Assignment 2

Quentin Moret

Set similarity join

Introduction

In this assignment, we implement two algorithms - one naïve and one more optimized - outputting similar pair from a set of documents. This may have various application for example collaborative filtering, plagiarism detection, data deduplication, nearest neighbor searches, etc. It is an algorithmically simple problem, but very computationnaly expensive. We will see how performance can be improved.

1. Hadoop setup

Hadoop is configured in pseudo-distributed mode on MacOS 10.12, using version 2.7.3. Native libraries have been installed:

```
Hadoop 2.7.3
Subversion https://git-wip-us.apache.org/repos/asf/hadoop.git -r baa91f7c6bc9cb9
2be5982de4719c1c8af91ccff
Compiled by root on 2016-08-18T01:41Z
Compiled with protoc 2.5.0
From source with checksum 2e4ce5f957ea4db193bce3734ff29ff4
This command was run using /usr/local/Cellar/hadoop/2.7.3/libexec/share/hadoop/c
ommon/hadoop-common-2.7.3.jar
```

```
Native library checking:
hadoop:  true /usr/local/Cellar/hadoop/2.7.3/lib/libhadoop.dylib
zlib:    true /usr/lib/libz.1.dylib
snappy:  true /usr/local/lib/libsnappy.1.dylib
lz4:     true revision:99
bzip2:   false
openssl: false build does not support openssl.
```

Java files are compiled in the shell (using *–cp* command including Hadoop class path), then Jar file is created, and executed in Hadoop. Eventually, the output file is copied to local storage.

Example for PreProcessing:

```
$ javac PreProcessing.java -cp $(hadoop classpath);
$ jar cf pp.jar PreProcessing*.class;
$ hdfs dfs -rm -r PreProcessing/output;
$ hadoop jar pp.jar PreProcessing WordCount/input/pg100.txt PreProcessing/output
StopWords/stopwords.csv WordCount/output/part-r-00000;
$ rm out.txt;
$ hdfs dfs -get PreProcessing/output/part-r-00000 out.txt
```

2. Pre-processing

Pre-processing here consists in removing stop words, special characters, empty line. We then consider each line as a value, and each line number as a key. We keep only unique words, and order them as a function of their global frequency

Input : Data is uploaded on HDFS using bash command

```
$ hdfs dfs –put input input
```

We declare special characters as delimiters to remove them. We only pass to the reducers values of non-empty lines. Stop words are taken from the last assignment : we get the file, store all of the stop words in a string and then test in the mapper whether each word belongs to the stop words list.

We don't need the reducer to perform any action as there is only one (key, value) pair outputted by the mapper for each unique key (only one line corresponding to a unique line number).
However, for better readability we use the reducer to perform the following task: order the words of the sentence with their global frequency.
To do so, we use the output of the classic WordCount that we convert in the `setup()` function into a HashMap. In the `reduce()` function, we get the sub-HashMap which keys are those contained in the line processed, sort it according to values, and output thus sorted keys, then joined into a String.

The job is executed with commands as seen on the previous page.

Output extract :

```
1    EBook Complete Works Shakespeare William Gutenberg Project The
2    Shakespeare William
3    anyone anywhere eBook cost use This
4    restrictions whatsoever copy almost away give may You
5    included License Gutenberg Project terms under use
6    online gutenberg www eBook org
7    Details COPYRIGHTED Below eBook Gutenberg Project This
8    guidelines copyright file Please follow
9    Title Complete Works Shakespeare William The
10   Author Shakespeare William
11   September Posting 2011 Date EBook 100 1
12   1994 Release Date January
```

The words seem well sorted in ascending order of global frequency.

We also store the outputted number of line in HDFS, using a global variable lineNumber of type IntWritable incremented each time we write a line to the output, and get the following : `114880`

Implementation:
Mapper:

```
public static class PreProcessingMapper
         extends Mapper<Object, Text, IntWritable, Text>{
    private Text wordText = new Text();
    private String word = new String();
    String stopWords = "";
    protected void setup(Context context) throws IOException, InterruptedException {
        String stopWordsPath = context.getConfiguration().get("stopWords");
        Path ofile = new Path(stopWordsPath);
        FileSystem fs = FileSystem.get(new Configuration());
        BufferedReader br =new BufferedReader(new InputStreamReader(fs.open(ofile)));
        String line;
        line=br.readLine();
        try{
            while (line != null){
                stopWords += line + ",";
                line = br.readLine();
                }
        } finally {
          br.close();
        }
    }
    public void map(Object key, Text value, Context context
                                    ) throws IOException, InterruptedException {
        String inLine = value.toString();
        String outLine = "";
        StringTokenizer itr = new StringTokenizer(inLine ,
"\t\n\r&\\_!.;,()\"\'/:+=$[]§?#*|{}~<>@`");
        while (itr.hasMoreTokens()) {
            wordText.set(itr.nextToken());
            word = wordText.toString();
            if (!stopWords.contains(word) && !outLine.contains(word)){
                outLine = outLine + word + " ";
            }
        }
        if (outLine.length()>0) {
            lineNumber.set(lineNumber.get()+1);
            outLine = outLine.substring(0, outLine.length()-1);
            Text outLineText = new Text(outLine);
            context.write(lineNumber, outLineText);
        }
    }
}
```

Reducer

```
public static class OrderReducer
          extends Reducer<IntWritable,Text,IntWritable,Text> {

    private HashMap<String, Integer> hm = new HashMap<String, Integer>();
    protected void setup(Context context) throws IOException, InterruptedException {
        String wordCountPath = context.getConfiguration().get("wordCount");
        Path ofile = new Path(wordCountPath);
        FileSystem fs = FileSystem.get(new Configuration());
        BufferedReader br=new BufferedReader(new InputStreamReader(fs.open(ofile)));
        String line;
        line=br.readLine();
        try{
            while (line != null){
```

```
                String [] keyvalue = line.split("\t");
                String word = keyvalue[0];
                int count = Integer.parseInt(keyvalue[1]);
                hm.put(word, count);
                line = br.readLine();
            }
        } finally {
          br.close();
        }
    }
    public void reduce(IntWritable key, Iterable<Text> values, Context context
                       ) throws IOException, InterruptedException{
        for (Text value : values){
         // Get line's words and associated counts
            String line = value.toString();
            StringTokenizer itr = new StringTokenizer(line);
            HashMap<String, Integer> subhm = new HashMap<String, Integer>();
            while (itr.hasMoreTokens()) {
                String word = itr.nextToken();
                subhm.put(word, hm.get(word));
            }
          // Order as a function of count
            List<String> mapKeys = new ArrayList<String>(subhm.keySet());
            List<Integer> mapValues = new ArrayList<Integer>(subhm.values());
            Collections.sort(mapValues);
            LinkedList<String> list = new LinkedList<String>();

            Iterator<Integer> valueIt = mapValues.iterator();
            while (valueIt.hasNext()) {
                Integer val = valueIt.next();
                Iterator<String> keyIt = mapKeys.iterator();
                while (keyIt.hasNext()) {
                    String sortedword = keyIt.next();
                    Integer comp1 = subhm.get(sortedword);
                    Integer comp2 = val;
                    if (comp1.equals(comp2)) {
                        keyIt.remove();
                        list.add(sortedword);
                    }
                }
            }
            String sortedString  = String.join(" ", list);
            Text result = new Text(sortedString);
            context.write(key, result);
        }
    }
}
```

Main

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

    Configuration conf = new Configuration();
    conf.set("stopWords", args[2]);
    conf.set("wordCount", args[3]);

    Job job = Job.getInstance(conf, "PreProcessing");
    job.setJarByClass(PreProcessing.class);
    job.setMapperClass(PreProcessingMapper.class);
    job.setReducerClass(OrderReducer.class);
    job.setOutputKeyClass(IntWritable.class);
```

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

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

    job.waitForCompletion(true);

    Path pt = new Path(
"hdfs://localhost:9000/user/quentin/PreProcessing/counter");
    FileSystem fs = pt.getFileSystem(conf);
    BufferedWriter bw=new BufferedWriter(new OutputStreamWriter(fs.create(pt,true)));
     bw.write("Number of remaining lines after preproccessing : " +
     String.valueOf(lineNumber) + "\n");
    bw.close();
}
```

Execution time

```
real    0m6.870s
user    0m13.212s
sys     0m0.853s
```

Output extract:

```
4389,4710  PAROLLES // PAROLLES // sim = 1.0
4413,4939  Exeunt // Exeunt // sim = 1.0
4414,4997  V ACT 2 SCENE // IV ACT 2 SCENE // sim = 0.8
4415,4808  WIDOW S house The // Florence WIDOW S house The // sim = 0.8
4467,4472  Bequeathed ancestors many // Bequeathed ancestors many // sim = 1.0
4468,4473  obloquy greatest world Which // obloquy greatest world Which // sim = 1.0
4493,4900  Exit // Exit // sim = 1.0
4634,4640  SOLDIER set Well FIRST // SOLDIER set Well FIRST // sim = 1.0
4716,4739  cat // cat // sim = 1.0
4809,4950  WIDOW DIANA HELENA Enter // ATTENDANTS WIDOW DIANA HELENA Enter // sim = 0.8
671,1571   sort such love But thee I // sort such love But thee I // sim = 1.0
672,1572   report being mine As // report being mine As // sim = 1.0
```

Set similarity joins

a) Pairwise comparisons

The goal of this algorithm is to output similar sentences from our preprocessed document. The first naïve approach to this problem is to compare all pairwise comparison.

To do so, in the mapper, for each document we loop over all the other documents, and output to the reducer each (document id, other document id) as a key, and document value as value.
More specifically, we first get the line number being treated in the mapper (that we will call line_i and then loop over all other keys (`line_j`) with a basic for loop bounded by the total number of lines that we calculated in the preprocessing step. The implemented key was:

"`line_i,line_j`" if `line_i < line_j`
"`line_j,line_i`" if `line_i > line_j`

so that the key is unique to each pair.

In the reducer, we get the two values associated to each key, and compute the Jaccard similarity. We only keep pairs whose similarity is greater than a given threshold: `0.8`.

The output format is as follows :

Key : the same as before
Value : content of line_i, content of line_j // sim = sim

Since the calculation requires O(`totalLinesNumber`^{2}) items outputted to the reducer, and as many comparisons performed, it is extremely computationally expensive. Since my machine could not handle that, I bounded the number of lines processed to 5000.

We run the jar file with as input the output of the preprocessing and the number of lines:

```
hadoop jar pw.jar PairWise PreProcessing/output PairWise/output
PreProcessing/lineNumber;
```

With this configuration, the execution time is:

```
real    1m53.403s
user    1m43.011s
sys     0m16.175s
```

The exact number of operations should be n*(n-1)/2, which gives 12 497 500 for n=10000. We check with a counter that we increment in the reducer each time a comparison is performed, and obtain the right number:

```
PairWise$counters
        NUMBER_OF_COMPARISONS=12497500
```

Implementation

Mapper

```
public static class PairWiseMapper
         extends Mapper<Object, Text, Text, Text>{

    private int totalLineNumber;

    private int maxLine = 5000;

    private IntWritable lineNumber = new IntWritable(0);

    public void setup(Context context) throws IOException, InterruptedException {
        String lineNumberPath = context.getConfiguration().get("lineNumber");
        Path ofile = new Path(lineNumberPath);
        FileSystem fs = FileSystem.get(new Configuration());
        BufferedReader br =new BufferedReader(new InputStreamReader(fs.open(ofile)));
        String totalLineNumberString = br.readLine();
        totalLineNumber = Integer.parseInt(totalLineNumberString);
        br.close();
        // Bound its value to consider only a subset and reduce calculation
        totalLineNumber = Math.min(totalLineNumber, maxLine);
    }

    public void map(Object key, Text value, Context context
                                    ) throws IOException, InterruptedException {
        lineNumber.set(lineNumber.get()+1);
        if (lineNumber.get() < maxLine){
            String [] inLine = value.toString().split("\t");
            String lineNumberString = inLine[0];
            int lineNumber = Integer.parseInt(inLine[0]);
            String line = inLine[1];
            Text lineText = new Text(line);

            // Generate pairs of treated line with all other lines
            for (int otherLine = 1; otherLine<=totalLineNumber; otherLine++){
                String newKey = new String();
                String otherLineString = String.valueOf(otherLine);
                if (!(otherLine == lineNumber)) {
                    if (lineNumber < otherLine){
                        newKey = lineNumberString + "," + otherLineString;
                    } else {
                        newKey = otherLineString + "," + lineNumberString;
                    }
                    if (newKey.isEmpty()){
                        System.out.println("Vide !");
                        System.out.println(lineNumberString);
                        System.out.println(otherLineString);
                    }
                    Text newKeyText = new Text(newKey);
                    context.write(newKeyText, lineText);
                }
            }
        }
    }
}
```

Reducer

```
public static class ComparisonReducer
         extends Reducer<Text,Text,Text,Text> {

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

        // Put all lines associated with the key in an array
        List<String> valuesArray = new ArrayList<String>(2);
        for (Text value : values){
            valuesArray.add(value.toString());
        }

        // Make sure that the same pair of documents is compared no more than once
        if (valuesArray.size() == 2){
            String s1 = valuesArray.get(0);
            String s2 = valuesArray.get(1);
            String [] v1 = s1.split(" ");
            String [] v2 = s2.split(" ");

            // Jaccard similarity
            int inter = 0;
            int union = 0;

            for (String word : v1){
                if (s2.contains(word)){
                    inter++;
                    union++;
                }
                else
                    union++;
            }

            for (String word : v2){
                if (!(s1.contains(word)))
                    union++;
            }

            float sim = (float) inter / union;

            // Output pairs that are more similar than a given threshold
            if (sim > 0.8){
                String out = s1 + " // " + s2 + " // sim = " + String.valueOf(sim);
                Text value = new Text(out);
                context.write(key, value);
            }
            context.getCounter(counters.NUMBER_OF_COMPARISONS).increment(1);
        }
    }
}
```

Output extract :

```
4389,4710 PAROLLES // PAROLLES // sim = 1.0
4413,4939 Exeunt // Exeunt // sim = 1.0
4415,4808 WIDOW S house The // Florence WIDOW S house The // sim = 0.8
4467,4472 Bequeathed ancestors many // Bequeathed ancestors many // sim = 1.0
4468,4473 obloquy greatest world Which // obloquy greatest world Which // sim = 1.0
4493,4900 Exit // Exit // sim = 1.0
4634,4640 SOLDIER set Well FIRST // SOLDIER set Well FIRST // sim = 1.0
4716,4739 cat // cat // sim = 1.0
4809,4950 ATTENDANTS WIDOW DIANA HELENA Enter // WIDOW DIANA HELENA Enter // sim = 0.8
671,1571  sort such love But thee I // sort such love But thee I // sim = 1.0
672,1572  report being mine As // report being mine As // sim = 1.0
```

b) Using the prefix filtering principle

In order to overcome the complexity issue, the algorithm we implement consists in considering only the first $|d| - \lceil t * |d| \rceil + 1$ words of each line, where d is the number of words in the line, and compute an inverted index for these words.

For each of the first words of the processed line (`line_i`), thanks to the inverted index, we get a list of lines to compare to. We finally add these lines to a set of unique lines to compare.

For each other line (`line_j`) in the obtained set, we thus generate a key in the same way as in the PairWise class:

"line_i,line_j" if line_i < line_j
"line_j,line_i" if line_i > line_j

The reducer function is the same as in the PairWise class.

In order to compare with the PairWise implementation, we still only use 5000 lines of the preprocessed input. The number of comparisons thus performed is calculated by the same counter:

```
SetSimilarityJoins$counters
        NUMBER_OF_COMPARISONS=13856
```

Execution time :

```
real    0m5.938s
user    0m9.557s
sys     0m0.626s
```

The job is executed by first processing the inverted index :

```
$ hdfs dfs –mkdir InvertedIndex2
$ cd InvertedIndex/
$ javac InvertedIndex.java –cp $(hadoop classpath);
$ jar cf ii.jar InvertedIndex*.class;
$ hadoop jar ii.jar InvertedIndex PreProcessing/output InvertedIndex2/output;
```

Then executing the similarity join using the inverted index as argument and the PreProcessing output as input :

```
$ hdfs dfs –mkdir SetSimilarityJoin
$ cd SetSimilarityJoin/
$ javac SetSimilarityJoins.java –cp $(hadoop classpath);
$ jar cf ssj.jar SetSimilarityJoins*.class;
$    hadoop    jar    ssj.jar    SetSimilarityJoins    PreProcessing/output
SetSimilarityJoins/output InvertedIndex2/output/part-r-00000;
```

Implementation

Mapper

```
public static class SetSimilarityJoinsMapper
         extends Mapper<Object, Text, Text, Text>{

    private int maxLine = 5000;
    private IntWritable lineNumber = new IntWritable(0);

    private HashMap<String, String> hm = new HashMap<String, String>();
    protected void setup(Context context) throws IOException, InterruptedException{
        // Store the inverted index in a HashMap
        String IIPath = context.getConfiguration().get("InvertedIndex");
        Path ofile = new Path(IIPath);
        FileSystem fs = FileSystem.get(new Configuration());
        BufferedReader br =new BufferedReader(new InputStreamReader(fs.open(ofile)));
        String line;
        line=br.readLine();
        try{
            while (line != null){
                String [] keyvalue = line.split("\t");
                String word = keyvalue[0];
                String indexes = keyvalue[1];
                hm.put(word, indexes);
                line = br.readLine();
            }
        } finally {
          br.close();
        }
    }

    public void map(Object key, Text value, Context context
                                    ) throws IOException, InterruptedException {
        lineNumber.set(lineNumber.get()+1);
        if (lineNumber.get() < maxLine){
            String [] inLine = value.toString().split("\t");
            String lineNumberString = inLine[0];
            int lineNumber = Integer.parseInt(inLine[0]);
            String [] line = inLine[1].split(" ");
            Text lineText = new Text(inLine[1]);


            // Compute the number of words to consider
            int d = line.length;
            int td = (int) Math.ceil(0.8*d);
            int firstWords = d - td + 1;

            Set<String> linesToCompare = new HashSet<String>() ;

            // Loop on the words to consider and get lines associated
            // in the inverted index
            for (int i = 0; i<firstWords; i++){
                String word = line[i];
                String [] otherLines = hm.get(word).split(", ");
                for (String otherLine:otherLines){
                    int otherLineInt = Integer.parseInt(otherLine);
                    if (otherLineInt < maxLine)
                       linesToCompare.add(otherLine);
                }
            }
```

```
            // Create obtained pairs keys and output them to the reducer
            for(String otherLineString : linesToCompare){
                int otherLine = Integer.parseInt(otherLineString);
                String newKey = new String();
                if (!(otherLine == lineNumber)) {
                    if (lineNumber < otherLine){
                        newKey = lineNumberString + "," + otherLineString;
                    } else {
                        newKey = otherLineString + "," + lineNumberString;
                    }
                    Text newKeyText = new Text(newKey);
                    context.write(newKeyText, value);
                }

            }
        }
    }
}
```

Reducer

```
public static class ComparisonReducer
         extends Reducer<Text,Text,Text,Text> {

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

        List<String> valuesArray = new ArrayList<String>(2);
        for (Text value : values){
            valuesArray.add(value.toString());
        }
        if (valuesArray.size() == 2){
            String s1 = valuesArray.get(0);
            String s2 = valuesArray.get(1);
            String [] v1 = s1.split(" ");
            String [] v2 = s2.split(" ");

            int inter = 0;
            int union = 0;

            for (String word : v1){
                if (s2.contains(word)){
                    inter++;
                    union++;
                }
                else
                    union++;
            }

            for (String word : v2){
                if (!(s1.contains(word)))
                    union++;
            }

            float sim = (float) inter / union;

            if (sim > 0.8){
                String out = s1 + " // " + s2 + " // sim = " + String.valueOf(sim);
                Text value = new Text(out);
                context.write(key, value);
            }
            context.getCounter(counters.NUMBER_OF_COMPARISONS).increment(1);
        }
    }
}
```

Main

```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    conf.set("InvertedIndex", args[2]);
    Job job = Job.getInstance(conf, "SetSimilarityJoins");
    job.setJarByClass(SetSimilarityJoins.class);
    job.setMapperClass(SetSimilarityJoinsMapper.class);
    job.setReducerClass(ComparisonReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(Text.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    job.waitForCompletion(true);
}
```

Output extract

```
4319,4947MEMBERSHIP DOWNLOAD SERVICE CHARGES THAT TIME FOR // MEMBERSHIP DOWNLOAD
SERVICE CHARGES THAT TIME FOR // sim = 1.0
4389,4710PAROLLES // PAROLLES // sim = 1.0
4413,4939Exeunt // Exeunt // sim = 1.0
4415,4808WIDOW S house The // Florence WIDOW S house The // sim = 0.8
4467,4472Bequeathed ancestors many // Bequeathed ancestors many // sim = 1.0
4468,4473obloquy greatest world Which // obloquy greatest world Which // sim = 1.0
4493,4900Exit // Exit // sim = 1.0
4634,4640SOLDIER set Well FIRST // SOLDIER set Well FIRST // sim = 1.0
4716,4739cat // cat // sim = 1.0
4809,4950ATTENDANTS WIDOW DIANA HELENA Enter // WIDOW DIANA HELENA Enter // sim = 0.8
671,1571 sort such love But thee I // sort such love But thee I // sim = 1.0
672,1572 report being mine As // report being mine As // sim = 1.0
```

c) Compare both algorithms in the number of performed comparisons, as well as their difference in execution time

We notice a drastic drop in the number of comparison performed when using the prefix filtering principle.
Indeed only perform comparison to a very small subset of the pairs :
13856 instead of 12 497 500, which is a ratio of 0.1%.

The idea behind the prefix filtering principle is to compare only pairs which are likely to have a similarity greater than the threshold, which are pairs that share some of their least commons words. We actually consider only the n first least common words of each lines, where n depends on the number of unique words, and the similarity threshold that we consider.

In the pairwise method, we compare every pair, which means that a lot of the comparisons are done though very few words are in common.

	Number of comparisons performed	Number of pairs more similar than threshold
PairWise	12497500	319
Prefix filtering	13856	317

As it is implemented, the prefix filtering method does not allow us to retrieve all pairs, two are missing.
One of these is the pair

```
4414,4997   IV ACT 2 SCENE // V ACT 2 SCENE // sim = 0.8
```

the number of words for both lines is 4, which leads to only 1 (4 - ⌈4*0.8⌉ + 1) word kept, which is the first, which they do not have in common. Either this comes from a lack of preprocessing, or it show a limit to the approach. Anyway it is only 0.6% of all pairs, which is quite negligible.

The gain in number of comparison also comes at a cost : in this implementation we need to keep in memory for all workers a full copy of the invertexd index. Depending on the size of the corpus, it can be come memory expensive. We also need to add a first step of computing the inverted index.

Nonetheless, the gain in perfomance: from 1m53s to 5.94s is truly remarkable and justify some additional memory cost and recall tradeoff

Conclusion

In this assignment, we understood the importance of two major points in massive data processing:

1. The pre-processing : it proves to have a great influence on the algorithmic steps then performed, and allow one to implement these algorithm with a lot more ease.
2. The impact of computation optimization: between two algorithms implementing the same task, we can get tremendous performance differences at a low cost of precision.

Concerning the Java implementation, having some basic understanding of Hadoop helped a lot. We see that there is room for improvement in data manipulation: how to define variables in an optimal way, how to define custom classes so that you can manipulate your data more easily… For example for managing the key of the pairs through a proper class. Other next step would be to define a better workflow, by redesigning packages and classes. Nevertheless we see that with a basic understanding of Java and Hadoop we get our prototypes functional quite easily.

Sources

1. Searches on www.stackoverflow.com
2. S. Chaudhuri, V. Ganti, and R. Kaushik. A primitive operator for similarity joins in data cleaning, In Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006, 3-8 April 2006, Atlanta, GA, USA, page 5, 2006.
3. Tom White, Hadoop: The Definitive Guide, 4th Edition, O'Reilly Media, March 2015
4. Hadoop documentation : http://hadoop.apache.org/docs/current/