INDIVIDUAL ASSIGNMENT

CT107-3-3-TXSA

TEXT ANALYTICS AND SENTIMENT ANALYSIS

CSSE___CT107-3-3-TXSA-L-4___2022-12-05

HAND OUT DATE : 20 DECEMBER 2022

HAND IN DATE : 28 FEBRUARY 2023

WEIGHTAGE : 25%

NAME : EDWARD LEONARDO

INTAKE : TP058284

INSTRUCTIONS TO CANDIDATES:

1 **Submit your assignment via Moodle.**

2 **Students are advised to underpin their answers with the use of references (cited using the 7th Edition of APA Referencing Style).**

3 **Late submission will be awarded zero (0) unless Extenuating Circumstances (EC) are upheld.**

4 **Cases of plagiarism will be penalized.**

5 **You must obtain 50% overall to pass this module.**

Table of Contents

Q1. Form Tokenization

1. Demonstrate sentence segmentation and report the output.

1. Demonstrate sentence segmentation and report the output.

```
#import Library
import nltk

#text Input
text_raw = text_1

#sentence segmentation using NLTK
nltk_sent_tokens = nltk.sent_tokenize(text_raw)

#display results
print(nltk_sent_tokens)
print("\nAmount of sentences: ", len(nltk_sent_tokens))
```

```
['Sentiment analysis is "contextual mining of text which identifies and extracts subjective informati
on" in source material, and helping a business to understand the social sentiment of their brand, pro
duct or service while monitoring online conversations.', 'However, analysis of social media streams i
s usually restricted to just basic sentiment analysis and count based metrics.', 'This is akin to jus
t scratching the surface and missing out on those high value insights that are waiting to be discover
ed.', 'So what should a brand do to capture that low hanging fruit?']

Amount of sentences:  4
```

Sentence segmentation means to separate a text by each sentence. To do this, NLTK library must be imported first, and then save the text in a variable, which is text_raw in the above picture. Next, NLTK function sent_tokenize will be used to do the sentence segmentation. sent_tokenize have one parameter, which is the variable that contain the paragraph. Finally, the result of the sentence segmentation can be printed to be displayed.

2. Demonstrate word tokenisation using the Split Function, Regular Expression, and NLTK packages separately and report the output.

1. Python split() function

```
#Using Python split() function
py_word = text_raw.split()

#result
print(py_word)
```

```
['Sentiment', 'analysis', 'is', '"contextual', 'mining', 'of', 'text', 'which', 'identifies', 'and', 'extracts', 'subjective',
'information"', 'in', 'source', 'material,', 'and', 'helping', 'a', 'business', 'to', 'understand', 'the', 'social', 'sentimen
t', 'of', 'their', 'brand,', 'product', 'or', 'service', 'while', 'monitoring', 'online', 'conversations.', 'However,', 'analys
is', 'of', 'social', 'media', 'streams', 'is', 'usually', 'restricted', 'to', 'just', 'basic', 'sentiment', 'analysis', 'and',
'count', 'based', 'metrics.', 'This', 'is', 'akin', 'to', 'just', 'scratching', 'the', 'surface', 'and', 'missing', 'out', 'o
n', 'those', 'high', 'value', 'insights', 'that', 'are', 'waiting', 'to', 'be', 'discovered.', 'So', 'what', 'should', 'a', 'br
and', 'do', 'to', 'capture', 'that', 'low', 'hanging', 'fruit?']
```

Word segmentation means to segment or separate a text by each word. To do word segmentation using Python function's split, we can use the function split() on the text variable to get the word segmentation. Python split works by separating a string by the character inside the split function parameter. If the parameter is empty, then split function will split the text by

each whitespace, thus resulting in a word segmentation. From the result above, split function will keep punctuation marks in the text, but did not separate it from the word it was written with.

2. Regular Expression

```
#import library
import re

#Using Regular Expression
re_word = re.findall(r"\w+", text_raw)

#result
print(re_word)
```

```
['Sentiment', 'analysis', 'is', 'contextual', 'mining', 'of', 'text', 'which', 'identifies', 'and', 'extracts', 'subjective',
'information', 'in', 'source', 'material', 'and', 'helping', 'a', 'business', 'to', 'understand', 'the', 'social', 'sentiment',
'of', 'their', 'brand', 'product', 'or', 'service', 'while', 'monitoring', 'online', 'conversations', 'However', 'analysis', 'o
f', 'social', 'media', 'streams', 'is', 'usually', 'restricted', 'to', 'just', 'basic', 'sentiment', 'analysis', 'and', 'coun
t', 'based', 'metrics', 'This', 'is', 'akin', 'to', 'just', 'scratching', 'the', 'surface', 'and', 'missing', 'out', 'on', 'tho
se', 'high', 'value', 'insights', 'that', 'are', 'waiting', 'to', 'be', 'discovered', 'So', 'what', 'should', 'a', 'brand', 'd
o', 'to', 'capture', 'that', 'low', 'hanging', 'fruit']
```

Using Regular Expression library, Word segmentation also can be achieved. First, import the Regular Expression library, named re library. After that, use findall function from the library. This function works by searching for all occurrence of the word inside the text variable based on the predetermined pattern of strings. By using “\w+”, it will search all one or more-word characters, which resulted in word segmentation. findall function have two main parameters, the first one is the regular expression syntax, to determine the patterns of text to be search on the variable. The second parameter is the text variable to apply the findall function. from the result, findall will not include any punctuation marks in the array result.

3. NLTK package

```
#import library
import nltk

#Using NLTK word tokenisation
nltk_word = nltk.word_tokenize(text_raw)

#result
print(nltk_word)
```

```
['Sentiment', 'analysis', 'is', '```', 'contextual', 'mining', 'of', 'text', 'which', 'identifies', 'and', 'extracts', 'subjecti
ve', 'information', "'''", 'in', 'source', 'material', ',', 'and', 'helping', 'a', 'business', 'to', 'understand', 'the', 'socia
l', 'sentiment', 'of', 'their', 'brand', ',', 'product', 'or', 'service', 'while', 'monitoring', 'online', 'conversations',
'.', 'However', ',', 'analysis', 'of', 'social', 'media', 'streams', 'is', 'usually', 'restricted', 'to', 'just', 'basic', 'sen
timent', 'analysis', 'and', 'count', 'based', 'metrics', '.', 'This', 'is', 'akin', 'to', 'just', 'scratching', 'the', 'surfac
e', 'and', 'missing', 'out', 'on', 'those', 'high', 'value', 'insights', 'that', 'are', 'waiting', 'to', 'be', 'discovered',
'.', 'So', 'what', 'should', 'a', 'brand', 'do', 'to', 'capture', 'that', 'low', 'hanging', 'fruit', '?']
```

Finally, word segmentation also can be done by using NLTK library. First, NLTK library need to be imported. After that, NLTK function word_tokenize will be used, where it has a single parameter, which is the text variable to apply the word segmentation. from the result of NLTK’s word segmentation, it still includes any punctuation marks from the text into the array result but separate it from the word it is originally written with.

3. Explain the differences of the tokenisation operations performed in Q1.2 using the reported output.

The difference between the three methods:

1. Python split() would only do tokenisation based on the parameter input. Since the input is a whitespace, any special characters attached to a word, in the example would be a double quote, a question mark, and the period would be stuck on the word itself.

2. Regular Expression findall() method allow us to do tokenisation based on their predetermined pattern. since we use \w+, which split for each word, resulted in all special characters would be deleted. But on the other hand, Regex gives a better flexibility since the filter can be customized.

3. NLTK word_tokenize() method gives the best of both worlds, where the special characters are still available in the tokenisation result, but are separated from the words, making it easy for any next step to be taken.

4. Justify the most suitable tokenisation operation for text analysis. Support your answer using obtained outputs.

From the result above, NLTK's word_tokenize() method give the best result overall. It separates the special/punctuation characters from the word its associated with, which makes it more convenient to Python's split(), but did not delete it immediately, which provides a better output from Regular Expression's findall(), since special characters should not be removed in some cases, making the NLTK’s function result more versatile to use (Greve, 2019).

Q2. Form Word Stemming

1. Explain the importance of stemming in text analysis.

Stemming is a Natural Language Processing (NLP) technique that lowers inflection in words to the basic, root forms. Inflection itself is a process where a word is modified to be suitable to many special grammatical situations (Sharma, 2021).

So, in a sense, stemming is a process to reduce any variants of a word to their basic form, even if the basic form is not a legitimate word in the language.

Stemming is an important step in NLP, because having a word that exist in several inflected forms can add data redundancy to the text corpus result, rendering the NLP or ML models ineffective.

2. Explain the differences among Regular Expression Stemmer and Porter Stemmer use for word stemming.

1. Porter Stemmer

Porter Stemmer is one of the most popular stemming algorithms. Truncating algorithms aim at cutting the words into a fixed length. Porter Stemmer is a part of the nltk.stem library, PorterStemmer module. Porter Stemmer already have a fixed affixes included in the module, so no affix configuration is needed before Porter Stemmer can work (Sharma, 2021).

2. Regular Expression Stemmer

Regular Expression Stemmer, or Regex Stemmer is stemming technique that uses the regular expressions to identify the same affixes, where any substrings that match the pattern will be removed. this means the substring that is chosen for stemming can be modified. there is also a minimum length of string parameter that make sure any words below the minimum length will not be affected. Regex Stemmer also a part of nltk.stem library, but under the RegexpStemmer module (NLTK Project, 2023).

3. Demonstrate and report the output for word stemming using any techniques such as Regular Expression or Porter Stemmer.

```
#import library for word tokenization
import nltk
import re

#text Input
text_raw = text_1

#making all text Lowercase
text_lower = text_raw.lower()

#removing punctuation and special characters
text_no_punc = re.sub(r'[^\w\s]+','', text_lower)

#stripping trailing whitespaces
text_no_wspace = text_no_punc.strip()

#word segmentation/tokenisation
nltk_word = nltk.word_tokenize(text_no_wspace)
```

Before doing the stemming, the text needs to be pre-processed/cleaned first. Part of this pre-processing is making all the words in the text to be lowercase. This is to reduce the amount of variants of a word by having one with a uppercase first letter, and one with lowercase first letter. The next step is to remove all special characters, and finally to strip any trailing whitespaces from the previous cleaning process. Finally, word segmentation will be applied to the text, so a stemming process for each word can be done.

1. Porter Stemmer

```
#import library
from nltk import PorterStemmer

#using Porter Stemmer
ps = PorterStemmer()

#printing result
for word in nltk_word:
    print(word," ---> ", ps.stem(word))
```

```
sentiment  --->  sentiment
analysis  --->  analysi
is  --->  is
contextual  --->  contextu
mining  --->  mine
of  --->  of
text  --->  text
which  --->  which
identifies  --->  identifi
and  --->  and
extracts  --->  extract
subjective  --->  subject
information  --->  inform
in  --->  in
source  --->  sourc
material  --->  materi
and  --->  and
helping  --->  help
a  --->  a
business  --->  busi
to  --->  to
understand  --->  understand
the  --->  the
social  --->  social
sentiment  --->  sentiment
of  --->  of
their  --->  their
brand  --->  brand
product  --->  product
or  --->  or
service  --->  servic
while  --->  while
monitoring  --->  monitor
online  --->  onlin
conversations  --->  convers
however  --->  howev
analysis  --->  analysi
of  --->  of
social  --->  social
```

```
media  --->  media
streams  --->  stream
is  --->  is
usually  --->  usual
restricted  --->  restrict
to  --->  to
just  --->  just
basic  --->  basic
sentiment  --->  sentiment
analysis  --->  analysi
and  --->  and
count  --->  count
based  --->  base
metrics  --->  metric
this  --->  thi
is  --->  is
akin  --->  akin
to  --->  to
just  --->  just
scratching  --->  scratch
the  --->  the
surface  --->  surfac
and  --->  and
missing  --->  miss
out  --->  out
on  --->  on
those  --->  those
high  --->  high
value  --->  valu
insights  --->  insight
that  --->  that
are  --->  are
waiting  --->  wait
to  --->  to
be  --->  be
discovered  --->  discov
so  --->  so
what  --->  what
should  --->  should
a  --->  a
brand  --->  brand
do  --->  do
to  --->  to
capture  --->  captur
that  --->  that
low  --->  low
hanging  --->  hang
fruit  --->  fruit
```

The pictures above show the results of Porter Stemming. To perform Porter Stemming, PorterStemmer module from NLTK library needs to be imported first. Next, the PorterStemmer() function need to be assigned to a variable to be used. Finally, use the stem() function to do stemming for each word, with the result above using a loop to show all of the original word from the text, and the stemmed version of the word using Porter Stemmer.

2. Regular Expression

```
#import library
from nltk import RegexpStemmer

#using Regular Expression Stemmer
#the below Regex Stemmer will only remove -ing, -s, -e, and -able, with
# a minimum length character to be stemmed is 4 characters long
res = RegexpStemmer('ing$|s$|e$|able$', min=4)

#printing result
for word in nltk_word:
    print(word," ---> ", res.stem(word))
```

```
sentiment  --->  sentiment
analysis  --->  analysi
is  --->  is
contextual  --->  contextual
mining  --->  min
of  --->  of
text  --->  text
which  --->  which
identifies  --->  identifie
and  --->  and
extracts  --->  extract
subjective  --->  subjectiv
information  --->  information
in  --->  in
source  --->  sourc
material  --->  material
and  --->  and
helping  --->  help
a  --->  a
business  --->  busines
to  --->  to
understand  --->  understand
the  --->  the
social  --->  social
sentiment  --->  sentiment
of  --->  of
their  --->  their
brand  --->  brand
product  --->  product
or  --->  or
service  --->  servic
while  --->  whil
monitoring  --->  monitor
online  --->  onlin
conversations  --->  conversation
however  --->  however
analysis  --->  analysi
of  --->  of
```

```
social  --->  social
media  --->  media
streams  --->  stream
is  --->  is
usually  --->  usually
restricted  --->  restricted
to  --->  to
just  --->  just
basic  --->  basic
sentiment  --->  sentiment
analysis  --->  analysi
and  --->  and
count  --->  count
based  --->  based
metrics  --->  metric
this  --->  thi
is  --->  is
akin  --->  akin
to  --->  to
just  --->  just
scratching  --->  scratch
the  --->  the
surface  --->  surfac
and  --->  and
missing  --->  miss
out  --->  out
on  --->  on
those  --->  thos
high  --->  high
value  --->  valu
insights  --->  insight
that  --->  that
are  --->  are
waiting  --->  wait
to  --->  to
be  --->  be
discovered  --->  discovered
so  --->  so
what  --->  what
should  --->  should
a  --->  a
brand  --->  brand
do  --->  do
to  --->  to
capture  --->  captur
that  --->  that
low  --->  low
hanging  --->  hang
fruit  --->  fruit
```

The pictures above show the results of Regular Expression/Regex Stemming. To perform Regex Stemming, RegexpSemmer module from NLTK library needs to be imported first. Next, the RegexpStemmer() function need to be assigned to a variable to be used, including with the search pattern to be matched. From the example above, it means any words that have -ing, -s, -e, and -able will get stemmed. Last, use the stem() function to do stemming for each word, with the result above using a loop to show all of the original word from the text, and the stemmed version of the word using Regex Stemmer.

3. Justify the most suitable stemming operation for text analytics. Support your answer using the obtained output.

For basic, most widespread use of stemming, Porter Stemmer would be the best choice for text analytics, since it is simpler to use. But on some cases where the task only needs to apply stemming on some specific suffix, Regex Stemming can be a good choice, but Regex Stemming can cause a mis stemming or over stemming, where the definition of the word is changed after the stemming (Srinidhi, 2021). in the output above, "mining" becomes min in regex stemmer, which changes the meaning of the word. This is caused by the search pattern is not defined properly, meaning Regex Stemming require more preparation before being used. Meanwhile, Porter Stemmer managed to reduce it to mine, where the same meaning still holds true.

Q3. Filter Stop Words and Punctuation

1. Demonstrate stop words and punctuations removal from the given text corpus and report the output suitably.

```
#text Input
text_raw = text_1
print(text_raw)
```

```
Sentiment analysis is "contextual mining of text which identifies and extracts subjective informatio
n" in source material, and helping a business to understand the social sentiment of their brand, prod
uct or service while monitoring online conversations. However, analysis of social media streams is us
ually restricted to just basic sentiment analysis and count based metrics. This is akin to just scrat
ching the surface and missing out on those high value insights that are waiting to be discovered. So
what should a brand do to capture that low hanging fruit?
```

```
#import library
from nltk.corpus import stopwords
import re

#get the stopwords corpus list based on the language
stop_words = set (stopwords.words('english'))

#set the text into all lowercase
text_lower = text_raw.lower()
print(text_lower)
```

```
sentiment analysis is "contextual mining of text which identifies and extracts subjective informatio
n" in source material, and helping a business to understand the social sentiment of their brand, prod
uct or service while monitoring online conversations. however, analysis of social media streams is us
ually restricted to just basic sentiment analysis and count based metrics. this is akin to just scrat
ching the surface and missing out on those high value insights that are waiting to be discovered. so
what should a brand do to capture that low hanging fruit?
```

```
#since there are no number in the text, number removal/subtitution will be skipped
#removing punctuation
text_no_punc = re.sub(r'[^\w\s]+','',text_lower)
print(text_no_punc)
```

```
sentiment analysis is contextual mining of text which identifies and extracts subjective information
in source material and helping a business to understand the social sentiment of their brand product o
r service while monitoring online conversations however analysis of social media streams is usually r
estricted to just basic sentiment analysis and count based metrics this is akin to just scratching th
e surface and missing out on those high value insights that are waiting to be discovered so what shou
ld a brand do to capture that low hanging fruit
```

```
#removing trailing whitespace
text_no_wspace = text_no_punc.strip()
print(text_no_wspace)
```

```
sentiment analysis is contextual mining of text which identifies and extracts subjective information in source material and hel
ping a business to understand the social sentiment of their brand product or service while monitoring online conversations howe
ver analysis of social media streams is usually restricted to just basic sentiment analysis and count based metrics this is aki
n to just scratching the surface and missing out on those high value insights that are waiting to be discovered so what should
a brand do to capture that low hanging fruit
```

```
#converting string into list
#either using nltk word_tokenise,
#or split() since the text has been pre-processed
text_word_token = nltk.word_tokenize(text_no_wspace)
print(text_word_token)
```

```
['sentiment', 'analysis', 'is', 'contextual', 'mining', 'of', 'text', 'which', 'identifies', 'and', 'extracts', 'subjective',
'information', 'in', 'source', 'material', 'and', 'helping', 'a', 'business', 'to', 'understand', 'the', 'social', 'sentiment',
'of', 'their', 'brand', 'product', 'or', 'service', 'while', 'monitoring', 'online', 'conversations', 'however', 'analysis', 'o
f', 'social', 'media', 'streams', 'is', 'usually', 'restricted', 'to', 'just', 'basic', 'sentiment', 'analysis', 'and', 'coun
t', 'based', 'metrics', 'this', 'is', 'akin', 'to', 'just', 'scratching', 'the', 'surface', 'and', 'missing', 'out', 'on', 'tho
se', 'high', 'value', 'insights', 'that', 'are', 'waiting', 'to', 'be', 'discovered', 'so', 'what', 'should', 'a', 'brand', 'd
o', 'to', 'capture', 'that', 'low', 'hanging', 'fruit']
```

```
#removing stopwords
text_no_stopwords = []
stopwords_included = []
for word in text_word_token:
    if not word in stop_words:
        text_no_stopwords.append(word)
    else:
        stopwords_included.append(word)

#result of the results after stopwords removal
print(text_no_stopwords)
```

```
['sentiment', 'analysis', 'contextual', 'mining', 'text', 'identifies', 'extracts', 'subjective', 'information', 'source', 'mat
erial', 'helping', 'business', 'understand', 'social', 'sentiment', 'brand', 'product', 'service', 'monitoring', 'online', 'con
versations', 'however', 'analysis', 'social', 'media', 'streams', 'usually', 'restricted', 'basic', 'sentiment', 'analysis', 'c
ount', 'based', 'metrics', 'akin', 'scratching', 'surface', 'missing', 'high', 'value', 'insights', 'waiting', 'discovered', 'b
rand', 'capture', 'low', 'hanging', 'fruit']
```

Before removing the punctuation, some pre-processing can be done to the text first, as a good habit, to make sure that the text is optimal for NLP. A few of this good habit is making sure the text is all in lowercase, using lower() function. after that, punctuation removal can be done using re library, sub() function, where all punctuation will be replaced with ‘’, causing them to be deleted from the text. After that, removing any trailing whitespace is also a good practice since punctuation removal may caused a trailing whitespace to be created. After that, using NLTK’s word_tokenize() function or Python’s split() function, the text will undergo word tokenization.

After the tokenization, the text is ready for stop words removal. This step will utilize the stopwords module from nltk.corpus library. Before using it, the module need to be set according to the text’s language, so the stop words filter will be compatible with the text. Accordance to the text used, the language for stop words removal will be set to English. Next, by going through the text array using a loop, a checking can be done, where if a stop word is encountered, it can be deleted or in this case, the stop words found will be put into another array. After the checking loop is done, the result will be the text array without any stop words in it.

2. Report the stop words found in the given text corpus

```
#result of the stopwords that has been removed
print(stopwords_included)
```

```
['is', 'of', 'which', 'and', 'in', 'and', 'a', 'to', 'the', 'of', 'their', 'or', 'while', 'of', 'is', 'to', 'just', 'and', 'thi
s', 'is', 'to', 'just', 'the', 'and', 'out', 'on', 'those', 'that', 'are', 'to', 'be', 'so', 'what', 'should', 'a', 'do', 'to',
'that']
```

Since the stop words removed from the text is saved into another array during the checking process, all that needs to be done is to display the array that contains all the removed stop words.

3. Explain the importance of filtering the stop words and punctuations in text analytics.

Stop words are usually the words that does not add any value or new information to the text itself, hence during most of NLP tasks, stop words can be removed entirely to reduce the size of the model, making it more optimized. The same case can be said about punctuations too (Balodi, 2021).

But stop words and punctuation filtering is dependent on the NLP task at hand, since some tasks can have better result with the inclusion of stop words and punctuations (Khanna, 2021).

Q4. Form Parts of Speech (POS) Taggers & Syntactic Analysers

1. Demonstrate POS tagging using NLTK POS tagger, the Regular Expression tagger and report the output.

```
#import library
import nltk
import re
from nltk import word_tokenize
from nltk import pos_tag
from nltk.tag import RegexpTagger

#cleaning and tokenization process

#remove non-alphabetical words
text_clean = re.sub("[^a-zA-Z]", " ", text_raw)

#remove multiple whitespaces
text_clean = " ".join(text_clean.split())

#word tokenization
nltk_word = word_tokenize(text_clean)
```

```
#Applying NLTK POS tagging
nltk_pos_tag = pos_tag(nltk_word)

print(nltk_pos_tag)
```

```
[('A', 'DT'), ('videogame', 'NN'), ('or', 'CC'), ('computergame', 'NN'), ('is', 'VBZ'), ('an', 'DT'),
('electronic', 'JJ'), ('game', 'NN'), ('that', 'WDT'), ('involves', 'VBZ'), ('interaction', 'NN'),
('with', 'IN'), ('a', 'DT'), ('user', 'JJ'), ('interface', 'NN'), ('or', 'CC'), ('input', 'NN'), ('de
vice', 'NN')]
```

Before doing any Part-of-Speech Tagging, some pre-processing is required to be done. First, to clean the text, remove all non-alphabetical words and any multiple whitespaces inside the text. This can be done using re's sub() function. Finally, word tokenization can be done using nltk's word_tokenize().

Now for the NLTK's POS tagging, it can be done using the pos_tag() function from the NLTK library, with the array of tokenized words as the parameter.

```
#Applying Regex POS Tagging
#https://tedboy.github.io/nlps/generated/generated/nltk.RegexpTagger.html#nltk.RegexpTagger

#defining the tag by associating it with the words
regexp_tagger = RegexpTagger(
    [(r'^-?[0-9]+(.[0-9]+)?$', 'CD'),   # cardinal numbers
    (r'(The|the|A|a|An|an)$', 'AT'),   # articles
    (r'.*able$', 'JJ'),                # adjectives
    (r'.*ness$', 'NN'),                # nouns formed from adjectives
    (r'.*ly$', 'RB'),                  # adverbs
    (r'.*s$', 'NNS'),                  # plural nouns
    (r'.*ing$', 'VBG'),                # gerunds
    (r'.*ed$', 'VBD'),                 # past tense verbs
    (r'.*', 'NN')                      # nouns (default)
])

regex_pos_tag = regexp_tagger.tag(nltk_word)

print(regex_pos_tag)
```

```
[('A', 'AT'), ('videogame', 'NN'), ('or', 'NN'), ('computergame', 'NN'), ('is', 'NNS'), ('an', 'AT'),
('electronic', 'NN'), ('game', 'NN'), ('that', 'NN'), ('involves', 'NNS'), ('interaction', 'NN'), ('w
ith', 'NN'), ('a', 'AT'), ('user', 'NN'), ('interface', 'NN'), ('or', 'NN'), ('input', 'NN'), ('devic
e', 'NN')]
```

For Regex's Tagging, the RegexpTagger function from nltk.tag needs to be imported first. Next is to declare the function inside a variable. This includes the tag definition, where the tag

defined in the function will be used for the POS tagging of the words. Finally, the tagging process can be done by calling the variable with a tag() function attached to it, where the tokenized text array included as the parameter.

2. Explain the differences of the POS taggers using the output obtained in the above question

From the results above, NLTK POS Tagging will use the NLTK POS tag that has been determined. On the other hand, regex POS tagging need to define the tag by associating it with the words, using the regular expressions. this means the user can create customised tags if necessary. also, if the word is not associated with any of the tags defined, it will use the default one. the example above is "with", where in NLTK, it is classified as IN (preposition/subordinating conjunction), but in Regex, since the tag is not defined, it is classified as NN (nouns), which is the default tag.

3. Justify the most suitable POS tagger for text analytics. Support your answer using the output obtained.

The most suitable POS tagging for normal text analytics would be NLTK, mainly due to its ease of use. Regex Tagging can still be useful for other cases, where the analysis requires a specialized/custom tagging. But due to its nature requiring defining the tag first, it can be redundantly more complex to do a normal text POS tagging using regex compared to NLTK's function.

4. Draw possible parse trees for the given sentences using python codes and report the Parse Trees along with the Python code.

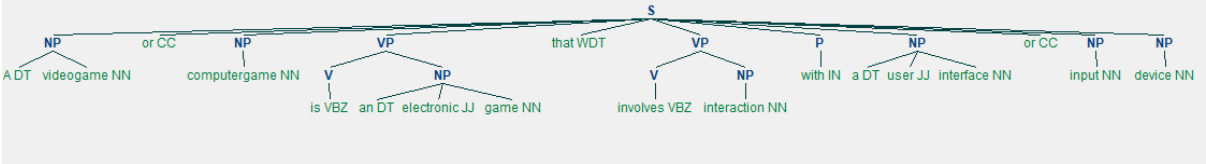
```
#import library for creating CFG and Parse Tree
from nltk import RegexpParser

#Creating the CFG Form
cfg_form = RegexpParser("""
NP: {<DT>?<JJ>*<NN>} #To extract Noun Phrases
P: {<IN>}            #To extract Prepositions
V: {<V.*>}           #To extract Verbs
PP: {<p> <NP>}       #To extract Prepositional Phrases
VP: {<V> <NP|PP>*}   #To extract Verb Phrases
                    """)
```

```
#using NLTK POS Tag's result
nltk_cfg = cfg_form.parse(nltk_pos_tag)
print("NLTK POS Tag Result:")
print(nltk_cfg)
nltk_cfg.draw()
```

```
NLTK POS Tag Result:
(S
  (NP A/DT videogame/NN)
  or/CC
  (NP computergame/NN)
  (VP (V is/VBZ) (NP an/DT electronic/JJ game/NN))
  that/WDT
  (VP (V involves/VBZ) (NP interaction/NN))
  (P with/IN)
  (NP a/DT user/JJ interface/NN)
  or/CC
  (NP input/NN)
  (NP device/NN))
```

```
#using Regex POS Tag's result
regex_cfg = cfg_form.parse(regex_pos_tag)
print("Regex POS Tag Result:")
print(regex_cfg)
regex_cfg.draw()
```

```
Regex POS Tag Result:
(S
  A/AT
  (NP videogame/NN)
  (NP or/NN)
  (NP computergame/NN)
  is/NNS
  an/AT
  (NP electronic/NN)
  (NP game/NN)
  (NP that/NN)
  involves/NNS
  (NP interaction/NN)
  (NP with/NN)
  a/AT
  (NP user/NN)
  (NP interface/NN)
  (NP or/NN)
  (NP input/NN)
  (NP device/NN))
```



Before drawing the parse tree or the CFG diagram, all that is needed to do is to create the Context-Free Grammar form using the RegexpParser function, imported from the nltk library. Next is to feed the tagged array from the previous answer using the parse() function, where in this case, the NLTK POS Tag result will be the one used to create the parse tree. Finally, a Parse Tree can be created using the draw() function, and the result can also be displayed in the lexical form. The parse tree and the lexical form output shows the syntactic structure of the text.

References

Balodi, T. (2021, July 31). NLTK Python Tutorial For Beginners. AnalyticsSteps. Retrieved February 14, 2023, from https://www.analyticssteps.com/blogs/nltk-python-tutorial-beginners

Greve, B. (2019). Special characters. A Beginner's Guide to Clean Data. Retrieved February 13, 2022, from https://b-greve.gitbook.io/beginners-guide-to-clean-data/text-mining-problems/special-characters

Khanna, C. (2021, February 10). Text pre-processing: Stop words removal using different libraries. Medium. Retrieved February 15, 2023, from https://towardsdatascience.com/text-pre-processing-stop-words-removal-using-different-libraries-f20bac19929a

NLTK Project. (2023, January 23). nltk.stem.regexp module. NLTK Documentation. Retrieved February 13, 2023, from https://www.nltk.org/api/nltk.stem.regexp.html

Sharma, P. (2021, November 25). An Introduction to Stemming in Natural Language Processing. Analytics Vidhya. Retrieved February 13, 2022, from https://www.analyticsvidhya.com/blog/2021/11/an-introduction-to-stemming-in-natural-language-processing/

Srinidhi, S. (2021, December 13). Stemming of words in Natural Language Processing, what is it? Medium. Retrieved February 15, 2023, from https://towardsdatascience.com/stemming-of-words-in-natural-language-processing-what-is-it-41a33e8996e2