AIM: NLP program for Stop word analysis , Stemming ,Morphological analysis

STEMMING:

Stemming is the process of producing morphological variants of a root/base word. Stemming programs are commonly referred to as stemming algorithms or stemmers. A stemming algorithm reduces the words “chocolates”, “chocolatey”, “choco” to the root word, “chocolate” and “retrieval”, “retrieved”, “retrieves” reduce to the stem “retrieve”. Stemming is an important part of the pipelining process in Natural language processing. The input to the stemmer is tokenized words. How do we get these tokenized words? Well, tokenization involves breaking down the document into different words.
Some more example of stemming for root word "like" include:
->"likes"
->"liked"
->"likely"
->"liking"

Errors in Stemming:

There are mainly two errors in stemming –

- over-stemming
- Under-stemming

Over-stemming occurs when two words are stemmed from the same root that are of different stems. Over-stemming can also be regarded as false-positives. Under-stemming occurs when two words are stemmed from the same root that are not of different stems. Under-stemming can be interpreted as false-negatives.

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.

Some Stemming algorithms are:

- **Porter’s Stemmer algorithm**

  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'.
- 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.

STOP WORDS:

A stop word is a commonly used word (such as "the", "a", "an", "in") that a search engine has been programmed to ignore, both when indexing entries for searching and when retrieving them as the result of a search query.
We would not want these words to take up space in our database, or taking up valuable processing time. For this, we can remove them easily, by storing a list of words that you consider to stop words. NLTK(Natural Language Toolkit) in python has a list of stopwords stored in 16 different languages. You can find them in the nltk_data directory.

MORPHOLOGICAL ANALYSIS:

Morphological analysis is the process of examining possible resolutions to unquantifiable, complex problems involving many factors. The root of the word morphology comes from the Greek word, morphe, for form.
Morphological analysis takes a problem with many known solutions and breaks them down into their most basic elements, or forms, in order to more completely understand them.
Morphological analysis is used in general problem solving, linguistics and biology. In many fields of study morphology facilitates clearer instruction for teachers to help students understand problems and their solutions.
For general problem solving, morphological analysis provides a formalized structure to help examine the problem and possible solutions. The elements of a problem and its solutions are arranged in a matrix to help eliminate illogical solutions.
In biology, the study of forms helps understand mutations, adaptation and evolution. The study of the features and structure of organisms helps us understand organisms and their place in the greater environment.
In linguistics, words are broken down into the smallest units of meaning: morphemes. Morphemes can sometimes be words themselves as in the case of free morphemes, which can stand on their own. Other morphemes can add meaning but not stand as words on their own; bound morphemes need to be used along with another morpheme to make a word. Cats, for example, is a two-morpheme word. Its base, cat, is a free morpheme and its suffix an s, to denote pluralization, a bound morpheme.

A) STEMMING:

SOURCE CODE:

```
import nltk
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
True
```

```
from nltk.stem import PorterStemmer
from nltk.tokenize import sent_tokenize, word_tokenize

ps = PorterStemmer()

example_words = ["python", "pythoner", "pythoning", "pythoned", "pythonly"]

for w in example_words:
  print(ps.stem(w))
```

```
python
python
python
python
pythonli
```

```
new_text = "It is important to by very pythonly while you are pythoning with python. All pythoners have pythoned poorly at least once."

words = word_tokenize(new_text)

for w in words:
  print(ps.stem(w))
```

```
It
is
import
to
by
veri
pythonli
while
you
are
python
with
python
.
all
python
have
python
poorli
at
least
onc
.
```

```
def stemSentence (new_text):
  words = word_tokenize(new_text)

  for w in words:
    print(ps.stem(w))

sent = "hello man what is dogy"
stemSentence(sent)
```

```
hello
man
what
is
dogi
```

B) STOP WORDS:

SOURCE CODE:

```
nltk.download('stopwords')
nltk.download('punkt')

from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
```

```
example_sent = "This is a sample sentence, showing off the stop words filtration."
stop_words = set(stopwords.words('english'))
print("stop words:",stop_words)
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
stop words: {"mightn't", 'both', 'it', 'or', "wouldn't", 'they', 'your', 'to', "needn't", "she's", 'its', 'be', "shan't", 'is', 'my', 'how', "that'll", 'her', "is
```

stop words: {"mightn't", 'both', 'it', 'or', "wouldn't", 'they', 'your', 'to', "needn't", "she's", 'its', 'be', "shan't", 'is', 'my', 'how', "that'll", 'her', "isn't", 'do', 'me', 'can', 'themselves', "mustn't", 'haven', 'been', 'yours', 'doesn', "it's", 'himself', 'wasn', 'hasn', "won't", 'what', 'until', 'you', 'don', 'that', "you'd", 'will', "don't", 'all', 'in', 'he', 'not', "weren't", 'should', 'when', 'each', 'there', 'once', 'and', 'weren', 'them', 'through', 'most', 'where', "hasn't", 'mightn', 'out', "wasn't", 'on', 'yourselves', "you're", 'a', 'for', 'now', "didn't", 'we', 'ours', 'did', 'again', 'ain', 'won', 'the', 'before', 'down', 'd', 'she', "aren't", 'such', 'but', 'only', 'myself', 'above', 'some', 'herself', 'as', 'ourselves', 'his', 'am', 'about', 'why', "you've", 'these', 'too', 'y', 'at', 'whom', 'ma', 'than', 'yourself', 'has', 'other', 'was', 'needn', "hadn't", 'after', 'o', 'mustn', 'very', 'aren', 'own', 'wouldn', 'more', 'being', 'theirs', 's', 'had', 'which', 'itself', 'up', 'then', 'having', 'doing', 'from', 'll', 'shouldn', 'does', 'who', 'during', 'this', 'by', 'their', 'are', 'against', 'isn', 'no', 'under', 'few', 'hadn', 'couldn', 'shan', 'an', 'same', 'further', 'here', "you'll", 'below', "couldn't", 'nor', 'with', 'into', 't', 'over', 'were', 'because', 'hers', 're', 'our', 'have', 've', 'any', 'while', "shouldn't", "haven't", "should've", 'those', 'off', 'just', 'if', 'so', 'i', 'of', 'didn', 'between', "doesn't", 'm', 'him'}

```
word_tokens = word_tokenize(example_sent)
filtered_sentence = [w for w in word_tokens if not w in stop_words]

filtered_sentence = []
stop_words_list_ofsentence = []

for w in word_tokens:
        if w not in stop_words:
                filtered_sentence.append(w)
        else:
                stop_words_list_ofsentence.append(w)

print("token list of sentence", word_tokens)
print("Stop word list of sentence ", stop_words_list_ofsentence)
print("token list of sentence with stop words removed", filtered_sentence)
```

```
token list of sentence ['This', 'is', 'a', 'sample', 'sentence', ',', 'showing', 'off', 'the', 'stop', 'words', 'filtration', '.']
Stop word list of sentence  ['is', 'a', 'off', 'the']
token list of sentence with stop words removed ['This', 'sample', 'sentence', ',', 'showing', 'stop', 'words', 'filtration', '.']
```

C) MORPHOLOGICAL ANALYSIS:

SOURCE CODE:

```
import spacy
nlp = spacy.load('en_core_web_sm')

def morphologicalAnalysis(text):
doc= nlp(text)

        for token in doc:
            print("The original word is :",token)
            print("The stem word is :", token.lemma_)
            print("The morphological analysis is :")
            print(' '.join(token.morph))

morphologicalAnalysis("cat")
morphologicalAnalysis("played")
morphologicalAnalysis("liking")
morphologicalAnalysis("bats")
```

OUTPUT:

```
The original word is : cat
The stem word is : cat
The morphological analysis is :
Number_sing NounType_prop
The original word is : played
The stem word is : play
The morphological analysis is :
Aspect_perf VerbForm_part
The original word is : liking
The stem word is : like
The morphological analysis is :
Aspect_prog VerbForm_part
The original word is : bats
The stem word is : bat
The morphological analysis is :
Number_plur
```

CONCLUSION:

From this practical, I have learned about Stop word analysis , Stemming , Morphological analysis.