Introduction to Stemming

Stemming is a method in **text processing** that eliminates prefixes and suffixes from words, transforming them into their fundamental or root form, The main objective of stemming is to streamline and standardize words, enhancing the effectiveness of the **natural language processing** tasks. The article explores more on the stemming technique and how to perform stemming in Python.

What is Stemming in NLP?

Simplifying words to their most basic form is called stemming, and it is made easier by stemmers or stemming algorithms. For example, “chocolates” becomes “chocolate” and “retrieval” becomes “retrieve.” This is crucial for pipelines for natural language processing, which use tokenized words that are acquired from the first stage of dissecting a document into its constituent words.

Stemming in natural language processing reduces words to their base or root form, aiding in text normalization for easier processing. This technique is crucial in tasks like text classification, information retrieval, and text summarization. While beneficial, stemming has drawbacks, including potential impacts on text readability and occasional inaccuracies in determining the correct root form of a word.

Why is Stemming important?

It is important to note that stemming is different from Lemmatization. Lemmatization is the process of reducing a word to its base form, but unlike stemming, it takes into account the context of the word, and it produces a valid word, unlike stemming which may produce a non-word as the root form.

> **Note:** Do must go through concepts of ‘tokenization.’

```
 Some more example of stemming for root word "like" include:
->"likes"
->"liked"
->"likely"
->"liking"
```

Types of Stemmer in NLTK

Python NLTK contains a variety of stemming algorithms, including several types. Let’s examine them down below.

1. Porter's Stemmer

It is one of the most popular stemming methods proposed in 1980. It is based on the idea that the suffixes in the English language are made up of a combination of smaller and simpler suffixes. This stemmer is known for its speed and simplicity. The main applications of Porter Stemmer include data mining and Information retrieval. However, its applications are only limited to English words. Also, the group of stems is mapped on to the same stem and the output stem is not necessarily a meaningful word. The algorithms are fairly lengthy in nature and are known to be the oldest stemmer.

Example: EED → EE means "if the word has at least one vowel and consonant plus EED ending, change the ending to EE" as 'agreed' becomes 'agree'.

Implementation of Porter Stemmer

Python3

```
from nltk.stem import PorterStemmer

porter_stemmer = PorterStemmer()

words = [``"running"``, "jumps"``, "happily"``, "running"``, "happily"``]

stemmed_words = [porter_stemmer.stem(word) for word in words]

print``(``"Original words:"``, words)

print``(``"Stemmed words:"``, stemmed_words)
```

Output:

```
Original words: ['running', 'jumps', 'happily', 'running', 'happily']
Stemmed words: ['run', 'jump', 'happili', 'run', 'happili']
```

- **Advantage:** It produces the best output as compared to other stemmers and it has less error rate.
- **Limitation:** Morphological variants produced are not always real words.

2. 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.
Example: sitting → sitt → sit

- **Advantage:** It is fast and handles irregular plurals like 'teeth' and 'tooth' etc.
- **Limitation:** It is time consuming and frequently fails to form words from stem.

3. Dawson Stemmer

It is an extension of Lovins stemmer in which suffixes are stored in the reversed order indexed by their length and last letter.

- **Advantage:** It is fast in execution and covers more suffices.
- **Limitation:** It is very complex to implement.

4. Krovetz Stemmer

It was proposed in 1993 by Robert Krovetz. Following are the steps:

1. Convert the plural form of a word to its singular form.
2. Convert the past tense of a word to its present tense and remove the suffix 'ing'.
   Example: 'children' → 'child'

- **Advantage:** It is light in nature and can be used as pre-stemmer for other stemmers.
- **Limitation:** It is inefficient in case of large documents.

5. Xerox Stemmer

Capable of processing extensive datasets and generating valid words, it has a tendency to over-stem, primarily due to its reliance on lexicons, making it language-dependent. This constraint implies that its effectiveness is limited to specific languages.

Example:

'children' → 'child'

'understood' → 'understand'

'whom' → 'who'

'best' → 'good'

6. N-Gram Stemmer

The algorithm, aptly named n-grams (typically n=2 or 3), involves breaking words into segments of length n and then applying statistical analysis to identify patterns. An n-gram is a set of n consecutive characters extracted from a word in which similar words will have a high proportion of n-grams in

common.
Example: 'INTRODUCTIONS' for n=2 becomes : *I, IN, NT, TR, RO, OD, DU, UC, CT, TI, IO, ON, NS, S*

- **Advantage:** It is based on string comparisons and it is language dependent.
- **Limitation:** It requires space to create and index the n-grams and it is not time efficient.

7. Snowball Stemmer

The Snowball Stemmer, compared to the Porter Stemmer, is multi-lingual as it can handle non-English words. It supports various languages and is based on the 'Snowball' programming language, known for efficient processing of small strings.

The Snowball stemmer is way more aggressive than Porter Stemmer and is also referred to as Porter2 Stemmer. Because of the improvements added when compared to the Porter Stemmer, the Snowball stemmer is having greater computational speed.

Implementation of Snowball Stemmer

Python3

```
from nltk.stem import SnowballStemmer

stemmer = SnowballStemmer(language``=``'english'``)

words_to_stem = [``'running'``, 'jumped'``, 'happily'``, 'quickly'``, 'foxes'``]

stemmed_words = [stemmer.stem(word) for word in words_to_stem]

print``(``"Original words:"``, words_to_stem)

print``(``"Stemmed words:"``, stemmed_words)
```

Output:

```
Original words: ['running', 'jumped', 'happily', 'quickly', 'foxes']
Stemmed words: ['run', 'jump', 'happili', 'quick', 'fox']
```

8. Lancaster Stemmer

The Lancaster stemmers are more aggressive and dynamic compared to the other two stemmers. The stemmer is really faster, but the algorithm is really confusing when dealing with small words. But they are not as efficient as Snowball Stemmers. The Lancaster stemmers save the rules externally and basically uses an iterative algorithm.

Implementation of Lancaster Stemmer

Python3

```
from nltk.stem import LancasterStemmer

stemmer = LancasterStemmer()

words_to_stem = [``'running'``, 'jumped'``, 'happily'``, 'quickly'``, 'foxes'``]

stemmed_words = [stemmer.stem(word) for word in words_to_stem]

print``(``"Original words:"``, words_to_stem)

print``(``"Stemmed words:"``, stemmed_words)
```

Output:

```
Original words: ['running', 'jumped', 'happily', 'quickly', 'foxes']
Stemmed words: ['run', 'jump', 'happy', 'quick', 'fox']
```

9. Regexp Stemmer

The Regexp Stemmer, or Regular Expression Stemmer, is a stemming algorithm that utilizes regular expressions to identify and remove suffixes from words. It allows users to define custom rules for stemming by specifying patterns to match and remove.

This method provides flexibility and control over the stemming process, making it suitable for specific applications where custom rule-based stemming is desired.

Implementation of Regexp Stemmer

Python3

```
from nltk.stem import RegexpStemmer

custom_rule = r``'ing$'

regexp_stemmer = RegexpStemmer(custom_rule)

word = 'running'

stemmed_word = regexp_stemmer.stem(word)
```

```
print``(f``'Original Word: {word}'``)

print``(f``'Stemmed Word: {stemmed_word}'``)
```

Output:

```
Original Word: running
Stemmed Word: runn
```

Applications of Stemming

1. Stemming is used in information retrieval systems like search engines.
2. It is used to determine domain vocabularies in domain analysis.
3. To display search results by indexing while documents are evolving into numbers and to map documents to common subjects by stemming.
4. Sentiment Analysis, which examines reviews and comments made by different users about anything, is frequently used for product analysis, such as for online retail stores. Before it is interpreted, stemming is accepted in the form of the text-preparation mean.
5. A method of group analysis used on textual materials is called document clustering (also known as text clustering). Important uses of it include subject extraction, automatic document structuring, and quick information retrieval.

Disadvantages in Stemming

There are mainly two errors in stemming –

- **Over-stemming**: Over-stemming in natural language processing occurs when a stemmer produces incorrect root forms or non-valid words. This can result in a loss of meaning and readability. For instance, “arguing” may be stemmed to “argu,” losing meaning. To address this, choosing an appropriate stemmer, testing on sample text, or using lemmatization can mitigate over-stemming issues. Techniques like semantic role labeling and sentiment analysis can enhance context awareness in stemming.
- **Under-stemming**: Under-stemming in natural language processing arises when a stemmer fails to produce accurate root forms or reduce words to their base form. This can result in a loss of information and hinder text analysis. For instance, stemming “arguing” and “argument” to “argu” may lose meaning. To mitigate under-stemming, selecting an appropriate stemmer, testing on sample text, or opting for lemmatization can be beneficial. Techniques like semantic role labeling and sentiment analysis enhance context awareness in stemming.

Advantages of Stemming

Stemming in natural language processing offers advantages such as text normalization, simplifying word variations to a common base form. It aids in information retrieval, text mining, and reduces feature dimensionality in machine learning. Stemming enhances computational efficiency, making it a valuable step in text pre-processing for various NLP applications.

Frequently Asked Questions (FAQs)

1. What is stemming in NLP?

> Stemming is a text normalization technique in natural language processing that reduces words to their base or root form, removing prefixes or suffixes.

2. How does stemming differ from lemmatization?

> While both aim to reduce words, stemming may produce non-real words, whereas lemmatization ensures valid words by considering the context.

3. What are common stemming algorithms?

> Popular stemming algorithms include Porter Stemmer, Snowball Stemmer, Lancaster Stemmer, and Regexp Stemmer.

4. What issues can arise from over-stemming?

> Over-stemming occurs when too many words share the same stem, leading to loss of meaning and readability in the text.