Program – 4

AIM: Write a program for Stop word elimination in a large textual file.

Introduction & Theory

Stop Words

In computing, stop words are words which are filtered out before or after processing of natural language data (text). Though "stop words" usually refers to the most common words in a language, there is no single universal list of stop words used by all natural language processing tools, and indeed not all tools even use such a list. Some tools specifically avoid removing these stop words to support phrase search.

Any group of words can be chosen as the stop words for a given purpose. For some search engines, these are some of the most common, short function words, such as the, is, at, which, and on. In this case, stop words can cause problems when searching for phrases that include them, particularly in names such as "The Who", "The The", or "Take That".

Below are the steps for MapReduce data flow:

- **Step 1:** One block is processed by one **mapper** at a time. In the mapper, a developer can specify his own business logic as per the requirements. In this manner, Map runs on all the nodes of the cluster and process the data blocks in parallel.
- **Step 2:** Output of Mapper also known as intermediate output is written to the local disk. An output of mapper is not stored on HDFS as this is temporary data and **writing on HDFS** will create unnecessary many copies.
- **Step 3:** Output of mapper is shuffled to **reducer** node (which is a normal slave node but reduce phase will run here hence called as reducer node). The shuffling/copying is a physical movement of data which is done over the network.
- **Step 4:** Once all the mappers are finished and their output is shuffled on reducer nodes then this intermediate output is merged & sorted. Which is then provided as input to reduce phase.
- **Step 5:** Reduce is the second phase of processing where the user can specify his own custom business logic as per the requirements. An input to a reducer is provided from all the mappers. An output of reducer is the final output, which is written on HDFS.

Code

Mapper

```java
import java.io.BufferedReader;
import java.io.FileReader;
```

Program – 4

```java
import java.io.IOException;
import java.util.*;
import java.util.Iterator;
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 stopWordMapper extends Mapper<LongWritable, Text, Text,
IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();
    private Set StopWords = new HashSet();

    @Override
    protected void setup(Context context) throws IOException,
InterruptedException {
        readFile(stopWords.stopWordsFiles);
    }
    /**
     * map function of Mapper parent class takes a line of text at a
time splits to
     * tokens and passes to the context as word along with value as
one
     */
    @Override
    protected void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException {

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

        while (tokenizer.hasMoreTokens()) {
            String token = tokenizer.nextToken();
            if (StopWords.contains(token)) {

context.getCounter(stopWords.COUNTERS.STOPWORDS).increment(1L);
            } else {
context.getCounter(stopWords.COUNTERS.GOODWORDS).increment(1L);
                word.set(token);
                context.write(word, null);
            }
        }
    }
    private void readFile(Path filePath) {
        try {
            BufferedReader bufferedReader = new BufferedReader(new
FileReader(filePath.toString()));
            String stopWord = null;
            while ((stopWord = bufferedReader.readLine()) != null) {
                StopWords.add(stopWord.toLowerCase());
            }
            bufferedReader.close();
```

Program – 4

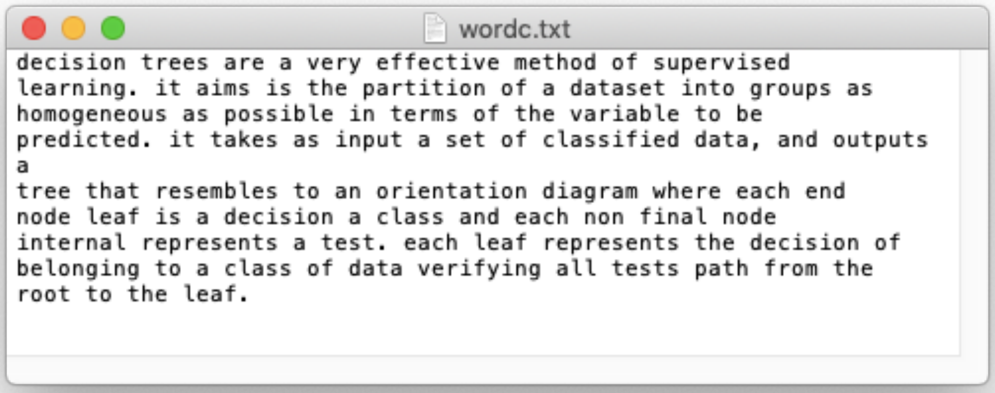
```
        } catch (IOException ex) {
            System.err.println("Exception while reading stop words 
file: " + ex.getMessage());
        }
    }
}
```

Main

```
import java.util.ArrayList;
import java.util.List;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.filecache.DistributedCache;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Counters;
import org.apache.hadoop.mapreduce.Job;
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;
import org.apache.hadoop.util.GenericOptionsParser;

@SuppressWarnings("deprecation")
public class stopWords {

    public static Path stopWordsFiles;

    public enum COUNTERS {
        STOPWORDS, GOODWORDS
    }
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        GenericOptionsParser parser = new GenericOptionsParser(conf, 
args);
        args = parser.getRemainingArgs();
        Job job = new Job(conf, "StopWordSkipper");
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        job.setJarByClass(stopWords.class);
        job.setMapperClass(stopWordMapper.class);
        job.setNumReduceTasks(0);
        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);
        stopWordsFiles = new Path(args[2]);
        FileInputFormat.setInputPaths(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        job.waitForCompletion(true);
        Counters counters = job.getCounters();
        System.out.printf("Good Words: %d, Stop Words: %d\n", 
counters.findCounter(COUNTERS.GOODWORDS).getValue(), 
            counters.findCounter(COUNTERS.STOPWORDS).getValue());
    }
}
```

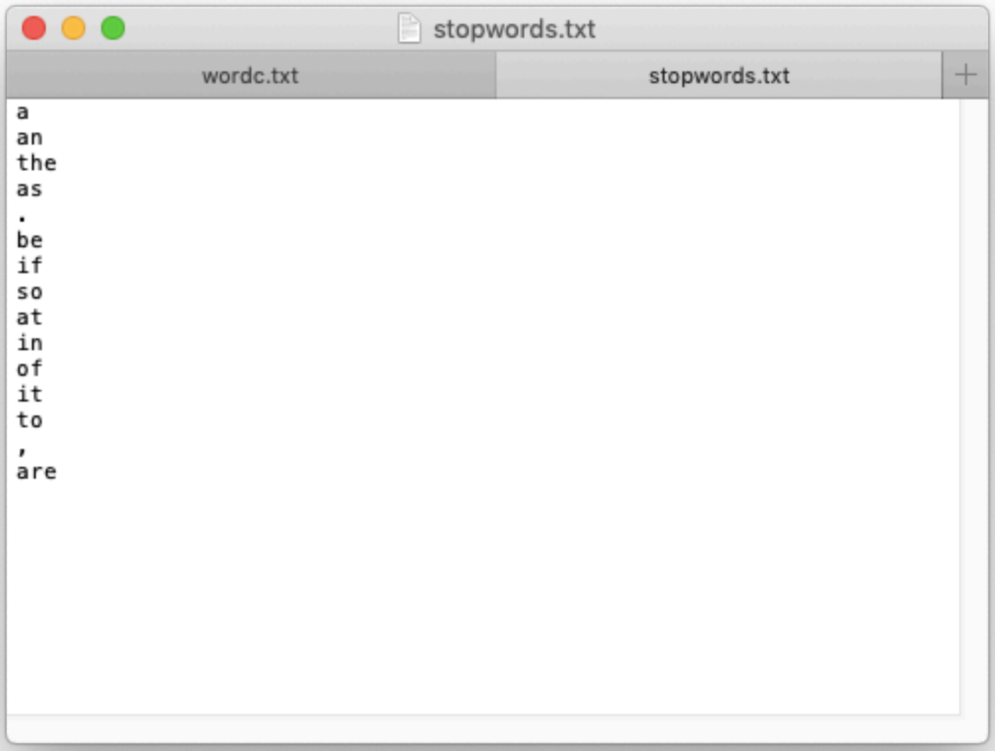
Program – 4

Output



wordc.txt

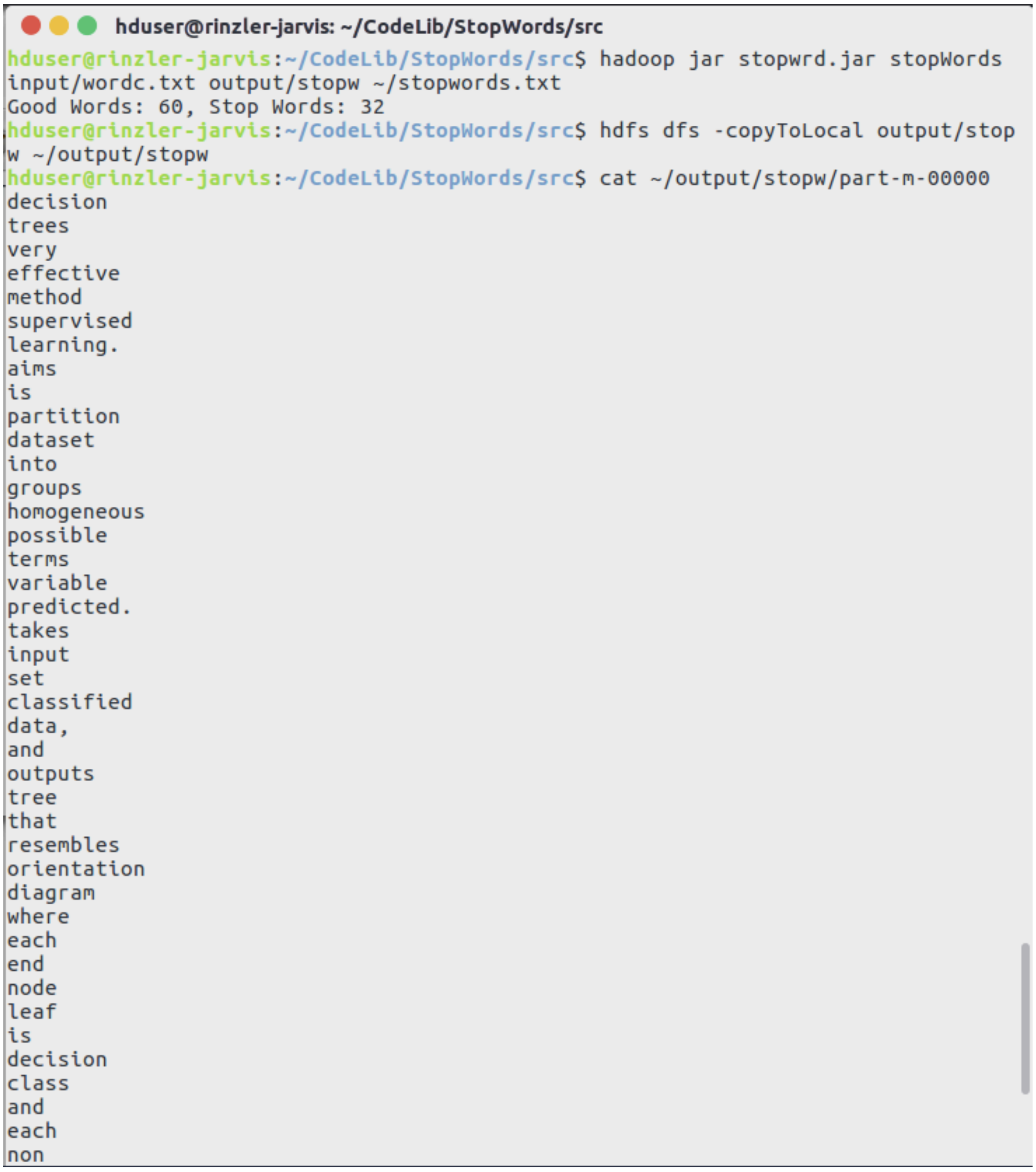
```
decision trees are a very effective method of supervised
learning. it aims is the partition of a dataset into groups as
homogeneous as possible in terms of the variable to be
predicted. it takes as input a set of classified data, and outputs
a
tree that resembles to an orientation diagram where each end
node leaf is a decision a class and each non final node
internal represents a test. each leaf represents the decision of
belonging to a class of data verifying all tests path from the
root to the leaf.
```



stopwords.txt

```
a
an
the
as
.
be
if
so
at
in
of
it
to
,
are
```

Program – 4



```
hduser@rinzler-jarvis: ~/CodeLib/StopWords/src
hduser@rinzler-jarvis:~/CodeLib/StopWords/src$ hadoop jar stopwrd.jar stopWords
input/wordc.txt output/stopw ~/stopwords.txt
Good Words: 60, Stop Words: 32
hduser@rinzler-jarvis:~/CodeLib/StopWords/src$ hdfs dfs -copyToLocal output/stop
w ~/output/stopw
hduser@rinzler-jarvis:~/CodeLib/StopWords/src$ cat ~/output/stopw/part-m-00000
decision
trees
very
effective
method
supervised
learning.
aims
is
partition
dataset
into
groups
homogeneous
possible
terms
variable
predicted.
takes
input
set
classified
data,
and
outputs
tree
that
resembles
orientation
diagram
where
each
end
node
leaf
is
decision
class
and
each
non
```

Findings and Learnings:

1. We learned how map-reduce works.
2. We learned how to code using hadoop in Java.
3. We learned how input and output in hadoop works.
4. We learned stop word removal in hadoop.