**Practical 1**

**Aim:** Write a program to implement sentence segmentation and word tokenization.

**Background Information:**

**Sentence Segmentation:** Sentence tokenization (also called sentence segmentation) is the problem of dividing a string of written language into its component sentences.

**Word Tokenization:** Tokenization is used in natural language processing to split paragraphs and sentences into smaller units that can be more easily assigned meaning.

**Code:**

import nltk

import spacy

nlp = spacy.load("en\_core\_web\_sm")

# sentence segmentation (part 1)

doc = nlp(u"I Love Coding. Geeks for Geeks helped me in this regard very much. I Love Geeks

for Geeks.")

for sent in doc.sents:

print(sent)

# word Tokenization (part 2)

text = "A good traveler has no fixed plans and is not intent on arriving"

sentences = nltk.sent\_tokenize(text)

for sentence in sentences:

words = nltk.word\_tokenize(sentence) print(words)

**Output:**

I Love Coding.

Geeks for Geeks helped me in this regard very much.

I Love Geeks for Geeks.

['A', 'good', 'traveler', 'has', 'no', 'fixed', 'plans', 'and', 'is', 'not', 'intent', 'on', 'arriving']

**Conclusion:** Thus, we have successfully implemented sentence segmentation and word

tokenization.

**Practical 2**

**Aim:** Write a program to Implement stemming and lemmatization.

**Background Information:**

**Stemming:** Stemming is the process of reducing a word to its stem that affixes to suffixes and prefixes or to the roots of words known as "lemmas".

**Lemmatization:** Lemmatization is a text pre-processing technique used in natural language processing (NLP) models to break a word down to its root meaning to identify similarities. For example, a lemmatization algorithm would reduce the word better to its root word, or lemme, good.

**Code:**

import nltk

# nltk.download('punkt')

# nltk.download('wordnet')

words = ['eating', 'eats', 'eaten', 'eat']

stemmer = nltk.stem.PorterStemmer()

stemmed\_words = [stemmer.stem(word) for word in words]

print("Stemmed words:", stemmed\_words)

lemmatizer = nltk.stem.WordNetLemmatizer()

lemmatized\_words = [lemmatizer.lemmatize(word) for word in words]

print("Lemmatized words:", lemmatized\_words)

**Output:**

Stemmed words: ['eat', 'eat', 'eaten', 'eat']

Lemmatized words: ['eating', 'eats', 'eaten', 'eat']

**Conclusion:** Thus, we have successfully implemented stemming and lemmatization.

**Practical 3**

**Aim:** Write a program to Implement a tri-gram model.

**Background Information:**

N-gram: N-grams are contiguous sequences of items that are collected from a sequence of text or speech corpus or almost any type of data. The n in n-grams specify the size of number of items to consider, unigram for n =1, bigram for n = 2, and trigram for n = 3, and so on.

**Code:**

import nltk

from nltk.util import ngrams

text = "The flame that burns Twice as bright burns half as long"

words = nltk.word\_tokenize(text)

trigrams = ngrams(words, 3)

for trigram in trigrams:

print(trigram)

**Output:**

('The', 'flame', 'that')

('flame', 'that', 'burns')

('that', 'burns', 'Twice')

('burns', 'Twice', 'as')

('Twice', 'as', 'bright')

('as', 'bright', 'burns')

('bright', 'burns', 'half')

('burns', 'half', 'as')

('half', 'as', 'long')

**Conclusion:** Thus, we have successfully a tri-gram model.

**Practical 4**

**Aim:** Write a program to Implement POS tagging using HMM & Neural Model.

**Background Information:**

**POS Tagging:** In corpus linguistics, part-of-speech tagging (POS tagging or PoS tagging or POST), also called grammatical tagging is the process of marking up a word in a text (corpus) as corresponding to a particular part of speech, based on both its definition and its context.

**HMM:** HMM (Hidden Markov Model) is a Stochastic technique for POS tagging. Hidden Markov models are known for their applications to reinforcement learning and temporal pattern recognition such as speech, handwriting, gesture recognition, musical score following, partial

discharges, and bioinformatics.

**Neural Network:** A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates.

**Code:**

import nltk

nltk.download('punkt')

nltk.download('averaged\_perceptron\_tagger')

nltk.download('universal\_tagset')

text = "Joe waited for the train, but the train was late"

words = nltk.word\_tokenize(text)

hmm\_tagged = nltk.pos\_tag(words)

print("PoS tagging using HMM:", hmm\_tagged)

nn\_tagged = nltk.pos\_tag(words, tagset='universal')

print("PoS tagging using NN:", nn\_tagged)

**Output:**

PoS tagging using HMM: [('Joe', 'NNP'), ('waited', 'VBD'), ('for', 'IN'), ('the', 'DT'), ('train', 'NN'),

(',', ','), ('but', 'CC'), ('the', 'DT'), ('train', 'NN'), ('was', 'VBD'), ('late', 'JJ')]

PoS tagging using NN: [('Joe', 'NOUN'), ('waited', 'VERB'), ('for', 'ADP'), ('the', 'DET'), ('train',

'NOUN'), (',', '.'), ('but', 'CONJ'), ('the', 'DET'), ('train', 'NOUN'), ('was', 'VERB'), ('late', 'ADJ')

**Conclusion:** Thus, we have successfully implemented PoS tagging using HMM & Neural Model

**Practical 5**

**Aim:** Write a program to Implement syntactic parsing of a given text.

**Background Information:**

**Syntactic Parsing:** Syntactic parsing involves the analysis of words in the sentence for grammar and their arrangement in a manner that shows the relationships among the words.

**Code:**

import nltk

# nltk.download('punkt')

# nltk.download('averaged\_perceptron\_tagger')

# nltk.download('maxent\_ne\_chunker')

# nltk.download('words')

# nltk.download('treebank')

text = "I ate hot ice-cream ,before match start"

words = nltk.word\_tokenize(text)

tagged\_words = nltk.pos\_tag(words)

syntactic\_tree = nltk.ne\_chunk(tagged\_words, binary=True)

print("Syntactic tree:", syntactic\_tree)

**Output:**

Syntactic tree: (S I/PRP ate/VBP hot/JJ ice-cream/NN ,/, before/IN match/JJ start/NN)

**Conclusion:** Thus, we have successfully implemented syntactic parsing of a given text.

**Practical 6**

**Aim:** Write a program to Implement dependency parsing of a given text.

**Background Information:**

**Dependency Parsing:** The term Dependency Parsing (DP) refers to the process of examining the dependencies between the phrases of a sentence in order to determine its grammatical structure. A sentence is divided into many sections based mostly on this. The process is based on the assumption that there is a direct relationship between each linguistic unit in a sentence. These

hyperlinks are called dependencies.

**Code:**

import spacy

nlp = spacy.load("en\_core\_web\_sm") #nlp model

text = "John likes Mary because she is beautiful."

doc = nlp(text)

for token in doc:

print(token.text, token.dep\_, token.head.text, token.head.pos\_,[child for child in token.children])

**Output:**

John nsubj likes VERB []

likes ROOT likes VERB [John, Mary, is, .]

Mary dobj likes VERB []

because mark is AUX []

she nsubj is AUX []

is advcl likes VERB [because, she, beautiful]

beautiful acomp is AUX []

. punct likes VERB []

**Conclusion:** Thus, we have successfully implemented Implement dependency parsing of a given text.

**Practical 7**

**Aim**: Write a program to Implement Named Entity Recognition (NER).

**Background Information:**

**Named Entity Recognition:** The named entity recognition (NER) is one of the most popular data preprocessing task. It involves the identification of key information in the text and classification into a set of predefined categories. An entity is basically the thing that is consistentlytalked about or refer to in the text.

Some of the categories that are the most important architecture in NER such that:

Person

Organization

Place/ location

**Code:**

import spacy

nlp = spacy.load("en\_core\_web\_sm")

text = "Nitin is studying at Indian Institute of technology Bombay."

doc = nlp(text)

for entity in doc.ents:

print(entity.label\_, entity.text)

**Output:**

PERSON Nitin

ORG Indian Institute of technology

GPE Bombay

**Conclusion:** Thus, we have successfully implemented Named Entity Recognition (NER)

**Practical 8**

**Aim:** Path Write a program to Implement Text Summarization for the given sample text.

**Background Information:**

Text Summarization:NLP text summarization is the process of breaking down lengthy text into digestible paragraphs or sentences. This method extracts vital information while also preserving the meaning of the text. This reduces the time required for grasping lengthy pieces such

as articles without losing vital information.

**Code:**

import nltk

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize, sent\_tokenize

from heapq import nlargest

text = """

Natural language processing (NLP) is a branch of artificial intelligence that focuses on the

interaction between computers and human language. NLP has been around for several decades, but

recent advances in machine learning and deep learning have dramatically improved its capabilities.

NLP is used in a wide range of applications, from virtual assistants like Siri and Alexa to

sentiment analysis, machine translation, and even content generation. NLP involves a range of

techniques, including tokenization, part-of-speech tagging, named entity recognition, and

sentiment analysis, among others. These techniques can be used to analyze and understand human

language in a variety of contexts, from social media posts to scientific literature. Despite its many

successes, NLP remains a challenging field, as natural language is complex and often ambiguous.

As NLP continues to evolve, it has the potential to transform the way we interact with technology

and with each other, opening up new possibilities for communication, collaboration, and creativity.

"""

num\_sentences = 2

sentences = sent\_tokenize(text)

words = word\_tokenize(text)

stop\_words = set(stopwords.words('english'))

word\_freq = {}

for word in words:

if word not in stop\_words:

if word not in word\_freq:

word\_freq[word] = 1

else:

word\_freq[word] += 1

max\_freq = max(word\_freq.values())

for word in word\_freq.keys():

word\_freq[word] = (word\_freq[word]/max\_freq)

sent\_scores = {}

for sentence in sentences:

for word in word\_tokenize(sentence.lower()):

if word in word\_freq.keys():

if len(sentence.split(' ')) < 30:

if sentence not in sent\_scores.keys():

sent\_scores[sentence] = word\_freq[word]

else:

sent\_scores[sentence] += word\_freq[word]

summary\_sentences = nlargest(num\_sentences, sent\_scores, key=sent\_scores.get)

summary = ' '.join(summary\_sentences)

print(summary)

**Output:**

NLP involves a range of techniques, including tokenization, part-of-speech tagging, named entity

recognition, and sentiment analysis, among others. NLP is used in a wide range of applications,

from virtual assistants like Siri and Alexa to sentiment analysis, machine translation, and even

content generation.

**Conclusion:** Thus, we have successfully implemented Text Summarization for the given sample

text.