

GOKHALE EDUCATION SOCIETY'S



N. B. MEHTA (V) SCIENCE COLLEGE

ACHARYA BHISE VIDYANAGAR, BORDI

TAL - DAHANU, DIST. PALGHAR, 401701, Tel. 02528 - 254357



**DEPARTMENT OF INFORMATION TECHNOLOGY
AND
COMPUTER SCIENCE**

Certificate

Class _____

Year _____

This is to certify that the work entered in this journal is the work of
Shri/Kumari _____

Of _____ division _____ Roll No. _____

Uni. Exam No. _____ has satisfactorily completed the required University
of Mumbai Practicals on Subject PSCSP202: Natural Language Processing and worked for
the Master of Computer Science 1st term / 2nd term/ both the terms of the Year
_____ in the college laboratory as laid down by the university.

Head of the
Department

External
Examiner

Internal Examiner
Subject teacher

Date: / / 2023 Department of IT-CS

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

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:

```
import nltk
from nltk.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:

```
import nltk
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from nltk.tag import HiddenMarkovModelTrainer
from nltk.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 the 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

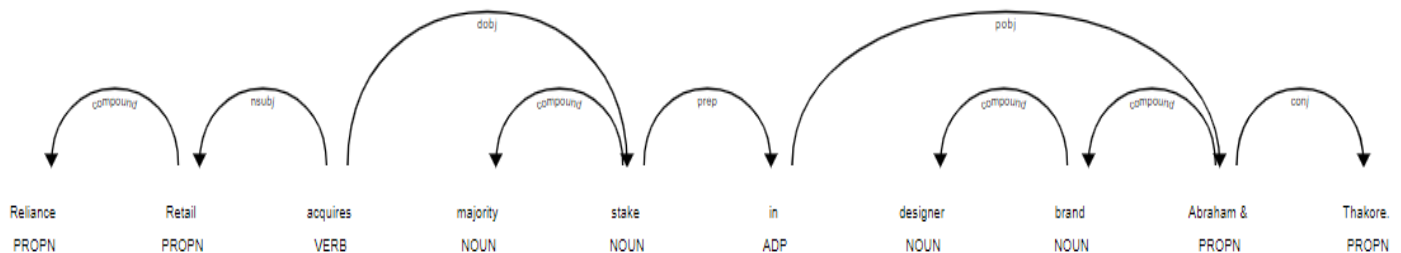
Aim: Write a program to implement syntactic parsing of given text

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