**YouTube Data Analysis**

This blog is about, how to perform YouTube data analysis in [Hadoop](https://acadgild.com/big-data/big-data-development-training-certification) MapReduce.  
This YouTube data is publicly available and the YouTube data set is described below under the heading Data Set Description.  
Using that dataset we will perform some Analysis and will draw out some insights like what are the top 10 rated videos on YouTube, who uploaded the most number of videos.  
By reading this blog you will understand how to handle data sets that do not have proper structure and how to sort the output of reducer.

**DATA SET DESCRIPTION**

**Column 1:** Video id of 11 characters.

**Column 2:** uploader of the video

**Column 3:** Interval between the day of establishment of Youtube and the date of uploading of the video.

**Column 4:** Category of the video.

**Column 5:** Length of the video.

Bottom of Form

Top of Form

Bottom of Form

**Column 6:** Number of views for the video.

**Column 7:** Rating on the video.

**Column 8:** Number of ratings given for the video

**Column 9:** Number of comments done on the videos.

**Column 10:** Related video ids with the uploaded video.

You can download the data set from the below link.

**DATA SET LINK**

https://drive.google.com/file/d/0ByJLBTmJojjzR2x0MzVpc2Z6enM/view

**PROBLEM STATEMENT 1**

Here we will find out what are the top 5 categories with maximum number of videos uploaded.

**SOURCE CODE**

Now from the mapper, we want to get the *video category as key* and final int value *‘1’ as values* which will be passed to the *shuffle* and *sort* phase and are further sent to the reducer phase where the aggregation of the values is performed.

|  |
| --- |
| //top 5 categories of videos in youtube |
|  |

|  |
| --- |
|  |
|  |
|  |
|  |
| import java.io.IOException; |
|  |
|  |
|  |
|  |
|  |
| import org.apache.hadoop.fs.Path; |
|  |
|  |
| import org.apache.hadoop.conf.\*; |
|  |
|  |
| import org.apache.hadoop.io.\*; |
|  |
|  |
| import org.apache.hadoop.mapreduce.\*; |
|  |
|  |
| 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; |
|  |
|  |
|  |
|  |
|  |
| public class Top5\_categories { |
|  |
|  |
|  |
|  |
|  |
| public static class Map extends Mapper<LongWritable, Text, Text, |
|  |
|  |
| IntWritable> { |
|  |
|  |
|  |
|  |
|  |
| private Text category = new Text(); |
|  |
|  |
| private final static IntWritable one = new IntWritable(1); |
|  |
|  |
| public void map(LongWritable key, Text value, Context context ) |
|  |
|  |
| throws IOException, InterruptedException { |
|  |
|  |
| String line = value.toString(); |
|  |
|  |
| String str[]=line.split("\t"); |
|  |
|  |
|  |
|  |
|  |
| if(str.length > 5){ |
|  |
|  |
| category.set(str[3]); |
|  |
|  |
| } |
|  |
|  |
|  |
|  |
|  |
| context.write(category, one); |
|  |
|  |
| } |
|  |
|  |
|  |
|  |
|  |
| } |
|  |
|  |
|  |
|  |
|  |
| public static class Reduce extends Reducer<Text, IntWritable, |
|  |
|  |
| Text, IntWritable> { |
|  |
|  |
|  |
|  |
|  |
| public void reduce(Text key, Iterable<IntWritable> values, |
|  |
|  |
| Context context) |
|  |
|  |
| throws IOException, InterruptedException { |
|  |
|  |
| int sum = 0; |
|  |
|  |
| for (IntWritable val : values) { |
|  |
|  |
|  |
|  |
|  |
| sum += val.get(); |
|  |
|  |
| } |
|  |
|  |
| context.write(key, new IntWritable(sum)); |
|  |
|  |
| } |
|  |
|  |
| } |
|  |
|  |
|  |
|  |
|  |
| public static void main(String[] args) throws Exception { |
|  |
|  |
| Configuration conf = new Configuration(); |
|  |
|  |
|  |
|  |
|  |
| @SuppressWarnings("deprecation") |
|  |
|  |
| Job job = new Job(conf, "categories"); |
|  |
|  |
| job.setJarByClass(Top5\_categories.class); |
|  |
|  |
|  |
|  |
|  |
| job.setMapOutputKeyClass(Text.class); |
|  |
|  |
| job.setMapOutputValueClass(IntWritable.class); |
|  |
|  |
| //job.setNumReduceTasks(0); |
|  |
|  |
| job.setOutputKeyClass(Text.class); |
|  |
|  |
| job.setOutputValueClass(IntWritable.class); |
|  |
|  |
|  |
|  |
|  |
| job.setMapperClass(Map.class); |
|  |
|  |
| job.setReducerClass(Reduce.class); |
|  |
|  |
|  |
|  |
|  |
| job.setInputFormatClass(TextInputFormat.class); |
|  |
|  |
| job.setOutputFormatClass(TextOutputFormat.class); |
|  |
|  |
|  |
|  |
|  |
| FileInputFormat.addInputPath(job, new Path(args[0])); |
|  |
|  |
| FileOutputFormat.setOutputPath(job, new Path(args[1])); |
|  |
|  |
| Path out=new Path(args[1]); |
|  |
|  |
| out.getFileSystem(conf).delete(out); |
|  |
|  |
| job.waitForCompletion(true); |
|  |
|  |
| } |
|  |
|  |
|  |
|  |

|  |
| --- |
| } |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

In **line 1** we are taking a class by name Top5\_categories,  
In **line 2** we are extending the Mapper default class having the arguments keyIn as LongWritable and ValueIn as Text and KeyOut as Text and ValueOut as IntWritable.

In **line 3** we are declaring a private Text variable ‘category’ which will store the category of videos in youtube

In **line 4** we are declaring a private final static IntWritable variable ‘one’ which will be constant for every value. MapReduce deals with Key and Value pairs.Here we can set the key as gender and value as age.

In **line 5** we are overriding the map method which will run one time for every line.

In **line 7** we are storing the line in a string variable ‘line’

In **line 8** we are splitting the line by using tab “\t” delimiter and storing the values in a String Array so that all the columns in a row are stored in the string array.

In **line 9** we are taking a condition if we have the string array of length greater than 6 which means if the line or row has at least 7 columns then it will enter into the if condition and execute the code to eliminate the **ArrayIndexOutOfBoundsException.**

In **line 10** we are storing the category which is in the 4th column.

In **line 12** we are writing the key and value into the context which will be the output of the map method.

### REDUCER CODE

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | public static class Reduce extends Reducer<Text, IntWritable,Text,IntWritable>{     public void reduce(Text key, Iterable<IntWritable> values,Context context throws IOException, InterruptedException {             int sum = 0;             for (IntWritable val : values) {                 sum += val.get();  }             context.write(key, new IntWritable(sum));  }  } |

#### ****While coming to the Reducer code****

**line 1** extends the default Reducer class with arguments KeyIn as Text and ValueIn as IntWritable which are same as the outputs of the mapper class and KeyOut as Text and ValueOut as IntWritbale which will be final outputs of our MapReduce program.

In **line 2** we are overriding the Reduce method which will run each time for every key.

In **line 3** we are declaring an integer variable sum which will store the sum of all the values for each key.

In **line 4** a for each loop is taken which will run each time for the values inside the “Iterable values” which are coming from the shuffle and sort phase after the mapper phase.

In **line 5** we are storing and calculating the sum of the values.

In **line 7** writes the respected key and the obtained sum as value to the context.

### CONF CODE

|  |  |
| --- | --- |
| 1  2 | job.setMapOutputKeyClass(Text.class);  job.setMapOutputValueClass(IntWritable.class); |

This two configuration classes are included in the main class whereas to clarify the Output key type of mapper and the output value type of the Mapper.