St. Francis Institute of Technology

Department of Computer Engineering

Academic Year: 2021-2022 **Semester: VIII**

Subject: Natural Language Processing **Class/Branch/: BE/CMPNA**

Name :- Nithin Menezes **Roll Number: 56**

Installing Necessary Packages

```
# Basic Preprocessing
import nltk
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('stopwords')
from·nltk.corpus·import·stopwords
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Unzipping corpora/wordnet.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

1. Tokenization

```
#sentence = input('Enter a sentence: ')
sentence= "Coding helps to improve your logical thinking"
tokenized_sent = nltk.tokenize.word_tokenize(sentence)
print(f'After Sentence Tokenization: {tokenized_sent}')
```

```
After Sentence Tokenization: ['Coding', 'helps', 'to', 'improve', 'your', 'logical', 'th
```

2. Stemming

```
stemmer = nltk.stem.PorterStemmer()
print(f'After Stemming: {[stemmer.stem(x) for x in tokenized_sent]}')
```

```
After Stemming: ['code', 'help', 'to', 'improv', 'your', 'logic', 'think']
```

```
from nltk.stem import PorterStemmer
from nltk.tokenize import sent_tokenize, word_tokenize
sentence="Coding helps to improve your logical thinking"
words = word_tokenize(sentence)
ps = PorterStemmer()
for w in words:
    rootWord=ps.stem(w)
    print(rootWord)
```

```
code
help
to
improv
your
logic
think
```

3. Lemmatization

```
lemmatizer = nltk.stem.WordNetLemmatizer()
print(f'After Lemmatization: {[lemmatizer.lemmatize(x) for x in tokenized_sent]}')
```

```
After Lemmatization: ['Coding', 'help', 'to', 'improve', 'your', 'logical', 'thinking']
```

4. Stop word Removal

```
stopwords = set(stopwords.words('english'))
removed_stopwords = [w for w in tokenized_sent if not w in stopwords]
print(f'After removing Stopwords: {removed_stopwords}')
```

```
After removing Stopwords: ['Coding', 'helps', 'improve', 'logical', 'thinking']
```

Punctuation Removal

```
new_words= [word for word in removed_stopwords if word. isalnum()]
print(new_words)
```

```
['Coding', 'helps', 'improve', 'logical', 'thinking']
```

5. Filtration

```
sentence_mix ="How's the Josh? हाई सर"
print('After Filtration: ' + ''.join(list(filter(lambda x: ord(x) < 123, sentence_mix))))
```

```
After Filtration: How's the Josh?
```

6. Script Validation

```
validated = []
for w in removed_stopwords:
    validated.append(''.join([e for e in w if e.isalnum()]))
print(f'After script validation: {validated}')
```

```
After script validation: ['Coding', 'helps', 'improve', 'logical', 'thinking']
```

Extra Processing

Convert to Lowercase

```
import re

text = input(" enter string: ")

print("Original String:")
print(text)

# lower() function to convert
# string to lower_case
print("\nConverted String:")
print(text.lower())
```

```
 enter string: MICHAEL JETSON
Original String:
MICHAEL JETSON

Converted String:
michael jetson
```

Remove Whitespaces

```
import re
input_str = input("Enter string: ")
result = re.sub(r' ', '', input_str)
print(result)
```

```
Enter string: Michael Jetson
MichaelJetson
```

```
#Extra Preprocessing

import re
def remove_emoji(string):
    emoji_pattern = re.compile("["
                           u"\U0001F600-\U0001F64F"  # emoticons
                           u"\U0001F300-\U0001F5FF"  # symbols & pictographs
                           u"\U0001F680-\U0001F6FF"  # transport & map symbols
                           u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                           u"\U00002702-\U000027B0"
                           u"\U000024C2-\U0001F251"
                           "]+", flags=re.UNICODE)
    return emoji_pattern.sub(r'', string)
emoji_sent = input("Enter text with emojis:")
print(f'After removing emojis: {remove_emoji(emoji_sent)}')
def remove_urls(text):
    url_pattern = re.compile(r'https?://\S+|www\.\S+')
    return url_pattern.sub(r'', text)
url_string = input(" Enter text with URL")
print(f'After removing URL: {remove_urls(url_string)}')
```

```
Enter text with emojis:😅Hello
After removing emojis: Hello
 Enter text with URLhttps://iq.opengenus.org/porter-stemmer/  Porter Stemmer Algorithm
After removing URL:   Porter Stemmer Algorithm
```

Porter Stemmer Algorithm

```
class PorterStemmer:
    def isCons(self, letter):
        if letter == 'a' or letter == 'e' or letter == 'i' or letter == 'o' or letter == 'u':
            return False
        else:
            return True

    def isConsonant(self, word, i):
        letter = word[i]
        if self.isCons(letter):
            if letter == 'y' and isCons(word[i-1]):
                return False
            else:
                return True
        else:
            return False

    def isVowel(self, word, i):
        return not(isConsonant(word, i))

    # *S
    def endsWith(self, stem, letter):
        if stem.endswith(letter):
            return True
        else:
            return False

    # *v*
    def containsVowel(self, stem):
        for i in stem:
            if not self.isCons(i):
                return True
        return False

    # *d
    def doubleCons(self, stem):
        if len(stem) >= 2:
            if self.isConsonant(stem, -1) and self.isConsonant(stem, -2):
                return True
            else:
                return False
        else:
            return False

    def getForm(self, word):
        form = []
        formStr = ''
        for i in range(len(word)):
            if self.isConsonant(word, i):
                if i != 0:
                    prev = form[-1]
                    if prev != 'C':
                        form.append('C')
                else:
                    form.append('C')
            else:
                if i != 0:
                    prev = form[-1]
                    if prev != 'V':
                        form.append('V')
                else:
                    form.append('V')
        for j in form:
            formStr += j
        return formStr

    def getM(self, word):
        form = self.getForm(word)
        m = form.count('VC')
        return m

    # *o
    def cvc(self, word):
        if len(word) >= 3:
            f = -3
            s = -2
            t = -1
            third = word[t]
            if self.isConsonant(word, f) and self.isVowel(word, s) and self.isConsonant(word,
                if third != 'w' and third != 'x' and third != 'y':
                    return True
                else:
                    return False
            else:
                return False
        else:
            return False

    def replace(self, orig, rem, rep):
        result = orig.rfind(rem)
        base = orig[:result]
        replaced = base + rep
        return replaced

    def replaceM0(self, orig, rem, rep):
        result = orig.rfind(rem)
        base = orig[:result]
        if self.getM(base) > 0:
            replaced = base + rep
            return replaced
        else:
            return orig

    def replaceM1(self, orig, rem, rep):
        result = orig.rfind(rem)
        base = orig[:result]
        if self.getM(base) > 1:
            replaced = base + rep
            return replaced
        else:
            return orig

    def step1a(self, word):
        if word.endswith('sses'):
            word = self.replace(word, 'sses', 'ss')
        elif word.endswith('ies'):
            word = self.replace(word, 'ies', 'i')
        elif word.endswith('ss'):
            word = self.replace(word, 'ss', 'ss')
        elif word.endswith('s'):
            word = self.replace(word, 's', '')
        else:
            pass
        return word
    def step1b(self, word):
        flag = False
        if word.endswith('eed'):
            result = word.rfind('eed')
            base = word[:result]
            if self.getM(base) > 0:
                word = base
                word += 'ee'
        elif word.endswith('ed'):
            result = word.rfind('ed')
            base = word[:result]
            if self.containsVowel(base):
                word = base
                flag = True
        elif word.endswith('ing'):
            result = word.rfind('ing')
            base = word[:result]
            if self.containsVowel(base):
                word = base
                flag = True
        if flag:
            if word.endswith('at') or word.endswith('bl') or word.endswith('iz'):
                word += 'e'
            elif self.doubleCons(word) and not self.endsWith(word, 'l') and not self.endsWith
                word = word[:-1]
            elif self.getM(word) == 1 and self.cvc(word):
                word += 'e'
            else:
                pass
        else:
            pass
        return word

    def step1c(self, word):
        if word.endswith('y'):
            result = word.rfind('y')
            base = word[:result]
            if self.containsVowel(base):
                word = base
                word += 'i'
        return word

    def step2(self, word):
        if word.endswith('ational'):
            word = self.replaceM0(word, 'ational', 'ate')
        elif word.endswith('tional'):
            word = self.replaceM0(word, 'tional', 'tion')
        elif word.endswith('enci'):
            word = self.replaceM0(word, 'enci', 'ence')
        elif word.endswith('anci'):
            word = self.replaceM0(word, 'anci', 'ance')
        elif word.endswith('izer'):
            word = self.replaceM0(word, 'izer', 'ize')
        elif word.endswith('abli'):
            word = self.replaceM0(word, 'abli', 'able')
        elif word.endswith('alli'):
            word = self.replaceM0(word, 'alli', 'al')
        elif word.endswith('entli'):
            word = self.replaceM0(word, 'entli', 'ent')
        elif word.endswith('eli'):
            word = self.replaceM0(word, 'eli', 'e')
        elif word.endswith('ousli'):
            word = self.replaceM0(word, 'ousli', 'ous')
        elif word.endswith('ization'):
            word = self.replaceM0(word, 'ization', 'ize')
        elif word.endswith('ation'):
            word = self.replaceM0(word, 'ation', 'ate')
        elif word.endswith('ator'):
            word = self.replaceM0(word, 'ator', 'ate')
        elif word.endswith('alism'):
            word = self.replaceM0(word, 'alism', 'al')
        elif word.endswith('iveness'):
            word = self.replaceM0(word, 'iveness', 'ive')
        elif word.endswith('fulness'):
            word = self.replaceM0(word, 'fulness', 'ful')
        elif word.endswith('ousness'):
            word = self.replaceM0(word, 'ousness', 'ous')
        elif word.endswith('aliti'):
            word = self.replaceM0(word, 'aliti', 'al')
        elif word.endswith('iviti'):
            word = self.replaceM0(word, 'iviti', 'ive')
        elif word.endswith('biliti'):
            word = self.replaceM0(word, 'biliti', 'ble')
        return word

    def step3(self, word):
        if word.endswith('icate'):
            word = self.replaceM0(word, 'icate', 'ic')
        elif word.endswith('ative'):
            word = self.replaceM0(word, 'ative', '')
        elif word.endswith('alize'):
            word = self.replaceM0(word, 'alize', 'al')
        elif word.endswith('iciti'):
            word = self.replaceM0(word, 'iciti', 'ic')
        elif word.endswith('ful'):
            word = self.replaceM0(word, 'ful', '')
        elif word.endswith('ness'):
            word = self.replaceM0(word, 'ness', '')
        return word

    def step4(self, word):
        if word.endswith('al'):
            word = self.replaceM1(word, 'al', '')
        elif word.endswith('ance'):
            word = self.replaceM1(word, 'ance', '')
        elif word.endswith('ence'):
            word = self.replaceM1(word, 'ence', '')
        elif word.endswith('er'):
            word = self.replaceM1(word, 'er', '')
        elif word.endswith('ic'):
            word = self.replaceM1(word, 'ic', '')
        elif word.endswith('able'):
            word = self.replaceM1(word, 'able', '')
        elif word.endswith('ible'):
            word = self.replaceM1(word, 'ible', '')
        elif word.endswith('ant'):
            word = self.replaceM1(word, 'ant', '')
        elif word.endswith('ement'):
            word = self.replaceM1(word, 'ement', '')
        elif word.endswith('ment'):
            word = self.replaceM1(word, 'ment', '')
        elif word.endswith('ent'):
            word = self.replaceM1(word, 'ent', '')
        elif word.endswith('ou'):
            word = self.replaceM1(word, 'ou', '')
        elif word.endswith('ism'):
            word = self.replaceM1(word, 'ism', '')
        elif word.endswith('ate'):
            word = self.replaceM1(word, 'ate', '')
        elif word.endswith('iti'):
            word = self.replaceM1(word, 'iti', '')
        elif word.endswith('ous'):
            word = self.replaceM1(word, 'ous', '')
        elif word.endswith('ive'):
            word = self.replaceM1(word, 'ive', '')
        elif word.endswith('ize'):
            word = self.replaceM1(word, 'ize', '')
        elif word.endswith('ion'):
            result = word.rfind('ion')
            base = word[:result]
            if self.getM(base) > 1 and (self.endsWith(base, 's') or self.endsWith(base, 't'))
                word = base
            word = self.replaceM1(word, '', '')
        return word

    def step5a(self, word):
        if word.endswith('e'):
            base = word[:-1]
            if self.getM(base) > 1:
                word = base
            elif self.getM(base) == 1 and not self.cvc(base):
                word = base
        return word

    def step5b(self, word):
        if self.getM(word) > 1 and self.doubleCons(word) and self.endsWith(word, 'l'):
            word = word[:-1]
        return word

    def stem(self, word):
        word = self.step1a(word)
        word = self.step1b(word)
        word = self.step1c(word)
        word = self.step2(word)
        word = self.step3(word)
        word = self.step4(word)
        word = self.step5a(word)
        word = self.step5b(word)
        return word

stemmer = PorterStemmer()
stemmer.stem(input('enter word to get stem word '))
```

```
enter word to get stem word attraction
'attract'
```

MINI PROJECT - HINDI

Tokenization

```
sentence= "कोडिंग आपकी तार्किक सोच को बेहतर बनाने में मदद करती है"
tokenized_sent = nltk.tokenize.word_tokenize(sentence)
print(f'After Sentence Tokenization: {tokenized_sent}')
```

```
After Sentence Tokenization: ['कोडिंग', 'आपकी', 'तार्किक', 'सोच', 'को', 'बेहतर', 'बनाने', '
```

Stemming

```
stemmer = nltk.stem.PorterStemmer()
print(f'After Stemming: {[stemmer.stem(x) for x in tokenized_sent]}')
```

```
After Stemming: ['कोडिंग', 'आपकी', 'तार्किक', 'सोच', 'को', 'बेहतर', 'बनाने', 'में', 'मदद', '
```

Lemmitization

```
lemmatizer = nltk.stem.WordNetLemmatizer()
print(f'After Lemmatization: {[lemmatizer.lemmatize(x) for x in tokenized_sent]}')
```

```
After Lemmatization: ['कोडिंग', 'आपकी', 'तार्किक', 'सोच', 'को', 'बेहतर', 'बनाने', 'में', 'मदद
```

Conclusion: in this experiment we learnt about text processing and also implemented the same using python.