Table of Contents

1. Introduction

CNGLSummarizer is an automatic summarization tool built at the Center of Next Generation Localisation (CNGL). It uses sentence extraction to generate summaries. The selection methods are based on Information Retrieval and Natural Language Processing techniques.

Below is a basic guide for getting the tool up and running. Further details on its inner workings can be found by reading the javadocs or exploring its source code.

2. Installation Guide

2.1 Requirements

- Java 6+ - http://java.com/en/download/index.jsp
- Maven 3+ - http://maven.apache.org/download.html

2.2 Installation Using Maven

Maven is used to manage the project's dependencies. To build the summarizer follow the instructions here - http://maven.apache.org/run-maven/index.html#Quick_Start

```
mvn clean install
```

2.3 Manual Installation

In cases where usage of Maven is not desired the project can be built manually. CNGLSummarizer depends on the following libraries -

- Apache Lucene 3.6.1
- Apache Commons Configuration 1.8
- Apache Commons IO 2.4
- Apache Commons Lang 2.6

- Apache Commons Logging 1.1.1

3. Usage

3.1 Configuration

Numerous components of CNGLSummarizer are configurable. Below is a detailed explaination of each option.

3.1.1 summarizer.xml

Files

Several features used to score sentences require files containing lists of words or phrases. Different lists may work better for different domains, so these are configurable.

- ***stopwords***

  Used during indexing and tokenization. Any words listed in the stopwords file will be ignored.

- ***cuephrases***

  The cue phrase feature scores sentences containing phrases listed in the cue phrase file. These can be either positive or negative scores.

  Format: <phrases>,<score> eg. to conclude,5

- ***affixes***

  The affix presence feature scores sentences containing affixes from the affixes file positively. These can be domain specific.

  Format: <optional hyphen><affix><optional hypen> eg. -iate

  The hyphen denotes the position of the word's stem.

- ***sections***

  Certain sections contain sentences better fit for use in a summary. The names of these sections are stored along with their scores in the sections file.

  Format: <section>,<score> eg. introduction,5

- ***importantTerms***

  For some domains certain terms will be useful in a summary. These can be listed in the importantTerms file. Any sentences which contain these terms will be scored positively.

Multipliers

Some features may prove better for sentence selection in certain domains. It is possible to give specific weights to features in the configuration file, which will either improve that features contribution, disimprove it, or even cancel it completely (a multiplier of 0). Any value can be used for the weight, except for the cases detailed below. These features are -

- ***skimming***
- ***namedEntity***
- ***TSISF***
- ***titleTerm***
- ***cuePhrase***
- ***shortSentence***

A multiplier of either 1 or 0 is used here. The multiplier either enables or disables the feature.

- ***LongSentence***

A multiplier of either 1 or 0 is used here. The multiplier either enables or disables the feature.

- ***queryBias***
- ***globalBushy***
- ***punctuation***

A multiplier of either 1 or 0 is used here. The multiplier either enables or disables the feature.

- ***affixPresence***
- ***basicWords***
- ***sectionImportance***
- ***clusterKeyword***

Feature Settings

Some features allow the user to change more specific settings. This gives the user more control over the inner workings of that feature.

- ***affixPresence***

  extraLettersForMatch

  To count as a definite match, a predefined number of characters must exist at the appropriate end of a discovered affix.

  Example:

  Affix: -iate

  Word: asphixiate

  Number of letters: 6.

  If the number of letters required is 6 or less we have a match.

  Default: 3

- ***clusterKeyword***

minSignificantWordSeperation

To be considered a cluster of keywords, significant words must be no greater than a predefined number of insignificant words apart.

Example:

Significant words: scoring, information, structural

Sentence: The sentence [[scoring process utilises information] both from the structural] organization.

If the minimum separation is 3 we have a cluster of 3 significant terms. If it is 2 our largest cluster has 2 significant terms.

Default: 5

- ***globalBushy***

minimumSimilarity

Any sentence comparisons below this level will not be counted. They are deemed to be too dissimilar. These high scores are not good indicators of central nodes.

Default: 0.8

MaximumSimilarity

Any sentence comparisons above this level will not be counted. Some sentences are almost carbon copies, and will score artificially high. These high scores are not good indicators of central nodes.

Default: 1.0

minimumSentenceQueryLength

To calculate the similarity of each sentence a query is provided to a lucene index containing all sentences. Sentences with few query terms score artifically high, and are not good indicators of central nodes. Increasing this value improves central node detection, as well as performance (due to a reduction in the number of queries).

Default: 20

- ***lucene***

topTermCutOff

This value specifies where to cutoff the term list for query. The text is loaded into an in-memory index, a sentence per Lucene Document. Then the index is queried for terms and their associated frequency in the index. The topTermCutoff is a ratio from 0 to 1 which specifies how far to go down the frequency ordered list of terms. The terms considered have a frequency greater than topTermCutoff *topFrequency. topTermCutoff a ratio specifying where the term list will be cut off. Must be between 0 and 1. Default is to consider all terms if this variable is not set, ie topTermCutoff == 0. But it is recommended to set an appropriate value (such as 0.5).

Default: 0.3

- ***punctuation***

  maxRatio

  The max ratio of punctuation to terms. If the ratio is higher than this the sentence is marked negatively.

  Default: 0.3

- ***shortSentence***

  minLength

  If the number of terms in a sentence is below this figure it is weighted negatively.

- ***longSentence***

  maxLength

  If the number of terms in a sentence is above this figure it is weighted negatively.

3.2.1 tokenizer.xml

Configuration files are needed at word level and sentence level for tokenization. These are configurable to allow for different languages.

Word

- ***abbreviations***

  Certain abbreviations can be missed by general rule tokenizers. A list of these can be specified in order to ensure they are discovered.

  Format: "<abbreviation>"

Sentence

- ***badStart***

  A list of bad sentence starts can be listed in the badStart file. A sentence cannot start with any of these.

  Format: "<bad sentence start>"

- ***possibleEnd***

  A list of possible sentence ends can be listed in the possibleEnd file. A sentence cannot end with any of these.

  Format: "<possible sentence end>"

- ***badEnd***

  A list of bad sentence ends can be listed in the badEnd file. A sentence cannot end with any of these.

  Format: "<bad sentence end>"

3.2 Summarization (on the fly)

Content can be summarized on the fly, though this may be slow for long content, especially in the case where computationally expensive features are used.

Example:

```
public class Example {
  public static void main(String [] args) throws Exception {
    //Reading file to string
    String text = FileUtils.readFileToString(new File(<fileWithPath>));
    //Structurer will tokenize, sentenize and paragraph content.
    Structurer structurer = new Structurer();
    //Weighter will weight the sentences
    Weighter weighter = new Weighter();
    //Aggregator will combine the scores and rank the sentences
    Aggregator aggregator = new Aggregator();
    //The summarizer will use all of these components to summarize the content
    Summarizer summarizer = new Summarizer(structurer, weighter, aggregator);
    //The number of sentences we require
    summarizer.setNumSentences(3);
    String summary = summarizer.summarize(text);
    System.out.println(summary);
  }
}
```

3.3 Summarization (offline)

Because it can be slow to summarize content on the fly it may be necessary to do certain calculations offline.

Offline calculations (with storage):

```
public class SavingWeights {
  public static void main(String[] args) throws IOException {
    String text = FileUtils.readFileToString(new File(<fileWithPath>));
    Structurer structurer = new Structurer();
    PageStructure structure = structurer.getStructure(text);
    Weighter weighter = new Weighter();
    ArrayList<Double[]> weights = new ArrayList<Double[]>();
    weighter.setStructure(structure);
    //Using weighter to only calculate weights
    weighter.calculateWeights(weights);

    try {
      FileOutputStream fileOut = new FileOutputStream(new File(<outputFile>));
      ObjectOutputStream out = new ObjectOutputStream(fileOut);
      out.writeObject(weights);
      out.close();
      fileOut.close();
    } catch(IOException e) {
      e.printStackTrace();
    }
  }
}
```

Using offline calculations:

```
public class UsingSavedWeights {
  public static void main(String[] args) throws IOException {
    ArrayList<Double[]> weights = new ArrayList<Double[]>();
    try {
      FileInputStream fileIn = new FileInputStream(new File(<outputFile>));
      ObjectInputStream in = new ObjectInputStream(fileIn);
      weights = (ArrayList<Double[]>) in.readObject();
      in.close();
      fileIn.close();
    } catch(Exception e) {
      e.printStackTrace();
    }

    String text = FileUtils.readFileToString(new File(<fileWithPath>));
    Structurer structurer = new Structurer();
    Weighter weighter = new Weighter();
    Aggregator aggregator = new Aggregator();
    Summarizer summarizer = new Summarizer(structurer, weighter, aggregator);
    summarizer.setNumSentences(3);
    //Using saved weights.
    summarizer.setWeights(weights);
    summarizer.setTitle(<fileTitle>);
    String summary = summarizer.summarize(text);
    System.out.println(summary);
  }
}
```