

Experiment No 02: Text Processing

Rebecca Dias

Roll no: 18 BE CMPN A2

Pid: 182027

Aim: To perform the various preprocessing techniques of text in Python.

▼ 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.
```

▼ Tokenization

Tokenization is the process by which a big quantity of text is divided into smaller parts called tokens.

▼ Word Tokenization : This involves splitting a sentence into words.

```
#sentence = input('Enter a sentence: ')
sentence= "This is a test statement for tokenization."
tokenized_sent = nltk.tokenize.word_tokenize(sentence)
print(f'After Sentence Tokenization: {tokenized_sent}')
```

After Sentence Tokenization: ['This', 'is', 'a', 'test', 'statement', 'for', 'tokenization', '.']

▼ Sentence Tokenization

```
from nltk.tokenize import sent_tokenize
```

```
text = "This is a test statement for tokenization."
print(sent_tokenize(text))
```

```
['This is a test statement for tokenization.']
```

▼ Stemming

Stemming is a kind of normalization where a set of words in a sentence are converted into a sequence to shorten its lookup.

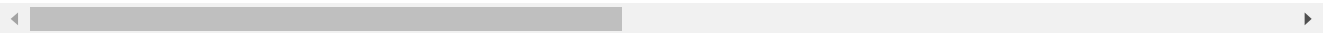
```
stemmer = nltk.stem.PorterStemmer()
print(f'After Stemming: {[stemmer.stem(x) for x in tokenized_sent]}')
```

```
After Stemming: ['thi', 'is', 'a', 'test', 'statement', 'for', 'token', '.']
```

Example01

```
tokenized_sent = nltk.tokenize.word_tokenize("In computing, the term text processing refers to the t
print(f'After Stemming: {[stemmer.stem(x) for x in tokenized_sent]}')
```

```
After Stemming: ['In', 'comput', ',', 'the', 'term', 'text', 'process', 'refer', 'to', 'the', '']
```



Example02

```
tokenized_sent = nltk.tokenize.word_tokenize("The text processing of a regular expression is a virtua
print(f'After Stemming: {[stemmer.stem(x) for x in tokenized_sent]}')
```

```
After Stemming: ['the', 'text', 'process', 'of', 'a', 'regular', 'express', 'is', 'a', 'virtual']
```



Example03

```
tokenized_sent = nltk.tokenize.word_tokenize("Text processing involves computer commands which invoke
print(f'After Stemming: {[stemmer.stem(x) for x in tokenized_sent]}')
```

```
After Stemming: ['text', 'process', 'involv', 'comput', 'command', 'which', 'invok', 'content',
```



```
from nltk.stem import PorterStemmer
from nltk.tokenize import sent_tokenize, word_tokenize
sentence="Recent researchs reported that the virus can remain viable for up to 72 hours on plastic ar
words = word_tokenize(sentence)
ps = PorterStemmer()
for w in words:
    rootWord=ps.stem(w)
    print(rootWord)
```

```

recent
research
report
that
the
viro
can
remain
viabl
for
up
to
72
hour
on
plastic
and
stainless
steel
and
up
to
24
hour
on
cardboard
.

```

▼ Lemmatization

Lemmatization is the algorithmic process of finding the lemma of a word depending on their meaning.


Example01

```

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

After Lemmatization: ['Text', 'processing', 'involves', 'computer', 'command', 'which', 'invoke

```




Example02

```

tokenized_sent = nltk.tokenize.word_tokenize("Text processing involves computer commands which invoke
print(f'After Lemmatization: {[lemmatizer.lemmatize(x) for x in tokenized_sent]}')

After Lemmatization: ['Text', 'processing', 'involves', 'computer', 'command', 'which', 'invoke

```



▼ Stop Word Removal

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.

```
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: ['Text', 'processing', 'involves', 'computer', 'commands', 'invoke',

▼ Punctuation Removal

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

['Text', 'processing', 'involves', 'computer', 'commands', 'invoke', 'content', 'content', 'cha

▼ Filtration

Filtration is the process of removing other language components from the given text.

Example 01

```
sentence_mix = "कौन है class monitor ?"
print('After Filtration: ' + ''.join(list(filter(lambda x: ord(x) < 123, sentence_mix))))
```

After Filtration: class monitor ?

Example 02

```
sentence_mix = "कितना आया exam में ?"
print('After Filtration: ' + ''.join(list(filter(lambda x: ord(x) < 123, sentence_mix))))
```

After Filtration: exam ?

▼ Script Validation

Script Validation is the process of removing special characters and punctuations.

```
validated = []
for w in removed_stopwords:
    validated.append(''.join([e for e in w if e.isalnum()]))
```

```
print('After script validation: {validated}')
```

```
After script validation: ['Text', 'processing', 'involves', 'computer', 'commands', 'invoke', '']
```

▼ Extra_Preprocessing

```
#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 = "😄Removing emojis for text preprocessing 🤖👨🏻👩🏻👧🏻👦🏻"
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 = 'URL of whatsapp https://web.whatsapp.com/'
print(f'After removing URL: {remove_urls(url_string)}')
```

```
After removing emojis: Removing emojis for text preprocessing
After removing URL: URL of whatsapp
```

▼ 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, t):
            if third != 'w' and third != 'x' and third != 'y':
                return True
            else:
                return False
    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, 'l'):
            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('affectionate')

stemmer.stem('passionate')
```

Conclusion:

Implemented various text processing techniques in Python. We also implemented the porter stemmer algorithm.