How to Build an Automated Text Summarizer:

An extraction-based summarizer

CS486/686 Spring 2010

Assignment 4 (due Wed July 28 6:00pm)

In this assignment, you will build an extraction-based text summarizer that will input a text, decide on the most significant sentences in the text according to a metric you will specify, then list these significant sentences as a summary of the original text.

To build an automated text summarizer, for example, a typical word-frequency–based summarizer, you will first need to understand the basic components of any text summarizer. An overview of text summarization is given in the paper by Dr. Eduard Hovy on our UW-ACE website. You should begin by reviewing this paper, then review the starter code given to you in "TextSummarizer.zip" on the website. The starter code is written in Java and contains the following components: (Note: In your implementation, you may use a programming language other than Java):

1. **The `Main` module:**
    a. Calls the `Generator` module to produce the summary of a text.
2. **The `Generator` module contains the following methods:**
    a. `setKeywords`: Reads in the text, preprocesses it (i.e., converts to lower case, removes punctuation, removes stopwords (see below), removes suffixes (this is called "word stemming"), then assigns the variable `keywords` to be the set of significant words in the document.
    b. `stopwords.txt:` It is necessary to have a list of English "stop words". These are small function words, like "the", "and", "a", which do not contribute meaning to the text summary. The file `stopwords.txt` contains a list of common English stop words, to which you may add others.
    c. `***calcAllSentenceScores:` Calculates a variable `scores` giving the value of each sentence in the document. This sentence

scorer calculates the value of a sentence according to a single metric, or a combined metric. The sentence scores will then be used to decide which sentences to keep in the final summary. You will need to devise a metric and implement this method. (We will discuss types of metrics during the Project Workweek working sessions.)

d. `**generateSignificantSentences:` Ranks the sentences in the document according to their scores and decides which ones to keep (i.e., these sentences will have scores above a certain threshold). You will need to implement this method.

e. `*generateSummary:` Prints out the most significant sentences in the document. You will need to implement this method.

f. The methods above are labelled in order of easiest and shortest to write ("*") to most difficult and time-consuming ("*").**

3. **Helper classes given to you:**

a. `Main.java:` Calls the `Generator` to produce the text summary.

b. `Word.java:` Some useful utilities for processing individual words in a document. (Note: You may not need all these methods for your summarizer.)

c. `TextExtractor.java:` Contains methods to extract the individual terms and sentences from a document.

d. `TermPreprocessor.java:` Contains utilities to convert a word to lower case, remove punctuation, remove stop words, and produce a word's stem (i.e., remove suffixes).

e. `TermCollectionProcessor.java:` Contains utilities for keeping track of the number of occurrences of each term (word) in the document, for sorting words according to their scores, etc. (May or may not be needed depending on the metric you choose.)

f. `TermCollection.java:` Contains utilities for managing a collection of terms. (May or may not be needed.)

g. `StringTrimmer.java:` Removes leading and trailing characters from a string.

h. `Stemmer.java:` The Porter stemmer in Java. Returns a word's "stem" (i.e., its root form, minus suffixes).

i. `InputDocument.java:` Contains utilities for setting up various methods to process a document. (May or may not be needed.)

4. **Helper files given to you:**
   a. `stopwords.txt:` Common function words ("the", "a", "and", etc.) to be removed before extracting a summary. You may add others.
   b. `inputLarge.txt, inputNews.txt, inputTech.txt:` Sample input texts from different genres to use in testing. You are expected to test your summarizer on additional texts. In developing a text summarizer, it is usual to focus on texts from one specific genre (e.g., news articles, blogs, email, product reviews, etc.).
   c. `outputLarge.txt,outputNews.txt,outputTech.txt:` Summaries of the above sample input texts. Note: These summaries are for comparison only. There is no "right" summary of a text.
5. **How to get started:**
   a. Download the zipfile "TextSummarizer.zip" from the Assignments subfolder in the "Communications" folder under "Lessons".
   b. Walk through the code following the sequence of classes and helper methods shown above to make sure you understand the overall structure of a summarizer.
   c. Decide on a metric or combined metric for scoring sentences. For example, you may use a word frequency-based metric (as the starter code is set up to do), or a metric based on a sentence's position in the text, or some other criterion. We will discuss types of metrics in Lecture 20 during the Project Workweek, but the Hovy paper (in Handouts in the "Communications" folder) gives many good ideas.
   d. Implement the `Generator` class methods for scoring sentences (`calcAllSentenceScores`), for generating the most significant sentences (`generateSignificantSentences`), and for generating the final summary (`generateSummary`).
   e. (Optional) You may find it useful to debug your text summarizer on the `input*.txt` files provided.

6. **What to hand in:**
    a. (50 marks) A working text summarizer using at least one type of scoring method.
    b. (35 marks) Tests of your summarizer on 3-4 texts of reasonable length in a specific genre. Note: A summarizer that works well on one type of text (e.g., news articles) will generally not work as well on documents from other genres.
    c. (15 marks) A short write-up on how you would evaluate how well your summarizer works.
    d. (Bonus 20 marks) Perform an evaluation of your summarizer. We will discuss forms of evaluation in Lecture 21 during the Project Workweek, but the Hovy paper (in Handouts in the "Communications" folder) describes various methods of evaluation.