CS6370: Natural Language Processing

Assignment 1 (Part-A)

Release Date: 20th Feb 2025 Deadline: 8th March 2025

Name: Roll No.:

|  |  |
| --- | --- |
| Jay Gupta | AE21B026 |

General Instructions:

1. This assignment consists of two parts: A and B. Part B will be released later with a separate deadline.
2. The template for the code (in Python) is provided in a separate zip file. You are expected to fill in the template wherever instructed. Note that any Python library, such as nltk, stanfordcorenlp, spacy, etc, can be used.
3. The programming questions for the Spell Check and WordNet parts need to be done in separate Python files.
4. A folder named ‘Roll\_number.zip’ that contains a zip of the code folder and your responses to the questions (a PDF of this document with the solutions written in the text boxes) must be uploaded on Moodle by the deadline.
5. You may discuss this assignment in a group, but the implementation must be completed individually and submitted separately.
6. Any submissions made after the deadline will not be graded.
7. Answer the theoretical questions concisely. All the codes should contain proper comments.
8. The institute’s academic code of conduct will be strictly enforced.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

The goal of this assignment is to build a search engine from scratch, which is an example of an Information Retrieval system. In the class, we have seen the various modules that serve as the building blocks for a search engine. We will be progressively building the same as the course progresses. This assignment requires you to build a basic text processing module that implements sentence segmentation, tokenization, stemming/lemmatization, spell check, and stopword removal. You will also explore some aspects of WordNet as a part of this assignment. The Cranfield dataset, which has been uploaded, will be used for this purpose.

Part 1: Sentence Segmentation [Theory + Implementation]

1. Suggest a simplistic top-down approach to sentence segmentation for English texts. Do you foresee issues with your proposed approach in specific situations? Provide supporting examples and possible strategies that can be adopted to handle these issues. [2 marks]

|  |
| --- |
| A simplistic **top-down** approach to **sentence segmentation** can be based on splitting text using common sentence-ending punctuation marks such as **periods (.), exclamation marks (!), and question marks (?).**          # Split the text using the delimiters '.', '!', '?'          split\_text = re.split(r'[.!?]\s', text)            # Remove empty strings and strip spaces from the sentences          segmentedText = [s.strip() for s in split\_text if s.strip()]  Consider the example:   * Dr. Smith went to Washington D.C. for a meeting.   The output using the above code will be:   * ['Dr', 'Smith went to Washington D.C', 'for a meeting']   The naive method mistakenly splits "Dr. Smith" and "Washington D.C. for a meeting"  This method might incorrectly split after abbreviations. |

1. Python NLTK is one of the most commonly used packages for Natural Language Processing. What does the Punkt Sentence Tokenizer in NLTK do differently from the simple top-down approach? [1 marks]

|  |
| --- |
| The **Punkt Sentence Tokenizer** in **NLTK** is a **machine learning-based, unsupervised sentence segmentation algorithm**. Unlike a simple top-down approach that **splits sentences based on fixed punctuation rules**, Punkt **learns patterns** in text to intelligently determine sentence boundaries.  Punkt uses unsupervised machine learning to understand the context in which punctuation appears. The algorithm estimates the likelihood that a punctuation mark is a sentence boundary based on the statistics of the text. This probabilistic approach makes it more robust compared to a deterministic, naive splitting method.  The output of punkt tokenizer for above exmaple is:   * ['Dr. Smith went to Washington D.C. for a meeting.']   Punkt recognizes abbreviations and does not split. |

1. Perform sentence segmentation on the documents in the Cranfield dataset using:
   1. The proposed top-down method and
   2. The pre-trained Punkt Tokenizer for English

State a possible scenario where

1. Your approach performs better than the Punkt Tokenizer
2. Your approach performs worse than the Punkt Tokenizer [4 marks]

|  |
| --- |
|  |

Part 2: Tokenization [Theory + Implementation]

1. Suggest a simplistic top-down approach for tokenization in English text. Identify specific situations where your proposed approach may fail to produce expected results. [2 marks]

|  |
| --- |
|  |

1. Study about NLTK’s Penn Treebank tokenizer. What type of knowledge does it use - Top-down or Bottom-up? [1 mark]

|  |
| --- |
|  |

1. Perform word tokenization of the sentence-segmented documents using
   1. The proposed top-down method and
   2. Penn Treebank Tokenizer

State a possible scenario along with an example where:

1. Your approach performs better than Penn Treebank Tokenizer
2. Your approach performs worse than Penn Treebank Tokenizer

[4 marks]

|  |
| --- |
|  |

Part 3: Stemming and Lemmatization [Theory + Implementation]

1. What is the difference between stemming and lemmatization? Give an example to illustrate your point. [1 marks]

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Both stemming and lemmatization are techniques used in Natural Language Processing (NLP) to reduce words to their base or root form. However, they differ in how they process words and the accuracy of the results.   |  |  |  | | --- | --- | --- | | Feature | Stemming | Lemmatization | | Definition | Reduces words to their root form by **removing suffixes**. | Converts words to their **base dictionary form** (lemma) using **linguistic rules**. | | Method | Uses **rule-based truncation** (e.g., removing "-ing", "-ed"). | Uses **morphological analysis** and a **lexicon** (dictionary). | | Accuracy | |  | | --- | |  |  |  | | --- | | Less accurate (may produce non  -existent words). | | More accurate (returns valid dictionary words). | | Speed | |  | | --- | |  |  |  | | --- | | Faster (just removes suffixes). | | Slower (requires **context & lookup** in a dictionary). |   **Input Words:**  ["running", "better", "flies", "studies", "played"]  Stemming Output:  ['run', 'better', 'fli', 'studi', 'play']  Lemmatization Output:  ['run', 'good', 'fly', 'study', 'play'] |

1. Using Porter's stemmer, perform stemming/lemmatization on the word-tokenized text from the Cranfield Dataset. [1 marks]

|  |
| --- |
|  |

Part 4: Stopword Removal [Theory + Implementation]

1. Remove stopwords from the tokenized documents using a curated list, such as the list of stopwords from NLTK. [1 marks]

|  |
| --- |
|  |

1. Can you suggest a bottom-up approach for creating a list of stopwords specific to the corpus of documents? [1 marks]

|  |
| --- |
| Stopwords are commonly used words in a language that do not carry significant meaning and are often filtered out in Natural Language Processing (NLP) tasks to improve efficiency and relevance.  Steps to Build a Corpus-Specific Stopword List:   1. Tokenize the Corpus: Split text into words. 2. Compute Word Frequencies: Count occurrences of each word. 3. Identify High-Frequency Words: Words that appear in a large percentage of documents (e.g., >80%) are likely stopwords. 4. Filter Out Domain-Specific Terms: Stopwords should be general and **not** domain-specific (e.g., in medical text, “patient” may be frequent but **not** a stopword). |

1. Implement the strategy proposed in the previous question and compare the stopwords obtained with those obtained from NLTK on the Cranfield dataset. [2 marks]

|  |
| --- |
|  |

Part 5: Retrieval [Theory]

1. Given a set of queries Q and a corpus of documents D, what would be the number of computations involved in estimating the similarity of each query with every document? Assume you have access to the TF-IDF vectors of the queries and documents over the vocabulary V. [1 marks]

|  |
| --- |
|  |

1. Suggest how the idea of the inverted index can help reduce the time complexity of the approach in (2). You can introduce additional variables as needed. [3 marks]

|  |
| --- |
|  |