K-Means Mapper

```
package com.clustering.mapreduce; import java.io.IOException;
import java.util.LinkedList; import java.util.List;
import org.apache.hadoop.conf.Configuration; 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.Mapper;


import com.clustering.model.ClusterCenter; import com.clustering.model.DistanceMeasurer; import
com.clustering.model.Vector;
public class KMeansMapper extends Mapper<ClusterCenter, Vector, ClusterCenter, Vector>{
List<ClusterCenter> centers = new LinkedList<ClusterCenter>();
@Override
protected void setup(Context context) throws IOException, InterruptedException {
super.setup(context);
Configuration conf = context.getConfiguration(); Path centroids = new
Path(conf.get("centroid.path")); FileSystem fs = FileSystem.get(conf);
SequenceFile.Reader reader = new SequenceFile.Reader(fs,centroids,conf); ClusterCenter key = new
ClusterCenter();
IntWritable value = new IntWritable(); while (reader.next(key,value))
{
centers.add(newClusterCenter(key));
}
reader.close();
}
@Override
protected void map(ClusterCenter key, Vector value, Context context) throws IOException,
InterruptedException {
ClusterCenter nearest = null;
double nearestDistance = Double.MAX_VALUE; for (ClusterCenter c : centers) {
double dist = DistanceMeasurer.measureDistance(c, value);
if (nearest == null) { nearest = c; nearestDistance = dist;
} else {
if (nearestDistance > dist) { nearest = c; nearestDistance = dist;
}          17
}
}
context.write(nearest, value);
}
}
```

K-Means Reducer

```
package com.clustering.mapreduce; import java.io.IOException;
import java.util.LinkedList; import java.util.List;
import org.apache.hadoop.conf.Configuration; 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.Reducer; import com.clustering.model.ClusterCenter; import
com.clustering.model.Vector;
```

```
public class KMeansReducer extends Reducer<ClusterCenter, Vector, ClusterCenter, Vector>{ public
static enum Counter{
CONVERGED
}
List<ClusterCenter> centers = new LinkedList<ClusterCenter>(); protected void
reduce(ClusterCenter key, Iterable<Vector> values, Context context) throws IOException,
InterruptedException{
Vector newCenter = new Vector();
List<Vector> vectorList = new LinkedList<Vector>(); int vectorSize =
key.getCenter().getVector().length; newCenter.setVector(new double[vectorSize]); for(Vector value
:values){
vectorList.add(new Vector(value));
for(int i=0;i<value.getVector().length;i++){ newCenter.getVector()[i]+=value.getVector()[i];
}
}
for(int i=0;i<newCenter.getVector().length;i++){ newCenter.getVector()[i] =
newCenter.getVector()[i]/vectorList.size();
}
ClusterCenter center = new ClusterCenter(newCenter); centers.add(center);
for(Vector vector:vectorList){ context.write(center, vector);
}
if(center.converged(key)) context.getCounter(Counter.CONVERGED).increment(1);
}          18
protected void cleanup(Context context) throws IOException,InterruptedException{
super.cleanup(context);
Configuration conf = context.getConfiguration(); Path outPath = new Path(conf.get("centroid.path"));
FileSystem fs = FileSystem.get(conf); fs.delete(outPath,true);
final SequenceFile.Writer out = SequenceFile.createWriter(fs, context.getConfiguration(),
outPath, ClusterCenter.class, IntWritable.class); final IntWritable value = new IntWritable(0);
for(ClusterCenter center:centers){ out.append(center, value);
}
out.close();
}
}
```