Experiment No. 6

Title: Lemmatization of Text using NLTK

Batch: 1 **Roll No.: 16010420008** **Experiment No.:6**

Aim: To implement lemmatization of Text using NLTK

Resources needed: Text Editor, Python Interpreter

Theory:

Lemmatization is the algorithmic process of finding the lemma of a word depending on their meaning. Lemmatization usually refers to the morphological analysis of words, which aims to remove inflectional endings. It helps in returning the base or dictionary form of a word, which is known as the lemma.
The NLTK Lemmatization method is based on WordNet's built-in morph function. To perform lemmatization using NLTK, two steps are to be performed as follows:

Step 1: Download wordnet using NLTK

```
from nltk.stem import WordNetLemmatizer
nltk.download('wordnet')
```

Step 2: Use lemmatize() from WordNetLemmatizer to perform lemmatization
Once the wordnet is downloaded, then lemmatization can be performed. Lemmatization can be done with or without a POS tag. A POS or part-of-speech tag assigns a tag to each word, and hence increases the accuracy of the lemma in the context of the dataset.
For example, the word 'leaves' without a POS tag would get lemmatized to the word 'leaf', but with a verb tag, its lemma would become 'leave'.

Example of using lemmatize( ):

```
words = ["grows","leaves","fairly","cats","trouble","running","friendships","easily", "was",
"relational","has"]

#an instance of Word Net Lemmatizer
lemmatizer = WordNetLemmatizer()

 #lemmatized words
lemmatized_words = [lemmatizer.lemmatize(word) for word in words]
print("The lemmatized words: ", lemmatized_words)

#POS tagged lemmatized words
lemmatized_words_pos = [lemmatizer.lemmatize(word, pos = "v") for word in words]
print("The lemmatized words using a POS tag: ", lemmatized_words_pos)
```

Activity:

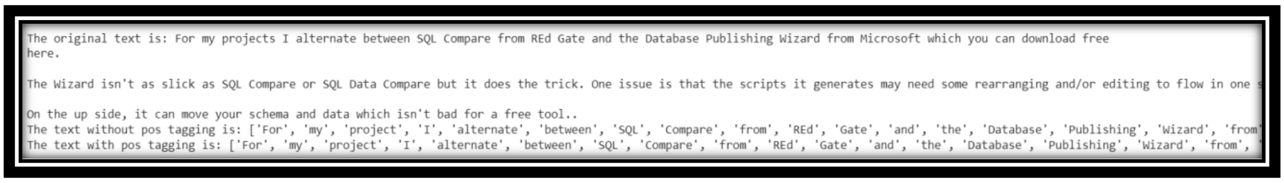
1. Perform lemmatization of Text , with and without POS Tagging to large corpus of text
2. Write down the observations about the difference in the results of lemmatization with POS and without POS Tagging, when both are applied on the same text corpus

Results: (Program with snapshot of output)

Code:

```
import pandas as pd
import re
import nltk
nltk.download('punkt')
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
from nltk import word_tokenize
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')

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 without_pos_tagging(tokenized_words):
  lemmatized_words_without_pos = [lemmatizer.lemmatize(word) for word in
tokenized_words]
  return lemmatized_words_without_pos

def with_pos_tagging(treebank_tag):
    if treebank_tag.startswith('J'):
        return wordnet.ADJ
    elif treebank_tag.startswith('V'):
        return wordnet.VERB
    elif treebank_tag.startswith('N'):
        return wordnet.NOUN
    elif treebank_tag.startswith('R'):
        return wordnet.ADV
    else:
        return wordnet.NOUN

for i in answer_column:
    lemmatizer = WordNetLemmatizer()
    html_removal_text = html_removal(i)
    print(f"The original text is: {html_removal_text}.")
    tokenized_words = word_tokenize(html_removal_text)
    print(type(tokenized_words))
    without_pos_tagging = without_pos_tagging(tokenized_words)
    print(f"The text without pos tagging is: {without_pos_tagging}.")
    pos_tags = nltk.pos_tag(tokenized_words)
    lemmatized_words_with_pos = [lemmatizer.lemmatize(word,
with_pos_tagging(tag)) for word, tag in pos_tags]
    print(f"The text with pos tagging is: {without_pos_tagging}.")
```

Output:



```
The original text is: For my projects I alternate between SQL Compare from REd Gate and the Database Publishing Wizard from Microsoft which you can download free
here.

The Wizard isn't as slick as SQL Compare or SQL Data Compare but it does the trick. One issue is that the scripts it generates may need some rearranging and/or editing to flow in one s

On the up side, it can move your schema and data which isn't bad for a free tool..
The text without pos tagging is: ['For', 'my', 'project', 'I', 'alternate', 'between', 'SQL', 'Compare', 'from', 'REd', 'Gate', 'and', 'the', 'Database', 'Publishing', 'Wizard', 'from'
The text with pos tagging is: ['For', 'my', 'project', 'I', 'alternate', 'between', 'SQL', 'Compare', 'from', 'REd', 'Gate', 'and', 'the', 'Database', 'Publishing', 'Wizard', 'from',
```

Observations:

Lemmatization without POS Tagging:
- Lemmatization without POS tagging often defaults to noun lemmatization.
- Most words in the text are lemmatized, but some words remain unchanged because the lemmatizer doesn't know their specific POS tags.
- For example, "alternate" is lemmatized to "alternate," "rearranging" to "rearranging," and "editing" to "editing." These words are not changed since their POS tags are not specified.
- Words like "isn't" and "does" are not lemmatized; they remain unchanged.

Lemmatization with POS Tagging:
- Lemmatization with POS tagging takes into account the specific POS tags of words.
- Words are lemmatized based on their known POS tags, which can lead to more accurate results.
- For example, "be" is used for "is" and "isn't," and "do" is used for "does," which are correct verb forms.
- Nouns like "project" and "issue" are not changed because their POS tags are not specified as verbs.
- Words like "rearrange" and "edit" are lemmatized to their base verb forms, which is more accurate.

Questions:

1. Explain the difference between stemming and lemmatization of text

A) Stemming:
Stemming is a technique that aims to reduce words to their root or base forms, known as "stems," by removing common suffixes. It's a heuristic and rule-based process that doesn't consider the meaning of the words. Instead, it focuses on string manipulation.

Mechanism: Stemming algorithms apply a series of predefined rules to trim off suffixes. For example, the stemming algorithm might remove the "-ing" or "-ed" suffix to reduce "running" to "run" or "jumped" to "jump." It aims to produce a rough approximation of the root form, even if it results in non-words.

Speed: Stemming is computationally efficient, and it's a good choice when processing speed is a primary concern. It's commonly used in information retrieval systems and search engines to match different word forms with a common base for efficient searching.

Accuracy: While stemming is fast, it's less accurate. It may generate stems that are not valid words and may not accurately represent the true meaning of the original words. For instance, it might convert "happily" to "happi," which is not a valid word.

Lemmatization:
Lemmatization, in contrast, is a linguistically-informed technique that reduces words to their dictionary or lemma forms. It takes into account the meaning and context of words and is more concerned with linguistic accuracy.

Mechanism: Lemmatization uses linguistic knowledge, dictionaries, and language-specific rules to determine the lemma of a word. It considers the part of speech of a word and its context to produce accurate lemmas. For example, it lemmatizes "running" to "run" and "better" to "good."

Accuracy: Lemmatization is highly accurate, as it produces valid words in the language. It ensures that the reduced form is linguistically correct and maintains the integrity of the original words. This accuracy is particularly valuable in applications that require a deep understanding of language, such as machine translation or sentiment analysis.

Speed: Lemmatization can be slower and more computationally intensive compared to stemming. It may not be the best choice when speed is of the essence but is essential when linguistic accuracy matters.

The choice between stemming and lemmatization depends on your specific text processing needs:

- Use stemming when you need speed and a basic level of text normalization for tasks like information retrieval or text indexing, where the exact meaning of words is less critical.
- Use lemmatization when linguistic accuracy is paramount, such as in natural language understanding tasks like sentiment analysis, machine translation, or text summarization. Lemmatization is also valuable when you want to maintain the correctness of words and their meanings. However, be prepared for potentially longer processing times.

Outcomes:

CO3: Establish concept of Structure and Semantics

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

<u>We understood the concept of lemmatization and the ways of performing lemmatization. We also implemented both the ways of lemmatization on our corpus of data and observed the differences in them.</u>

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