BDPA winter 2017: Assignment 2

Rajaa EL HAMDANI

March 17, 2017

1 Pre-processing the input

1.1 Word counts job

1.1.1 Mapper

The Mapper tokenizes each line and ouput pairs where the keys are tokens and values are 1.

```java

public class WordCountMapper extends Mapper<LongWritable, Text, Text,
    IntWritable> {
    private Text word = new Text();
    private final static IntWritable ONE = new IntWritable(1);

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

      String line = value.toString().toLowerCase();
      StringTokenizer tokenizer = new StringTokenizer(line);

       while (tokenizer.hasMoreTokens()) {
          word.set(tokenizer.nextToken());
          context.write(word, ONE);
       }
    }
  }
```

1.1.2 Reducer

The reducer count the frequency of each word by summing over the received values.

```java
public class WordCountReducer extends Reducer<Text, IntWritable, Text,
    IntWritable> {

  @Override
  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));
  }
}
```

1.1.3 Driver

```
public class WordCount {

  public static void main(String[] args) throws Exception {

    if (args.length != 2) {
      System.out.printf("Usage: WordCount <input dir> <output dir>\n");
      System.exit(-1);
    }

    Job job = new Job();
    job.setJarByClass(WordCount.class);
    job.setJobName("Word count");

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    job.setMapperClass(WordCountMapper.class);
    job.setCombinerClass(WordCountReducer.class);
    job.setReducerClass(WordCountReducer.class);

    System.exit(job.waitForCompletion(true) ? 0:1);
  }
}
```

1.2 Preprocessing job

To carry the task of preprocessing, I wrote two MapReduce jobs. The first job counts the input words. The second job performs the preprocessing.

1.2.1 Mapper

The mapper performs the following tasks:
- Loading the stop-words file (generated in the 1st assignment) in a HashSet.

```
HashSet<String> stopwords = new HashSet<String>();
        BufferedReader Reader = new BufferedReader(new FileReader(new File("/
    home/cloudera/stopwords.csv")));
        String line;
        while ((line = Reader.readLine()) != null) {
            stopwords.add(line.split("\\s+")[0].toLowerCase());
        }
```

- Removing stopwords. - Removing words that contain special characters by matching each word with a regular expression.

```

    String lineF = value.toString().toLowerCase();
    StringTokenizer tokenizer = new StringTokenizer(lineF);

    Pattern p = Pattern.compile("[^A-Za-z0-9]");
    String token;
    while (tokenizer.hasMoreTokens()) {
      token = tokenizer.nextToken().toLowerCase();
      if (!(stopwords.contains(token) || p.matcher(token).find() || token.
  isEmpty())){
        word.set(token);
            context.write(key, word);
      }
       }
```

- Removing empty lines.

1.2.2 Reducer

The reducer performs the following tasks: - Remove repetition of words by loading the words of its line input in a HashSet.

```
    ArrayList<String> tokens = new ArrayList<String>();

    for (Text word : values) {
      tokens.add(word.toString());
    }

    HashSet<String> tokensU = new HashSet<String>(tokens);
```

- Sort the words, of the line, by their global frequency. This is done by the following steps:

1. Loading the word count file (output of the WordCount job).
2. Storing the word counts (only for the words that appear in the current line) in a HashMap, where the keys are the word and values are their frequencies.
3. Sorting the HashMap by its values in ascending order.

```
BufferedReader reader = new BufferedReader(new FileReader(new File("/home/
  cloudera/workspace/SetSimJoins/wordcount.txt")));

    Map<String, Integer> wordcount = new HashMap<String, Integer>(); //
  HashMap table to store words as keys and their frequency as vales
    String line;
    while ((line = reader.readLine()) != null) {
      String[] word = line.split("\\s+");
      if (tokensU.contains(word[0])){
        wordcount.put(word[0], Integer.parseInt(word[1]));
      }

    }
    // Sort table by values
    Map<String, Integer> sortedMap = sortByValue(wordcount);
          /* sortByValue is a function that sort a hashMap by its values.
  Its code is in          the appendix
          */
```

- Writing sorted words in a string buffer

```
// Write the ordered words in a StringBuffer
    StringBuffer bf = new StringBuffer();
    for (Entry<String, Integer> entry : sortedMap.entrySet()) {
      if(bf.length()!=0){
        bf.append(" ");
      }
      bf.append(entry.getKey());
    }
```

- Outputing the line in the HDFS (only if it not empty), and incrementing the counter of output records.

```

      if(bf.length()!=0){
        context.getCounter(RECORDS_COUNTER.NB_RECORDS).increment(1);
      context.write(key, new Text(bf.toString()));
      }
```

The variable `RECORDS_COUNTER` is instantiated in the driver.

1.2.3 Driver

The driver performs the following tasks:
- Instantiation and configuration of the MapReduce Job.

```
  public static void main(String[] args) throws IOException,
   ClassNotFoundException, InterruptedException {
     /*
       * Validate that two arguments were passed from the command line.
       */
      if (args.length != 2) {
        System.out.printf("Usage: StopWords <input dir> <output dir>\n");
        System.exit(-1);
      }

      /*
       * Instantiate a Job object for your job's configuration.
       */
      Configuration conf = new Configuration();
      Job job = new Job(conf);
      job.setJarByClass(Preprocessing.class);
      job.setJobName("Part1_Preprocessing");

      FileInputFormat.addInputPath(job, new Path(args[0]));
      FileOutputFormat.setOutputPath(job, new Path(args[1]));

```

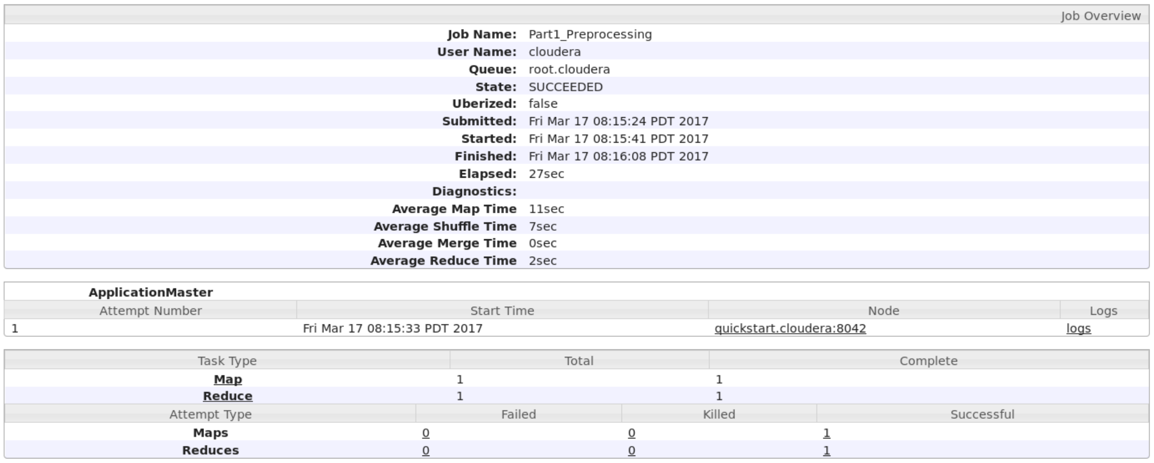
```
        job.setMapperClass(PreprocessingMapper.class);
        job.setReducerClass(PreprocessingReducer.class);

        job.setOutputKeyClass(LongWritable.class);
        job.setOutputValueClass(Text.class);

        job.setOutputFormatClass(TextOutputFormat.class);

        job.getConfiguration().set(
          "mapreduce.output.textoutputformat.separator", "; ");

        FileSystem fs = FileSystem.get(new Configuration());
        if (fs.exists(new Path(args[1]))) {
          fs.delete(new Path(args[1]));
        }

        job.waitForCompletion(true);
```

- Instantiation of the counter of the output records.

```
public static enum RECORDS_COUNTER {
        NB_RECORDS,
  };
```

- Writing the value of the counter in HDFS.

```

        long counter = job.getCounters().findCounter(RECORDS_COUNTER.NB_RECORDS)
          .getValue();
      Path counterFile = new Path("NB_RECORDS.txt");
      BufferedWriter bf = new BufferedWriter(new OutputStreamWriter(
          fs.create(counterFile, true)));
      bf.write(String.valueOf(counter));
      bf.close();

      System.exit(0);
```



Job Overview

Job Name:	Part1_Preprocessing
User Name:	cloudera
Queue:	root.cloudera
State:	SUCCEEDED
Uberized:	false
Submitted:	Fri Mar 17 08:15:24 PDT 2017
Started:	Fri Mar 17 08:15:41 PDT 2017
Finished:	Fri Mar 17 08:16:08 PDT 2017
Elapsed:	27sec
Diagnostics:	
Average Map Time	11sec
Average Shuffle Time	7sec
Average Merge Time	0sec
Average Reduce Time	2sec

ApplicationMaster

Attempt Number	Start Time	Node	Logs
1	Fri Mar 17 08:15:33 PDT 2017	quickstart.cloudera:8042	logs

Task Type	Total	Complete
Map	1	1
Reduce	1	1

Attempt Type	Failed	Killed	Successful
Maps	0	0	1
Reduces	0	0	1

Figure 1: Logs in Yarn of the pre-processing job

2 Set-similarity joins

2.1 First approach

In the first approach we perform all the pair-wise comparison between documents. To make sure that the same pair of documents is only compared once I implemented a custom WritablComparable class `DocPair` (the code is in the appendix).

2.1.1 Mapper

The mapper performs the following tasks:
- Read the pre-processed file.
- Associate the current document id with the ids of the rest of documents. The result is stored in a DocPair object and output as a key from the mapper.

```

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

        reader = new BufferedReader(new FileReader(new File("/home/cloudera/
    workspace/SetSimJoins/preprocessing_output_sample.txt")));
        String line;
        System.out.println(value.toString());
        String[] valueS = value.toString().split(";");
        while ((line = reader.readLine()) != null) {
            String key2 = line.split(";")[0];
            if (!key.toString().equals(key2)){
              pairKeys.set(new Text(valueS[0]), new Text(key2));
              context.write(pairKeys, new Text(valueS[1]));
            }
        }
        }
```

2.1.2 Reducer

The reducer performs the following tasks:—— - Computation of the Jaccard similarity between pairs of documents.

```
public class SetsimjoinsReducer extends
    Reducer<DocPair, Text, Text, Text> {

    private BufferedReader reader;


    @Override
    public void reduce(DocPair key, Iterable<Text> values, Context context)
            throws IOException, InterruptedException {

        HashMap<String, String> allLines = new HashMap<String, String>();
        reader = new BufferedReader(new FileReader(new File("/home/cloudera/
  workspace/SetSimJoins/preprocessing_output_sample.txt")));
        String line;
        while ((line = reader.readLine()) != null) {
            String[] lineS = line.split(";");
            allLines.put(lineS[0], lineS[1]);
        }

        HashSet<String> words1 = new HashSet<String>();

        for (String word : values.iterator().next().toString().split(" ")) {
          words1.add(word);
        }

        HashSet<String> words2 = new HashSet<String>();
        String doc2 = allLines.get(key.getSecond()
                .toString());
        for (String word : doc2.split(" ")) {
          words2.add(word);
        }
        double sim = jaccardsim(words2, words2);


}
}
```

-Output the similar pairs in HDFS.

```
        if (sim >= 0.8) {
            context.write(new Text("(" + key.getFirst() + ", " + key.getSecond
  () + ")"),
                        new Text(String.valueOf(sim)));
        }
```

- Increment the number of comparisons. (The `COUNTER` variable is instantiated in the driver)

```
 context.getCounter(COUNTER.NB_COMPARISIONS_I).increment(1);
```

2.1.3 Driver

The driver performs the following tasks:
- Instantiation and configuration of the MapReduce Job.

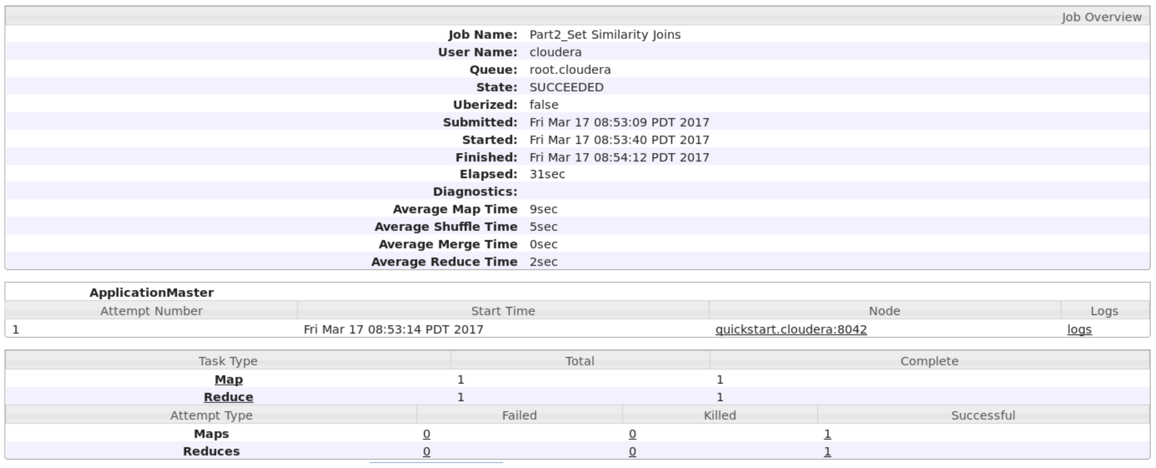
```
public class Setsimjoins {

  public static void main(String[] args) throws IOException,
   ClassNotFoundException, InterruptedException {

      if (args.length != 2) {
        System.out.printf("Usage: StopWords <input dir> <output dir>\n");
        System.exit(-1);
      }

      Configuration conf = new Configuration();
      Job job = new Job(conf);
      job.setJarByClass(Setsimjoins.class);
      job.setJobName("Part2_Set Similarity Joins");

      FileInputFormat.addInputPath(job, new Path(args[0]));
      FileOutputFormat.setOutputPath(job, new Path(args[1]));

      job.setMapperClass(SetsimjoinsMapper.class);
      job.setReducerClass(SetsimjoinsReducer.class);

    job.setMapOutputKeyClass(DocPair.class);
    job.setMapOutputValueClass(Text.class);

      job.setOutputKeyClass(Text.class);
      job.setOutputValueClass(Text.class);

      job.setOutputFormatClass(TextOutputFormat.class);


      FileSystem fs = FileSystem.get(new Configuration());
      if (fs.exists(new Path(args[1]))) {
        fs.delete(new Path(args[1]));
      }

      job.waitForCompletion(true);

  }
}
```

- Instantiation of the counter of the number of comparisons.

```
  public static enum COUNTER {
      NB_COMPARISIONS_I,
  };
```

- Writing the value of the counter in HDFS.

```
      long counter = job.getCounters()
              .findCounter(COUNTER.NB_COMPARISIONS_I).getValue();
      Path outFile = new Path("NB_COMPARISIONS_I.txt");
      BufferedWriter br = new BufferedWriter(new OutputStreamWriter(
              fs.create(outFile, true)));
      br.write(String.valueOf(counter));
      br.close();

    System.exit(0);
```



	Job Overview
Job Name:	Part2_Set Similarity Joins
User Name:	cloudera
Queue:	root.cloudera
State:	SUCCEEDED
Uberized:	false
Submitted:	Fri Mar 17 08:53:09 PDT 2017
Started:	Fri Mar 17 08:53:40 PDT 2017
Finished:	Fri Mar 17 08:54:12 PDT 2017
Elapsed:	31sec
Diagnostics:	
Average Map Time	9sec
Average Shuffle Time	5sec
Average Merge Time	0sec
Average Reduce Time	2sec

ApplicationMaster

Attempt Number	Start Time	Node	Logs
1	Fri Mar 17 08:53:14 PDT 2017	quickstart.cloudera:8042	logs

Task Type	Total	Complete
Map	1	1
Reduce	1	1

Attempt Type	Failed	Killed	Successful
Maps	0	0	1
Reduces	0	0	1

Figure 2: Logs in Yarn of the comparison job for the 1st approach

2.2 Second approach

In the second approach we compare only pairs of documents that have in common the first $|d| - \lceil t.|d| \rceil + 1$ words, where $|d|$ is the number of words in document d, and t is the Jaccard similarity threshold.

2.2.1 Mapper

The mapper implements an inverted index for the first $|d| - \lceil t.|d| \rceil + 1$ words of each document.

```java
public class Setsimjoins2Mapper extends
    Mapper<LongWritable, Text, Text, Text> {

      private Text word = new Text();

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

        String doc = value.toString().split(";")[1];
        String docID = value.toString().split(";")[0];
        String[] words = doc.split(" ");
          long keptWordsNumber = Math.round(words.length - (words.length *
    0.8) + 1);
          String[] keptWords = Arrays.copyOfRange(words, 0,(int)
    keptWordsNumber);


          for (String keptWord : keptWords) {

              word.set(keptWord);
              //System.out.println(docID);
              context.write(word, new Text(docID));
          }
      }
}
```

2.2.2 Reducer

The reducer create pairs of documents the belong to the same line in the inverted index, ad then computes the Jaccard similarity.

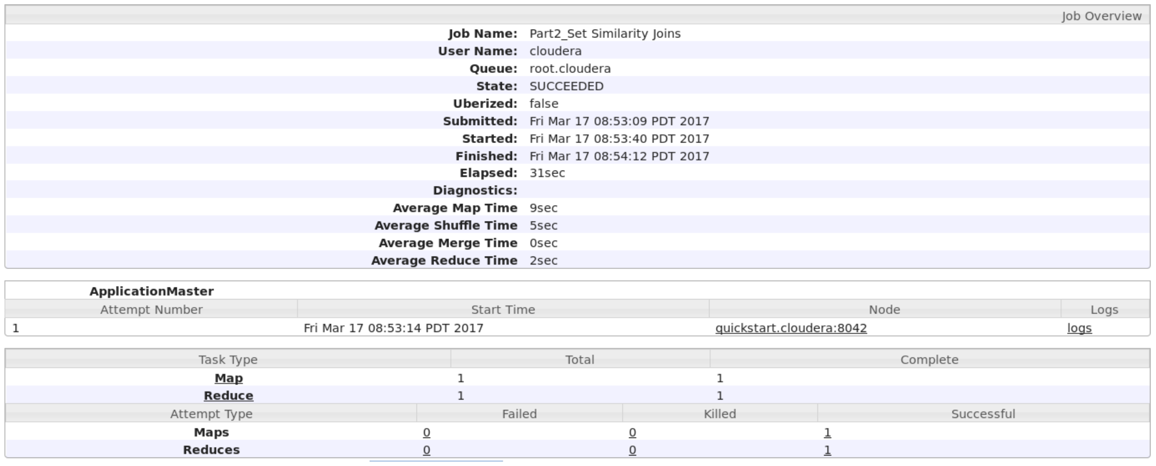
```
public class Setsimjoins2Reducer extends Reducer<Text, Text, Text, Text> {

    private BufferedReader reader;

    @Override
  public void reduce (Text key, Iterable<Text> values, Context context)
            throws IOException, InterruptedException {

      HashMap<String, String> allLines = new HashMap<String, String>();
        reader = new BufferedReader(new FileReader(new File("/home/cloudera/workspace/SetSimJoins/preprocessing_output_sample.txt")));
        String line;
        while ((line = reader.readLine()) != null) {
            String[] lineS = line.split(";");
            allLines.put(lineS[0], lineS[1]);
        }

        List<String> docSet = new ArrayList<String>();
        for (Text id : values){
          docSet.add(id.toString());
        }

      //System.out.println(docSet[0]);

      if (docSet.size() > 1) {
          ArrayList<String> pairs = new ArrayList<String>();
          for (int i = 0; i < docSet.size(); ++i) {
              for (int j = i + 1; j < docSet.size(); ++j) {
                  String pair = new String(docSet.get(i) + " "
                          + docSet.get(j));
                  pairs.add(pair);
              }
          }
          //System.out.println("****************");
          //System.out.println(pairs.size());
          for (String pair : pairs) {
            HashSet<String> words11 = new HashSet<String>();
              String words12 = allLines.get(pair.split(" ")[0].toString());
              for (String word : words12.split(" ")) {
                  words11.add(word);
              }

              HashSet<String> words21 = new HashSet<String>();
              String words22 = allLines.get(pair.split(" ")[1].toString());
              for (String word : words22.split(" ")) {
                words21.add(word);
              }

              context.getCounter(COUNTER.NB_COMPARISIONS_II).increment(1);
              double sim = jaccardsim(words11,
                  words21);
              System.out.println("************");
              if (sim >= 0.1) {
                System.out.println(pair.split(" ")[0]);
                  context.write(new Text("(" + pair.split(" ")[0] + ", "
                          + pair.split(" ")[1] + ")"),
                          new Text(String.valueOf(sim)));
              }
          }
      }
  }
}
```

2.2.3 Driver

The driver of this second approach has the same structure as the first approach.

```
public class Setsimjoins2 {

  public static enum COUNTER {
      NB_COMPARISIONS_II,
  };

  public static void main(String[] args) throws IOException,
   ClassNotFoundException, InterruptedException {
    /*
       * Validate that two arguments were passed from the command line.
       */
      if (args.length != 2) {
        System.out.printf("Usage: StopWords <input dir> <output dir>\n");
        System.exit(-1);
      }

      /*
       * Instantiate a Job object for your job's configuration.
       */
      Configuration conf = new Configuration();
      Job job = new Job(conf);
      job.setJarByClass(Setsimjoins.class);
      job.setJobName("Part2_Set Similarity Joins_2nd method");

      FileInputFormat.addInputPath(job, new Path(args[0]));
      FileOutputFormat.setOutputPath(job, new Path(args[1]));

      job.setMapperClass(Setsimjoins2Mapper.class);
      job.setReducerClass(Setsimjoins2Reducer.class);


      job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(Text.class);
```

```java

      job.setOutputKeyClass(Text.class);
      job.setOutputValueClass(Text.class);

      job.setOutputFormatClass(TextOutputFormat.class);


      FileSystem fs = FileSystem.get(new Configuration());
      if (fs.exists(new Path(args[1]))) {
        fs.delete(new Path(args[1]));
      }

      job.waitForCompletion(true);

      long counter = job.getCounters()
              .findCounter(COUNTER.NB_COMPARISIONS_II).getValue();
      Path outFile = new Path("NB_COMPARISIONS_II.txt");
      BufferedWriter br = new BufferedWriter(new OutputStreamWriter(
              fs.create(outFile, true)));
      br.write(String.valueOf(counter));
      br.close();

    System.exit(0);
  }
}
```



Job Overview

Job Name:	Part2_Set Similarity Joins
User Name:	cloudera
Queue:	root.cloudera
State:	SUCCEEDED
Uberized:	false
Submitted:	Fri Mar 17 08:53:09 PDT 2017
Started:	Fri Mar 17 08:53:40 PDT 2017
Finished:	Fri Mar 17 08:54:12 PDT 2017
Elapsed:	31sec
Diagnostics:	
Average Map Time	9sec
Average Shuffle Time	5sec
Average Merge Time	0sec
Average Reduce Time	2sec

ApplicationMaster

Attempt Number	Start Time	Node	Logs
1	Fri Mar 17 08:53:14 PDT 2017	quickstart.cloudera:8042	logs

Task Type	Total	Complete	
Map	1	1	
Reduce	1	1	
Attempt Type	Failed	Killed	Successful
Maps	0	0	1
Reduces	0	0	1

Figure 3: Logs in Yarn of the comparison job for the 2nd approach

2.3 Comparison of the two approaches

	————	NB of comparisons	Execution time
Result on sample text	1st method	258	31 sec
	2nd method	212	36 sec

As expected the second approach performs less comparisons than the 1st approach. However, the execution time of the 2nd method is larger than the 1st method. I think it is due to the fact that the comparison is carried on a short sample file and therefore it won't be a large difference in elapsed time in addition to that the elapsed time of a same job can fluctuate by few seconds, therefore we obtain a larger elapsed time.

3 Appendix

3.1 sortByValue code

```
    private static Map<String, Integer> sortByValue(Map<String, Integer>
    wordcount) {

      List<Map.Entry<String, Integer>> list =
                new LinkedList<Map.Entry<String, Integer>>(wordcount.entrySet
    ());

        Collections.sort(list, new Comparator<Map.Entry<String, Integer>>() {
            public int compare(Map.Entry<String, Integer> o1,
                               Map.Entry<String, Integer> o2) {
                return (o1.getValue()).compareTo(o2.getValue());
            }
        });

        Map<String, Integer> sortedMap = new LinkedHashMap<String, Integer>();
        for (Map.Entry<String, Integer> entry : list) {
            sortedMap.put(entry.getKey(), entry.getValue());
        }
        return sortedMap;
    }
```

3.2 Implementation of WritableComparable Class

```

public class DocPair implements WritableComparable<DocPair> {

  private Text key1;
    private Text key2;

    public DocPair(Text key1, Text key2) {
        set(key1, key2);
    }
    public DocPair() {
        set(new Text(), new Text());
    }

    public DocPair(String key1, String key2) {
        set(new Text(key1), new Text(key2));
    }

    public Text getFirst() {
        return key1;
    }

    public Text getSecond() {
        return key2;
    }

    public void set(Text key1, Text key2) {
```

```java
        this.key1 = key1;
        this.key2 = key2;
    }

    @Override
    public void readFields(DataInput in) throws IOException {
      key1.readFields(in);
      key2.readFields(in);
    }

    @Override
    public void write(DataOutput out) throws IOException {
      key1.write(out);
      key2.write(out);
    }

    @Override
    public String toString() {
        return key1 + " " + key2;
    }

    @Override
    public int compareTo(DocPair other) {
        int cmpFirstFirst = key1.compareTo(other.key1);
        int cmpSecondSecond = key2.compareTo(other.key2);
        int cmpFirstSecond = key2.compareTo(other.key2);
        int cmpSecondFirst = key2.compareTo(other.key1);

        if (cmpFirstFirst == 0 && cmpSecondSecond == 0 || cmpFirstSecond == 0
                && cmpSecondFirst == 0) {
            return 0;
        }

        Text thisSmaller;
        Text otherSmaller;

        Text thisBigger;
        Text otherBigger;

        if (this.key1.compareTo(this.key2) < 0) {
            thisSmaller = this.key1;
            thisBigger = this.key2;
        } else {
            thisSmaller = this.key2;
            thisBigger = this.key1;
        }

        if (other.key1.compareTo(other.key2) < 0) {
            otherSmaller = other.key1;
            otherBigger = other.key2;
        } else {
            otherSmaller = other.key2;
            otherBigger = other.key1;
        }

        int cmpThisSmallerOtherSmaller = thisSmaller.compareTo(otherSmaller);
        int cmpThisBiggerOtherBigger = thisBigger.compareTo(otherBigger);

        if (cmpThisSmallerOtherSmaller == 0) {
            return cmpThisBiggerOtherBigger;
        } else {
            return cmpThisSmallerOtherSmaller;
        }
    }
    @Override
    public int hashCode() {
        return key1.hashCode() * 163 + key2.hashCode();
    }

    @Override
    public boolean equals(Object o) {
        if (o instanceof DocPair) {
          DocPair tp = (DocPair) o;
            return key1.equals(tp.key1) && key2.equals(tp.key2);
        }
        return false;
    }
}
```

3.3 Jaccard similarity code

```
    public double jaccardsim(HashSet<String> v1, HashSet<String> v2) {

      HashSet<String> intersect1 = v1;
      intersect1.retainAll(v2);
      int intertsect = intersect1.size();

        if (v1.size() < v2.size()) {
          HashSet<String> unionSet = v1;
          unionSet.addAll(v2);
          int union = unionSet.size();
            return (double) intertsect / union;
        } else {
          HashSet<String> unionSet = v2;
            unionSet.addAll(v1);
            int union = unionSet.size();
            return (double) intertsect / union;
        }
}
```