BIGDATA

제출일 : 2018.05.08

TEXT ANALYSIS (N- GRAM)

개발 환경 : Java

사용 라이브러리 :

- Maven 으로 관리 (Apache commons-IO , Apache commons-Lang)

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
    <scope>test</scope>
  </dependency>
  <!-- Apache Commons IO » 2.5 -->
  <dependency>
      <groupId>commons-io</groupId>
      <artifactId>commons-io</artifactId>
      <version>2.5</version>
  </dependency>
      <dependency>
      <groupId>commons-lang</groupId>
      <artifactId>commons-lang</artifactId>
      <version>2.6</version>
  </dependency>
</dependencies>
```

위의 사진은 Maven 에 라이브러리를 추가한 소스코드 입니다. 처음 300MB 이상의 텍스트 파일을 불러 올 때 단순히 BufferedReader 를 사용하여 한 줄씩 텍스트를 읽어왔을 때 속도가 너무 느림을 확인 하였습니다. 아래의 소스코드는 처음 텍스트 파일을 읽어 오기 위해서 작성한 코드입니다. 따라서 검색결과 Apache commons 에서 제공하는 라이브러리를 찾게 되었고 파일 IO 부터 다양한 스트링 연산 기능을 제공 하기 때문에 사용하게 되었습니다. 또한, 아래 소스코드에서는 파일을 읽어올 때 여러 예외처리를 직접 해줘야 하지만 라이브러리를 사용함으로써 이러한 불편한 과정을 줄여주고 소스코드가 한결 간결해 졌습니다.

아래 두번째 그림은 라이브러리를 사용함으로 써 바뀐 소스 코드 입니다.

FileUtils.readFileToString 을 통해서 한번에 String 타입으로 읽어 올 수 있었고 속도를 훨씬 단축 시킬 수 있었습니다. 하지만 osx 에서 텍스트 파일을 읽어올 때 한글 깨짐에 발생하여서 해당 파일을 utf-8 로 변환하여 주었습니다. 아래 명령 입니다.

$ iconv -c -f euc-kr -t utf-8 test.txt > text_1.txt

```
        /*
         *  read text

        try {
                br = new BufferedReader(new InputStreamReader(new FileInputStream(inFile),"
                // 파일 형식에 따라 euc-kr , utf-8
                out= new BufferedWriter(new FileWriter(outFile, true)); // true : subsequen

                String line;


                // too slow to append total lines !
                while( (line=br.readLine() ) != null) // read by line
                {

                        //line.trim(); //
                        line = line.replaceAll("(^\\p{Z}+|\\p{Z}+$)", "");

                        //sb.append(line); // join lines
                        //System.out.println(line);
                        //sb.trimToSize();

                        //StringTokenizer token = new StringTokenizer(line); // split by sp
                        //totalWords =  token.countTokens();
                        //System.out.println("total number of words : "+totalWords);

                        input = line.split(" "); // easier to access with array than tokeni

                        for(int i=0; i< input.length + 2 ; i++)  // add 2 in length to conf
                        {
                                solve(i-(N-1), 0, input.length);
                                writing.append("\n");
                        }
                        writing.trimToSize();
                }

                System.out.println("complete Ngram classification!");
                //writingTxt(writing.toString());

        } catch ( FileNotFoundException e) {
                e.printStackTrace();
        } catch ( IOException e) {
                e.printStackTrace();
        } finally {
                        if(br != null ) try {br.close(); } catch (IOException e) {}
                }
```

라이브러리 사용 후 변경된 소스 코드

```
import org.apache.commons.io.FileUtils;
import org.apache.commons.io.filefilter.FileFilterUtils;
import org.apache.commons.lang.StringUtils;

public class TextDataSplit
{


        N =3; // test bigram

        String dir = "test2.txt";
        File dirFile = new File(dir);

        try {
                /*
                List<String> readList = FileUtils.readLines(dirFile,"euc-kr");
                for(int i=0;i< readList.size(); i++)
                {
                        System.out.println(readList.get(i));
                }*/
                String readFile = FileUtils.readFileToString(dirFile,"utf-8");
                System.out.println("read success! ");
                readFile.trim();
                long start = System.currentTimeMillis();
                //input = StringUtils.split(readFile);
                readFile= readFile.replaceAll("[[:]\\\\/?[*]]", "");

                //readFile = readFile.replaceAll("!\"#[$]%&\\(\\)\\{\\}@`[*]:[+];-.<>,\\^~
                //readFile= readFile.replaceAll("/[\\{\\}\\[\\]\\/?.,;:|\\)*~`!^\\-_+<>@\\
                //input = readFile.split("^[ㄱ-ㅎ가-힣0-9]*$");
                long end = System.currentTimeMillis();
                System.out.println("split 실행시간 : " +(end-start)/1000.0+"초 ");
```

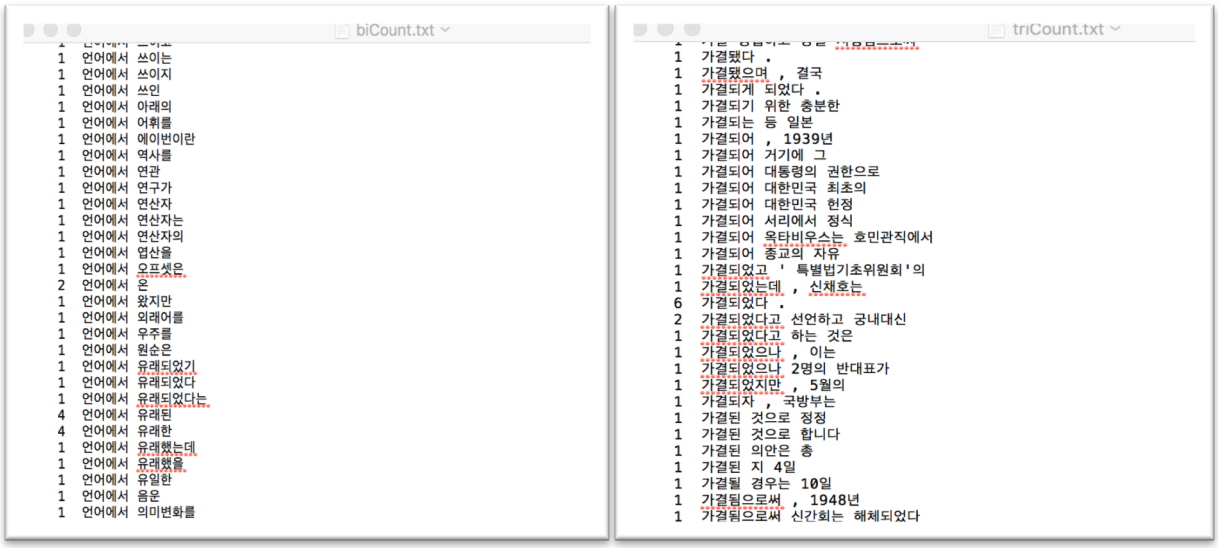
또한 String 타입으로 읽어 온 후 WordCount.exe 로 각 단어의 갯수를 Count 하기 위해서 bigram, trigram 형식에 맞게 라인 별로 단어를 나눴습니다. 전체 String 타입으로 불러왔기 때문에 split 함수를 사용하여 공백 기준으로 단어로 나누었습니다. 이때는 약 20 초 ~ 30 초 정도 소요 됐으며, N-gram 확률을 구할 때 불필요한 특수문자를 제거하기 위해 정규식을 사용하였습니다. readFile = readFile.replaceAll("[[:]\\\\/?[*]]", ""); 이와 같이 정규식을 사용하였고 아직은 미숙하여 완전하게 특수문자를 제거하지 못해서 조금 더 연구해 보겠습니다. 그 후 미리 작성해 놓은 함수를 이용하여 bigram, trigram 형식에 맞게 단어를 라인 별로 나누었습니다. 라인 별로 작성하여 output.txt 파일로 따로 작성을 하였고, 이때도 기존의 자바에서 제공하는 bufferedWrite 를 사용 하였을 때 너무 속도가 느려서 똑같이 Apache 에서 제공하는 라이브러리를 사용하였습니다. 하지만 이때도 한 줄씩 작성하다 보니 속도가 느렸고 10 분~30 분 정도가 소요 됐습니다.

```
for(int i=0; i< input.length ; i++)  // add 2 in length to confirm text to the enc
{
    solve(i-(N-1), 0, input.length);

    FileUtils.writeStringToFile(outFile,writing.toString()+"\n",true);
    writing.setLength(0);
}
```

위의 소스 코드는 단어를 공백기준으로 모두 나누어 N gram 라인 별로 나누는 그림입니다. Input.length 는 단어의 갯수 만큼 for 문을 돌면서 파일에 write 해주는 코드입니다. 이 부분에서 시간이 오래 소요 되었으므로 추후 좀더 효율적인 방법을 생각해 보겠습니다.

아래는 bigram, trigram 모두 텍스트 파일로 작성되어 wordcount.exe 를 라인 별로 실행시킨 사진이다.



그 후 이 데이터를 바탕으로 확률을 구하기 위한 클래스를 하나 더 만들었습니다.(TextAnalysis.class)

저는 데이터 접근을 효율적으로 하기 위해 hashmap 을 사용하여 스트링과 count 를 mapping 한 후 확률을 구했습니다. 아래의 그림처럼 먼저 라인 별로 앞 뒤 불필요한 공백을 제거 해준 후 hashmap 으로 전체 파일을 맵핑을 해주었습니다

```
public class TextAnalysis {
    public static Map<String, String> unigramMap = new HashMap<String, String>();
    public static Map<String, String> bigramMap = new HashMap<String, String>();
    public static Map<String, String> trigramMap = new HashMap<String, String>();
    public final static double unigramN = 3175482; // value obtained from wordcount.exe
    public final static double bigramN = 3009662;
    public final static double triaramN = 2984332:
```

위의 사진과 같이 unigram 부터 trigram 까지 Hashmap 을 선언하였고, 전체 N 의 숫자는 wordcount.exe 에서 나온 단어 갯수를 참고 하였습니다. 아래 사진과 같이 전체 라인별로 for 문을 돌면서 hashmap 을 설정해 주었는데 wordcount.exe 에서 갯수 와 스트링 사이 \t 으로

결과가 나오기 때문에 이를 이용하여 split 을 하였습니다. 또한 편의를 위해서 스트링 맨 앞과 맨 뒤의 공백을 제거하였습니다.

```
String[] input;
for(int i=0; i< readList.size(); i++)
{
    // remove front-end space
    String result = readList.get(i).replaceAll("(^\\p{Z}+|\\p{Z}+$)", "");

    input = result.split("\t"); // split count number and string value

    if(input.length == 2) map.put(input[1], input[0]);

}
```

```
uniCount.txt Time to read: 0.686 second
Time to make hashMap: 7.077 second
biCount.txt Time to read: 2.742 second
Time to make hashMap: 14.197 second
triCount.txt Time to read: 0.627 second
Time to make hashMap: 7.86 second
Puni(str) : 2.705319265790755E-9
Pbi(str) : 9.613424906214796E-9
Pbi(str) : 1.3403334481552322E-6
```

위의 사진은 최종 결과 화면입니다. wordcount.exe 를 거친 파일들을 각각 읽어오고 시간을 실행 시간을 계산해 주고 있고 해당 스트링을 계산하여 확률을 구하였습니다.