University of CINCINNATI

BIG DATA ANALYTICS

CS 7070

ASSIGNMENT - 3

Submitted on:

03/08/2019

Submitted by

Siva Sai Krishna Paladugu

Problem Statement:

3. Now consider a "Term" to mean a 2-gram (two words occurring sequentially) in a document. Modify the programs given to you to compute the TFIDF for each Term (2-gram). Submit the following items:

a.) List of top 15 "Terms" (2-grams) for each document, having the highest TFIDF values. The task of selecting the top 15 terms does not need to be done by the MapReduce program.

Code:

- Used same code provided in the zip and changed some code in mapper in phase1 as below. Also, used same standalone java code which is submitted in assignment 2 but with 2 changes (while reading output files of Phase1 and 3) as mentioned below to calculate TFIDF based on output files of above step.
- Putting only changed code here. (Entire code base has been attached as zip for reference).

Mapper Code (Changed part in phase 1):

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

            public void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException
            {
                String doc = value.toString();
                String docPart[] = doc.split(" "); //spliting input string to get
individual words
                String docName = docPart[0]; //getting the document number or the
document name
                String tempStr=""; //temp string to construct the key part

                String first = "";
                String second = "";

                if(docPart.length>0){
                    first = docPart[1].replaceAll("\\p{P}", "");
                  }
                //loop to collect all the words
                //for loop counter i is starting as we have first element of each
line as document number and doc as first word
                for(int i=2;i<docPart.length;i++)
                {
                        second = docPart[i].replaceAll("\\p{P}", ""); //removing
special character and punctuation from the word
                        tempStr = first+' '+second+","+docName;
                        word.set(tempStr);//converting string to text writable
                        context.write(word,one);
                        first=second;
                }
            }
    }
```

Standalone Java Code to calculate TFIDF:

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.Comparator;
import java.util.HashMap;
import java.util.Map;
import java.util.StringTokenizer;
import java.util.TreeMap;

public class CalTdIdf {
        //Method to sort Map by VALUE
        public static <K, V extends Comparable<V>> Map<K, V>
sortByValues(final Map<K, V> map) {
        Comparator<K> valueComparator = new Comparator<K>() {
                public int compare(K k1, K k2) {
                        int compare =map.get(k2).compareTo(map.get(k1));
                                if (compare == 0)
                                        return 1;
                                else
                                        return compare;
                }
        };

        Map<K, V> sortedByValues = new TreeMap<K, V>(valueComparator);
        sortedByValues.putAll(map);
        return sortedByValues;
    }
        //Method to sort Map by KEY
        public static void sortbykey(Map<String, TreeMap<String, Double>> map)
        {
                TreeMap<String, TreeMap<String, Double>> sorted = new
                                                                TreeMap<>();
                sorted.putAll(map);
                for (Map.Entry<String, TreeMap<String, Double>> entry:
                                                        sorted.entrySet())
                        for (Map.Entry<String, Double> child:
                                                entry.getValue().entrySet())
                                System.out.println(entry.getKey() + " " +
                                        child.getKey() + " " + child.getValue());
         }
        //Method to print top 15 words with highest tfidf in each doc
        public static void getTopFifteenSorted(Map<String, Map<String,
Double>> map) {
                TreeMap<String, Map<String, Double>> sorted = new TreeMap<>();
                sorted.putAll(map);
                for (Map.Entry<String, Map<String, Double>> entry:
                                                        sorted.entrySet()) {
                        int counter = 0;
                        System.out.println("The top Fifteen words with Highest
TF*IDF value in document: " + entry.getKey());
                    for (Map.Entry<String, Double> child:
```

```
                                                entry.getValue().entrySet()) {
                                counter = counter + 1;
                                if (counter <= 15) {
                                        System.out.println(child.getKey() + "
" + child.getValue());
                                }
                        }
                }
        }
//Method to change "term : (docName, TFIDF)" format to docName : (term,TFIDF)
        public static void interChangeDocNameTerm(Map<String, TreeMap<String,
Double>> map) {
                Map<String, TreeMap<String, Double>> fb = new HashMap<>();
                for (Map.Entry<String, TreeMap<String, Double>> entry:
                                                                map.entrySet()) {
                        for (Map.Entry<String, Double> child:
entry.getValue().entrySet()) {

                                TreeMap<String, Double> temp;
                                if (fb.containsKey(child.getKey())) {
                                        temp = fb.get(child.getKey());
                                        temp.put(entry.getKey(),
child.getValue());
                                } else {
                                        temp = new TreeMap<>();
                                        temp.put(entry.getKey(),
child.getValue());
                                        fb.put(child.getKey(), temp);
                                }
                        }
                }

                Map<String, Map<String, Double>> fbb = new HashMap<>();
                for (Map.Entry<String, TreeMap<String, Double>> entry :
fb.entrySet()) {
                        Map<String, Double> child =
sortByValues(entry.getValue());
                        fbb.put(entry.getKey(), child);
                }
                getTopFifteenSorted(fbb);
        }

        public static void main(String[] args) {
                BufferedReader br1 = null;
                BufferedReader br2 = null;
                BufferedReader br3 = null;

                Map<String, TreeMap<String, Double>> termtfIdf = new
HashMap<>();
                Map<String, Integer> docNameTotalTerms = new HashMap<>();
                Map<String, Double> termIdf = new HashMap<>();
                try {
                //Change path to output files of 3 phases for ques 1&2
```

```
                        br1 = new BufferedReader(new
FileReader("./Phase1Ques3OP/part-r-00000"));
                        br2 = new BufferedReader(new
FileReader("./Phase2Ques3OP/part-r-00000"));
                        br3 = new BufferedReader(new
FileReader("./Phase3Ques3OP/part-r-00000"));
                        String line1;
                        String line2;
                        String line3;

                        StringTokenizer word_list;
                        int totalDocs = 0;
                        //Calculating total documents
                        while ((line2 = br2.readLine()) != null) {
                                word_list = new StringTokenizer(line2);
                                String docName = word_list.nextToken();
                                int totalTermInDoc =
Integer.parseInt(word_list.nextToken());
                                docNameTotalTerms.put(docName, totalTermInDoc);
                                totalDocs += 1;
                        }
                        //Calculating IDF for each word
                        while ((line3 = br3.readLine()) != null) {
                                word_list = new StringTokenizer(line3);
                                String term = word_list.nextToken()+'
'+word_list.nextToken(); //Change for bigram
//String term = word_list.nextToken(); // Change for other than bigram
                                int docWithTerm =
Integer.parseInt(word_list.nextToken());
                                double tm = (double) totalDocs / docWithTerm;
                                double idf = Math.log(tm);
                                termIdf.put(term, idf);
                        }
//Calculating TF for each word in each doc and then TF*IDF
                        while ((line1 = br1.readLine()) != null) {
                                word_list = new StringTokenizer(line1);
                                String keyPhase1 = word_list.nextToken()+'
'+word_list.nextToken(); //Change for bigram
// String keyPhase1 = word_list.nextToken();//Change for other than bigram
                                String[] termDocName = keyPhase1.split(",");
                                String term = termDocName[0];
                                String docName = termDocName[1];
                                int tFrequency =
Integer.parseInt(word_list.nextToken());

                                TreeMap<String, Double> temp;
                                if (termtfIdf.containsKey(term)) {
                                        temp = termtfIdf.get(term);
                                        int totalTerms =
docNameTotalTerms.get(docName);
                                        double tf = (double) tFrequency /
totalTerms;
```

```
                                        double tfIdf = (double) tf *
termIdf.get(term);
                                        temp.put(docName, tfIdf);
                                } else {
                                        temp = new TreeMap<>();
                                        int totalTerms =
docNameTotalTerms.get(docName);
                                        double tf = (double) tFrequency /
totalTerms;
                                        double tfIdf = (double) tf *
termIdf.get(term);
                                        temp.put(docName, tfIdf);
                                        termtfIdf.put(term, temp);
                                }

                        }

                        System.out.println("All words TF*IDF values sorted by
terms of all documents");
                        sortbykey(termtfIdf);
                        System.out.println("Top Fifteen words TF*IDF values
sorted by tf*idf values in each document");

                        interChangeDocNameTerm(termtfIdf);
                        br1.close();
                        br2.close();
                        br3.close();
                } catch (IOException e) {
                        // TODO Auto-generated catch block
                        System.out.println("Erro Occured"+e.printStackTrace());
                }
        }
}
```

Formulae Used:

totalDocs (Total Number of documents) = *number of lines in phase2 output file*.
docWithTerm (Number of documents with term t in it) = *from phase 3 output file*.
Calculating IDF:
IDF = log(totalDocs / docWithTerm)
tFrequency (Number of times term t appears in a document) = *from phase1 output file*.
totalTerms (Total number of terms in the given document) = *from phase2 output file*.
Calculating TF:
TF = tFrequency / totalTerms
Calculating TF-IDF:
*tfidf = TF * IDF*

Results:

```
The top Fifteen words with Highest TF*IDF value in document: 0001
Bigrarm             --- TF*IDF
dummy text          --- 0.03981687709395678
1500s when          --- 0.01990843854697839
1960s with          --- 0.01990843854697839
Aldus PageMaker     --- 0.01990843854697839
Ipsum has           --- 0.01990843854697839
Ipsum passages      --- 0.01990843854697839
It was              --- 0.01990843854697839
Letraset sheets     --- 0.01990843854697839
PageMaker including --- 0.01990843854697839
a galley            --- 0.01990843854697839
a type              --- 0.01990843854697839
also the            --- 0.01990843854697839
an unknown          --- 0.01990843854697839
and more            --- 0.01990843854697839
and scrambled       --- 0.01990843854697839

The top Fifteen words with Highest TF*IDF value in document: 0002
Bigrarm             --- TF*IDF
45 BC               --- 0.02799624170668836
comes from          --- 0.02799624170668836
from a              --- 0.02799624170668836
of the              --- 0.017165817010439215
11033 of            --- 0.01399812085334418
2000 years          --- 0.01399812085334418
BC This             --- 0.01399812085334418
BC making           --- 0.01399812085334418
Cicero written      --- 0.01399812085334418
College in          --- 0.01399812085334418
Contrary to         --- 0.01399812085334418
Evil by             --- 0.01399812085334418
Extremes of         --- 0.01399812085334418
Good and            --- 0.01399812085334418
HampdenSydney College --- 0.01399812085334418

The top Fifteen words with Highest TF*IDF value in document: 0003
Bigrarm             --- TF*IDF
11033 from          --- 0.038951292809305545
1500s is            --- 0.038951292809305545
1914 translation    --- 0.038951292809305545
Cicero are          --- 0.038951292809305545
English versions    --- 0.038951292809305545
H Rackham           --- 0.038951292809305545
Ipsum used          --- 0.038951292809305545
Malorum by          --- 0.038951292809305545
Sections 11032      --- 0.038951292809305545
The standard        --- 0.038951292809305545
accompanied by      --- 0.038951292809305545
also reproduced     --- 0.038951292809305545
are also            --- 0.038951292809305545
below for           --- 0.038951292809305545
by English          --- 0.038951292809305545
```

```
The top Fifteen words with Highest TF*IDF value in document: 0004
Bigrarm             --- TF*IDF
Content here        --- 0.017395723002214123
English Many        --- 0.017395723002214123
Ipsum as            --- 0.017395723002214123
It is               --- 0.017395723002214123
Many desktop        --- 0.017395723002214123
The point           --- 0.017395723002214123
Various versions    --- 0.017395723002214123
a long              --- 0.017395723002214123
a moreorless        --- 0.017395723002214123
a page              --- 0.017395723002214123
a reader            --- 0.017395723002214123
a search            --- 0.017395723002214123
accident sometimes  --- 0.017395723002214123
and a               --- 0.017395723002214123
and the             --- 0.017395723002214123

The top Fifteen words with Highest TF*IDF value in document: 0005
Bigrarm             --- TF*IDF
humour or           --- 0.029862657820467584
the Internet        --- 0.029862657820467584
injected humour     --- 0.018310204811135163
on the              --- 0.018310204811135163
200 Latin           --- 0.014931328910233792
All the             --- 0.014931328910233792
If you              --- 0.014931328910233792
Internet It         --- 0.014931328910233792
Internet tend       --- 0.014931328910233792
Ipsum available     --- 0.014931328910233792
Ipsum generators    --- 0.014931328910233792
Ipsum which         --- 0.014931328910233792
Ipsum you           --- 0.014931328910233792
It uses             --- 0.014931328910233792
The generated       --- 0.014931328910233792
The top Fifteen words with Highest TF*IDF value in document: 0006
Bigrarm             --- TF*IDF
sit amet            --- 0.049561456631644046
Aliquam consectetur --- 0.018683508645022558
Cras aliquet        --- 0.018683508645022558
Cras eu             --- 0.018683508645022558
Curae Suspendisse   --- 0.018683508645022558
Donec feugiat       --- 0.018683508645022558
Donec imperdiet     --- 0.018683508645022558
Duis et             --- 0.018683508645022558
Duis magna          --- 0.018683508645022558
Nulla feugiat       --- 0.018683508645022558
Pellentesque malesuada --- 0.018683508645022558
Praesent tincidunt  --- 0.018683508645022558
Suspendisse vulputate --- 0.018683508645022558
Ut dictum           --- 0.018683508645022558
Vestibulum ante     --- 0.018683508645022558
```

```
The top Fifteen words with Highest TF*IDF value in document: 0007
Bigrarm             --- TF*IDF
Aliquam varius      --- 0.025356190303959185
Cras imperdiet      --- 0.025356190303959185
Curabitur elementum --- 0.025356190303959185
Donec euismod       --- 0.025356190303959185
Donec viverra       --- 0.025356190303959185
Maecenas sollicitudin --- 0.025356190303959185
Mauris ut           --- 0.025356190303959185
Nulla facilisi      --- 0.025356190303959185
Nunc justo          --- 0.025356190303959185
Sed non             --- 0.025356190303959185
Suspendisse pretium --- 0.025356190303959185
ac placerat         --- 0.025356190303959185
aliquet facilisis   --- 0.025356190303959185
amet mi             --- 0.025356190303959185
ante sed            --- 0.025356190303959185
The top Fifteen words with Highest TF*IDF value in document: 0008
Bigrarm             --- TF*IDF
dummy text          --- 0.03981687709395678
1500s when          --- 0.01990843854697839
1960s with          --- 0.01990843854697839
Aldus PageMaker     --- 0.01990843854697839
Ipsum has           --- 0.01990843854697839
Ipsum passages      --- 0.01990843854697839
It was              --- 0.01990843854697839
Letraset sheets     --- 0.01990843854697839
PageMaker including --- 0.01990843854697839
a galley            --- 0.01990843854697839
a type              --- 0.01990843854697839
also the            --- 0.01990843854697839
an unknown          --- 0.01990843854697839
and more            --- 0.01990843854697839
and scrambled       --- 0.01990843854697839
The top Fifteen words with Highest TF*IDF value in document: 0009
Bigrarm             --- TF*IDF
45 BC               --- 0.02799624170668836
comes from          --- 0.02799624170668836
from a              --- 0.02799624170668836
of the              --- 0.017165817010439215
11033 of            --- 0.01399812085334418
2000 years          --- 0.01399812085334418
BC This             --- 0.01399812085334418
BC making           --- 0.01399812085334418
Cicero written      --- 0.01399812085334418
College in          --- 0.01399812085334418
Contrary to         --- 0.01399812085334418
Evil by             --- 0.01399812085334418
Extremes of         --- 0.01399812085334418
Good and            --- 0.01399812085334418
HampdenSydney College --- 0.01399812085334418
```

The top Fifteen words with Highest TF*IDF value in document: 0010

Bigrarm	**TF*IDF**
11033 from	0.038951292809305545
1500s is	0.038951292809305545
1914 translation	0.038951292809305545
Cicero are	0.038951292809305545
English versions	0.038951292809305545
H Rackham	0.038951292809305545
Ipsum used	0.038951292809305545
Malorum by	0.038951292809305545
Sections 11032	0.038951292809305545
The standard	0.038951292809305545
accompanied by	0.038951292809305545
also reproduced	0.038951292809305545
are also	0.038951292809305545
below for	0.038951292809305545
by English	0.038951292809305545

The top Fifteen words with Highest TF*IDF value in document: 0011

Bigrarm	**TF*IDF**
Content here	0.017395723002214123
English Many	0.017395723002214123
Ipsum as	0.017395723002214123
It is	0.017395723002214123
Many desktop	0.017395723002214123
The point	0.017395723002214123
Various versions	0.017395723002214123
a long	0.017395723002214123
a moreorless	0.017395723002214123
a page	0.017395723002214123
a reader	0.017395723002214123
a search	0.017395723002214123
accident sometimes	0.017395723002214123
and a	0.017395723002214123
and the	0.017395723002214123

The top Fifteen words with Highest TF*IDF value in document: 0012

Bigrarm	**TF*IDF**
humour or	0.029862657820467584
the Internet	0.029862657820467584
injected humour	0.018310204811135163
on the	0.018310204811135163
200 Latin	0.014931328910233792
All the	0.014931328910233792
If you	0.014931328910233792
Internet It	0.014931328910233792
Internet tend	0.014931328910233792
Ipsum available	0.014931328910233792
Ipsum generators	0.014931328910233792
Ipsum which	0.014931328910233792
Ipsum you	0.014931328910233792
It uses	0.014931328910233792
The generated	0.014931328910233792

b. Which output – obtained in 3(a) or in 2(b) – better characterizes the documents? Give reasons for your answers.

- Here before comparing 3a and 2b, I have observed few things among outputs of 3a. Document 0001 and 0008 have same order and same terms in list of top 15 highest TF*IDF values. Similarly documents (0002,0009) (0003,0010) (0004,0011) (0005,0012) have same terms correspondingly.
- TF*IDF value for a particular word in 2b has more tf*idf value compared with the bigram words that contain that particular word. For ex, Donec in doc 0007 in 2b and Donec euismod, Donec viverra bigram words in doc 0007 in 3a.
- Also, understood reason for decrease in TFIDF values is as term frequency will be decreased linearly and idf (uniqueness) increases log linearly.
- Actually, choosing 3a or 2b for better characterizing the documents depends on specific requirements. For example, in output of 3a for doc 0001, "Aldus Pagemaker" appeared as one word which indicates more information compared with "Pagemaker" which appeared in output of 2b for doc 0002. Similarly, in case of doc 0009 the term "HampdenSydney College" is giving more specific information about word.
- Also, bigrams like "and a" and "and the" in output of 3a in doc 0011 are not giving much information which are not necessary to be with top tfidf values. But for same doc 0011, output of 2a is not having those words so helping in removing the un-necessary information. **So, I think combination of 2b and 3a will give best results for characterizing the documents. Selecting one from 2b or 3a for better characterizing will entirely depends on the specific requirements we need. If more particular information from words is needed then bigram/3a is preferred and if we want to remove more unnecessary words we can use 2b.**

Problem Statement:

4. Once your program is working for the above two parts, run the programs on a larger collection of documents (to be provided to you next week) and repeat the above two tasks. Discuss the results for 1(b), 2(b), and 3(a) in the context of the new set of documents.

Code:

- As new input documents (first 4 files) contain more special characters and special characters alone as words, I am doing preprocessing for **sentence (doc read by mapper)** instead for **word** which was done in given code as shown in below code. Also, converted all to **lowercase** to get more useful insights without redundancy.
- Used different mappers in all **phase one** to calculate three parts - 1(b), 2(b) and 3(a) with new input files.
- Used phase 2 and phase 3 as given in code zip for all three parts (1b,2b and 3a).
- Used same standalone program as in question 1 to calculate top 15 highest TF*IDF terms by changing path to new output files except change for reading bigram outputs as commented in code.
- Putting only changed code here. (Entire code base has been attached as zip for reference)

Mapper for Phase 1 for part 1 and part2 (To calculate outputs for 1b and 2b):

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

            public void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException
            {
                  //NEW CHANGE
                String doc = value.toString().toLowerCase().replaceAll("\\p{P}", "");
                  StringTokenizer docPart = new StringTokenizer(doc);
                String              docName              =              ((FileSplit)
context.getInputSplit()).getPath().getName().toString();
                String tempStr=""; //temp string to construct the key part
                  while (docPart.hasMoreTokens()) {
                  tempStr = docPart.nextToken(); //removing special character and
punctuation from the word
                  tempStr = tempStr+","+docName;
                  word.set(tempStr);//converting string to text writable
                  context.write(word,one);
                }
            }
    }
```

Reducer for Phase 1 for part2 (To calculate outputs for 2b):

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

             public void reduce(Text key, Iterable<IntWritable> values, Context
context)
                throws IOException, InterruptedException
               {
                         int sum = 0;
                         for (IntWritable val : values)
                         {
                             sum += val.get();
                         }
                         //Condition to consider words that occure more than once
                         if (sum > 1) {
                                context.write(key, new IntWritable(sum));
                         }
               }
    }
```

Mapper for Phase 1 for part 3 (To calculate outputs for 3a):

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

             public void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException
             {
                 String doc = value.toString().toLowerCase().replaceAll("\\p{P}",
"");
                   StringTokenizer docPart = new StringTokenizer(doc);
                 String docName = ((FileSplit)
context.getInputSplit()).getPath().getName().toString();
                 String tempStr=""; //temp string to construct the key part
                 String first = "";
                 String second = "";
                 if(docPart.hasMoreTokens()){
                      first = docPart.nextToken();
               }
                   while (docPart.hasMoreTokens()) {
                   second = docPart.nextToken(); //removing special character and
punctuation from the word
                   tempStr = first+' '+second+","+docName;
                   word.set(tempStr);//converting string to text writable
                   context.write(word,one);
                   first=second;
                 }
             }
    }
```

Results:

FOR PART 1 (Results for 1b with new inputs)

```
The top Fifteen words with Highest TF*IDF value in document: 11-0.txt
Word                --- TF*IDF
alice               --- 0.018160031610451103
queen               --- 0.0032074861026251296
mock                --- 0.002641459143338342
turtle              --- 0.002641459143338342
gryphon             --- 0.002594290230064443
hatter              --- 0.002594290230064443
rabbit              --- 0.0020282632707776557
dormouse            --- 0.0018395876176820595
hare                --- 0.0014622363114908678
king                --- 0.0014386518548539185
caterpillar         --- 0.001273560658395272
youre               --- 0.0010848850052996761
duchess             --- 9.197938088410298E-4
mouse               --- 8.962093522040803E-4
cat                 --- 8.254559822932319E-4
The top Fifteen words with Highest TF*IDF value in document: 1342-0.txt
Word                --- TF*IDF
elizabeth           --- 0.006611843704625833
mr                  --- 0.004357806078048844
darcy               --- 0.004129619552889198
mrs                 --- 0.0038179501526711453
bennet              --- 0.0032613976522817656
bingley             --- 0.002860679852001412
wickham             --- 0.0017920990512538028
collins             --- 0.001736443801214865
miss                --- 0.001575043576101945
jane                --- 0.0014637330760240688
lydia               --- 0.001458167551020175
catherine           --- 0.0012244155008566356
lizzy               --- 0.001046318700732034
lady                --- 0.001018491075712565
longbourn           --- 9.795324006853085E-4
The top Fifteen words with Highest TF*IDF value in document: 1952-0.txt
Word                --- TF*IDF
wallpaper           --- 0.002593200279383598
jennie              --- 0.0018304943148590104
pattern             --- 0.001601682525501634
john                --- 0.0013611717776657764
yellow              --- 0.0010677883503344228
daylight            --- 7.627059645245877E-4
nursery             --- 7.627059645245877E-4
creeping            --- 6.864353680721289E-4
arbors              --- 6.101647716196701E-4
color               --- 6.101647716196701E-4
creep               --- 6.101647716196701E-4
daytime             --- 6.101647716196701E-4
gilman              --- 6.101647716196701E-4
perkins             --- 6.101647716196701E-4
physician           --- 6.101647716196701E-4
```

```
The top Fifteen words with Highest TF*IDF value in document: 219-0.txt
Word                --- TF*IDF
kurtz               --- 0.003151208065508514
pilgrims            --- 0.001016518730809198
ivory               --- 9.826347731155581E-4
river               --- 8.979248788814583E-4
mr                  --- 8.640409211878182E-4
men                 --- 8.132149846473584E-4
forest              --- 7.793310269537185E-4
black               --- 7.115631115664386E-4
kurtzs              --- 7.115631115664386E-4
manager             --- 6.607371750259786E-4
steamer             --- 6.437951961791587E-4
earth               --- 6.268532173323388E-4
steamboat           --- 6.099112384855187E-4
bush                --- 5.760272807918789E-4
darkness            --- 5.25201344251419E-4
```

FOR PART 2 (Results for 2b with new inputs)

```
The top Fifteen words with Highest TF*IDF value in document: 11-0.txt
Word                --- TF*IDF
alice               --- 0.019171096588762856
queen               --- 0.0033860638130801925
mock                --- 0.002788523140183688
turtle              --- 0.002788523140183688
gryphon             --- 0.0027387280841089795
hatter              --- 0.0027387280841089795
rabbit              --- 0.002141187411212475
dormouse            --- 0.0019420071869136399
duchess             --- 0.0019420071869136399
mouse               --- 0.0018922121308389312
hare                --- 0.00154364673831597
king                --- 0.001518749210278616
caterpillar         --- 0.0013444665140171352
youre               --- 0.0011452862897183005
cat                 --- 8.714134813074025E-4
The top Fifteen words with Highest TF*IDF value in document: 1342-0.txt
Word                --- TF*IDF
elizabeth           --- 0.006775097089937758
mr                  --- 0.004465404900186249
darcy               --- 0.004231584209371899
mrs                 --- 0.003912219363381568
bennet              --- 0.003341924995541689
jane                --- 0.0029997483748377615
bingley             --- 0.002931313050696976
wickham             --- 0.0018363478644444094
collins             --- 0.0017793184276604214
miss                --- 0.0016139330609868565
lydia               --- 0.001494171243740482
catherine           --- 0.001254647609247733
lizzy               --- 0.001072153411538972
lady                --- 0.001043638693146978
longbourn           --- 0.0010037180873981864
```

```
The top Fifteen words with Highest TF*IDF value in document: 1952-0.txt
Word                --- TF*IDF
john                --- 0.00370713044329324
pattern             --- 0.0036209181074026995
wallpaper           --- 0.0029312194202783756
jennie              --- 0.002069096061372971
creeping            --- 0.0015518220460297284
creep               --- 0.0013793973742486474
yellow              --- 0.0012069727024675665
daylight            --- 8.621233589054045E-4
nursery             --- 8.621233589054045E-4
arbors              --- 6.896986871243237E-4
color               --- 6.896986871243237E-4
darling             --- 6.896986871243237E-4
daytime             --- 6.896986871243237E-4
gilman              --- 6.896986871243237E-4
perkins             --- 6.896986871243237E-4
The top Fifteen words with Highest TF*IDF value in document: 219-0.txt
Word                --- TF*IDF
kurtz               --- 0.00346080518573403
black               --- 0.0015629442774282717
manager             --- 0.0014513054004691094
darkness            --- 0.00115360172857801
pilgrims            --- 0.0011163887695916227
ivory               --- 0.0010791758106052352
river               --- 9.861434131392666E-4
mr                  --- 9.489304541528793E-4
men                 --- 8.931110156732981E-4
forest              --- 8.558980566869107E-4
station             --- 8.37291577193717E-4
kurtzs              --- 7.814721387141358E-4
wilderness          --- 7.814721387141358E-4
steamer             --- 7.07046220741361E-4
earth               --- 6.884397412481672E-4
```

FOR PART 3 (Results for 3a with new inputs)

```
The top Fifteen words with Highest TF*IDF value in document: 11-0.txt
Bigram              --- TF*IDF
said alice          --- 0.005783173259332075
the queen           --- 0.0031260395996389597
the mock            --- 0.002761334979681081
mock turtle         --- 0.0026571336596931156
the gryphon         --- 0.002605032999699133
the hatter          --- 0.002605032999699133
said the            --- 0.0022488676740067054
the duchess         --- 0.0019277244197773585
the dormouse        --- 0.0017714224397954105
march hare          --- 0.0015630197998194799
the king            --- 0.0014848688098285058
the march           --- 0.0014588184798315145
the caterpillar     --- 0.0013025164998495666
the mouse           --- 0.0013025164998495666
thought alice       --- 0.0013025164998495666
```

The top Fifteen words with Highest TF*IDF value in document: 1342-0.txt

Bigrarm	---	TF*IDF
mr darcy	---	0.0027844102534914468
mrs bennet	---	0.0015021160578045961
mr collins	---	0.00147769140645818
lady catherine	---	0.0010624723335691047
mr bingley	---	0.0010136230308762722
mr bennet	---	9.159244254906074E-4
miss bingley	---	8.182258201049426E-4
she could	---	7.510580289022981E-4
elizabeth was	---	7.449518660656941E-4
am sure	---	7.083148890460698E-4
mr wickham	---	6.716779120264454E-4
her mother	---	6.350409350068211E-4
miss bennet	---	6.350409350068211E-4
of her	---	6.183659191500275E-4
mrs gardiner	---	6.10616283660405E-4

The top Fifteen words with Highest TF*IDF value in document: 1952-0.txt

Bigrarm	---	TF*IDF
john is	---	0.001517471326937365
john says	---	0.0010116475512915767
yellow wallpaper	---	0.0010116475512915767
by daylight	---	8.430396260763139E-4
the paper	---	7.587356634686825E-4
a physician	---	6.744317008610511E-4
around the	---	6.744317008610511E-4
charlotte perkins	---	6.744317008610511E-4
john and	---	6.744317008610511E-4
john would	---	6.744317008610511E-4
perkins gilman	---	6.744317008610511E-4
the daytime	---	6.744317008610511E-4
the pattern	---	6.744317008610511E-4
the wallpaper	---	6.744317008610511E-4
is one	---	5.901277382534197E-4

The top Fifteen words with Highest TF*IDF value in document: 219-0.txt

Bigrarm	---	TF*IDF
mr kurtz	---	0.0014415154930857924
as though	---	0.0012567058144850498
the manager	---	0.0012197438787649015
i saw	---	7.392387144029705E-4
the pilgrims	---	7.02276778682822E-4
the forest	---	5.913909715223765E-4
he was	---	5.676018066824451E-4
of darkness	---	5.174671000820793E-4
the station	---	5.174671000820793E-4
the wilderness	---	5.174671000820793E-4
kurtz was	---	4.8050516436193085E-4
the steamer	---	4.8050516436193085E-4
of his	---	4.448770917240785E-4
the river	---	4.435432286417823E-4
kurtz had	---	3.6961935720148525E-4

Comments: (For new input 4 files)

- TF*IDF values are increased for same words → (So if we want to rank with decent threshold we are increasing TFIDF values to same words by ignoring rare words/ noise words.)
- Almost all words which appear in output (top 15 in each doc) of part-1 (1b) came up in output (top 15 in each doc) of part-2. (2b) also but with lightly high TFIDF values.
  (Observed that removing rare words is important because they're so rare, the association between them and other words is dominated by noise**.)**
- So, part-2 (2b) is better characterizing the documents compared with part-1 (1b)
- Also, part-3(3a) is giving more specific information compared with part-1(1b) and so it part-3(3a) is preferred over part-1(1b)
- Output **of part-3(3a) can be preferred over part-2(2a)** because part-3 output is giving more specific information, for example **gender information** can be easily extracted from part3 output compared with part 1 or part2 for "**1324-0.txt**" input file.
- Also, for **"11-0.txt" & 219-0.txt** input files, the output words are appearing almost similar in all three but with **specific articles for those words in bigram/part3 output. (**This information is useful in some analysis in characterizing the document.)
- Also, for **"1952-0.txt"** input file, the bigrams like "`charlotte perkins`" and "`perkins gilman`" in output of part3 will have more similar detailed information when compared with word like "perkins" in output of part1 and part2. Example consider when someone searches for "**Project Gutenberg's The Yellow Wallpaper, by Charlotte Perkins Gilman"** the system like part1 or part2 returns all pages/documents that are with high tfidf "perkins" or "wallpaper" which may irrelevant. But the system like part3 will give pages with high "charlotte parkins" or "perkins gilman" or "yellow whitepaper". Here **Author name** is more specific in case of part 3 and so search results will be refined and will get more relevant documents.

Note:

- Code files and output folders are attached as zip for reference.