NLP Laboratory

Module 1

1) **Tokenizing -Design a Python program to splitting up a larger body of text into smaller lines, words or even create words for a non-English language.**

```
import nltk
nltk.download('punkt')    # Downloading necessary NLTK data (if not already downloaded)

from nltk.tokenize import sent_tokenize, word_tokenize

# French text
text = "Bonjour le monde! Ceci est un texte simple. "

# Tokenize sentences
sentences = sent_tokenize(text)
print("Sentences:", sentences)

# Tokenize words
words = word_tokenize(text, language='french')
print("Words:", words)
```

Output:

```
Sentences: ['Bonjour le monde!', 'Ceci est un texte simple.']
Words: ['Bonjour', 'le', 'monde', '!', 'Ceci', 'est', 'un', 'texte', 'simple', '.']
```

2) **Corpus- Design a Python program to illustrate corpus.**

```
import nltk

from nltk.corpus import brown

nltk.download('brown')


brown_corpus = brown.words()

word_count = brown_corpus.count('the')


print("\nOccurrences of 'the' in the brown corpus", word_count)


collocations = nltk.Text(brown_corpus).collocation_list()
```

```
print("\nCollocations in the brown corpus.")

print(collocations[:10])

fdist = nltk.FreqDist(brown_corpus)

print("\nMost common words in the brown corpus.")

print(fdist.most_common(10))
```

Output:

```
Occurrences of 'the' in the brown corpus 62713

Collocations in the brown corpus.

[('United', 'States'), ('New', 'York'), ('per', 'cent'), ('Rhode', 'Island'), ('years', 'ago'), ('Los',
'Angeles'), ('White', 'House'), ('Peace', 'Corps'), ('World', 'War'), ('San', 'Francisco')]

Most common words in the brown corpus.

[('the', 62713), (',', 58334), ('.', 49346), ('of', 36080), ('and', 27915), ('to', 25732), ('a',
21881), ('in', 19536), ('that', 10237), ('is', 10011)]
```

3) **Lemmatizing- Design a Python program to group together the different inflected forms of a word so they can be analyzed as a single item.**

```
import nltk
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize

# Sample text
text = "The cats are chasing mice in the garden. "

# Tokenize the text
words = word_tokenize(text)

# Initialize the WordNetLemmatizer
lemmatizer = WordNetLemmatizer()

# Lemmatize each word
lemmatized_words = [lemmatizer.lemmatize(word) for word in words]

# Print original and lemmatized words
print("Original words:", words)
print("Lemmatized words:", lemmatized_words)
```

Output:

```
Original words: ['The', 'cats', 'are', 'chasing', 'mice', 'in', 'the', 'garden', '.']
Lemmatized words: ['The', 'cat', 'are', 'chasing', 'mouse', 'in', 'the', 'garden', '.']
```

4) **Process-Implement a python program to process the given text.**

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

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

text = "Hello! This is a sample text. It includes punctuation marks, like commas, periods, and exclamation marks!"

# Lowercasing and removing punctuation
text = text.lower()
text = ''.join(char for char in text if char not in string.punctuation)

# Tokenization
tokens = word_tokenize(text)

# Removing stop words
stop_words = set(stopwords.words('english'))
tokens = [token for token in tokens if token not in stop_words]

print("Processed text:", tokens)
```

Output: Processed text: ['hello', 'sample', 'text', 'includes', 'punctuation', 'marks', 'like', 'commas', 'periods', 'exclamation', 'marks']

Module 2

1) **Getting text to analyze- Design a Python program to analyze the given text**

```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
text = "I absolutely love this movie! the acting is fantastic and the storyline is captivating"
sid= SentimentIntensityAnalyzer()
sentiment_scores = sid.polarity_scores(text)

if sentiment_scores['compound'] >= 0.05:
   sentiment = "Positive"

elif sentiment_scores['compound'] <= -0.05:
   sentiment = "Negative"

else:
    sentiment = "Neutral"

print("Text: ", text)
print("Sentiment: ", sentiment)
print("Sentiment scores: ", sentiment_scores)
```

Output:

```
Text:  I absolutely love this movie! the acting is fantastic and the storyline is captivating
Sentiment:  Positive
Sentiment scores:  {'neg': 0.0, 'neu': 0.567, 'pos': 0.433, 'compound': 0.855}
```

2) **POS Tagger- Design python program to perform part-of-speech tagging on the text scraped from a website.**

```
import requests
from bs4 import BeautifulSoup

url = "https://www.snickers.com/"  # Replace with the actual website URL
response = requests.get(url)
soup = BeautifulSoup(response.content, "html.parser")
text = soup.get_text()  # Extract text from HTML

# Tokenize the text
tokens = nltk.word_tokenize(text)

# Perform part-of-speech tagging
tagged_tokens = nltk.pos_tag(tokens)

print(tagged_tokens)
```

Output:

```
[('Just', 'RB'), ('a', 'DT'), ('moment', 'NN'), ('...', ':'), ('Enable', 'JJ'), ('JavaScript', 'NNP'), ('and', 'CC'), ('cookies', 'NNS'), ('to', 'TO'), ('continue', 'VB')]
```

3) **Default Tagger- Design python program to illustrate default tagger.**

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.tag import DefaultTagger

text = "This is an example sentence for illustrating default tagger"
words = word_tokenize(text)

default_tagger = DefaultTagger('NN')
tagged_words = default_tagger.tag(words)

print("Tagged words: ")
for word, tag in tagged_words:
    print(f"{word} : {tag}")
```

Output: Tagged words:

```
This : NN
is : NN
an : NN
example : NN
sentence : NN
for : NN
illustrating : NN
default : NN
tagger : NN
```

4) **Chunking- Design a python program to group similar words together based on the nature of the word.**

```
import nltk
from nltk import word_tokenize, pos_tag

def pos_tagging(text):
    tokens = word_tokenize(text)
    tagged_words = nltk.pos_tag(tokens)
    return tagged_words

def group_similar_words(tagged_words):
    grouped_words ={}
    for word, pos_tag in tagged_words:
        if pos_tag not in grouped_words:
            grouped_words[pos_tag] = []
        grouped_words[pos_tag].append(word)
    return grouped_words

text = "The cat is chasing the mouse. A dog is barking loudly."
tagged_words = pos_tagging(text)
grouped_words = group_similar_words(tagged_words)
for pos_tag, words in grouped_words.items():
    print(f"POS Tag: {pos_tag}, Words: {words} ")
```

Output: POS Tag: DT, Words: ['The', 'the', 'A']

```
POS Tag: NN, Words: ['cat', 'mouse', 'dog']
POS Tag: VBZ, Words: ['is', 'is']
POS Tag: VBG, Words: ['chasing', 'barking']
POS Tag: ., Words: ['.', '.']
POS Tag: RB, Words: ['loudly']
```

5) **Chinking- Design a Python program to remove a sequence of tokens from a chunk.**

```
def chink_text(text, chink_pattern):
    tokens = nltk.word_tokenize(text)
    chinked_tokens = [(word, tag) for (word,tag) in nltk.pos_tag(nltk.word_tokenize(chink_pattern))]
    tagged_tokens = nltk.pos_tag(tokens)
    cleaned_tokens = [token for token in tagged_tokens if token not in chinked_tokens]
    cleaned_text = " ".join([word for word, _ in cleaned_tokens])
    return cleaned_text

if __name__ == "__main__":
    text = "The quick brown fox jumps over the lazy dog"

    chink_pattern = "quick brown fox"
    cleaned_text = chink_text(text, chink_pattern)
    print("Cleaned Text: ")
    print(cleaned_text)
```

Output: Cleaned Text:
The jumps over the lazy dog

Module 3

1) **N grams- Implement a Python program to implement N-Gram**

```
def generate_ngrams(text,n):
    words = text.split()
    ngrams = []
    for i in range(len(words)-n+1):
        ngrams.append(words[i:i+n])
    return ngrams
text = "This is a sample text for generating n-grams"
n=3
result = generate_ngrams(text,n)
print(result)
```

Output: [['This', 'is', 'a'], ['is', 'a', 'sample'], ['a', 'sample', 'text'], ['sample', 'text', 'for'], ['text', 'for', 'generating'], ['for', 'generating', 'n-grams']]

2) **Smoothing-Design a Python program to perform smoothing using various methods in Python.**

```
def laplace_smoothing(word_counts, vocab_size):
    smoothed_counts = {}
    total_words = sum(word_counts.values())

    for word, count in word_counts.items():
        smoothed_counts[word] = (count+1)/(total_words + vocab_size)
    return smoothed_counts

word_counts = {'apple':3,'banana':2, 'orange':1}
vocab_size =100
```

```
smoothed_count=laplace_smoothing(word_counts,vocab_size)
print('Original count',word_counts)
print("Smooth count",smoothed_count)
```

Output: Original count {'apple': 3, 'banana': 2, 'orange': 1}
Smooth count {'apple': 0.03773584905660377, 'banana': 0.02830188679245283, 'orange': 0.018867924528301886}

3) **Good turing- Develop a Python program to calculate good turing frequency.**

```
from collections import Counter
def good_turing(frequencies):
    freq_of_freq = Counter(frequencies)

    good_turing_frequencies = {}
    for freq, freq_count in freq_of_freq.items():
        if freq+1 in freq_of_freq:
            good_turing_frequencies[freq] = (freq +1) * (freq_of_freq[freq +1]/freq_count)
        else:
            good_turing_frequencies[freq] = freq_count/len(frequencies)
    return good_turing_frequencies

text = "The quick brown fox jumps over the lazy dog"

word_lengths = [len(word) for word in text.split()]
word_length_counts = Counter(word_lengths)
good_turing_frequencies = good_turing(word_lengths)
print("Word Length\tFrequency\tGood-turing Frequency")
for length, freq in word_length_counts.items():
    gt_freq = good_turing_frequencies[length]
    print(f"{length}\t\t{freq}\t\t{gt_freq}")
```

Output:

```
Word Length  Frequency     Good-turing Frequency
3            4             2.0
5            3             0.3333333333333333
4            2             7.5
```

Module 4

1) **Lexical Semantics- Design Python program to do text classification.**