Experiment No. 3

Title: Implementation of Stemming of Text using different stemming modules in NLTK

Batch: 1 **Roll No.: 16010420008** **Experiment No.:3**

Aim: To implement stemming of Text using different stemming modules in NLTK

Resources needed: Text Editor, Python Interpreter

Theory:

After stop words removal, the next step is NLP pipeline stemming.

A stemming algorithm is a computational procedure which reduces all words with the same root, to a common form, usually by stripping each word of its derivational and inflectional suffixes.

For example, ‘boat’ is the root of the words: ‘boats’, ‘boater’, or ‘boating’.In stem, the root of the word is called the stem. So, in this case, ‘boat’ is called a stem.

NLTK provides many inbuilt stemmers such as the Porter Stemmer, Snowball Stemmer, Lancaster Stemmer and Regexp Stemmer

1. **Porter Stemmer**

It is one of the most commonly used stemmers, developed by M.F. Porter in 1980. Porter’s stemmer consists of five different phases. These phases are applied sequentially. Within each phase, there are certain conventions for selecting rules. The entire porter algorithm is small and thus fast and simple. The drawback of this stemmer is that it supports only the English language, and the stem obtained may or may not be linguistically correct.

Example:

```
from nltk.stem import PorterStemmer
porter = PorterStemmer()
words=
['Connects','Connecting','Connections','Connected','Connection','Con
nect']
for word in words:
    print(word,"--->",porter.stem(word))
```

Output :

```
Connects ---> connect
Connecting ---> connect
Connections ---> connect
Connected ---> connect
Connection ---> connect
Connect ---> connect
```

2. Snowball Stemmer

M.F. Porter also developed snowball stemmer. Snowball is a string processing language that is mainly developed to create stemming algorithms. It was created by Porter as an improvement over his previously created porter algorithm. It supports multiple languages, including English, Russian, Danish, French, Finnish, German, Italian, Hungarian, Portuguese, Norwegian, Swedish, and Spanish. The snowball stemmer presenting the English language stemmer is called Porter2.

According to M.F. Porter, the stemming of stopwords like ‘being’ to ‘be’ is useless because they don’t have any shared meaning, although there could be a grammatical connection between these two words. Snowball provides another parameter called ignore_stopwords, which is set to false by default. If it is set to true, then snowball will not perform the stemming of stopwords.

Example:

```
from nltk.stem  import  SnowballStemmer
snowball = SnowballStemmer(language='english')
words = ['generous','generate','generously','generation']
for word in words:
    print(word,"--->",snowball.stem(word))
```

Output :

```
generous ---> generous
generate ---> generat
generously ---> generous
generation ---> generat
```

3. Lancaster Stemmer

Lancaster stemmer is also called Paice or Husk stemmer. It was developed by C.D. Paice at Lancaster University in 1990. It uses an iterative approach, and this makes it the most aggressive algorithm among the three stemmers described. Due to its iterative approach, it may lead to over-stemming, which may result in the linguistically incorrect roots. It is not as efficient as a porter or snowball stemmer. Also, it only supports the English language.

Example:

```
from nltk.stem import LancasterStemmer
lancaster = LancasterStemmer()
words = ['eating','eats','eaten','puts','putting']
for word in words:
    print(word,"--->",lancaster.stem(word))
```

Output :

```
eating ---> eat
eats ---> eat
eaten ---> eat
puts ---> put
putting ---> put
```

4. **Regexp Stemmer**

Regex stemmer identifies morphological affixes using regular expressions. Substrings matching the regular expressions will be discarded

Example:

```
from nltk.stem import RegexpStemmer
regexp = RegexpStemmer('ing$|s$|e$|able$', min=4)
words = ['mass','was','bee','computer','advisable']
for word in words:
    print(word,"--->",regexp.stem(word))
```

Output :

```
mass ---> mas
was ---> was
bee ---> bee
computer ---> computer
advisable ---> advis
```

Activity:

1. Apply Porter Stemmer, Snowball Stemmer, Lancaster Stemmer and Regexp Stemmer to a large corpus of Text.
2. Write down the observations about differences in the working of each of the Stemmer by analyzing the result obtained after applying each Stemmer on large corpus of text

---

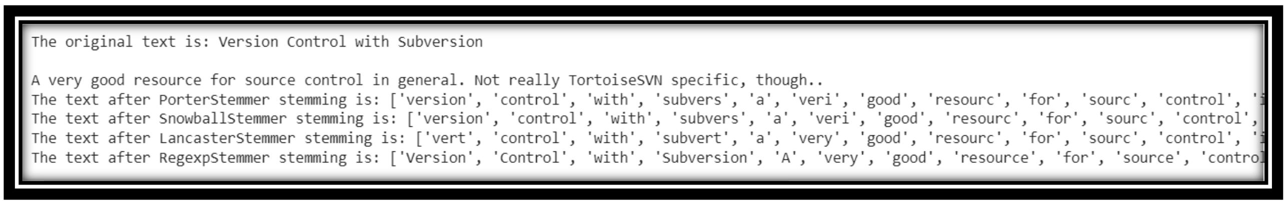
Results: (Program with snapshot of output)
Code:

```
import pandas as pd
import re
import nltk
nltk.download('punkt')
from nltk import word_tokenize
from nltk.stem import PorterStemmer , SnowballStemmer , LancasterStemmer ,
RegexpStemmer
```

```
csv_file = pd.read_csv('/content/Answers.csv')
csv_file_column = csv_file[csv_file['Body'].notna()]

answer_column = []
for i in range(0, 1):
    answer_column.append(csv_file_column['Body'][i])

def html_removal(text):
    pattern = r"<[^>]+>"
    html_removal_text = re.sub(pattern, "", text)
    return html_removal_text

def porterstemmer(text):
    porter = PorterStemmer()
    stemmed = []
    for i in text:
        stemmed.append(porter.stem(i))
    return stemmed

def snowballstemmer(text):
    snowball = SnowballStemmer(language='english')
    stemmed = []
    for i in text:
        stemmed.append(snowball.stem(i))
    return stemmed

def lancasterstemmer(text):
    lancaster = LancasterStemmer()
    stemmed = []
    for i in text:
        stemmed.append(lancaster.stem(i))
    return stemmed

def regexpstemmer(text):
    regexp = RegexpStemmer('ing$|s$|es$|able$')
    stemmed = []
    for i in text:
        stemmed.append(regexp.stem(i))
    return stemmed

for i in answer_column:
    html_removal_text = html_removal(i)
    print(f"The original text is: {html_removal_text}.")
```

```
tokens = word_tokenize(html_removal_text)
porterstemmed = porterstemmer(tokens)
print(f"The text after PorterStemmer stemming is: {porterstemmed}.")
snowballstemmed = snowballstemmer(tokens)
print(f"The text after SnowballStemmer stemming is: {snowballstemmed}.")
lancasterstemmed = lancasterstemmer(tokens)
print(f"The text after LancasterStemmer stemming is: {lancasterstemmed}.")
regexpstemmed = regexpstemmer(tokens)
print(f"The text after RegexpStemmer stemming is: {regexpstemmed}.")
```

Output:



```
The original text is: Version Control with Subversion

A very good resource for source control in general. Not really TortoiseSVN specific, though..
The text after PorterStemmer stemming is: ['version', 'control', 'with', 'subvers', 'a', 'veri', 'good', 'resourc', 'for', 'sourc', 'control', 'i
The text after SnowballStemmer stemming is: ['version', 'control', 'with', 'subvers', 'a', 'veri', 'good', 'resourc', 'for', 'sourc', 'control',
The text after LancasterStemmer stemming is: ['vert', 'control', 'with', 'subvert', 'a', 'very', 'good', 'resourc', 'for', 'sourc', 'control', 'i
The text after RegexpStemmer stemming is: ['Version', 'Control', 'with', 'Subversion', 'A', 'very', 'good', 'resource', 'for', 'source', 'contro
```

Observations:

PorterStemmer:
1) Words are stemmed to their root form, e.g., "version" becomes "version," "control" becomes "control," and "specific" becomes "specif."
2) It tends to remove common word endings and suffixes but does not always produce words in their dictionary form.

SnowballStemmer:
1) Produces results similar to PorterStemmer with some variations.
2) "General" is stemmed to "gener" instead of "genera," and "really" is stemmed to "realli" instead of "real."
3) It is a more modern and extended version of the PorterStemmer, aiming to improve stemming performance for various languages.

LancasterStemmer:
1) This stemmer produces more aggressive stemming, often resulting in shorter stems.
2) Words like "version" become "vert," "control" becomes "control," and "specific" becomes "spec."
3) It can be too aggressive for some applications, potentially leading to the loss of meaningful parts of words.

RegexpStemmer:
1) Unlike the other stemmers, this one does not perform typical English stemming.
2) It seems to focus on capitalization and punctuation, e.g., "Version" remains "Version," "Control" remains "Control," and "specific" remains "specific."
3) It does not follow traditional stemming rules, making it suitable for specific cases where you want to target particular patterns.

Questions:

1. Explain Under-Stemming and Over-Stemming errors with the help of suitableexample.

A) Under-Stemming:

Under-stemming occurs when a stemming algorithm is too conservative, and it fails to reduce a word to its root form adequately. As a result, similar words with different inflections are not reduced to the same stem.

Example of Under-Stemming:

Original Words: "jumping," "jumps," "jumped," "jumper"

Stemmed Words (under-stemming): "jump," "jump," "jump," "jumper"

In this case, the stemming algorithm failed to properly reduce "jumping," "jumps," and "jumped" to the common root "jump." It also incorrectly treated "jumper" as a different word. Under-stemming can lead to issues in information retrieval and text analysis tasks where related words should be grouped together.

Over-Stemming:

Over-stemming occurs when a stemming algorithm is too aggressive and reduces words to a root form that is too generic. This can lead to the loss of meaning and distinctions between words.

Example of Over-Stemming:

Original Words: "universe," "university," "universal"

Stemmed Words (over-stemming): "univers," "univers," "univers"

In this case, the stemming algorithm over-stems by reducing "universe," "university," and "universal" to the same root "univers." This leads to the loss of important distinctions between these words. Over-stemming can result in poor retrieval precision and the loss of context in text analysis.

Outcomes:

CO2: Comprehend Words and Word Forms in NLP

Conclusion: (Conclusion to be based on the outcomes achieved)

We understood the concept of Stemming and the various types of Stemming modules available with NLTK. We also implemented on some of the types of Stemming and observed the variations.

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of faculty in-charge with date

References:

-Books/ Journals/ Websites:

1. Allen.James, Natural Language Understanding, Benjamin Cumming, Second Edition, 1995
2. Jurafsky, Dan and Martin, James, Speech and Language Processing, Prentice Hall, 2008
3. Palash Goyal, Karan Jain, Sumit Pandey,Deep Learning for Natural Language

Processing: Creating Neural Networks with Python, Apress, 2018