# Summarization of Document using Java

Priyanka Sarraf
M.Tech Scholar, Banasthali University
Jaipur, India

Yogesh Kumar Meena
Assistant Professor, MNIT
Jaipur, India

***Abstract*-- Automatic summarization is the process of reducing a document by a computer program to create summary which retains the important aspects of the original document. As the use of whole document is very tedious task so summary helps identifying the general overview and its importance for us. Generally, there are two approaches of automatic summarization: extraction and abstraction. Extractive methods work by selecting a subset of existing words, phrases, or sentences in the original text to form the summary. Whereas, abstractive methods build an internal semantic representation and then use natural language generation techniques to create a summary that is closer to what a human might generate. Such a summary might contain words not explicitly present in the original. The state-of-the-art abstractive methods are still quite weak, so most research has focused on extractive methods.**

**Each programming language is having its own set of libraries which helps for reading and summarizes the document. This paper presents basic steps of summarization of documents, its challenges and the set of APIs which are available in java and its related technologies.**

***Keywords:* Document summarization, extraction, Java, Preprocessing, Stemming.**

## I. INTRODUCTION

With the increasing amount of online information, it becomes extremely difficult to find relevant information to users. Information retrieval systems usually return a large amount of documents listed in the order of estimated relevance. It is not possible for users to read each document in order to find useful ones. Automatic text summarization systems helps in this task by providing a quick summary of the information contained in the document. An ideal summary in these situations will be one that does not contain repeated information and includes unique information from multiple documents on that topic.

The Single document summarizer is an application which is proposed to extract the most important information of the document. In automatic summarization, there are two distinct techniques either text extraction or text abstraction. Extraction is a summary consisting of a number of sentences selected from the input document. An abstraction based summary is generated where some text units are not present into the input document. With extraction based summary technique, some more features are added based on Information Retrieval. However, the total system is alienated into three segments: pre-processing the test document, sentence scoring based on text extraction and summarization based on sentence ranking.

In this approach generic based single document summarization system is proposed using extraction based summary techniques.

## II. SYSTEM DESCRIPTION

Goal of extractive text summarization is selecting the most relevant sentences of the text. The Proposed method uses statistical and Linguistic approach to find most relevant sentence. Summarization system consists of 3 major steps, Preprocessing ,Extraction of feature terms and algorithm for ranking the sentence based on the optimized feature weights.

### *A. Pre-processing*

This step involves Sentence segmentation, Sentence tokenization, Stop word Removal and Stemming.

#### *1) Sentence Segmentation*

It is the process of decomposing the given text document into its constituent sentences along with its word count. In English, sentence is segmented by identifying the boundary of sentence which ends with full stop ( . ) , question mark (?), exclamatory mark(!).

#### *2) Tokenization*

It is the process of splitting the sentences into words by identifying the spaces, comma and special symbols between the words. So list of sentences and words are maintained for further processing.

#### *3) Stop Word Removal*

Stop words are common words that carry less important meaning than keywords .This words should be eliminated otherwise sentence containing them can influence summary generated.

#### *4) Stemming*

A word can be found in different forms in the same document. These words have to be converted to their original form for simplicity. The stemming algorithm is used to transform words to their canonical forms. In this work, Porter’s stemmer is used that splits a word into its root form using a predefined suffix list.

### B. *Feature Extraction*

After an input document is tokenized and stemmed, it is split into a collection of sentences. The sentences are ranked based on four important features: Frequency, Sentence Position value, Cue words and Similarity with the Title.

#### 1) *Frequency*

Frequency is the number of times a word occurs in a document. If a word's frequency in a document is high, then it can be said that this word has a significant effect on the content of the document. The total frequency value of a sentence is calculated by sum up the frequency of every word in the document.

#### 2) *Sentence Position Value*

Position of the sentence in the text, decides its importance. Sentences in the beginning defines the theme of the document whereas end sentences conclude or summarize the document. The positional value of a sentence is computed by assigning the highest value to the first sentence and the lowest value to the last sentence of the document.

#### 3) *Cue Words*

Cue words are connective expressions (such as *therefore, hence, lastly, finally, meanwhile* or on the other hand) that links spans of communication and signals semantic relations in a text.

#### 4) *Similarity with the Title*

The similarity with the title consists of the words in titles and headers. These words are considered having some extra weights in sentence scoring for summarization.

### C. *Sentence Scoring*

The final score is a Linear Combination of frequency, Sentence positional value, weights of Cue Words and Similarity with the title of the document.

### D. *Sentence Ranking*

After scoring of each sentence, sentences are arranged in descending order of their score value i.e. the sentence whose score value is highest is in top position and the sentence whose score value is lowest is in bottom position.

### E. *Summary Extraction*

After ranking the sentences based on their total score the summary is produced selecting X number of top ranked sentences where the value of X is provided by the user. For the readers' convenience, the selected sentences in the summary are reordered according to their original positions in the document.

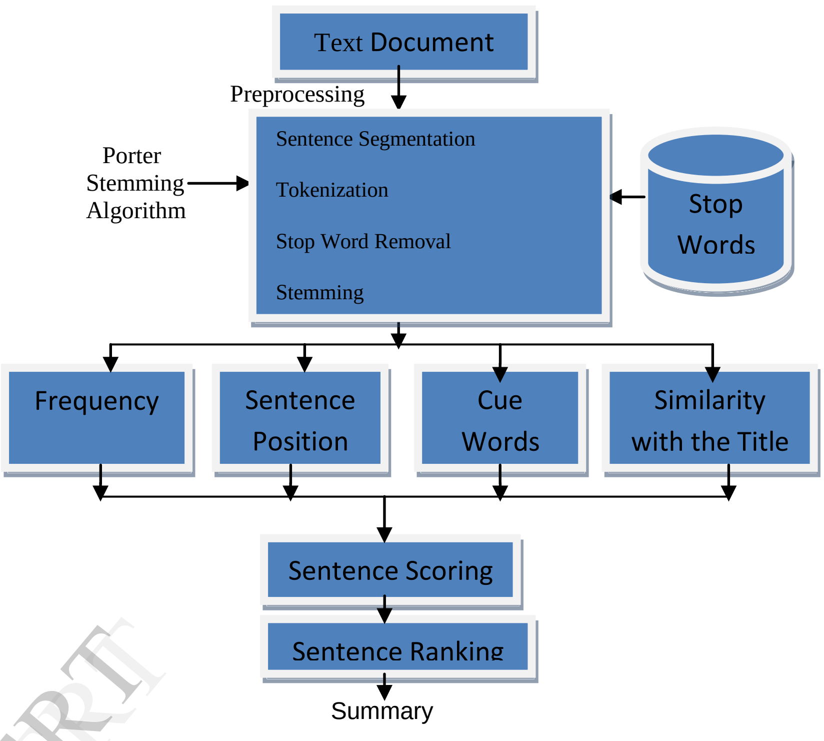

Fig. 1 Steps of the proposed text summarization technique

## III. JAVA API'S

A lot of java APIs is available for summarization of text. Basic steps of Java APIs can be categorized as follows:

- Reading the document
- Create summary of the document.

### A. *Reading the document:*

There might be different type of documents which might be available to get summary. Basically following four types of documents which we can get are:

- Text File
- Doc File
- Docx File
- PDF file

#### 1) *Text File*

A text file can be read just by using java.io package of j2sdk. For representing and reading the file, File class, FileInputStream and FileReader classes are available. A simple piece of code is given below for reading a text file.

```
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
public String readFile(String filename)
```

```
{   String fullContent = null;
   File file = new File (filename);
//Filename is name of file like f1.txt
   try {FileReader reader = new FileReader(file);
       char[] chars = new char[(int) file.length()];
       reader.read(chars);
       fullContent = new String(chars);
       reader.close()} catch (IOException e)
{e.printStackTrace();}
   return fullContent;}
```

There might several different ways also to read a text file from java.

*2) Doc/Docx File*

.doc files are part binary, part text files. There are several third party APIs available for reading a .doc file. The easiest way to "read" a Microsoft doc file **is** apache POI api. The Apache POI Project's mission is to create and maintain Java APIs for manipulating various file formats based upon the Office Open XML standards (OOXML) and Microsoft's OLE 2 Compound Document format (OLE2). In short, you can read and write MS Excel files using Java. In addition, you can read and write MS Word and MS PowerPoint files using Java. Apache POI is your Java Excel solution (for Excel 97-2008). We have a complete API for porting other OOXML and OLE2 formats and welcome others to participate.

OLE2 files include most Microsoft Office files such as XLS, DOC, and PPT as well as MFC serialization API based file formats. The project provides APIs for the OLE2 Filesystem (POIFS) and OLE2 Document Properties (HPSF).

Office OpenXML Format is the new standards based XML file format found in Microsoft Office 2007 and 2008. This includes XLSX, DOCX and PPTX. The project provides a low level API to support the Open Packaging Conventions using openxml4j.

For each MS Office application there exists a component module that attempts to provide a common high level Java api to both OLE2 and OOXML document formats. This is most developed for Excel workbooks (SS=HSSF+XSSF). Work is progressing for Word documents (HWPF+XWPF) and PowerPoint presentations (HSLF+XSLF).

A sample code for reading a .doc and docx file using POI api is given below:

```
if(filetype.equals(".Doc")||filetype.equals(".docx"))
{  String
extension=filename.substring(filename.lastIndexOf(".")
   + 1, filename.length());
   if(extension.equals("doc"))
   { File docFile=new File(filename);   // file object was
created
          FileInputStream finStream=new
          FileInputStream(docFile.getAbsolutePath()); //
file input
          stream with docFile
      HWPFDocument doc=new
HWPFDocument(finStream);// throws
              IOException and need to import
              org.apache.poi.hwpf.HWPFDocument;
      WordExtractor wordExtract=new WordExtractor(doc);
      // import  org.apache.poi.hwpf.extractor.WordExtractor
      String [] dataArray =wordExtract.getParagraphText();
      // dataArray stores the each line from the document
      for(int i=0;i<dataArray.length;i++)
      { document+=dataArray[i];
          // printing lines from the array}
      finStream.close(); //closing fileinputstream }
   else
   { OPCPackage d = OPCPackage.open(fin);
      XWPFWordExtractor xw = new
XWPFWordExtractor(d);
      document=xw.getText();
      System.out.println(filename); }}
```

*3) PDF file*

A PDF file is made up of a sequence of bytes. These bytes, grouped into tokens, make up the basic objects upon which higher level objects and structures are built. There are various set of APIs for reading pdf files. PDFBox is one of such APIs which was designed by apache. PDFBox makes these basic objects available in the *org.apache.pdfbox.cos* package (The COS Model). The organization of these objects, how to they are read and how to write them is defined in the file structure of the PDF. In addition a file can be encrypted to protect the document's content. PDFBox handles the reading in the *org.apache.pdfbox.pdfparser* package. Writing of PDF files is handled in the *org.apache.pdfbox.pdfwriter* package. Within the file structure basic objects are used to create a document structure building higher level objects such as pages, bookmarks, annotations.

PDFBox makes these higher level objects available through the *org.apache.pdfbox.pdfmodel* package (The PD Model).In addition there is a COS representation available for the PD model if there is a need to inspect the underlying structure or to handle special cases where the higher level PD model doesn't provide the functionality needed. It's always the COS model which is represented in the PDF file.

A sample code for reading a pdf file using POI api is given below:

```
import java.io.*;
import org.apache.pdfbox.pdmodel.*;
import org.apache.pdfbox.util.*;

 PDDocument pd;
 BufferedWriter wr;
 try {File input = new File ("C:\\Invoice.pdf");
        File output = new File ("C:\\SampleText.txt");
        pd = PDDocument.load(input);
        System.out.println(pd.getNumberOfPages());
        System.out.println(pd.isEncrypted());
              pd.save("CopyOfInvoice.pdf");
        PDFTextStripper stripper = new PDFTextStripper();
              wr = new BufferedWriter(new
             OutputStreamWriter(new
              FileOutputStream(output)));
        stripper.writeText(pd, wr);
        if (pd != null) { pd.close(); }
      wr.close(); } catch (Exception e){e.printStackTrace();
       }
    }
}
```

*B. Create summary of the document.*

- *String*

The String class represents character strings. All string literals in Java programs, such as "abc", are implemented as instances of this class. Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings. Because String objects are immutable they can be shared. For example:

```
String str = "abc";
```

is equivalent to:

```
char data[] = {'a', 'b', 'c'};

String str = new String (data);
```

Here are some more examples of how strings can be used:

```
System.out.println("abc");

String cde = "cde";

System.out.println("abc" + cde);

String c = "abc".substring(2,3);

String d = cde.substring(1, 2);
```

The class String includes methods for examining individual characters of the sequence, for comparing strings, for searching strings, for extracting substrings, and for creating a copy of a string with all characters translated to uppercase or to lowercase. Case mapping is based on the Unicode Standard version specified by the Character class.

The Java language provides special support for the string concatenation operator (+), and for conversion of other objects to strings. String concatenation is implemented through theStringBuilder(or StringBuffer) class and its append method. String conversions are implemented through the method toString, defined by Object and inherited by all classes in Java.

- *StringTokenizer*

The string tokenizer class allows an application to break a string into tokens. The tokenization method is much simpler than the one used by the StreamTokenizer class. The StringTokenizer methods do not distinguish among identifiers, numbers, and quoted strings, nor do they recognize and skip comments. The set of delimiters (the characters that separate tokens) may be specified either at creation time or on a per-token basis.

An instance of StringTokenizer behaves in one of two ways, depending on whether it was created with the returnDelims flag having the value true or false:

- If the flag is false, delimiter characters serve to separate tokens. A token is a maximal sequence of consecutive characters that are not delimiters.
- If the flag is true, delimiter characters are themselves considered to be tokens. A token is thus either one delimiter character, or a maximal sequence of consecutive characters that are not delimiters.

A StringTokenizer object internally maintains a current position within the string to be tokenized. Some operations advance this current position past the characters processed.

A token is returned by taking a substring of the string that was used to create the StringTokenizer object. The following is one example of the use of the tokenizer. The code:

```
StringTokenizer st = new StringTokenizer ("this is a test");
 while (st.hasMoreTokens()) {System.out.println(st.nextToken());}
```

Prints the following output:

StringTokenizer is a legacy class that is retained for compatibility reasons although its use is discouraged in new code. It is recommended that anyone seeking this functionality use the split method of String or the java.util.regex package instead.

The following example illustrates how the String.split method can be used to break up a string into its basic tokens:

```
String[] result = "this is a test".split("\\s");
for (int x=0; x<result.length; x++)
   System.out.println(result[x]);
```

Prints the following output:

## CONCLUSION

This Paper discusses the single document summarization using extraction method. This paper presents basic steps of summarization of documents, its challenges and the set of APIs which are available in java and its related technologies. This proposed method is implemented in java and is under development.

## REFERENCES

[1] H. P. Luhn, *The* automatic creation of literature abstracts, in IBM Journal of Research Development, volume 2, number 2, pages 159-165, 1958.

[2] Mani, I.: Automatic Summarization. John Benjamins Publishing Co., Amsterdam (2001).

[3] Wei Liu, WANG, The Document Summary Method based on Statements Weight and Genetic Algorithm, International Conference on Computer Science and Network Technology, 2011

[4] Automated Bangla Text Summarization by Sentence Scoring and Ranking, Md. Iftekharul Alam Efat, IEEE, 2013.

[5] The Porter Stemming Algorithm [Online].Available:http://tartarus.org/~martin/PorterStemmer/

[6] Mr. Vikrant Gupta, Ms. Priya Chauhan, Dr. Sohan Garg. Mrs. Anita Borude, Prof. Shobha Krishnan "A Statistical Tool for Multi-Document Summrization" , International Journal of Scientific and Research publication, Volume 2, Issue 5, may 2012 ISSN 2250-3153

[7] Chetana Thaokar, Dr.Latesh Malik, Test Model for Summarizing Hindi Text using Extraction Method, IEEE, 2013