Name of the student:	Tanmay Prashant Rane	**Roll No.**	8031
Practical Number:	7	**Date of Practical:**	
Relevant CO's	**At the end of the course students will be able to apply appropriate algorithms for extracting knowledge from given dataset.**		
Sign here to indicate that you have read all the relevant material provided before attempting this practical		**Sign:**	

Practical grading using Rubrics

Indicator	**Very Poor**	**Poor**	**Average**	**Good**	**Excellent**
Timeline (2)	Practical not submitted (0)	More than two session late (0.5)	Two sessions late (1)	One session late (1.5)	Early or on time (2)
Code design (3)	N/A	Very poor code design(0)	poor design (1)	design with good coding standards (2)	Accurate Design with better coding standards(3)
Execution (3)	N/A	Very less execution (0)	little execution.(1)	Major execution(2)	Entire code execution (3)
Postlab (2)	Both answers wrong(0)	N/A	One answer correct (1)	N/A	Both answers correct (2)

Total Marks (10)	**Sign of instructor with date**

Practical

Course title: Big Data Analytics
Course term: 2019-2020

Problem Statement: To implement K-means algorithm using map-reduce.

Theory:

The k-means algorithm takes the input data set D and parameter k, and then divides a data set D of n objects into k groups. This partition depends upon the similarity measure so that the resulting intra cluster similarity is high but the inter cluster similarity is low. Cluster similarity is measured regarding the mean value of the objects in a cluster, which can be showed as the cluster's mean. The k-means procedure works as follows.
First, it randomly chooses k of the objects, each of which initially defined as a cluster mean or center.
For each of the remaining objects, an object is moved to the cluster to which it is the most similar, based on the similarity measure which is the distance between the item and the cluster average.
It then calculates the new mean for each cluster. This process repeats until no change in the mean values in the clusters.

Algorithm: k-means
Input: E = {e1, e2,..., en} (set of objects to be clustered)
k (number of clusters)

Output: C = {c1, c2,...,ck} (set of cluster centroids)
L={l(e) | e = 1,2,...,n} (set of cluster labels of E)

Methods:
1. Randomly choose k points from the data set D as the initial cluster means (centroids);
2. Assign each object to the group to which is the most closest, based on the means values of the objects in the cluster;
3. Recalculate the mean value of the objects for each cluster;
4. Repeat the steps 2 and 3 until no change in the means values for the groups

USING MAPREDUCE FOR KMEANS:

The first step of designing MapReduce code Kmeans algorithm is to express and investigate the input and output of the implementation. Input is given as <key,value> pair, where "key" is the cluster mean and "value" is the serializable implementation of a vector in the dataset. The prerequisite to implement Map routine and Reduce routine is to have two files. The first one should involve clusters with their centroids values and the other one should have objects to be clustered. Chosen of centroids and the objects to be clustered are arranged in two spillited files is the initial step to cluster data by K-means algorithm using MapReduce method of Apache Hadoop. It can be done by following the algorithm to implement MapReduceroutines for Kmeans clustering. The initial set of centroid is stored in the input directory of HDFS prior to Map routine call and they form the "key" field in the <key, value> pair. The instructions required to compute the distance between the given data set and cluster centroid fed as a <key, value> pair is coded in the Mapper routine.
The Mapper function calculates the distance between the object value and each of the cluster centroidreferred in the cluster set and jointly keeping track of the cluster to which the given object is closest. Once the computation of distances is complete the object should be assigned to the closest cluster.

Once Mapper is invoked, the given object is assigned to the cluster that it is nearest related to.
After the assignment of all objects to their associated clusters is done the centroid of each cluster is recomputed.

The recalculation is done by the Reduce routine and also it restructures the cluster to avoid generation of clusters with extreme sizes. At the end, once the centroid of the given cluster is revised, the new set of objects and clusters is re-written to the memory and is ready for the next iteration.

Code:

Write map-reduce code to implement k-means algorithm

code for mapper:

Code for Reducer:

Code for Driver Class:

```
//MAPPER
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class kmapper extends Mapper<LongWritable, Text, IntWritable, Text> {
        public float[][] centroids = new float[3][3];

        public void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {

                Configuration conf = context.getConfiguration();
                int k = conf.getInt("k", 2);
        for (int  i=0; i<k; i++ ) {
                String coords = conf.get(String.valueOf(i));
//              System.out.println(coords);
                String[] coords_x_y = coords.split(",");

                centroids[i][0] = Float.parseFloat(coords_x_y[0]);
                centroids[i][1] = Float.parseFloat(coords_x_y[1]);
          }
                String line = value.toString();
                String[] x_y = line.split(",");
                float x = Float.parseFloat(x_y[0]);
                float y = Float.parseFloat(x_y[1]);
                float distance = 0;
                int winnercentroid = -1;
                float min_distance = 999999999.9f;
                for(int i = 0; i<k;i++) {
                        distance = ( x-centroids[i][0])*(x-centroids[i][0]) +
                                        (y - centroids[i][1])*(y-centroids[i][1]);
                        if (distance < min_distance) {
                                min_distance = distance;
                                winnercentroid = i;
                        }
                }

                IntWritable winnerCentroid = new IntWritable(winnercentroid);
                context.write(winnerCentroid, new Text(value));
//              System.out.printf("Map: Centroid = %d distance = %f\n", winnercentroid, min_distance);

        }

}
```

//REDUCER

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class kreducer extends Reducer<IntWritable, Text, IntWritable, Text> {

        public void reduce(IntWritable key, Iterable<Text>values, Context context) throws IOException,
InterruptedException {
                // process values
                System.out.println("Hi from reducer");
                String[] value;
                String coords = "";
                int num = 0;
                float mean_x = 0.0f;
                float mean_y = 0.0f;
                for (Text val : values) {
                        num++;
                        value = val.toString().split(",");
                        float x = Float.parseFloat(value[0]);
                        float y = Float.parseFloat(value[0]);
                        coords += String.valueOf(x)+","+String.valueOf(y)+",";
                        mean_x += x;
                   mean_y += y;

                }
                mean_x = mean_x/num;
                mean_y = mean_y/num;

                String preres = String.format("%f,%f", mean_x,mean_y)+";"+coords;
                Text result = new Text(preres);
                context.write(key, result);
        }

}
```

//DRIVER

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
```

```
public class kdriver {
	public static void main(String[] args) throws Exception {
		int rec = 0;
		Configuration conf = new Configuration();
		while(true) {
		if(rec==0) {
		int k = 3; //Set K here for K clusters
		int count = 0;
		try {
		     File myObj = new File("/media/tanmay/Data/SEM-8/BDA/EXP7/input/kmeaninput");
		     Scanner myReader = new Scanner(myObj);
		     while (myReader.hasNextLine() && count < k) {
		       String data = myReader.nextLine();
		       conf.set(String.valueOf(count),data);
		       System.out.println(conf);
		       count = count + 1;
		     }
		     myReader.close();
		    } catch (FileNotFoundException e) {
		     System.out.println("An error occurred.");
		     e.printStackTrace();
		    }
		conf.setInt("k", k);
		Job job = Job.getInstance(conf, "JobName");
		job.setJarByClass(kdriver.class);
		// TODO: specify a mapper
		job.setMapperClass(kmapper.class);
		// TODO: specify a reducer
		job.setReducerClass(kreducer.class);

		// TODO: specify output types
		job.setOutputKeyClass(IntWritable.class);
		job.setOutputValueClass(Text.class);


		// TODO: specify input and output DIRECTORIES (not files)
		FileInputFormat.setInputPaths(job, new Path("/media/tanmay/Data/SEM-8/BDA/EXP7/input"));
		FileOutputFormat.setOutputPath(job, new
Path("/media/tanmay/Data/SEM-8/BDA/EXP7/outfinal"+String.valueOf(rec)));
		if (!job.waitForCompletion(true))
			System.out.println("True"+String.valueOf(rec));
			rec++;
		}
		else {

			try {
				int count = 0;
				boolean flag = true;
				int k = conf.getInt("k", 3);
			    File myObj = new
File("/media/tanmay/Data/SEM-8/BDA/EXP7/outfinal"+String.valueOf(rec-1)+"/part-r-00000");
```

```
Scanner myReader = new Scanner(myObj);
                                while (myReader.hasNextLine()) {
                                  String data = myReader.nextLine();
                                  String[] inter = data.split(";");
                                  data = inter[0];
                                  String prev = conf.get(String.valueOf(count));
                                  conf.set(String.valueOf(count),data.substring(2));
//                                System.out.println("current"+data.substring(2));
//                                System.out.println("prev"+ prev);
                                  String[] temp = data.substring(2).split(",");
                                  float data_x = Float.parseFloat(temp[0]);
                                  float data_y = Float.parseFloat(temp[1]);
                                  String[] temp1 = prev.split(",");
                                  float prev_x = Float.parseFloat(temp1[0]);
                                  float prev_y = Float.parseFloat(temp1[1]);
                                  String result = String.format("%f %f    %f %f",data_x,prev_x,data_y,prev_y);
                                  System.out.println(result);
                                  if(Math.abs(data_x - prev_x)>1.5 || Math.abs(data_y - prev_y)>1.5) {
                                       flag = false;
                                  }
                                  count++;
                                }
                                if(rec==10) {
                                System.exit(0);
                                }
                                if(flag == true) {
                                        myReader.close();
                                        System.out.println("Done");
                                        break;
                                }
                                else
                                {
                                        Job job = Job.getInstance(conf, "JobName");
                                                job.setJarByClass(kdriver.class);
                                                // TODO: specify a mapper
                                                job.setMapperClass(kmapper.class);
                                                // TODO: specify a reducer
                                                job.setReducerClass(kreducer.class);

                                                // TODO: specify output types
                                                job.setOutputKeyClass(IntWritable.class);
                                                job.setOutputValueClass(Text.class);


                                                // TODO: specify input and output DIRECTORIES (not files)
                                                FileInputFormat.setInputPaths(job, new
Path("/media/tanmay/Data/SEM-8/BDA/EXP7/input"));
                                                FileOutputFormat.setOutputPath(job, new Path("/media/tanmay/Data/
SEM-8/BDA/EXP7/outfinal"+String.valueOf(rec)));
                                                if (!job.waitForCompletion(true))
                                                        System.out.println("True"+String.valueOf(rec));
                                                        rec++;
```

```
}

                                } catch (Exception e) {
                                  System.out.println("An error occurred.");
                                  e.printStackTrace();
                                }

                        }
                }

                }

}
```

PostLab:

Explain DisCo algorithm of clustering

Answer for postlab question

• DisCo is a distributed co-clustering algorithm with MapReduce.

• Given a data matrix, coclustering groups the rows and columns so that the resulting permuted matrix has concentrated nonzero elements.

• For example, co-clustering on a documents-to-words matrix finds document groups as well as word groups.

• For an m×n input matrix, co-clustering outputs the row and column labeling vector r ∈ {1,2,..., k}^m and c ∈ {1,2,...,l}^n , respectively, and k ×l group matrix G where k and l are the number of desired row and column partitions, respectively.

• Searching for an optimal cluster is NP-hard [12], and thus co-clustering algorithms perform a local search. In the local search, each row is iterated to be assigned to the best group that gives the minimum cost while the column group assignments are fixed.

• Then in the same fashion each column is iterated while the row group assignments are fixed. This process continues until the cost function stops decreasing.

• There are two important observations that affect the design of the distributed co-clustering algorithm on MapReduce:

◦ The numbers k and l are typically small, and thus the k ×l matrix G is small. Also, the row and the column labeling vectors r and c can fit in the memory.

◦ For each row (or column), finding the best group assignment requires only r, c, and G. It does not require other rows.

Explain BoW algorithm of clustering

Answer for postlab question

A bag-of-words model, or BoW for short, is a way of extracting features from text for use in modeling, such as with machine learning algorithms.

The approach is very simple and flexible, and can be used in a myriad of ways for extracting features from documents.

A bag-of-words is a representation of text that describes the occurrence of words within a document. It involves two things:

A vocabulary of known words.
A measure of the presence of known words.

It is called a “bag” of words, because any information about the order or structure of words in the document is discarded. The model is only concerned with whether known words occur in the document, not where in the document.

In this approach, we look at the histogram of the words within the text, i.e. considering each word count as a feature.

The algoithm goes like follows:
1) Collect the data as input for algorithm
2) Design the vocabulary tokenization followed by removing punctuation and ignoring the case
3) Creating Document Vectors: The objective is to turn each document of free text into a vector that we can use as input or output for a machine learning model. The simplest scoring method is to mark the presence of words as a boolean value, 0 for absent, 1 for present.
4) Managing Vocabulary : Removing Stop Words and Stemming
5) Scoring Words: Once a vocabulary has been chosen, the occurrence of words in example documents needs to be scored.
Counts: Count the number of times each word appears in a document.
Frequencies: Calculate the frequency that each word appears in a document out of all the words in the document.
6) Word Hashing : Words are hashed deterministically to the same integer index in the target hash space. A binary score or count can then be used to score the word.
7) TF-IDF : A problem with scoring word frequency is that highly frequent words start to dominate in the document (e.g. larger score), but may not contain as much “informational content” to the model as rarer but perhaps domain specific words.

One approach is to rescale the frequency of words by how often they appear in all documents, so that the scores for frequent words like “the” that are also frequent across all documents are penalized.

This approach to scoring is called Term Frequency – Inverse Document Frequency, or TF-IDF for short, where:

Term Frequency: is a scoring of the frequency of the word in the current document.
Inverse Document Frequency: is a scoring of how rare the word is across documents.

The scores are a weighting where not all words are equally as important or interesting.