = 0; j < distinct_words.size(); j++) {

                  String word = distinct_words.get(j);

                  int tf = Collections.frequency(allwords.get(i), word);

             double idf = calculateIdf(distinct_words.get(j), allwords);
             double wi = tf * idf;

             inner.add(wi);
        }
    wt_mat.add(inner);
}

    // Get query string from user

    System.out.println("Enter the query: ");

    Scanner sc1 = new Scanner(System.in);

    String[] query_words = sc1.nextLine().split(" ");

    // add row of query weights in the matrix

    ArrayList<Double> query = new ArrayList<Double>();

    for (int j = 0; j < distinct_words.size(); j++) {

           String word = distinct_words.get(j);

           int tf = Collections.frequency(Arrays.asList(query_words), word);
           double idf = calculateIdf(distinct_words.get(j), allwords);


           double wi = tf * idf;
           query.add(wi);
    }

    wt_mat.add(query);
    System.out.println((distinct_words));
    printWeightMatrix(wt_mat);


    double[] cosine_sim = new double[4];

    HashMap<Double, String> cosine_sim_map = new HashMap<>();
```

```
        for (int i = 0; i < cosine_sim.length; i++) {

              cosine_sim[i] = get_cosine_sim(wt_mat.get(wt_mat.size() - 1), wt_mat.get(i));
            cosine_sim_map.put(cosine_sim[i], String.format("doc%d.txt", (i + 1)));

        }


        // sort the cosine_sim

        Arrays.sort(cosine_sim);
        // print the docs to be fetched on basis of least cosine dist

        System.out.println();
        System.out.println("Documents are fetched in the order:    ");

        for (int i = 3; i >= 0; i--) {

              if ((cosine_sim[i]) > 0) {
             System.out.println(cosine_sim_map.get(cosine_sim[i]));
            }
        }


    }//close main()
}//close class



/*

OUTPUT:
Enter the name of the document:
file1
Enter the name of the document:
file2
Enter the name of the document:
file3
Enter the name of the document to compare
compare
Enter the query:
Jack Jill fell water
[Jack, Jill, went, hill, fetch, pail, water, fell, broke, his, crown, came, tumbling]
1.0 1.0 2.0 2.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 2.0 2.0 2.0 0.0 0.0 0.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0 0.0 0.0 0.0 2.0 2.0 2.0 2.0 0.0 0.0
0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 2.0 2.0
1.0 1.0 0.0 0.0 0.0 0.0 2.0 2.0 0.0 0.0 0.0 0.0 0.0

Documents are fetched in the order:
doc3.txt
doc2.txt
doc1.txt
doc4.txt

*/
```