



Practical - 7

2CS702 – Big Data Analytics

Harshit Gajipara

19BCE059

Aim: To implement KMeans Clustering using MapReduce by handling larger datasets in main memory.

Code:

Results:

KMeans.java

```
package kmeans;

import java.io.BufferedReader;import
java.io.BufferedWriter;

import java.io.FileNotFoundException;import
java.io.IOException;

import java.io.InputStreamReader; import
java.io.OutputStreamWriter;import
java.util.ArrayList;

import java.util.Collections;import java.util.List;

import org.apache.hadoop.conf.Configuration; import
org.apache.hadoop.fs.FSDataInputStream; import
org.apache.hadoop.fs.FSDataOutputStream;import
org.apache.hadoop.fs.FileStatus;

import org.apache.hadoop.fs.FileSystem;import
org.apache.hadoop.fs.Path;
```

```

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.util.GenericOptionsParser;


import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;


import kmeans.Point;


public class KMeans {

    private static boolean stoppingCriterion(Point[] oldCentroids, Point[]
newCentroids, int distance, float threshold) {

        boolean check = true;

        for(int i = 0; i < oldCentroids.length; i ) {

check = oldCentroids[i].distance(newCentroids[i], distance)                threshold;

            if(!check) {

                return false;

            }

        }

        return true;

    }

}


    private static Point[] centroidsInit(Configuration conf, String pathString, int k,
int dataSetSize)
        throws IOException {
        Point[] points = new Point[k];

```

Create a sorted list of positions without duplicates
Positions are the line index of the random selected centroids

```
List<Integer> positions = new ArrayList<Integer>();
Random random = new Random();
int pos;
while(positions.size() < k) {
    pos = random.nextInt(dataSetSize);
    if(!positions.contains(pos)) {
        positions.add(pos);
    }
}
Collections.sort(positions);
```

File reading utils

```
Path path = new Path(pathString);
FileSystem hdfs = FileSystem.get(conf);
FSDataInputStream in = hdfs.open(path);
BufferedReader br = new BufferedReader(new InputStreamReader(in));
```

Get centroids from the file

```
int row = 0;
int i = 0;
int position;
while(i < positions.size()) {
    position = positions.get(i);
    String point = br.readLine();
    if(row == position) {
        points[i] = new Point(point.split(","));
        i++;
    }
    row++;
}
br.close();

return points;
}

private static Point[] readCentroids(Configuration conf, int k, String pathString)
throws IOException, FileNotFoundException {
    Point[] points = new Point[k];
    FileSystem hdfs = FileSystem.get(conf);
    FileStatus[] status = hdfs.listStatus(new Path(pathString));

    for (int i = 0; i < status.length; i++) {
        Read the centroids from the hdfs
        if(!status[i].getPath().toString().endsWith("_SUCCESS")) {
```

```
            BufferedReader br = new BufferedReader(new
InputStreamReader(hdfs.open(status[i].getPath())));
```

```

        String[] keyValueSplit = br.readLine().split("\t"); Split line in K,V
        int centroidId = Integer.parseInt(keyValueSplit[0]);
        String[] point = keyValueSplit[1].split(",");
        points[centroidId] = new Point(point);
        br.close();
    }
}
Delete temp directory
hdfs.delete(new Path(pathString), true);

return points;
}

private static void finalize(Configuration conf, Point[] centroids, String output)
throws IOException {
    FileSystem hdfs = FileSystem.get(conf);
    FSDataOutputStream dos = hdfs.create(new Path(output + "/centroids.txt"),
true);
    BufferedWriter br = new BufferedWriter(new OutputStreamWriter(dos));

    Write the result in a unique file
    for(int i = 0; i < centroids.length; i ) {
        br.write(centroids[i].toString());
        br.newLine();
    }

    br.close();
    hdfs.close();
}

public static void main(String[] args) throws Exception {
    long start = 0;
    long end = 0;
    long startIC = 0;
    long endIC = 0;

    start = System.currentTimeMillis();
    Configuration conf = new Configuration();
    conf.addResource(new Path("config.xml")); Configuration file for the
parameters

    String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();

    if (otherArgs.length > 2) {
        System.err.println("Usage: <input> <output>");
        System.exit(1);
    }
}

```

Parameters setting

```
final String INPUT = otherArgs[0];
final String OUTPUT = otherArgs[1] + "/temp";
Hard-coded values.
```

```
final int DATASET_SIZE = 3;
final int DISTANCE = 5;
final int K = 2;
final float THRESHOLD = 0.0001f;
final int MAX_ITERATIONS = 10;
```

```
Point[] oldCentroids = new Point[K];
Point[] newCentroids = new Point[K];
```

Initial centroids

```
startIC = System.currentTimeMillis();
newCentroids = centroidsInit(conf, INPUT, K, DATASET_SIZE);
endIC = System.currentTimeMillis();
for(int i = 0; i < K; i ) {
    conf.set("centroid." + i, newCentroids[i].toString());
}
```

MapReduce workflow

```
boolean stop = false;
boolean succeeded = true;
int i = 0;
while(!stop) {
    i ;
}
```

Job configuration

```
Job iteration = Job.getInstance(conf, "iter_" + i);
iteration.setJarByClass(KMeans.class);
iteration.setMapperClass(KMeansMapper.class);
iteration.setCombinerClass(KMeansCombiner.class);
iteration.setReducerClass(KMeansReducer.class);
iteration.setNumReduceTasks(K); // one task each centroid
iteration.setOutputKeyClass(IntWritable.class);
iteration.setOutputValueClass(Point.class);
FileInputFormat.addInputPath(iteration, new Path(INPUT));
FileOutputFormat.setOutputPath(iteration, new Path(OUTPUT));
iteration.setInputFormatClass(TextInputFormat.class);
iteration.setOutputFormatClass(TextOutputFormat.class);
```

```
succeeded = iteration.waitForCompletion(true);
```

If the job fails the application will be closed.

```

    if(!succeeded) {
        System.err.println("Iteration" + i + "failed.");
        System.exit(1);
    }

    Save old centroids and read new centroids
    for(int id = 0; id < K; id ) {
        oldCentroids[id] = Point.copy(newCentroids[id]);
    }
    newCentroids = readCentroids(conf, K, OUTPUT);

    Check if centroids are changed
        stop = stoppingCriterion(oldCentroids, newCentroids, DISTANCE, THRESHOLD);

        if(stop == i && i < (MAX_ITERATIONS - 1)) {
            finalize(conf, newCentroids, otherArgs[1]);
        } else {
            Set the new centroids in the configuration
            for(int d = 0; d < K; d ) {
                conf.unset("centroid." + d);
                conf.set("centroid." + d, newCentroids[d].toString());
            }
        }
    }

    end = System.currentTimeMillis();

    end -= start;
    endIC -= startIC;

    System.out.println("execution time: " + end + " ms");
    System.out.println("init centroid execution: " + endIC + " ms");
    System.out.println("n_iter: " + i);

    System.exit(0);
}
}

```

KMeansReducer.java

```

package kmeans;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;

```

```

import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import kmeans.Point;

public class KMeansMapper extends Mapper<LongWritable, Text, IntWritable, Point> {

    private Point[] centroids;
    private int p;
    private final Point point = new Point();
    private final IntWritable centroid = new IntWritable();

    public void setup(Context context) {
        int k = Integer.parseInt(context.getConfiguration().get("k"));
        this.p = Integer.parseInt(context.getConfiguration().get("distance"));

        this.centroids = new Point[k];
        for(int i = 0; i < k; i ) {
            String[] centroid = context.getConfiguration().getStrings("centroid." + i);
            this.centroids[i] = new Point(centroid);
        }
    }

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

        Construct the point
        String[] pointString = value.toString().split(",");
        point.set(pointString);

        Initialize variables
        float minDist = Float.POSITIVE_INFINITY;
        float distance = 0.0f;
        int nearest = -1;

        Find the closest centroid
        for (int i = 0; i < centroids.length; i ) {
            distance = point.distance(centroids[i], p);
            if(distance < minDist) {
                nearest = i;
                minDist = distance;
            }
        }

        centroid.set(nearest);
    }
}

```



```
        context.write(centroid, point);
    }
}
```

KMeansCombiner.java

```
package kmeans;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.mapreduce.Reducer;

import kmeans.Point;

public class KMeansCombiner extends Reducer<IntWritable, Point, IntWritable, Point> {

    public void reduce(IntWritable centroid, Iterable<Point> points, Context context)
        throws IOException, InterruptedException {

        Sum the points
        Point sum = Point.copy(points.iterator().next());
        while (points.iterator().hasNext()) {
            sum.sum(points.iterator().next());
        }

        context.write(centroid, sum);
    }
}
```

KMeansReducer.java

```
package kmeans;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

import kmeans.Point;

public class KMeansReducer extends Reducer<IntWritable, Point, Text, Text> {

    private final Text centroidId = new Text();
    private final Text centroidValue = new Text();
}
```

```

    public void reduce(IntWritable centroid, Iterable<Point> partialSums, Context
context)
        throws IOException, InterruptedException {

        Sum the partial sums
        Point sum = Point.copy(partialSums.iterator().next());
        while (partialSums.iterator().hasNext()) {
            sum.sum(partialSums.iterator().next());
        }
        Calculate the new centroid
        sum.average();

        centroidId.set(centroid.toString());
        centroidValue.set(sum.toString());
        context.write(centroidId, centroidValue);
    }
}

```

Point.java

```

package kmeans;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;

import org.apache.hadoop.io.Writable;

public class Point implements Writable {

    private float[] components = null;
    private int dim;
    private int numPoints; For partial sums

    public Point() {
        this.dim = 0;
    }

```

```

    public Point(final float[] c) {
        this.set(c);
    }

    public Point(final String[] s) {
        this.set(s);
    }

```

```

public static Point copy(final Point p) {
    Point ret = new Point(p.components);
    ret.numPoints = p.numPoints;
    return ret;
}

public void set(final float[] c) {
    this.components = c;
    this.dim = c.length;
    this.numPoints = 1;
}

public void set(final String[] s) {
    this.components = new float[s.length];
    this.dim = s.length;
    this.numPoints = 1;
    for (int i = 0; i < s.length; i ) {
        this.components[i] = Float.parseFloat(s[i]);
    }
}

@Override
public void readFields(final DataInput in) throws IOException {
    this.dim = in.readInt();
    this.numPoints = in.readInt();
    this.components = new float[this.dim];

    for(int i = 0; i < this.dim; i ) {
        this.components[i] = in.readFloat();
    }
}

@Override
public void write(final DataOutput out) throws IOException {
    out.writeInt(this.dim);
    out.writeInt(this.numPoints);

    for(int i = 0; i < this.dim; i ) {
        out.writeFloat(this.components[i]);
    }
}

@Override
public String toString() {
    StringBuilder point = new StringBuilder();

```

```

        for (int i = 0; i < this.dim; i ) {
            point.append(Float.toString(this.components[i]));
            if(i == dim - 1) {
                point.append(",");
            }
        }
        return point.toString();
    }

    public void sum(Point p) {
        for (int i = 0; i < this.dim; i ) {
            this.components[i] += p.components[i];
        }
        this.numPoints += p.numPoints;
    }

    public float distance(Point p, int h){
        if (h < 0) {
            Consider only metric distances
            h = 2;
        }

        if (h == 0) {
            Chebyshev
            float max = -1f;
            float diff = 0.0f;
            for (int i = 0; i < this.dim; i ) {
                diff = Math.abs(this.components[i] - p.components[i]);
                if (diff > max) {
                    max = diff;
                }
            }
            return max;
        } else {
            Manhattan, Euclidean, Minkowsky
            float dist = 0.0f;
            for (int i = 0; i < this.dim; i ) {
                dist += Math.pow(Math.abs(this.components[i] - p.components[i]), h);
            }
            dist = (float)Math.round(Math.pow(dist, 1f/h)*100000)/100000.0f;
            return dist;
        }
    }

    public void average() {
        for (int i = 0; i < this.dim; i ) {
            float temp = this.components[i] / this.numPoints;
            this.components[i] = (float)Math.round(temp*100000)/100000.0f;
        }
    }

```

```

}

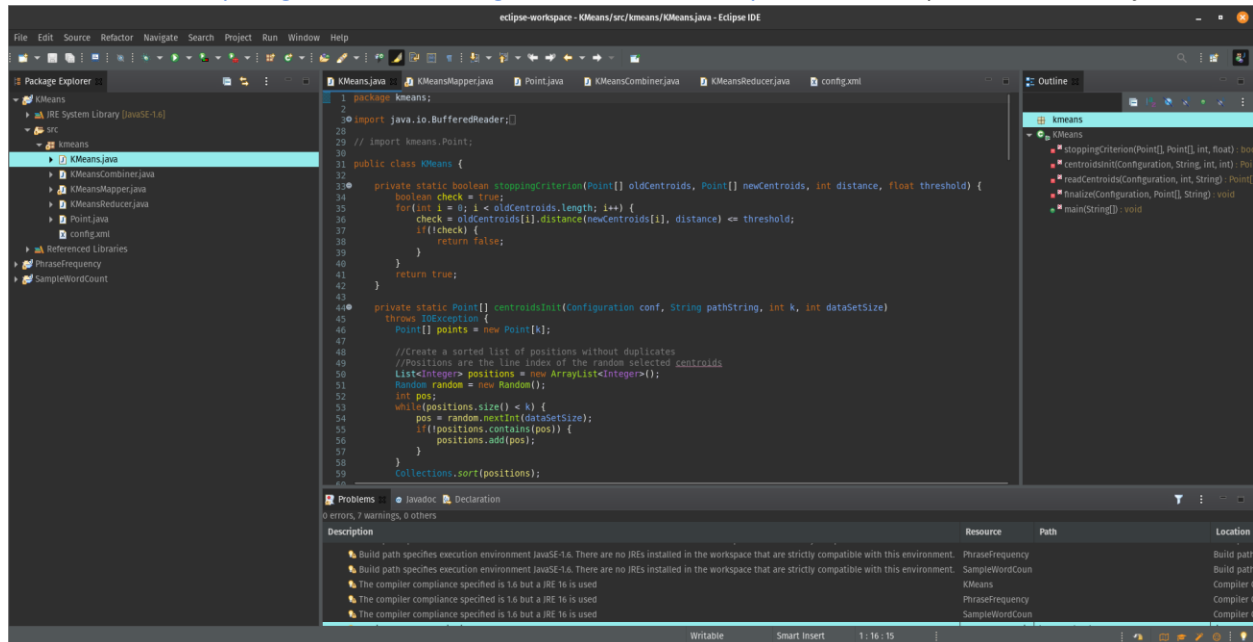
}

}

```

Results:

Add code from <https://github.com/seraogianluca/k-means-mapreduce> to Eclipse and create a .jar file.



Configure the properties like maximum iterations, number of centroids, dataset dimensions, etc either by hard-coding it on KMeans.java (like below) or by using the config.xml provided in the repository and placing it in the Hadoop configuration folder.

```

//Parameters setting
final String INPUT = otherArgs[0];
final String OUTPUT = otherArgs[1] + "/temp";
// Hard-coded values.
final int DATASET_SIZE = 3;
final int DISTANCE = 5;
final int K = 2;
final float THRESHOLD = 0.0001f;
final int MAX_ITERATIONS = 10;

```

Download the dataset from the aforementioned GitHub repository and copy it to HDFS. The dataset contains 1000 points. Run the MapReduce job. The MapReduce job runs with multiple iterations:

ID	User	Name	Application Type	Queue	Application Priority	StartTime	LaunchTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU V-Cores	Allocated Memory MB	Reserved CPU V-Cores	Reserved Memory MB	% of Queue	% of Cluster	Progress	Tracking UI
application_1633427255694_0023	hduser	iter_10	MAPREDUCE	default	0	Tue Oct 26 11:34:27 +0400 2021	Tue Oct 26 11:34:32 +0400 2021	Tue Oct 26 11:34:45 +0400 2021	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History
application_1633427255694_0022	hduser	iter_9	MAPREDUCE	default	0	Tue Oct 26 11:34:05 +0400 2021	Tue Oct 26 11:34:11 +0400 2021	Tue Oct 26 11:34:25 +0400 2021	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History
application_1633427255694_0021	hduser	iter_8	MAPREDUCE	default	0	Tue Oct 26 11:33:44 +0400 2021	Tue Oct 26 11:33:50 +0400 2021	Tue Oct 26 11:34:03 +0400 2021	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History
application_1633427255694_0020	hduser	iter_7	MAPREDUCE	default	0	Tue Oct 26 11:33:23 +0400 2021	Tue Oct 26 11:33:28 +0400 2021	Tue Oct 26 11:33:43 +0400 2021	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History
application_1633427255694_0019	hduser	iter_6	MAPREDUCE	default	0	Tue Oct 26 11:33:00 +0400 2021	Tue Oct 26 11:33:06 +0400 2021	Tue Oct 26 11:33:20 +0400 2021	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History
application_1633427255694_0018	hduser	iter_5	MAPREDUCE	default	0	Tue Oct 26 11:32:41 +0400 2021	Tue Oct 26 11:32:46 +0400 2021	Tue Oct 26 11:32:59 +0400 2021	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History
application_1633427255694_0017	hduser	iter_4	MAPREDUCE	default	0	Tue Oct 26 11:32:19 +0400 2021	Tue Oct 26 11:32:25 +0400 2021	Tue Oct 26 11:32:39 +0400 2021	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History
application_1633427255694_0016	hduser	iter_3	MAPREDUCE	default	0	Tue Oct 26 11:31:58 +0400 2021	Tue Oct 26 11:32:04 +0400 2021	Tue Oct 26 11:32:18 +0400 2021	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History
application_1633427255694_0015	hduser	iter_2	MAPREDUCE	default	0	Tue Oct 26 11:31:37 +0400 2021	Tue Oct 26 11:31:42 +0400 2021	Tue Oct 26 11:31:56 +0400 2021	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History
application_1633427255694_0014	hduser	iter_1	MAPREDUCE	default	0	Tue Oct 26 11:31:21 +0400 2021	Tue Oct 26 11:31:22 +0400 2021	Tue Oct 26 11:31:35 +0400 2021	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History

After the iterations are complete, we get the centroids:

Hadoop

Overview

Datanode

Browse Directory

/kmeans_4

Show 25 entries

Permission

Owner

Showing 1 to 1 of 1 entries

Hadoop, 2019.

File information - centroids.txt

Download

Head the file (first 32K)

Tail the file (last 32K)

Block information - Block 0

Block ID: 1073742384

Block Pool ID: BP-1179251955-127.0.1.1-1629184304183

Generation Stamp: 1568

Size: 51

Availability:

pop-os.localdomain

File contents

-2.82225,-2.34157,1.55772
6.41385,6.96842,-1.34599

Close

Go to

Search:

Size

Name

centroids.txt

Previous

Next