Assignment 1 Documentation

The java program contains three map reduce jobs that do the following:

1. Preprocesses review text: tokenization, case folding, stop word filtering, one character terms filtering; counts words, categories and <word, category> pairs
2. Calculates chi-square values for each term per category; sorts categories in ascending order and terms inside each category in descending order based on their chi-square value; creates an output where each line has the category and a list of 200 terms of that category that have the highest chi-square value
3. Creates a line containing top n terms from each category in ascending order

First Map Reduce job

The first map reduce job uses a mapper, a reducer and a combiner.

Mapper

The input of the mapper is Amazon Reviews Dataset. Before processing the values of the dataset, a hash set containing all the stop words is created in the setup method of the mapper.

Inside the map method, each line of the dataset is converted to json format for easier processing. For each line that comes in, there is a counter (Counters.RowsNo) that increments its value and stores the number of dataset rows after mapping is finished. In the next steps it is used for calculating the chi-square values.

Inside the map method, review text is tokenized using digits, spaces, tabs and some other characters as delimiters, it is also filtered from stop words and one character words. In order to not process duplicate words per review, an ArrayList<String> uniqueWords is created, to keep track of the words already processed for that line of dataset.

The map method outputs the key of type Text, and a value of type IntWritable(1) :

<new Text key, new IntWritable(1)>

The keys outputted by mapper are of type: “word-”+word, “category-”+category, category+”,”+word. So in this first job it is counted word, category and each <word, category> pair frequency, as well as the number of rows, so there is all that’s needed for calculating the chi-square value.

Mapper class declaration:

```
public static class TokenizerMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
        ……
        @Override
        public void map(LongWritable key, Text value, Context context) throws IOException,
        InterruptedException {
                ……
                context.getCounter(Counters.RowsNo).increment(1);
                context.write(new Text(category + "," + word), one);
                context.write(new Text("word-" + word), one);
                context.write(new Text("category-" + category), one);
        }
}
```

Reducer

The reducer sums up the values of the keys, and it writes them in a file in the following format:

“word-”+word sum

“category-”+category sum

category +”,” + word sum

…

Reducer class declaration:

```
public static class TokenizerSumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
        …….
}
```

So the key is of type Text, whereas the value is of type IntWritable:

<Text key, new IntWritable(sum)>

The combiner uses the same class as the reducer, and it’s used to improve the performance when summing up the values of the terms.

Second Map Reduce job

The second map reduce job uses a mapper, a partitioner, a group comparator, a key comparator and reducer.

Mapper

The mapper of this job calculates the chi-square value for each category, term pair. As in input, it takes the output of the first job. Inside the setup method are taken the values that contain “word-” and "category-" and stored in *words* and *categories* hash maps respectively. Then inside the map method are processed only the values of type category+ ”,” +word. There are calculated chi-squares, having all the information needed: the rows number, word frequency, categories count and <category, word> count.

Mapper class declaration:

```
public static class ChiSquareTopPairsMapper extends Mapper<LongWritable, Text,
CategoryWordChiPair, NullWritable> {
        …..
}
```

This mapper as a key uses a composite key, a class called *CategoryWordChiPair,* containing category as natural key and word, chi-square values concatenated in a string, *wordChiPair* as secondary key.

The value outputted by mapper is null. So the key value pair of the mapping method is:

<new CompositeWordChiPair key, NullWritable>

CompositeWordChiPair class declaration:

```
public class CategoryWordChiPair implements Writable,
WritableComparable<CategoryWordChiPair> {
    private String category;
    private String wordChiPair;
    …….
}
```

Partitioner

The partitioner class *PartitionerSecondarySort* is used for partitioning. It uses the natural key, the category, for partitioning:

```
public class PartitionerSecondarySort extends Partitioner<CategoryWordChiPair, NullWritable> {
        ….
}
```

Group Comparator

The group comparator class *GroupComparator* compares and groups by category (only by natural key):

```
public class GroupComparator extends WritableComparator {
        ……
}
```

Key Comparator

The key comparator class *KeySortComparator* is what sorts all the terms depending on chi-square values

```
public class KeySortComparator extends WritableComparator {
        ……
}
```

on descending order for each category.

Reducer

Inside the reducer, for each category are taken the top n terms with the highest chi-square value and concatenated in one string for each category.

Class declaration:

```
public static class ChiSquareTopPairsReducer
        extends Reducer<CategoryWordChiPair, NullWritable, Text, NullWritable> {
        …..
}
```

The key and value it outputs are:

<Text key, NullWritable>

The key contains category, then an empty space followed by top 200 (or n) terms with the highest chi-square value in the format:

Category1 word1:chisq1 word2:chisq2 ……. word200:chisq200

…..

For each category, this reducer outputs the above format in a file. For the final result, we only need to concatenate the terms in a single line space separated, which will be done by third map reduce job.

Third Map Reduce job

This job contains a mapper and a reducer.

Mapper

The mapper takes as an input the output of the second job. For each line of the document, it removes the category chi square values and only takes the terms. The key and value it outputs are:

<Text key, new IntWritable(1)> , where key is the term discussed above.

Mapper class declaration:

```
public static class AllTermsMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
    …..
}
```

Reducer

The reducer processes the keys sent by the mapper, i.e. the terms. Inside the reduce method it creates a line containing all the terms space separated, and in the cleanup method it outputs this line to a file.

Reducer class declaration:

```
public static class AllTermsReducer extends Reducer<Text, IntWritable, Text, NullWritable> {
        ….
}
```

The key and value of the output is of format:

<Text line, NullWritable> , where line contains all the terms in ascending order and space separated.