**EXPERIMENT NO.3**

**AIM:** Stemming of text

**RESOURCES REQUIRED:**

Python 3, NLTK toolkit, Text editor, 4 GB RAM and above, i5 processor and above

**THEORY:**

**STEMMING:**

Stemming is the process of reducing a word to its word stem that affixes to suffixes and prefixes or to the roots of words known as a lemma. Stemming is important in natural language understanding (NLU) and natural language processing (NLP).

Stemming is a part of linguistic studies in morphology and artificial intelligence (AI) information retrieval and extraction. Stemming and AI knowledge extract meaningful information from vast sources like big data or the Internet since additional forms of a word related to a subject may need to be searched to get the best results. Stemming is also a part of queries and Internet search engines.

Recognizing, searching and retrieving more forms of words returns more results. When a form of a word is recognized it can make it possible to return search results that otherwise might have been missed. That additional information retrieved is why stemming is integral to search queries and information retrieval.

When a new word is found, it can present new research opportunities. Often, the best results can be attained by using the basic morphological form of the word: the lemma. To find the lemma, stemming is performed by an individual or an algorithm, which may be used by an AI system. Stemming uses a number of approaches to reduce a word to its base from whatever inflected form is encountered.

It can be simple to develop a stemming algorithm. Some simple algorithms will simply strip recognized prefixes and suffixes. However, these simple algorithms are prone to error. For example, an error can reduce words like laziness to lazi instead of lazy. Such algorithms may also have difficulty with terms whose inflectional forms don't perfectly mirror the lemma such as with saw and see.

Examples of stemming algorithms include:

Lookups in tables of inflected forms of words. This approach requires all inflected forms be listed.

Suffix strippi . Algorithms recognize known suffixes on inflected words and remove them.

PORTER STEMMER:

A consonant in a word is a letter other than A, E, I, O or U, and other than Y preceded by a consonant. (The fact that the term **consonant** is defined to some extent in terms of itself does not make it ambiguous.) So in TOY the consonants are T and Y, and in SYZYGY they are S, Z and G. If a letter is not a consonant it is a vowel.

A consonant will be denoted by c, a vowel by v. A list ccc... of length greater than 0 will be denoted by C, and a list vvv... of length greater than 0 will be denoted by V. Any word, or part of a word, therefore has one of the four forms:

* CVCV ... C
* CVCV ... V
* VCVC ... C
* VCVC ... V

These may all be represented by the single form  
  
[C]VCVC ... [V]  
  
where the square brackets denote arbitrary presence of their contents. Using (VCmVCm) to denote VC repeated m times, this may again be written as  
  
[C](VCmVCm)[V]  
  
m will be called the measure of any word or word part when represented in this form. The case m = 0 covers the null word. Here are some examples:

* m=0 TR, EE, TREE, Y, BY.
* m=1 TROUBLE, OATS, TREES, IVY.
* m=2 TROUBLES, PRIVATE, OATEN, ORRERY.

The rules for removing a suffix will be given in the form  
  
(condition) S1 -> S2  
  
This means that if a word ends with the suffix S1, and the stem before S1 satisfies the given condition, S1 is replaced by S2. The condition is usually given in terms of m, e.g.  
  
(m > 1) EMENT ->  
  
Here S1 is 'EMENT' and S2 is null. This would map REPLACEMENT to REPLAC, since REPLAC is a word part for which m = 2.

The 'condition' part may also contain the following:

* \*S - the stem ends with S (and similarly for the other letters).
* \*v\* - the stem contains a vowel.
* m=2 TROUBLES, PRIVATE, OATEN, ORRERY.
* \*d - the stem ends with a double consonant (e.g. -TT, -SS).
* \*o - the stem ends cvc, where the second c is not W, X or Y (e.g. -WIL, -HOP).

And the condition part may also contain expressions with and, or and not, so that:

(m>1 and (\*S or \*T)) : tests for a stem with m>1 ending in S or T, while

(\*d and not (\*L or \*S or \*Z)) : tests for a stem ending witha double consonant other than L, S or Z. Elaborate conditions like this are required only rarely.

In a set of rules written beneath each other, only one is obeyed, and this will be the one with the longest matching S1 for the given word. For example, with

* SSES -> SS
* IES -> I
* SS -> SS
* S ->

(here the conditions are all null) CARESSES maps to CARESS since SSES is the longest match for S1. Equally CARESS maps to CARESS (S1=SS) and CARES to CARE (S1=S).

**CONCLUSION:**

Stemming is a text pre-processing task used in natural language processing. Stemming is the process of reducing words to their root form or stem. Stemming is useful to simplify text analysis in large corpora. A very common Stemming algorithm is the Porter Stemmer algorithm which has been implemented using the nltk toolkit.

**CODE:**

from random import choice

from nltk import word\_tokenize

from nltk.corpus import brown

from nltk.stem.porter import PorterStemmer

samples = choice(brown.paras(categories="humor"))

corpus = " ".join([" ".join(sample) for sample in samples])

print(f"Original corpus :\n{corpus}\n")

tokens = word\_tokenize(corpus)

print(f"Tokenized words : \n{tokens}\n")

porter = PorterStemmer()

stem\_words = [porter.stem(stem) for stem in tokens]

print(f"Stemmed words :\n{stem\_words}")

**OUTPUT:**

****