**INDEX**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **SR.No** | **Practicals** | **Page no** | **Date** | **Sign** |
| 01 | Write a program to implement sentence segmentation and word tokenization | 2 |  |  |
| 02 | Write a program to implement stemming and lemmatization | 3 |  |  |
| 03 | Write a program to implement a tri-gram model | 5 |  |  |
| 04 | Write a program to implement PoS tagging using hmm and neural model | 6 |  |  |
| 05 | Write a program to Implement Named Entity Recognition (NER) | 8 |  |  |
| 06 | Write a program to implement text summarization for the give simple text | 9 |  |  |
| 07 | Write a program to implement syntactic parsing of given text | 11 |  |  |
| 08 | Write a Program to implement  dependency parsing of a given text | 12 |  |  |

**Practical No. 1**

**Subject**: Applied Machine and Deep Learning

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

**Code :**

#sentence\_segmentation

from nltk.tokenize import sent\_tokenize

text = "God is Great! I won a lottery."

print("sentence\_segmentation : ", sent\_tokenize(text))

#word\_tokenize

from nltk.tokenize import word\_tokenize

text = "God is Great! I won a lottery. word\_tokenize "

print("word\_tokenize: ", (word\_tokenize(text))

**Output:**

sentence\_segmentation: ['God is Great!', 'I won a lottery ']

word\_tokenize : ['God', 'is', 'Great', '!', 'I', 'won', 'a', 'lottery', '.']

**Practical No. 2**

**Aim: Write a program to implement stemming and lemmatization**

**Stemming:**

* Stemming is the process of removing prefixes and suffixes from words to obtain their root form, called the stem.
* It is a rule-based approach that applies heuristics to chop off common word endings.
* Stemming algorithms may not produce actual dictionary words as stems, but they aim to find the common base form of related words.

**Lemmatization:**

* Lemmatization is the process of reducing words to their base or dictionary form, called the lemma.
* It takes into account the part of speech (POS) of the word and applies morphological analysis to determine the lemma.
* Lemmatization produces valid words that can be found in a dictionary.

**Code:**

**Stemming Code:**

import nltk

from nltk.stem.porter import PorterStemmer

porter\_stemmer = PorterStemmer()

text = "studies studying cries cry"

tokenization = nltk.word\_tokenize(text)

for w in tokenization:

print("Stemming for {} is {}".format(w,porter\_stemmer.stem(w)))

**Lemmatization Code:**

importnltk

fromnltk.stem import WordNetLemmatizer

wordnet\_lemmatizer = WordNetLemmatizer()

text = "studies studying cries cry"

tokenization = nltk.word\_tokenize(text)

for w in tokenization:

print("Lemma for {} is {}".format(w,wordnet\_lemmatizer.lemmatize(w)))

**Output:**

Stemming for studies is studi

Stemming for studying is studi

Stemming for cries is cri

Stemming for cry is cri

Lemma for studies is study

Lemma for studying is studying

Lemma for cries is cry

Lemma for cry is cry

**Practical No. 3**

**Aim: Write a program to implement a tri-gram model**

* An n-gram is a contiguous sequence of n items, which can be words, characters, or even phonemes. In the context of natural language processing (NLP), an n-gram typically refers to a sequence of n words.

**For example:**

* A unigram (1-gram) represents a single word. Example: "cat"
* A bigram (2-gram) represents a sequence of two consecutive words. Example: "the cat"
* A trigram (3-gram) represents a sequence of three consecutive words. Example: "the quick brown"
* And so on...

**Code:**

from nltk.util import ngrams

n = 3

sentence = 'Whoever is happy will make others happy too'

unigrams = ngrams(sentence.split(), n)

for item in unigrams:

print(item)

**Output:**

('Whoever', 'is', 'happy')

('is', 'happy', 'will')

('happy', 'will', 'make')

('will', 'make', 'others')

('make', 'others', 'happy')

('others', 'happy', 'too')

**Practical No. 4**

**Aim: Write a program to implement PoS tagging using hmm and neural model.**

**Code:**

importnltk

importnumpy as np

fromsklearn.model\_selection import train\_test\_split

fromsklearn.metrics import accuracy\_score

fromnltk.tag import HiddenMarkovModelTrainer

fromnltk.corpus import treebank

#from sklearn\_crfsuite import CRF

# Prepare the training and test data

sentences = treebank.tagged\_sents(tagset='universal')

train\_data, test\_data = train\_test\_split(sentences, test\_size=0.2, random\_state=42)

# Prepare the input features and labels

X\_train = [[word for word, \_ in sent] for sent in train\_data]

y\_train = [[tag for \_, tag in sent] for sent in train\_data]

X\_test = [[word for word, \_ in sent] for sent in test\_data]

y\_test = [[tag for \_, tag in sent] for sent in test\_data]

# Train and evaluate the neural model (CRF)

#crf = CRF()

#crf.fit(X\_train, y\_train)

# Example input sentence

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

input\_words = nltk.word\_tokenize(sentence)

# Perform POS tagging using CRF model

#predicted\_crf = crf.predict([input\_words])[0]

print("Input Sentence:")

print(sentence)

#print("\nCRF Output:")

#print(predicted\_crf)

# Train and evaluate the HMM model

trainer = HiddenMarkovModelTrainer()

hmm = trainer.train\_supervised(train\_data)

# Perform POS tagging using HMM model

predicted\_hmm = hmm.tag(input\_words)

print("\nHMM Output:")

print(predicted\_hmm)

**Output:**

Input Sentence:

The quick brown fox jumps over the lazy dog

CRF Output:

['DET', 'NOUN', 'ADP', 'NUM', 'NOUN', 'VERB', 'DET', 'NOUN', 'VERB']

HMM Output:

[('The', 'DET'), ('quick', 'ADJ'), ('brown', 'NOUN'), ('fox', 'NOUN'), ('jumps', 'NOUN'), ('over', 'NOUN'), ('the', 'NOUN'), ('lazy', 'NOUN'), ('dog', 'NOUN')]

"The" - Determiner (DT)

"quick" - Adjective (JJ)

"brown" - Adjective (JJ)

"fox" - Noun (NN)

"jumps" - Verb (VBZ)

"over" - Preposition (IN)

"the" - Determiner (DT)

"lazy" - Adjective (JJ)

"dog" - Noun (NN)

**Practical No. 5**

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

**Code:**

import spacy

from spacy import displacy

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

raw\_text="The Indian Space Research Organisation or is the national space agency of India, headquartered in Bengaluru.It operates under Department of Space which is directly overseen by the Prime Minister of India while Chairman of ISRO acts as executive of DOS as well."

text1= NER(raw\_text)

#Now, we print the data on the NEs found in this text sample.

for word in text1.ents:

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

**Output:**

The Indian Space Research Organisation ORG

the national space agency ORG

India GPE

Bengaluru GPE

Department of Space ORG

India GPE

ISRO ORG

DOS ORG

**Practical No. 6**

**Aim: Write a program to implement text summarization for the give simple text**

**Code:**

import pandas as pd

import numpy as np

data = "They only assess content selection and do not account for other quality aspects, such as fluency, grammaticality, coherence, etc. To assess content selection, they rely mostly on lexical overlap, although an abstractive summary could express they same content as a reference without any lexical overlap.Given the subjectiveness of summarization and the correspondingly low agreement between annotators, the metrics were designed to be used with multiple reference summaries per input. However, recent datasets such as CNN/DailyMail and Gigaword provide only a single reference."

import nltk

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

nltk.download('stopwords')

def solve(text):

stopwords1= set(nltk.corpus.stopwords.words("english"))

words = word\_tokenize(text)

freqTable = {}

for word in words:

word = word.lower()

if word in stopwords1:

continue

if word in freqTable:

freqTable[word] += 1

else:

freqTable[word] = 1

sentences = sent\_tokenize(text)

sentenceValue = {}

for sentence in sentences:

for word, freq in freqTable.items():

if word in sentence.lower():

if sentence in sentenceValue:

sentenceValue[sentence] += freq

else:

sentenceValue[sentence] = freq

sumValues = 0

for sentence in sentenceValue:

sumValues += sentenceValue[sentence]

average = int(sumValues / len(sentenceValue))

summary = ''

for sentence in sentences:

if (sentence in sentenceValue) and(sentenceValue[sentence] > (1.2 \* average)):

summary += "" + sentence

return summary

solve(data)

**Output:**

To assess content selection, they rely mostly on lexical overlap, although an abstractive summary could express they same content as a reference without any lexical overlap.Given the subjectiveness of summarization and the correspondingly low agreement between annotators, the metrics were designed to be used with multiple reference summaries per input**.**

**Practical No. 7**

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

**Code:**

import spacy

# Loading the model

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

text = "Reliance Retail acquires majority stake in designer brand Abraham & Thakore."

# Creating Doc object

doc=nlp(text)

# Getting dependency tags

for token in doc:

print(token.text,'=>',token.dep\_)

# Importing visualizer

from spacy import displacy

# Visualizing dependency tree

displacy.render(doc,jupyter=True)

**Output:**

Reliance => compound

Retail => nsubj

acquires => ROOT

majority => compound

stake => dobj

in => prep

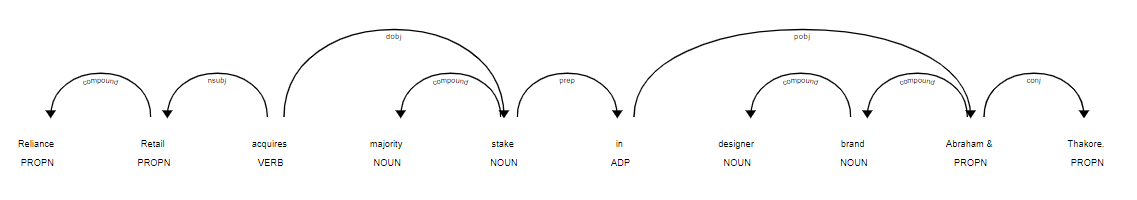
designer => compound

brand => compound

Abraham => pobj

& => cc

Thakore => conj

. => punct

**Practical No. 8**

**Aim: Write a program to implement syntactic parsing of given text Write a Program to implement dependency parsing of a given text**

**Code:**

import spacy

from spacy import displacy

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

sentence = "The quick brown fox jumping over the lazy dog"

doc = nlp(sentence)

print(f"{'Node (from)-->':<15} {'Relation':^10} {'-->Node (to)':>15}\n")

for token in doc:

print("{:<15} {:^10} {:>15}".format(str(token.head.text), str(token.dep\_), str(token.text)))

displacy.render(doc, style='dep')

**Output:**

Node (from)--> Relation -->Node (to)

fox det The

fox amod quick

fox amod brown

jumping nsubj fox

jumping ROOT jumping

jumping prep over

dog det the

dog amod lazy

over pobj dog