CS6370 Assignment 1
Information Retrieval System

Keerthana S
Computer Science Department
Indian Institute of Technology, Madras
Chennai, India
cs20b039@smail.iitm.ac.in

Vaibhav Mahapatra
Mechanical Engineering Department
Indian Institute of Technology, Madras
Chennai, India
me19b197@smail.iitm.ac.in

***Abstract*—In this document, we will be illustrating the approach we've taken to solve questions in Assignment 1 of CS6370: NLP. The assignment focuses on building a search engine from scratch, which is an example of an information retrieval system. This assignment involves building a basic text processing module that implements sentence segmentation, tokenization, stemming/lemmatization and stopword removal. We will also make use of the Cranfield dataset to meet the objectives of this assignment.**

I. The simplest top-down approach for sentence segmentation

The simplest way to do so would be to break sentences at key-punctuation characters like full-stop (.), question mark(?), exclamation mark (!), etc. These characters typically act as the endings of English sentences, and hence can be used as delimiters to separate sentences in a paragraph or collection of sentences.

II. Is the above approach always correct?

No, the punctuation based approach is not always effective for the task at hand. This naive approach fails in instances where "." is used for titles like Mr./Mrs./Dr. etc, or in the case of shortened names such as Johann S. Bach and in for urls such as www.overleaf.com. Examples of this approach failing are illustrated as follows:

- "Mr. Green is the culprit behind Mrs. White's murder." would get segmented into 3 sentences: ["Mr", "Green is the culprit behind Mrs", "White's murder"]
- "George W. Bush was the 43rd U.S. President" would get split into ["George W", "Bush was the 43rd U.S", "President"]
- "I like the resume templates on www.overleaf.com." would get split into ["I like the resume templates on www", "overleaf", "com"]

III. Punkt Sentence Tokenizer (NLTK) vs Simple Top-Down

The Punkt Sentence Tokenizer was built on the English text using an unsupervised algorithm to effectively segment sentences. It outperforms the previous approach on the following fronts:

1) It carefully recognises when "." is used for an abbreviation.
2) It takes into account other important features like collocations and words that typically start sentences.
3) It effectively deals with any parenthesis used in sentences.

Punkt is designed to learn parameters (a list of abbreviations, etc.) unsupervised from a corpus similar to the target domain. There's also a provision to train the tokenizer on a corpus of our choice, to derive more tailored results. [1]

IV. Sentence segmentation on the Cranfield dataset

A. Scenario where naive approach performs better than Punkt Tokenizer

When the text does not follow grammatical rules strictly and there is no space between two sentences, the naive approach splits them appropriately, while Punkt tokenizer tends to count them as a single sentence. For instance, "I ate breakfast.It was good" is split into ["I ate breakfast","It was good"] in the naive approach but stays as ["I ate breakfast.It was good"] when the Punkt tokenizer is used.

B. Scenario where Punkt Tokenizer performs better than naive approach

On performing sentence segmentation on the Cranfield dataset it was observed that the Punkt tokenizer recognized when the period '.' was used in abbreviations and not as a sentence ending. For example "i.e." was split into ["i","e"] in the naive approach, and not so when the tokenizer was used. The same goes for decimal numbers as well.

V. Simplest top-down approach to word tokenization

The simplest top-down approach to word tokenization would be to use any and all punctuations like period (.), comma (,), apostrophe ('), question-mark (?) etc. as delimiters. This is implemented by first using RegEx [2] to replace all non-alphanumeric characters with a space and then splitting the sentences using the input string *split()* function.

VI. Type of Knowledge Used by Penn Treebank Tokenizer

The Penn Treebank tokenizer uses regular expressions to tokenize text. Since the regular expressions are defined based on knowledge of the nature of the text (that words are separated by spaces or commas, that apostrophes indicate the shortening of a word, etc), this is a rule-based top-down approach. [3]

VII. Word Tokenization on the Sentence Segmented Documents

A. Scenario where naive approach performs better than Penn Treebank Tokenizer

The naive method outperforms the Penn Tokenizer in broadly 2 ways:

1) Typically this method is computationally less expensive as it doesn't make use of a corpus. This naive tokenizer will prove to be effective when we're working on big-data applications because it is significantly faster.
2) The naive approach does well in tokenizing words separated with a hyphen(-). Penn Treebank ignores hyphen's altogether. For example, naive tokenizer splits 'boundary-layer' into ['boundary','layer'], which will remain the same with Penn Treebank Tokenizer.
3) Penn Treebank Tokenizer fails to eliminate an ill-formatted periods(.) and commas(,). Typically, these characters are followed by space " " character, but ill-formatted documents may omit it. This is also observed in some cases within the Cranfield dataset. For example, "ratios.The" is split into ["ratios","."."the"].

B. Scenario where Penn Treebank Tokenizer performs better than naive approach

The naive method blindly splits words according to non alphanumeric characters. With the use of a corpus, the second method is better at recognising standard contractions in the English Language. For example the sentence "I can't eat egg today" is tokenized in the following way:

- naive method: ['I', 'can', 't', 'eat', 'egg', today']
- Penn Treebank method: ['I', 'ca', 'nt', 'eat', 'egg', today']

Penn's Treebank method will similarly perform better for other common contractions like wouldn't, shouldn't, etc.

VIII. Difference between stemming and lemmatization

Lemmatization and Stemming are Text Normalization techniques. These techniques are used to prepare words, text, and documents for further processing. Languages such as English, Hindi consists of several words which are often derived from one another. Further, Inflected Language is a term used for a language that contains derived words. For instance, word "historical" is derived from the word "history" and hence is the derived word. There is always a common root form for all inflected words. Further, degree of inflection varies from lower to higher depending on the language. To sum up, root form of derived or inflected words are attained using Stemming and Lemmatization.

Stemming is the process of removing the last few characters of a given word, to obtain a shorter form, even if that form doesn't have any meaning. For example, stemming would reduce the word "historical" to "histori", when the root form is actually "history".

Lemmatization comes into the picture to overcome stemming's drawback. Lemmatization takes more time as compared to stemming because it finds a meaningful word or representation. It makes use of a corpus to produce lemma (or the root word). Moreover, it also considers parts-of-speech to obtain a correct lemma for a word.

In an attempt to find the common base form of several variants of a word (the words eaten, eats, eater are related and have the root eat) stemming chops off letters from the end of the word until it reaches the stem, i.e. the base form. On the other hand, lemmatization uses a vocabulary and morphological analysis of words to return the *lemma* which is the base word (by analysing context, part of speech, etc). For example, for the word 'saw' (used in the verb sense and not the noun) stemming might return just s while lemmatization returns the root 'see'. [4]

IX. Which is better for Search Engine

Stemming and Lemmatization have their own pros and cons. Typically, it's a tradeoff between accuracy and computation time. Hence, the answer varies with the target language, specificity of domain, speed requirements and precision/recall recall standards of the target audience.

For general search engines on the scale of Google or Bing, handling large number of user requests at a high speed is the priority. The process of lemmatization would slow it down too much and hence stemming would be a more viable choice in this case.

For our use case, the Cranfield dataset is in English, is of fairly reasonable size and focused on aerospace engineering. Lemmatization is first built on a corpus to develop top-down model to link forms of words to their roots. Wordnet is an example, but it is trained on general purpose language, not specifically tuned for aerospace. Stemming on the other hand is able to reduce terms using simple rules and hence is domain agnostic. Finally, as stemming requires lesser compute, and can provide faster results, we have chosen stemming for our search engine.

X. Implementation of Stemming

We can implement stemming in multiple ways, by making use of the folllowing stemmers available in the nltk library:

- Porter Stemmer: the most popular stemming methods proposed in 1980 which is based on the idea that the suffixes in the English language are made up of a combination of smaller and simpler suffixes.

- Lovins Stemmer: It is proposed by Lovins in 1968, that removes the longest suffix from a word then the word is recorded to convert this stem into valid words.
- Krovetz Stemmer: It first converts the plural form of a word to its singular form before stemming. It also converts the past tense of a word to its present tense and removes the suffix 'ing'.

For our case, we implemented Porter Stemmer as it is the fastest, produces the best output as compared to other stemmers and also has a very less error rate.

XI. Stopword removal implementation

We implemented stopword removal iteratively, by checking if each word in the input query belongs to nltk library's list of stopwords. This was done by making using of python's handy list comprehension technique.

XII. Bottom-Up Approach to Stopword Removal

In any given corpus, not all words carry the same amount of information. Commonly occurring words that contain little meaningful information, or discriminative power, are called stopwords. Words like 'a', 'the', 'of', etc are examples of common stopwords in English since they typically don't add much to the meaning of the sentence, they exist to ensure that the structure of the sentence is sound. [5]

An intuitive bottom-up approach to generating a list of stopwords is to sort the types occurring in the corpus by their frequency (the total number of times a type occurs in the entire corpus) and consider the most frequent words as stopwords.

Alternately, we can use the Inverse Document Frequency (idf) to quantify the information contained in a word. It is defined as

$$idf_{type} = \ln\left(\frac{N}{n}\right)$$

where N is the total number of documents in the corpus and n is the number of documents that contain the type under consideration. A stopword list is made from the types that have the least idf and hence the least discriminating ability.

References

[1] Punkt Sentence Tokenizer. https://www.nltk.org/_modules/nltk/tokenize/punkt.html.
[2] Regular Expression Operations. https://docs.python.org/3/library/re.html.
[3] Penn Treebank Tokenizer. https://www.nltk.org/_modules/nltk/tokenize/treebank.html.
[4] Stemming and Lemmatization. https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html.
[5] Supervised Machine Learning for Text Analysis in R - Chapter 3 : Stopwords. https://smltar.com/stopwords.html.