```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.StringReader;
import java.util.ArrayList;
import java.util.logging.Level;
import java.util.logging.Logger;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.FileUtil;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
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.util.GenericOptionsParser;

public class KMeans
{
    static int nClusters;
    static CSVHelper objCsvHelper = new CSVHelper();
    public static class Map extends Mapper<LongWritable, Text, IntWritable, Text> {
      private double [][] nCentroids; // centroids: the center of clusters
      private int nDimensions;
      private int nClstrs;

      @Override
      public void setup(Mapper.Context context) throws IOException{
            Configuration conf = context.getConfiguration();
            if (conf.get ("centroids") == null)
                  throw new RuntimeException ("centroid file not found!!!");

            String filename = conf.get("centroids");
            ArrayList<String> cVal;
            Path p = new Path(filename);

            FileSystem fs = FileSystem.get(conf);
            BufferedReader br;
            try{
                  br = new BufferedReader(new BufferedReader(new
InputStreamReader(fs.open(p))));
                  nClstrs =1;
                  //read the line and get no of clusters
                  cVal = objCsvHelper.parseLine(br);
                  nDimensions = cVal.size()-1;
                  while(br.readLine()!=null)
                        nClstrs++;
                  nClusters=nClstrs;
                  br.close();

                  nCentroids = new double[nClstrs][];
```

```
                for (int i=0; i<nClstrs; i++)
                      nCentroids[i] = new double[nDimensions];

                //using csvHelper read the centroid file
                br = new BufferedReader(new BufferedReader(new
InputStreamReader(fs.open(p))));
                int rowVal=0;
                while ((cVal = objCsvHelper.parseLine(br))!=null){
                      double [] dv = new double[cVal.size()-1];
                      for (int i=0; i< cVal.size()-1; i++){
                            dv[i] = Double.parseDouble(cVal.get(i+1));
                      }
                      nCentroids[rowVal] = dv;
                      rowVal ++;
                }

                br.close();
          }catch(Exception e){
                System.err.println(e);
          }

          System.out.println("configuration completed..");

     }



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

          System.out.println("-----Map Start------");
          String line = value.toString();
          StringReader reader = new StringReader(line);
          ArrayList<String> dataArray = null;

          try {
                dataArray = objCsvHelper.parseLine(reader);
          } catch (Exception e) {
                System.out.println("caught exception");
          }
          int nDim=dataArray.size();

          double [] dataPts = new double[nDim-1];

          for(int i=0;i<nDim-1;i++){

                dataPts[i] = Double.parseDouble(dataArray.get(i+1));

          }

          //finding cluster label corresponding to data points
          int clustID;
          clustID = closest(dataPts);
          String datarecord="";
          datarecord = Double.toString(dataPts[0]);
          for(int j=1;j<nDimensions;j++){
                double x = dataPts[j];
                datarecord=datarecord+","+Double.toString(x);
          }
```

```
            //key value passed to reducer
            context.write(new IntWritable(clustID),new Text(datarecord));

      }

      // compute Euclidean distance between two vectors v1 and v2
      private double dist(double [] v1, double [] v2){
            double sum=0;
            for (int i=0; i<nDimensions; i++){
                  double d = v1[i]-v2[i];
                  sum += d*d;
            }
            return Math.sqrt(sum);
      }

      //find the closest label to data point and assign corresponding cluster ID
      private int closest(double [] v){
            double mindist = dist(v, nCentroids[0]);
            int label =0;
            for (int i=1; i<nClusters; i++){
                  double t = dist(v, nCentroids[i]);
                  if (mindist>t){
                        mindist = t;
                        label = i;
                  }
            }
            return label;
      }

    }
    public static class Reduce extends Reducer<IntWritable, Text, IntWritable,
Text> {

      public void reduce(IntWritable key, Iterable<Text> values,Context context)
                  throws IOException, InterruptedException {

            System.out.println("------------Reduce Start---------------");
            //processing text obtainde from map
            ArrayList<String> tempValues = new ArrayList<String>();
            for (Text value: values){
                  tempValues.add(value.toString());
            }

            StringReader reader = new StringReader(tempValues.get(0));
            ArrayList<String> objArrList = null;
            ArrayList<String> dArray = null;

            //parsing the first record to find the dimension
            try {
                  objArrList = objCsvHelper.parseLine(reader);

            } catch (Exception ex) {
                  System.out.println("caught an exception");
            }
            int nDim=objArrList.size();
            double [] total=new double[nDim];
            double [] average =new double[nDim];
            for(int j =0; j<tempValues.size(); j++){
```

```
                String val = tempValues.get(j);

                StringReader sr = new StringReader(val);

                try {
                      dArray = objCsvHelper.parseLine(sr);
                } catch (Exception ex) {
                      System.out.println("caught an exception");
                }

                double [] data_point = new double[nDim];

                // Finding total of each dimension
                for(int i=0;i<nDim;i++){
                      data_point[i] = Double.parseDouble(dArray.get(i));
                      total[i]=total[i]+data_point[i];
                }

          }
          //Finding average for each dimension
          for (int i=0;i<nDim;i++){
                average[i]=total[i]/tempValues.size();
          }
          String centroidrecord=Double.toString(average[0]);
          for(int i=1;i<nDim;i++){
                centroidrecord=centroidrecord+","+Double.toString(average[i]);
          }

          context.write(key,new Text(centroidrecord));
    }

  }

  private static double dist(double [] v1, double [] v2){
    double sum=0;
    for (int i=0; i<v1.length; i++){
        double d = v1[i]-v2[i];
        sum += d*d;
    }
    return Math.sqrt(sum);
  }

  private static double [][] returnCentroids(String fname) throws IOException{
          double [][] nCentroids=null;
          Configuration conf =new Configuration();
          ArrayList<String> cVal;
          BufferedReader br;
          Path path = new Path(fname);
          System.out.println(path);
          FileSystem fs = FileSystem.get(conf);
          try{
             br = new BufferedReader(new BufferedReader(new
InputStreamReader(fs.open(path))));
              int nClstrs =1;
              cVal = objCsvHelper.parseLine(br);
              int _ndims = cVal.size()-1;
              while(br.readLine()!=null)
                  nClstrs++;
              br.close();
```

```
                nCentroids = new double[nClstrs][];
                for (int i=0; i<nClstrs; i++)
                    nCentroids[i] = new double[_ndims];
                br = new BufferedReader(new InputStreamReader(fs.open(path)));
                int rowVal=0;
                while ((cVal = objCsvHelper.parseLine(br))!=null){
                    double [] dv = new double[cVal.size()-1];
                        for (int i=0; i< cVal.size()-1; i++){
                            dv[i] = Double.parseDouble(cVal.get(i+1));
                        }
                        nCentroids[rowVal] = dv;
                        rowVal ++;
                }
                br.close();
              }catch(Exception e){
              System.err.println(e);
            }

      return nCentroids;

    }


   // check convergence condition
   // max{dist(c1[i], c2[i]), i=1..numClusters < threshold
   private static boolean converge(double [][] c1, double [][] c2, double
threshold){
      // c1 and c2 are two sets of centroids
      double maxv = 0;
      for (int i=0; i< nClusters; i++){
          double d= dist(c1[i], c2[i]);
          if (maxv<d)
            maxv = d;
      }
       System.out.println("Euclidean distance between the older and newer
centroids");
       System.out.println(maxv);

      if (maxv <threshold)
          return true;
      else
          return false;

   }


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

        System.out.println("-----------KMeans Start----------------");
        Configuration conf =new Configuration();
          String pCentroid = "/KMeans/input/centroids";
        conf.set("centroids",pCentroid);
        String[] otherArgs = new GenericOptionsParser(conf,
args).getRemainingArgs();
          if (otherArgs.length != 2) {
            System.err.println("Usage: KMeans <in> <out>");
            System.exit(2);
          }
```

```
        FileSystem fs =FileSystem.get(conf);
      int iteration=1;
        //iterate 10 times
        for (int i=0;i<10;i++){
              iteration++;
              System.out.println("Iteration : "+iteration);
              Job job =new Job(conf,"KMeans");
              job.setJarByClass(KMeans.class);
              job.setOutputKeyClass(IntWritable.class);
              job.setOutputValueClass(Text.class);
              job.setMapperClass(Map.class);
              job.setReducerClass(Reduce.class);
              job.setNumReduceTasks(1);

              Path inPath =new Path(otherArgs[0]);
              FileInputFormat.addInputPath(job,inPath);
              Path outPath=new Path(otherArgs[1]);
              FileOutputFormat.setOutputPath(job,outPath);
              String outputPath = outPath.toString();
              String outputFile = outputPath+"/part-r-00000";

          job.waitForCompletion(true);
          String cf=conf.get("centroids");
          double [][]oldCenters;
          double [][]newCenters;
          oldCenters = returnCentroids(cf);
          //remove the old centroid file
          FileUtil.fullyDelete(fs,new Path(cf));
          Path centroidPath= new Path(pCentroid);
              fs.rename(new Path(outputFile), centroidPath);
              FileUtil.fullyDelete(fs, outPath);

          newCenters = returnCentroids(cf);

              System.out.println("checking convergence...");
          if (converge(oldCenters,newCenters,0.001)){
                   System.out.println("convergence condition obtained!");

              break;}
              else{
                  System.out.println("proceed for next iteration");

          }



      }

}
}
```