Map-Reduce program documentation

First job - Tokenization, Case folding, Stopwords

In the first map-reduce job as entry <k, v> pairs of the mapper we have the full dataset. Below is the declaration of the first mapper:

```
public static class TokenizerMapper extends Mapper<LongWritable, Text, Text,
IntWritable>
```

In the setup method of this mapper, we will fill an array with patterns to skip that we get from stopwords.txt. The map method of the mapper will convert the values in JSON format and will get the category and reviewText. In the reviewText will be used `toLowerCase()` function for case folding. Further the reviewText input will be tokenized by firstly replacing all numbers with spaces and then delimit with all delimiters. After tokenization, each word will be checked if it is a stopword or if already exists in the `uniqueWordsPerReview` array (this array is used because we want to send to the reducer only the unique words per line). The output of mapper will consist of three type of <k, v> pairs:

1. <<category,word>, 1>
2. <categoryCount~category, 1> and
3. <wordCount~word, 1>

categoryCount~category and wordCount~word types will be used in the mapper of the second job to calculate Chi-Square.

The reducer of the first job sums the occurrences of each pair sent by mapper and will store the result in `/tmp` directory. Below is the declaration of the first reducer:

```
public static class IntSumReducer extends Reducer<Text, IntWritable, Text,
IntWritable>
```

In this job we will also use the combiner. We will set it same as the reducer in order to speed up the process of summation.

Second job - Chi-Square Calculation, Sort and get Top N

In the second map-reduce job as entry <k, v> pairs of the mapper we have the output of the first job. Below is the declaration of the first mapper:

```
public static class SortTopNMapper extends Mapper<LongWritable, Text,
CompositeKeyWritable, NullWritable>
```

In the setup of this mapper we will open the output file and fill two HashMaps. One of them will contain all words of the dataset and a count of their unique occurrence across the reviews and the

other will contain all categories of the dataset and a count of their unique occurrence across the reviews. The map method of the mapper will calculate the chi-square value for pair, excluding categoryCount~category and wordCount~word pairs.

As an output of this mapper is used a composite key, consisting of a class called `CompositeKeyWritable` and null as a value. The CompositeKeyWritable class declaration is:

```
public class CompositeKeyWritable implements Writable,
WritableComparable<CompositeKeyWritable>
```

This class has two string attributes: `category` and `wordCSPair`. The output pairs of the mapper is <<category, (chi-square,word)>, NullWritable>.

This job also sets a partitioner class which is:

```
public class SecondarySortPartitioner implements Partitioner<CompositeKeyWritable,
NullWritable>
```

a sort comparator class:

```
public class SecondarySortKeySortComparator extends WritableComparator
```

and a grouping comparator class:

```
public class SecondarySortGroupComparator extends WritableComparator
```

The SecondarySortPartitioner class, partitions the data by category. The SecondarySortKeySortComparator class sorts the categories in the ascending order and also the chi-square values in the descending order for each category. Finally, the SecondarySortGroupComparator class, groups the results by category.

The reducer will concatenate all the words and chi-square values by "**:**". Below is the declaration of the second reducer:

```
public static class SortTopNReducer extends Reducer<CompositeKeyWritable,
NullWritable, Text, NullWritable>
```

The ouput <key,value> pairs of this reducer will be <<category word:chi-square>, NullWritable> where the key will have 200 word:chi-square tuples, space-separated for each category.

After the second job is completed, we no longer need the output of the first job therefore we will delete it.

Third job - Creating line of all terms space-separated and ordered alphabetically

In the third and final map-reduce job as entry <k, v> pairs of the mapper we have the output of the second job. Below is the declaration of the third mapper:

```
public static class SortWordsMapper extends Mapper<LongWritable, Text, Text,
IntWritable>
```

In map method of the mapper we will separate the category and chi-square from words and we will send only the words to the reducer. The output of the mapper will be: <word, 1>. Below is the declaration of the third reducer:

```
public static class SortWordsReducer extends Reducer<Text, IntWritable, Text,
NullWritable>
```

In the reduce method of the reducer we will concatenate all the unique words in a string, space-separated, while in the cleanup method of the reducer we will output the <key, value> pairs, with key consisting of one line of space-separated unique words and null as a value.

This job will output the results in a folder called `one_line_file`. After this job is completed, the program will exit.