K-Means Clustering Job

```
package com.clustering.mapreduce;
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileStatus;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.SequenceFile;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.SequenceFileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.SequenceFileOutputFormat;
import com.clustering.model.ClusterCenter;
import com.clustering.model.Vector;
public class KMeansClusteringJob {
private static final Log LOG =LogFactory.getLog(KMeansClusteringJob.class);
public static void main(String[] args) throws IOException,InterruptedException,
ClassNotFoundException {
int iteration = 1;
Configuration conf = new Configuration();
conf.set("num.iteration", iteration + "");
Path in = new Path("files/clustering/import/data");
Path center = new
Path("files/clustering/import/center/cen.seq");
conf.set("centroid.path", center.toString());
Path out = new Path("files/clustering/depth_1");
Job job = new Job(conf);
job.setJobName("KMeans Clustering");
job.setMapperClass(KMeansMapper.class);
job.setReducerClass(KMeansReducer.class);
job.setJarByClass(KMeansMapper.class);
SequenceFileInputFormat.addInputPath(job, in);
FileSystem fs = FileSystem.get(conf);
if (fs.exists(out))
fs.delete(out, true);
if (fs.exists(center))
fs.delete(out, true);
if (fs.exists(in))
fs.delete(out, true);
final SequenceFile.Writer centerWriter =SequenceFile.createWriter(fs,
```

```
conf, center, ClusterCenter.class,
IntWritable.class);
final IntWritable value = new IntWritable(0);
centerWriter.append(new ClusterCenter(new Vector(2,5.0)),value);
centerWriter.append(new ClusterCenter(new Vector(500,1.0)),value);
centerWriter.close();
final SequenceFile.Writer dataWriter = SequenceFile.createWriter(fs,conf, in,
ClusterCenter.class,Vector.class);
String csvFile = "/home/hadoop1/Desktop/Data/AllGujarat.csv";
BufferedReader br = null;
String line = "";
String cvsSplitBy = ",";

try {

   br = new BufferedReader(new FileReader(csvFile));
   while ((line = br.readLine()) != null) {

      // use comma as separator
      String[] DataLine = line.split(cvsSplitBy);
      double X=Double.parseDouble(DataLine[6]);
      String gramPanchayatName = DataLine[5];
//      double Y=Double.parseDouble(DataLine[7]);
      dataWriter.append(new ClusterCenter(new Vector(0,0)), new Vector(X, 0));

   }

}catch (FileNotFoundException e) {
   e.printStackTrace();
} catch (IOException e) {
   e.printStackTrace();
}finally {
   if (br != null) {
      try {
         br.close();
      } catch (IOException e) {
         e.printStackTrace();
      }
   }
dataWriter.close();
SequenceFileOutputFormat.setOutputPath(job, out);
job.setInputFormatClass(SequenceFileInputFormat.class);
job.setOutputFormatClass(SequenceFileOutputFormat.class);
job.setOutputKeyClass(ClusterCenter.class);
job.setOutputValueClass(Vector.class);
job.waitForCompletion(true);
long counter =
job.getCounters().findCounter(KMeansReducer.Counter.CONVERGED).getValue();
iteration++;
          while (counter > 0) {
                    conf = new Configuration();
```

```
conf.set("centroid.path",
center.toString());
conf.set("num.iteration", iteration
+ ""); job = new Job(conf);
job.setJobName("KMeans Clustering " +
iteration);
job.setMapperClass(KMeansMapper.class)
;
job.setReducerClass(KMeansReducer.clas
s);
job.setJarByClass(KMeansMapper.class);
in = new Path("files/clustering/depth_" +(iteration -
1) + "/"); out = new Path("files/clustering/depth_" +
iteration);
SequenceFileInputFormat.addInputPath(job, in);
if
(fs.exists(
out))
fs.delete(o
ut, true);
SequenceFileOutputFormat.setOutputPath(job, out);
job.setInputFormatClass(SequenceFileInputFormat.c
lass);
job.setOutputFormatClass(SequenceFileOutputForm
at.class);
job.setOutputKeyClass(ClusterCenter.class);
job.setOutputValueClass(Vector.class);
job.waitForCompletion(true);
iteration++;
counter = job.getCounters()
.findCounter(KMeansReducer.Counter.CONVERGED).getValue();
}
Path result = new
Path("files/clustering/depth_" + (iteration
- 1) + "/");
FileStatus[] stati =
fs.listStatus(result); for
(FileStatus status : stati) {
if (!status.isDir() &&
!status.getPath().toString().contains
("/_")) { Path path =
status.getPath();
LOG.info("FOUND " +
path.toString());
SequenceFile.Reader reader = new
SequenceFile.Reader(fs, path,
```

```
                    conf);
                    ClusterCenter key = new
                    ClusterCenter(); Vector v = new
                    Vector();

                    ArrayList<Integer> inputs = new

                    ArrayList<Integer>(); while (reader.next(key, v)) {
                            LOG.info(key + " / " + v);
                            inputs.add((int)(v.getVector()[0]));
                    }
                    ArrayList<Double> finalClusters = Algorithm.main(inputs, 2);
                    LOG.info("fC -> ");
                    LOG.info(final
                    Clusters);
                    reader.close();
                    }
                    }
                    }
                    }
}
```