Assignment 1

CS6370 - Natural Language Processing

Amogh Prabhunanda Patil - EE19B134
Abhigyan Chattopadhyay - EE19B146
Swathi G - CH19B094

July 24, 2022

1 Dataset

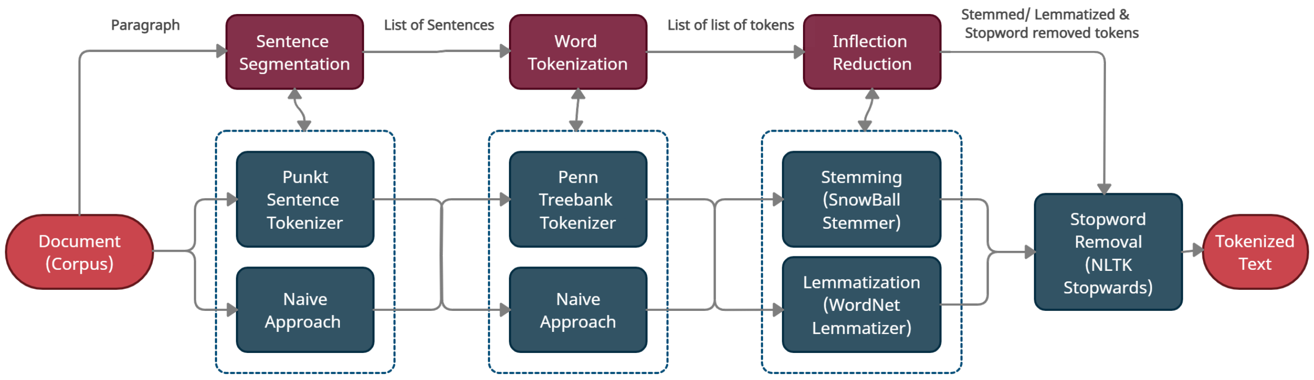

Figure 1: Flow of Data

The given dataset contains 3 files, cran_docs.json, cran_qrels.json, and cran_queries.json.

The documents form the corpus, the queries are what are used to retreive the data from the corpus, and the qrels file contains metadata regarding the documents and the queries.

From the README:

"The Cranfield dataset contains 1400 documents (cran_docs.json), 225 queries (cran_queries.json), and query-document relevance judgements (cran_qrels.json)."

More on the Cranfield dataset can be found here.

2 Sentence Segmentation

1. What is the simplest and obvious top-down approach to sentence segmentation for English texts?

A simple approach would be to detect either a period(“.”), question mark or an exclamation mark and create sentences by splitting the text using these delimiters.

2. Does the top-down approach (your answer to the above question) always do correct sentence segmentation? If Yes, justify. If No, substantiate it with a counter example.

Our simple top down approach cannot handle situations where given a sentence which ends with a period followed by another sentence without a space (for example “Cats are cute.So are dogs.”). The period is treated either as a decimal point or the words around the period are treated as acronyms as seen in words like Mr. or i.e.

3. 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? You can read about the tokenizer here.

The Punkt Tokenizer uses an Unsupervised Learning Algorithm to learn various features, such as a list of various abbreviations used in the corpus.

Its most important feature is that it doesn’t simply segment sentences based on punctuation marks and also doesn’t ignore them entirely. It retains punctuation as much as applicable, while separating out individual sentences.

4. Perform sentence segmentation on the documents in the Cranfield dataset using:

1. **The top-down method stated above**
2. **The pre-trained Punkt Tokenizer for English**

The code for the same is attached in the submission.

State a possible scenario along with an example where:

1. **the first method performs better than the second one (if any)**
2. **the second method performs better than the first one (if any)**

1. There isn’t any scenario in this dataset where the first method was able to outperform the second.
2. In case of abbreviations that use period signs, the punkt method was able to understand that it didn’t represent the end of a sentence and hence worked better. It was also able to work with multiple brackets and didn’t remove any punctuation that might otherwise have some meaning.

3 Word Tokenization

5. What is the simplest top-down approach to word tokenization for English texts?

The simplest top-down approach to word tokenization is identifying common delimiters in sentences and creating tokens based on splitting the sentences using that delimiter. With the knowledge of the language, we know that the words are always separated by spaces. So, just splitting the sentence based on spaces is the simplest top-down approach. However, we have cleaned the text before splitting the sentence to words.

We have first removed (') and join the word with s. For example, prandtl's turns into prandtls making it a single word. We have then removed all punctuation in the sentence except, letters, numbers, '.', '=', and '-' and replaced it with space. We have retained '.' to account for decimals (2.39, 10.8, etc) and latin abbreviations (i.e, viz.). Then, all multiple spaces and tabs are replaced with single space. The cleaned data string is then split into words with ' ' as the delimiter.

6. Study about NLTK's Penn Treebank tokenizer here. What type of knowledge does it use - Top-down or Bottom-up?

Penn Treebank tokenizer uses regular expressions to tokenize text similar to the tokenization used in the Penn Treebank. It is one of the largest syntactically annotated corpora (treebank) available. Because the sentences are parsed syntactically, knowledge about the language is necessary and hence, it is a Top-down approach

7. Perform word tokenization of the sentence-segmented documents using

1. **The simple method stated above**
2. **Penn Treebank Tokenizer**

The code for the same is attached in the submission.

State a possible scenario along with an example where

1. **the first method performs better than the second one (if any)**
2. **the second method performs better than the first one (if any)**

The first method performs better than the second one in cases where there are "," and "." as they are identified as separate tokens which doesn't contribute much.

The second method performs better than the first one in cases like handling common English contractions (') as prandtl's turns into (prandtl) + ('s) and splits them. It also splits commas and single quotes off from words, when they are followed by whitespace and periods that occur at the end of the sentence. But using a specific kind of tokenizer depends on the use case considering the dataset at hand.

4 Stemming vs Lemmatization

8. What is the difference between stemming and lemmatization?

The goal of both stemming and lemmatization is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form.

Stemming is a blind approach that converts the input words to stems based on their endings based on pre-programmed rules.

Lemmatization on the other hand, uses the word roots and context-based parts of speech to provide similar words rather than just a cut-off part of the input token. This process is more general and accurate, but comes at the cost of speed.

In the end, if we want to build a search engine that is highly specific and specialized for a given field and we have a (relatively) large dataset, we can use stemming and incur minimal losses in accuracy.

On the other hand, if we are designing a generalized search engine for handling many different kinds of fields and topics, and have a smaller dataset to handle, using lemmatization would be far better due to it being more accurate and not incurring much penalty in speed.

So essentially, in short, go with stemming when the vocabulary space is small and the documents are large. Conversely, go with word embeddings when the vocabulary space is large but the documents are small.

9. For the search engine application, which is better? Give a proper justification to your answer. This is a good reference on stemming and lemmatization.

The dataset given to us is extremely specific for aerodynamics and contains a lot of tokens specific to that field. As a result, using stemming would give us decent results.

However, since the dataset is not too large and has only 1400 samples in the documents, and 225 queries, we can also use lemmatization without incurring much of a speed penalty. This will also enable us to simplify our vectorization step going down the line as many words will get simplified to the same word roots

In our case, we have used Stemming as it is faster, and our dataset is highly specific to one type of inputs. We are assuming that we are building an information retrieval system for this specific dataset and not a generalized system.

In case we wanted to make a more generic system, using lemmatization would make sense, but since we have a small and specific dataset, using lemmatization would be using up too much time and the benefits gained from using it would be outweighed by the loss in speed.

10. Perform stemming/lemmatization (as per your answer to the previous question) on the word-tokenized text.

The code for the same is attached in the submission.

5 Stopword Removal

11. Remove stopwords from the tokenized documents using a curated list of stopwords (for example, the NLTK stopwords list).

Stop words are generally the most common words in a language. Example, the, what, where, how, has, had etc. These are filtered out before processing of natural language data.

Generally, these don't add any value to vectors and hence are always removed before the words are fed into encoders.

We used the NLTK stopwords list to filter them out via a double for loop in the code.

Otherwise, a number of stop words can be extracted to a feature column that can provide useful information about the structuring of sentences in cases of labelled data.

12. In the above question, the list of stopwords denotes top-down knowledge. Can you think of a bottom-up approach for stopword removal?

Using a pre-existing list of stopwords allows us to remove commonly seen words across the entire language, and is not curated based on the actual corpus.

However, there is no customization based on the corpus, which gives us room for us to go through the corpus and remove words that appear very commonly, using a metric like Rank Frequency Distribution. Zipf's law is one such metric measure we found while searching for bottom-up approach. It is used highly in quantitative linguistics and given a corpus, it gives a relation that the frequency of any word is inversely proportional to its rank in the frequency table. Therefore, the most frequent word will occur approximately twice as often as the second most frequent word, three times as often as the third most frequent word, etc, making the frequency distribuition very pronounced.

We can also start with bigrams or n-grams with words and remove the bigrams which occur very frequently. This will create a very skewed frequency distribution of the words and easily help us identify stopwords.

If we use a system like **Part of Speech Tagging** (POS Tagging), that would enable us to easily remove articles and prepositions, pronouns, particles and auxillary verbs. This would in turn remove a lot of stopwords with minimal effort.

A combination of top-down and bottom-up approaches would be the best as it would incorporate the best of both worlds.

6 References

1. Automatically Building a Stopword List for an Information Retrieval System, Rachel Tsz-Wai Lo, Ben He, Iadh Ounis http://terrierteam.dcs.gla.ac.uk/publications/rtlo_DIRpaper.pdf

2. Porter Stemmer http://people.scs.carleton.ca/~armyunis/projects/KAPI/porter.pdf

3. Snowball Stemmer NLP https://www.geeksforgeeks.org/snowball-stemmer-nlp/

4. Stemming vs Lemmatization https://www.baeldung.com/cs/stemming-vs-lemmatization

5. NLTK Penn Treebank Word Tokenizer https://www.nltk.org/_modules/nltk/tokenize/treebank.html#TreebankWordTokenizer

6. NLTK Lemmatization Implementation through WordNet https://wordnet.princeton.edu/documentation/morphy7wn