Big Data Analysis

Practical 7

Objective :

Implement any one of the analytic algorithm using mapreduce by handling larger datasets in main memory.

- PCY/Multi-Hash/SON algorithm
- Regression
- K-means Clustering

Roll No. & Name : 18bce183 & Prince Prajapati

Submitted to: Prof. Jaiprakash Verma

Regression in MapReduce

Sometimes, with big data, matrices are too big to handle, and it is possible to use tricks to numerically still do the map. Map-Reduce is one of those. With several cores, it is possible to split the problem, to map on each machine, and then to aggregate it back at the end.

Code :

```
import java.io.BufferedInputStream;
import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.URI;
import java.net.URISyntaxException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import org.apache.hadoop.filecache.DistributedCache;
import org.apache.hadoop.fs.Path;
```

```
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;

public class LinearRegression {

  public static class LinearRegressionMapper extends MapReduceBase implements
Mapper<LongWritable, Text, LongWritable, FloatWritable>
        {
                @Override
                public void map(LongWritable key, Text value, OutputCollector<LongWritable,
FloatWritable> output,
                                Reporter reporter) throws IOException {

                        /*
                          J(theta) = sum((y_predict(x)-y)^2)
                           y_predict(x) = theta(0)*x(0) + .... + theta(i)*x(i)
                         */

                        String line = value.toString();
                        String[] features = line.split(",");

                        List<Float> values = new ArrayList<Float>();

                        // read the values and convert them to floats
                        for(int i = 0; i<features.length; i++)
                        {
                                values.add(new Float(features[i]));
                        }

                        // calculate the costs
                        output.collect(new LongWritable(1), new FloatWritable(costs(values)));

                }

                private final float costs(List<Float> values)
                {

                        float costs = 0;
                        //all right we have all the theta values, lets convert them to floats
                        String[] theta = {"0.01","0.03",   "4",      "2",      "0.9", "2",         "0.8",
        "0.9",    "2",      "3",      "0.1"};

                        //first value is the y value
                        float y = values.get(0);

                        // Calculate the costs for each record in values
                        for(int j = 0; j < values.size(); j++)
                        {
                                        //bias calculation
                                        if(j == 0)
                                                costs += (new Float(theta[j]))*1;
```

```
                                        else
                                                costs += (new Float(theta[j]))*values.get(j);
                        }

                        costs = (costs -y)*(costs - y);
                        return costs;

                }

        }


        public static class LinearRegressionReducer extends MapReduceBase implements
Reducer<LongWritable, FloatWritable, LongWritable, FloatWritable>
        {

                @Override
                public void reduce(LongWritable key, Iterator<FloatWritable>
value,OutputCollector<LongWritable, FloatWritable> output, Reporter reporter)
                                throws IOException {


                        //The reducer just has to sum all the values for a given key


                        float sum = 0;

                        while(value.hasNext())
                        {
                                sum += value.next().get();
                        }

                        output.collect(key, new FloatWritable(sum));
                }

        }
        public static void main(String[] args) {

                JobConf conf = new JobConf(LinearRegression.class);

                conf.setJobName("linearregression");


                conf.setOutputKeyClass(LongWritable.class);
                conf.setOutputValueClass(FloatWritable.class);

                conf.setMapperClass(LinearRegressionMapper.class);
                conf.setCombinerClass(LinearRegressionReducer.class);
                conf.setReducerClass(LinearRegressionReducer.class);
                conf.setInputFormat(TextInputFormat.class);
                conf.setOutputFormat(TextOutputFormat.class);
```

```
                FileInputFormat.setInputPaths(conf, new Path(args[0]));
                FileOutputFormat.setOutputPath(conf, new Path(args[1]));
                try {
                        JobClient.runJob(conf);
                } catch (IOException e) {
                        e.printStackTrace();
                }

        }

}
```

Output:

hadoop jar C:\big-data\MapReduce.jar LinearRegression /practical7/diabetes.csv /practical7/output

```
C:\big-data\hadoop-3.3.0\sbin>hadoop jar C:\big-data\MapReduce.jar LinearRegression /practical7/diabetes.csv /practical7/output
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 2
        at LinearRegression.main(LinearRegression.java:117)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:498)
        at org.apache.hadoop.util.RunJar.run(RunJar.java:323)
        at org.apache.hadoop.util.RunJar.main(RunJar.java:236)

C:\big-data\hadoop-3.3.0\sbin>hadoop jar C:\big-data\MapReduce.jar LinearRegression /practical7/diabetes.csv /practical7/output
2021-11-12 19:29:03,145 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at /0.0.0.0:8032
2021-11-12 19:29:03,542 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at /0.0.0.0:8032
2021-11-12 19:29:05,257 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Too
2021-11-12 19:29:05,352 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/91926/.
2021-11-12 19:29:06,313 INFO mapred.FileInputFormat: Total input files to process : 1
2021-11-12 19:29:06,974 INFO mapreduce.JobSubmitter: number of splits:2
2021-11-12 19:29:07,955 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1636725029444_0001
2021-11-12 19:29:07,956 INFO mapreduce.JobSubmitter: Executing with tokens: []
2021-11-12 19:29:08,403 INFO conf.Configuration: resource-types.xml not found
2021-11-12 19:29:08,404 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2021-11-12 19:29:09,147 INFO impl.YarnClientImpl: Submitted application application_1636725029444_0001
2021-11-12 19:29:09,319 INFO mapreduce.Job: The url to track the job: http://LAPTOP-0MDL8L9R:8088/proxy/application_16367250294
2021-11-12 19:29:09,322 INFO mapreduce.Job: Running job: job_1636725029444_0001
2021-11-12 19:29:27,726 INFO mapreduce.Job: Job job_1636725029444_0001 running in uber mode : false
2021-11-12 19:29:27,730 INFO mapreduce.Job:  map 0% reduce 0%
2021-11-12 19:29:42,054 INFO mapreduce.Job:  map 100% reduce 0%
2021-11-12 19:29:55,277 INFO mapreduce.Job:  map 100% reduce 100%
2021-11-12 19:29:57,325 INFO mapreduce.Job: Job job_1636725029444_0001 completed successfully
```

Output showing square-error of regresion

```
C:\big-data\hadoop-3.3.0\sbin>hadoop fs -ls /practical7/output
Found 2 items
-rw-r--r--   1 91926 supergroup          0 2021-11-12 19:29 /practical7/output/_SUCCESS
-rw-r--r--   1 91926 supergroup         14 2021-11-12 19:29 /practical7/output/part-00000

C:\big-data\hadoop-3.3.0\sbin>hadoop fs -cat /practical7/output/part-00000
1       1.2834921E7
```

Conclusion:

After implementing this practical now I have clear understanding about regression in mapReduce and how it works with mapReduce.