Rayat Shikshan Sanstha's

KARMAVEER BHAURAO PATIL COLLEGE, VASHI

[Autonomous College]

Reaccredited by NAAC with Grade 'A+' (CGPA 3.53) | ISO 9001: 2008 Certified Institute

'Best College' Award by University of Mumbai

Natural Language Processing(NLP)

NAME : Shahzaad Khan

ROLL NO : 240726

CLASS : Msc Part 2(Data Science)

SUBJECT : Natural Language Processing

"Education through self-help is our Motto"-KARMAVEER

RAYAT SHIKSHAN SANSTHA'S

Karmaveer Bhaurao Patil College

Vashi, Navi Mumbai – 400703

(Empowered Autonomous College)

DEPARTMENT OF DATA SCIENCE

CERTIFICATE

This is to certify that **Mr.Shahzaad Firoz Khan** student of M.Sc. Part-II Data Science class from Karmaveer Bhaurao Patil College, Vashi, Navi Mumbai has satisfactorily completed the Practical cource in **Natural Language Processing(NLP)** during the academic year 2024-2025.

Roll No: PG - 240726

Exam No:

In charge Faculty
Date : / /2024

Examine

Head
Department of Data Science

Name:-Ketki Kumbhar Roll No:-240727

INDEX

Sr.No	Practical Name	Date	Signature
1	a. Convert the text into tokens	11/12/2024	
	b. Find the word frequency	17/12/2024	
	c. Demonstrate a bigram language model	19/12/2024	
	d. Demonstrate a trigram language model	19/12/2024	
	e. Generate regular expression for a given text	11/12/2024	
	f. Text Normalization	17/12/2024	
2	a. Perform Lemmatization	17/12/2024	
	b. Perform Stemming	19/12/2024	
	c. Identify parts-of Speech using Penn Treebank tag set.	07/01/2025	
	d. Implement HMM for POS tagging	21/01/2025	
	e. Build a Chunker	21/01/2025	
	f. Text summarization		
3	a. Find the synonym of a word using WordNet	05/02/2025	
	b. Find the antonym of a word	05/02/2025	
	c. Implement semantic role labeling to identify named entities	05/02/2025	
	d. Resolve the ambiguity	05/02/2025	
	e. Translate the text using First-order logic	05/02/2025	
4	a. Implement RNN for sequence labeling	12/02/2025	
	b. Implement POS tagging using LST	13/02/2025	
	c. Implement Named Entity Recognizer	13/02/2025	
	d. Word sense disambiguation by LSTM/GRU	13/02/2025	
5	a. Develop an application on a review system.	23/01/2025	
	b. Create a chatbot for HITS	23/01/2025	
6	Perform Sentiment Analysis	11/02/2025	
7	Develop a project on Social Media Platform Classification.	23/01/2025	

Name: Shahzaad Firoz Khan **Roll No. 240726**

Practical No:-1

Aim:- a. Convert the text into tokens

b. Find the word frequency

c. Demonstrate a bigram language model

d. Demonstrate a trigram language model

e. Generate regular expression for a given text

f. Text Normalization

a. Convert the text into tokens

Code:

```
import nltk

from nltk import word_tokenize, sent_tokenize, LineTokenizer

myself = " We are all different from each other and it is important to self-analyze and know about yourself. Only you can know everything about yourself. But, when it comes to describing yourself in front of others many students fail to do so. This happens due to the confusion generated by a student’s mind regarding what things to include in their description. This confusion never arises when someone is told to give any opinion about others. This blog will help students and children resolve the confusion and it also includes an essay on myself."

print("Word Tokenize:\n",nltk. word_tokenize(myself))

print("\nSentence Tokenize:", nltk.sent_tokenize(myself))

print("\nLine Tokenize:\n", nltk.line_tokenize(myself))

b=LineTokenizer()

print(b.tokenize(myself))
```

Output:

```
Word Tokenize:
 ['We', 'are', 'all', 'different', 'from', 'each', 'other', 'and', 'it', 'is', 'important', 'to', 'self-analyze', 'and', 'kno
w', 'about', 'yourself', '.', 'Only', 'you', 'can', 'know', 'everything', 'about', 'yourself', '.', 'But', ',', 'when', 'it',
'comes', 'to', 'describing', 'yourself', 'in', 'front', 'of', 'others', 'many', 'students', 'fail', 'to', 'do', 'so', '.', 'T
his', 'happens', 'due', 'to', 'the', 'confusion', 'generated', 'by', 'a', 'student', '’', 's', 'mind', 'regarding', 'what', '
things', 'to', 'include', 'in', 'their', 'description', '.', 'This', 'confusion', 'never', 'arises', 'when', 'someone', 'is',
'told', 'to', 'give', 'any', 'opinion', 'about', 'others', '.', 'This', 'blog', 'will', 'help', 'students', 'and', 'children'
, 'resolve', 'the', 'confusion', 'and', 'it', 'also', 'includes', 'an', 'essay', 'on', 'myself', '.']

Sentence Tokenize: [' We are all different from each other and it is important to self-analyze and know about yourself.', 'On
ly you can know everything about yourself.', 'But, when it comes to describing yourself in front of others many students fail
to do so.', 'This happens due to the confusion generated by a student’s mind regarding what things to include in their descri
ption.', 'This confusion never arises when someone is told to give any opinion about others.', 'This blog will help students
and children resolve the confusion and it also includes an essay on myself.']

Line Tokenize:
 [' We are all different from each other and it is important to self-analyze and know about yourself. Only you can know every
thing about yourself. But, when it comes to describing yourself in front of others many students fail to do so. This happens
due to the confusion generated by a student’s mind regarding what things to include in their description. This confusion neve
r arises when someone is told to give any opinion about others. This blog will help students and children resolve the confusi
on and it also includes an essay on myself.']
[' We are all different from each other and it is important to self-analyze and know about yourself. Only you can know everyt
hing about yourself. But, when it comes to describing yourself in front of others many students fail to do so. This happens d
ue to the confusion generated by a student’s mind regarding what things to include in their description. This confusion never
arises when someone is told to give any opinion about others. This blog will help students and children resolve the confusion
and it also includes an essay on myself.']
```

b. Find the word frequency

Code:

```
from nltk import word_tokenize
txt=input("Enter the String:")
def tokenization(str1):
    a=word_tokenize(str1)
    return a
tokenization(txt)
txt="Time after time there were cyclones coming in Karnataka"
a=tokenization(txt)
print("Word tokens are: ",a)
for i in range(len(a)):
    a[i]=a[i].lower()
print("After applying lower() mtehod: ",a)
count=[]
for i in a:
    i=i.strip(".")
    if i not in count:
        count.append(i)
print("\nFrequencies of words: ")
for i in range(0,len(count)):
    print(count[i], a.count(count[i]))
```

Output:

```
Enter the String:Ketki
Word tokens are:  ['Time', 'after', 'time', 'there', 'were', 'cyclones', 'coming', 'in', 'Karnataka']
After applying lower() mtehod:  ['time', 'after', 'time', 'there', 'were', 'cyclones', 'coming', 'in', 'karnataka']

Frequencies of words:
time 2
after 1
there 1
were 1
cyclones 1
coming 1
in 1
karnataka 1
```

c. Demonstrate a bigram language model

Code:

```
import nltk
txt=input("Enter the text: ")
no=2
def bi_gram(text,no):
    tokens=nltk.word_tokenize(txt)
    print("Tokens are: ",tokens)
    x=len(tokens)
    ngram_formula=x-(no-1)
    print("Number of grams are ", no, " So, the total number of word tokens by this grams are ",ngram_formula)
    n_gram=[]
    for i in range(ngram_formula):
        temp=[tokens[j] for j in range(i, i+no)]
        n_gram.append(" ".join(temp))
    print("Bi grams are:\n",n_gram)
bi_gram(txt, no)
```

Output:

```
Enter the text: Spark docker images are available from Dockerhub under the accounts of both The Apache Software Foundation an
d Official Images.
Tokens are:  ['Spark', 'docker', 'images', 'are', 'available', 'from', 'Dockerhub', 'under', 'the', 'accounts', 'of', 'both',
'The', 'Apache', 'Software', 'Foundation', 'and', 'Official', 'Images', '.']
Number of grams are  2  So, the total number of word tokens by this grams are  19
Bi grams are:
 ['Spark docker', 'docker images', 'images are', 'are available', 'available from', 'from Dockerhub', 'Dockerhub under', 'und
er the', 'the accounts', 'accounts of', 'of both', 'both The', 'The Apache', 'Apache Software', 'Software Foundation', 'Found
ation and', 'and Official', 'Official Images', 'Images .']
```

d. Demonstrate a trigram language model

Code:

```
import nltk
txt=input("Enter the text: ")
no=3
def bi_gram(text,no):
    tokens=nltk.word_tokenize(txt)
    print("Tokens are: ",tokens)
    x=len(tokens)
    ngram_formula=x-(no-1)
    print("Number of grams are ", no, " So, the total number of word tokens by this grams are ",ngram_formula)
    n_gram=[]
    for i in range(ngram_formula):
        temp=[tokens[j] for j in range(i, i+no)]
        n_gram.append(" ".join(temp))
    print("Bi grams are:\n",n_gram)
bi_gram(txt, no)
```

Output:

```
Enter the text: As new Spark releases come out for each development stream, previous ones will be archived, but they are stil
l available at Spark release archives.
Tokens are:  ['As', 'new', 'Spark', 'releases', 'come', 'out', 'for', 'each', 'development', 'stream', ',', 'previous', 'ones
', 'will', 'be', 'archived', ',', 'but', 'they', 'are', 'still', 'available', 'at', 'Spark', 'release', 'archives', '.']
Number of grams are  3  So, the total number of word tokens by this grams are  25
Bi grams are:
 ['As new Spark', 'new Spark releases', 'Spark releases come', 'releases come out', 'come out for', 'out for each', 'for each
development', 'each development stream', 'development stream ,', 'stream , previous', ', previous ones', 'previous ones will'
, 'ones will be', 'will be archived', 'be archived ,', 'archived , but', ', but they', 'but they are', 'they are still', 'are
still available', 'still available at', 'available at Spark', 'at Spark release', 'Spark release archives', 'release archives
.']
```

e. Generate regular expression for a given text

Code:

```
import re
text = """
Please contact support@example.com for technical issues.
You can also reach out to john.doe123@mail.co.uk or admin@organization.org for assistance.
"""
email_regex = r'[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}'
emails = re.findall(email_regex, text)
print("Extracted email addresses:", emails)
```

Output:

```
Extracted email addresses: ['support@example.com', 'john.doe123@mail.co.uk', 'admin@organization.org']
```

f. Text Normalization

Code:

```
import re
import unicodedata
def normalized_text(text):
    normalized_text = text.lower()
    normalized_text = re.sub(r'[^\w\s]', '', normalized_text)
    normalized_text
=unicodedata.normalize('NFKD',normalized_text).encode('ASCII','ignore').decode('utf-8')
    normalized_text = ''.join(normalized_text.split())
    return normalized_text
txt = input("Enter the text to normalize: ")
normalized_res = normalized_text(txt)
print("Normalized text: ", normalized_res)
```

Output:

```
Enter the text to normalize: ketkikumbhar005@gmail.com
Normalized text:  ketkikumbhar005gmailcom
```

Practical No:- 2

Aim:- a. Perform Lemmatization

b. Perform Stemming

c. Identify parts-of Speech using Penn Treebank tag set.

d. Implement HMM for POS tagging

e. Build a Chunker

f. Text summarization

a. Perform Lemmatization

Code:

```
import nltk
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('omw-1.4')
lemmatizer = WordNetLemmatizer()
sentence = "The cats are running quickly"
words = nltk.word_tokenize(sentence)
lemmatized_words = [lemmatizer.lemmatize(word, pos='v') if word.lower() in ['running'] else lemmatizer.lemmatize(word) for word in words]
print("Original sentence:", sentence)
print("Lemmatized sentence:", " ".join(lemmatized_words))
```

Output:

```
[nltk_data] Downloading package punkt to C:\Users\HP.DESKTOP-
[nltk_data]     JO7A8MB.000\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to C:\Users\HP.DESKTOP-
[nltk_data]     JO7A8MB.000\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to C:\Users\HP.DESKTOP-
[nltk_data]     JO7A8MB.000\AppData\Roaming\nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
Original sentence: The cats are running quickly
Lemmatized sentence: The cat are run quickly
```

b. Perform Stemming

Code:

```
import nltk
from nltk.stem import PorterStemmer, LancasterStemmer
nltk.download('punkt')
porter_stemmer = PorterStemmer()
lancaster_stemmer = LancasterStemmer()
sentence = "The cats are running quickly through the park"
words = nltk.word_tokenize(sentence)
porter_stemmed_words = [porter_stemmer.stem(word) for word in words]
lancaster_stemmed_words = [lancaster_stemmer.stem(word) for word in words]
print("Original sentence:", sentence)
print("Porter Stemmed sentence:", " ".join(porter_stemmed_words))
print("Lancaster Stemmed sentence:", " ".join(lancaster_stemmed_words))
```

Output:

```
Original sentence: The cats are running quickly through the park
Porter Stemmed sentence: the cat are run quickli through the park
Lancaster Stemmed sentence: the cat ar run quick through the park

[nltk_data] Downloading package punkt to C:\Users\HP.DESKTOP-
[nltk_data]     JO7A8MB.000\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

c. Identify parts-of Speech using Penn Treebank tag set.

Code:

```
import nltk
nltk.download('averaged_perceptron_tagger')
sent=input("Enter the sentence: ")
words=nltk.word_tokenize(sent)
tags=nltk.pos_tag(words)
print(tags)
```

Output:

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\HP.DESKTOP-
[nltk_data]     JO7A8MB.000\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!

Enter the sentence: Kaggle's community is a diverse group of 21 million data scientists, ML engineers & enthusiasts from arou
nd the world.
[('Kaggle', 'NNP'), ("'s", 'POS'), ('community', 'NN'), ('is', 'VBZ'), ('a', 'DT'), ('diverse', 'JJ'), ('group', 'NN'), ('o
f', 'IN'), ('21', 'CD'), ('million', 'CD'), ('data', 'NN'), ('scientists', 'NNS'), (',', ','), ('ML', 'NNP'), ('engineers',
'NNPS'), ('&', 'CC'), ('enthusiasts', 'NNS'), ('from', 'IN'), ('around', 'IN'), ('the', 'DT'), ('world', 'NN'), ('.', '.')]
```

d. Implement HMM for POS tagging

Code:

```
import nltk
#nltk.download('treebank)
import random
corpus=nltk.corpus.treebank
sent=corpus.sents()
tagged_sent=corpus.tagged_sents()
random.seed(123)
split_ratio=0.8
split_index=int(len(tagged_sent)*split_ratio)
training_sent = tagged_sent[:split_index]
testing_sent = tagged_sent[split_index:]
#create HMM
from nltk.tag import hmm
hmm_tagger = hmm.HiddenMarkovModelTrainer().train(training_sent)
accuracy=hmm_tagger.evaluate(testing_sent)
new_sent=input("Enter the sentence: ")
new_words=nltk.word_tokenize(new_sent)
predicted_tags = hmm_tagger.tag(new_words)
print("\nTraining data. ", new_words[:split_index])
print("\nPredicted POS tags for the new sentence:", predicted_tags)
print("\nHMM POS Tagger Accuray", accuracy)
```

Output:

```
Enter the sentence: A plagiarism detection system project typically involves creating a tool that analyzes text to identify s
imilarities and potential instances of plagiarism.

Training data.  ['A', 'plagiarism', 'detection', 'system', 'project', 'typically', 'involves', 'creating', 'a', 'tool', 'that
', 'analyzes', 'text', 'to', 'identify', 'similarities', 'and', 'potential', 'instances', 'of', 'plagiarism', '.']

Predicted POS tags for the new sentence: [('A', 'DT'), ('plagiarism', 'NNP'), ('detection', 'NNP'), ('system', 'NNP'), ('proj
ect', 'NNP'), ('typically', 'NNP'), ('involves', 'NNP'), ('creating', 'NNP'), ('a', 'NNP'), ('tool', 'NNP'), ('that', 'NNP'),
('analyzes', 'NNP'), ('text', 'NNP'), ('to', 'NNP'), ('identify', 'NNP'), ('similarities', 'NNP'), ('and', 'NNP'), ('potentia
l', 'NNP'), ('instances', 'NNP'), ('of', 'NNP'), ('plagiarism', 'NNP'), ('.', 'NNP')]

HMM POS Tagger Accuray 0.3647387594191327
```

e. Build a Chunker

Code:

```
import nltk
nltk.download("punkt")
nltk.download("averaged_perceptron_tagger")
sent=input("Enter the sentence: ")
words=nltk.word_tokenize(sent)
pos_tags=nltk.pos_tag(words)
grammer=r"NP: {<DT|JJ|NN.*>+}"
chunk_parser = nltk.RegexpParser(grammer)
chunked_sent = chunk_parser.parse(pos_tags)
for subtree in chunked_sent.subtrees():
    if subtree.label() =='NP':
        print(" ".join(word for word, tag in subtree.leaves()))
```

Output:

```
[nltk_data] Downloading package punkt to C:\Users\HP.DESKTOP-
[nltk_data]     JO7A8MB.000\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\HP.DESKTOP-
[nltk_data]     JO7A8MB.000\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
Enter the sentence: The quick brown fox jumps over the lazy dog.
The quick brown fox
the lazy dog
```

f. Text summarization

Code:

```
import nltk
from nltk.tokenize import sent_tokenize, word_tokenize
from collections import Counter
from sumy.parsers.plaintext import PlaintextParser
from sumy.nlp.tokenizers import Tokenizer
from sumy.summarizers.lex_rank import LexRankSummarizer
nltk.download("punkt")
def summarize_text(text, num_sentences=3):
    parser = PlaintextParser.from_string(text, Tokenizer("english"))
    summarizer = LexRankSummarizer()
    summary = summarizer(parser.document, num_sentences)
    return " ".join(str(sentence) for sentence in summary)

if __name__ == "__main__":
    text =input("Enter the sentence: ")
    print("Summary:")
    print(summarize_text(text))
```

Output:

```
[nltk_data] Downloading package punkt to C:\Users\HP.DESKTOP-
[nltk_data]     JO7A8MB.000\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
Enter the sentence: A plagiarism detection system project typically involves creating a tool that analyzes text to identify s
imilarities and potential instances of plagiarism.
Summary:
A plagiarism detection system project typically involves creating a tool that analyzes text to identify similarities and pote
ntial instances of plagiarism.
```

Practical No:-3

Aim:- a. Find the synonym of a word using WordNet

b. Find the antonym of a word

c. Implement semantic role labeling to identify named entities

d. Resolve the ambiguity

e. Translate the text using First-order logic

a. Find the synonym of a word using WordNet

Code:-

```
import nltk
nltk.download('wordnet')
from nltk.corpus import wordnet
word = input("Enter the word to find the synonyms:")
synonyms = []
for syn in wordnet.synsets(word):
  for lemma in syn.lemmas():
    synonyms.append(lemma.name())
synonyms = list(set(synonyms))
print("Synonyms of",word,"are:", synonyms)
```

Output:-

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
Enter the word to find the synonyms:sad
Synonyms of sad are: ['sorry', 'lamentable', 'distressing', 'deplorable', 'pitiful', 'sad']
```

b. Find the antonym of a word

Code:-

```
import nltk
nltk.download('wordnet')
from nltk.corpus import wordnet
word = input("Enter the word to find the synonyms:")
antonyms = []
for syn in wordnet.synsets(word):
  for lemma in syn.lemmas():
```

```
    for antonym in lemma.antonyms():
      antonyms.append(antonym.name())
antonyms = list(set(antonyms))
print("Antonyms of",word,"are:", antonyms)
```

Output:-

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
Enter the word to find the synonyms:good
Antonyms of good are: ['evil', 'bad', 'badness', 'evilness', 'ill']
```

c. Implement semantic role labeling to identify named entities

Code:-

```
import nltk
import spacy
nltk.download('maxent_ne_chunker')
nltk.download('words')
nlp= spacy.load("en_core_web_sm")
txt = "Apple Inc. was founded by Steve Jobs and Steve Wozinak in Cupertino, Callifornia "
doc = nlp(txt)
entities = [(ent.text,ent.label_)for ent in doc.ents]
for entity,label in entities:
  print(f"Entity: {entity}, Label: {label}")
```

Output:-

```
[nltk_data] Downloading package maxent_ne_chunker to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package maxent_ne_chunker is already up-to-date!
[nltk_data] Downloading package words to /root/nltk_data...
[nltk_data]   Package words is already up-to-date!
Entity: Apple Inc., Label: ORG
Entity: Steve Jobs, Label: PERSON
Entity: Steve Wozinak, Label: PERSON
Entity: Cupertino, Label: GPE
Entity: Callifornia, Label: GPE
```

d. Resolve the ambiguity

Code:-

```
import nltk
import spacy
nlp= spacy.load("en_core_web_sm")
```

```
txt = "I like to play football. I hated it in my childhood though"
doc = nlp(txt)
for token in doc:
  print(f"Token: {token.text}, POS: {token.pos_}, Sense: {token.lemma_}")
```

Output:-

```
/usr/local/lib/python3.11/dist-packages/spacy/util.py:1740: UserWarning: [W111] Jupyter notebook detected: if using `prefer_gpu()` or `require_gpu()`, include it
  warnings.warn(Warnings.W111)
Token: I, POS: PRON, Sense: I
Token: like, POS: VERB, Sense: like
Token: to, POS: PART, Sense: to
Token: play, POS: VERB, Sense: play
Token: football, POS: NOUN, Sense: football
Token: ., POS: PUNCT, Sense: .
Token: I, POS: PRON, Sense: I
Token: hated, POS: VERB, Sense: hate
Token: it, POS: PRON, Sense: it
Token: in, POS: ADP, Sense: in
Token: my, POS: PRON, Sense: my
Token: childhood, POS: NOUN, Sense: childhood
Token: though, POS: ADV, Sense: though
```

e. Translate the text using First-order logic

Code:-

```
!pip install pyDatalog
from pyDatalog import pyDatalog
pyDatalog.create_terms('human,mortal')
+human('John')
+human('Alice')
mortal<=human
res=mortal
if res:
   print("All humans are mortal.")
else:
      print("Not all humans are mortal.")
```

Output:-

```
Requirement already satisfied: pyDatalog in c:\programdata\anaconda3\lib\site-packages (0.17.4)
Not all humans are mortal.
WARNING: Ignoring invalid distribution - (c:\programdata\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -tk (c:\programdata\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -arkupsafe (c:\programdata\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -ltk (c:\programdata\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -rkupsafe (c:\programdata\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -tk (c:\programdata\anaconda3\lib\site-packages)
WARNING: Error parsing dependencies of pyodbc: Invalid version: '4.0.0-unsupported'
```

Name: Shahzaad Firoz Khan **Roll No. 240726**

Practical No:-4

Aim:- a. Implement RNN for sequence labeling

b. Implement POS tagging using LSTM

c. Implement Named Entity Recognizer

d. Word sense disambiguation by LSTM/GRU

a. Implement RNN for sequence labeling

Code:

```
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
vocab = {'T': 0, 'love': 1, 'natural': 2, 'language': 3, 'processing': 4,
'like': 5, 'deep': 6, 'learning': 7}
sequences = [['T', 'love', 'natural', 'language', 'processing']]
labels = [['PRON', 'VERB', 'ADJ', 'NOUN', 'NOUN']]
sequence_indices = [[vocab[word] for word in sequence] for sequence in
sequences]
label_vocab = {'PRON': 0, 'VERB': 1, 'ADJ': 2, 'NOUN': 3}
label_indices = [[label_vocab[label] for label in label_sequence] for
label_sequence in labels]
class RNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(RNN, self).__init__()
        self.hidden_size = hidden_size
        self.embedding = nn.Embedding(input_size, hidden_size)
        self.rnn = nn.RNN(hidden_size, hidden_size)
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        embedded = self.embedding(x)
        output, _ = self.rnn(embedded)
        output = self.fc(output)
        return output

input_size = len(vocab)
hidden_size = 100
output_size = len(label_vocab)
model = RNN(input_size, hidden_size, output_size)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

```
num_epochs = 30
for epoch in range(num_epochs):
    optimizer.zero_grad()
    inputs = torch.tensor(sequence_indices).long()
    labels = torch.tensor(label_indices).view(-1).long()
    outputs = model(inputs)
    outputs = outputs.view(-1, output_size)
    loss = criterion(outputs, labels)
    loss.backward()
    optimizer.step()
    print(f'Epoch [{epoch + 1}/{num_epochs}], Loss: {loss.item()}')

with torch.no_grad():
    test_sequence = [['T', 'like', 'deep', 'learning']]
    test_sequence_indices = [[vocab[word] for word in sequence] for
sequence in test_sequence]
    inputs = torch.tensor(test_sequence_indices).long()
    outputs = model(inputs)
    predicted_labels = torch.argmax(outputs, dim=2)
    predicted_labels =
[[list(label_vocab.keys())[list(label_vocab.values()).index(label)] for
label in sequence] for sequence in predicted_labels]
    print(f'\nPredicted Labels: {predicted_labels}')
```

Output:

```
Epoch [12/30], Loss: 0.3172128200531006
Epoch [13/30], Loss: 0.2732114791870117
Epoch [14/30], Loss: 0.23591656982898712
Epoch [15/30], Loss: 0.20437128841876984
Epoch [16/30], Loss: 0.1777113974094391
Epoch [17/30], Loss: 0.15517480671405792
Epoch [18/30], Loss: 0.13610169291496277
Epoch [19/30], Loss: 0.11992914974689484
Epoch [20/30], Loss: 0.10618207603693008
Epoch [21/30], Loss: 0.09446276724338531
Epoch [22/30], Loss: 0.0844397321343422
Epoch [23/30], Loss: 0.07583770155906677
Epoch [24/30], Loss: 0.06842862069606781
Epoch [25/30], Loss: 0.062023334205150604
Epoch [26/30], Loss: 0.05646521598100662
Epoch [27/30], Loss: 0.05162403732538223
Epoch [28/30], Loss: 0.047391679137945175
Epoch [29/30], Loss: 0.043677933514118195
Epoch [30/30], Loss: 0.04040742665529251

Predicted Labels: [['PRON', 'ADJ', 'NOUN', 'NOUN']]
```

b. Implement POS tagging using LSTM

Code:

```
import spacy
import tensorflow as tf
from tensorflow import keras
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
nlp = spacy.load("en_core_web_sm")
txt = "The quick brown fox jumps over the lazy dog"
doc = nlp(txt)
tokens = [token.text for token in doc]
pos_tags = [token.pos for token in doc]
label_encoder = LabelEncoder()
pos_labels = label_encoder.fit_transform(pos_tags)
X_train, X_test, y_train, y_test = train_test_split(tokens, pos_labels,
test_size=0.2, random_state=42)
tokenizer = keras.layers.TextVectorization()
tokenizer.adapt(X_train)
X_train = tokenizer(X_train)
X_test = tokenizer(X_test)
model = keras.Sequential([
    keras.layers.Embedding(input_dim=len(tokenizer.get_vocabulary()),
output_dim=128, mask_zero=True),
    keras.layers.LSTM(128, return_sequences=False),  # Output a single
    keras.layers.Dense(len(label_encoder.classes_), activation='softmax')
])
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
model.fit(X_train, y_train, epochs=5, validation_split=0.2)
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Loss: {loss}, Accuracy: {accuracy}')
```

Output:

```
/usr/local/lib/python3.11/dist-packages/spacy/util.py:1740: UserWarning: [W111] Jupyter notebook c
  warnings.warn(Warnings.W111)
Epoch 1/5
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step - accuracy: 0.0000e+00 - loss: 1.6120 - val_accuracy: 0.0000e+
Epoch 2/5
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 96ms/step - accuracy: 0.2000 - loss: 1.6053 - val_accuracy: 0.5000 - v
Epoch 3/5
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 108ms/step - accuracy: 1.0000 - loss: 1.5986 - val_accuracy: 0.5000 -
Epoch 4/5
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 97ms/step - accuracy: 1.0000 - loss: 1.5919 - val_accuracy: 0.5000 - v
Epoch 5/5
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 101ms/step - accuracy: 1.0000 - loss: 1.5852 - val_accuracy: 0.5000 -
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 61ms/step - accuracy: 0.0000e+00 - loss: 1.6114
Loss: 1.611444354057312, Accuracy: 0.0
```

c. Implement Named Entity Recognizer

Code:

```
!pip install spacy
!python -m spacy download en_core_web_sm
import spacy
# Load spaCy's pre-trained English model
nlp = spacy.load("en_core_web_sm")
# Sample text
text = "Elon Musk founded SpaceX in 2002 and was born in South Africa."
# Process the text
doc = nlp(text)
# Print named entities
print("Named Entities, Phrases, and Concepts:")
for ent in doc.ents:
    print(f"{ent.text} - {ent.label_}")
# Entity labels explanation
print("\nEntity Labels Explanation:")
print(spacy.explain("ORG"))  # Example: ORG -> Organizations
print(spacy.explain("GPE"))  # Example: GPE -> Countries, cities, states
```

Output:

```
Requirement already satisfied: markdown-it-py>=2.2.0 in /ur
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in
Requirement already satisfied: wrapt in /usr/local/lib/python3.11/dist-packages (from smart-oper
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-packages (from mark
✓ Download and installation successful
You can now load the package via spacy.load('en_core_web_sm')
⚠ Restart to reload dependencies
If you are in a Jupyter or Colab notebook, you may need to restart Python in
order to load all the package's dependencies. You can do this by selecting the
'Restart kernel' or 'Restart runtime' option.
/usr/local/lib/python3.11/dist-packages/spacy/util.py:1740: UserWarning: [W111] Jupyter noteboo
  warnings.warn(Warnings.W111)
Named Entities, Phrases, and Concepts:
Elon Musk - PERSON
2002 - DATE
South Africa - GPE

Entity Labels Explanation:
Companies, agencies, institutions, etc.
Countries, cities, states
```

d. Word sense disambiguation by LSTM/GRU

Code:

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, GRU, Dense
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
sentences = [
    "I deposited money at the bank","He sat on the bank of the river",
 "She took a loan from the bank","The river overflowed and flooded the
bank"]
labels = np.array([0, 1, 0, 1])
tokenizer = Tokenizer()
tokenizer.fit_on_texts(sentences)
vocab_size = len(tokenizer.word_index) + 1
sequences = tokenizer.texts_to_sequences(sentences)
max_len = max(len(seq) for seq in sequences)
X = pad_sequences(sequences, maxlen=max_len, padding="post")
model = Sequential([
    Embedding(input_dim=vocab_size, output_dim=50, input_length=max_len),
    LSTM(64, return_sequences=False),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')])
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
model.fit(X, labels, epochs=10, verbose=1)
new_sentence = ["He opened an account at the bank"]
new_seq = tokenizer.texts_to_sequences(new_sentence)
new_seq = pad_sequences(new_seq, maxlen=max_len, padding="post")
prediction = model.predict(new_seq)
predicted_sense = int(prediction[0] > 0.5)  # 0 or 1
print(f"Predicted sense: {predicted_sense}")
```

Output:

```
Epoch 1/10
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/embedding.py:90: UserWa
  warnings.warn(
1/1 ━━━━━━━━━━━━━━━━━━━━ 5s 5s/step - accuracy: 0.5000 - loss: 0.6907
Epoch 2/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 239ms/step - accuracy: 0.7500 - loss: 0.6869
Epoch 3/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 161ms/step - accuracy: 0.7500 - loss: 0.6837
Epoch 4/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 147ms/step - accuracy: 0.7500 - loss: 0.6805
Epoch 5/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 213ms/step - accuracy: 0.7500 - loss: 0.6770
Epoch 6/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 292ms/step - accuracy: 0.7500 - loss: 0.6733
Epoch 7/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 291ms/step - accuracy: 0.7500 - loss: 0.6693
Epoch 8/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 271ms/step - accuracy: 0.7500 - loss: 0.6649
Epoch 9/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 193ms/step - accuracy: 0.7500 - loss: 0.6600
Epoch 10/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 196ms/step - accuracy: 0.7500 - loss: 0.6544
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 890ms/step
Predicted sense: 0
<ipython-input-24-ace8be0ce36a>:41: DeprecationWarning: Conversion of an array with n
```

Practical No:-5

Aim:- a. Develop an application on a review system.

Code:

```
import pandas as pd
# Create an empty DataFrame for storing reviews
reviews_df = pd.DataFrame(columns=["User", "Rating", "Comment"])
def add_review(user, rating, comment):
    """Function to add a review."""
    global reviews_df
    new_review = pd.DataFrame([[user, rating, comment]], columns=["User", "Rating", "Comment"])
    reviews_df = pd.concat([reviews_df, new_review], ignore_index=True)
    print(f"Review added by {user}!")
def display_reviews():
    """Function to display all reviews."""
    if reviews_df.empty:
        print("No reviews yet!")
    else:
        print(reviews_df)
def average_rating():
    """Function to calculate the average rating."""
    if reviews_df.empty:
        return "No reviews yet!"
    return reviews_df["Rating"].mean()
def filter_reviews_by_rating(min_rating):
    """Function to filter reviews with rating greater than or equal to min_rating."""
    filtered_reviews = reviews_df[reviews_df["Rating"] >= min_rating]
    if filtered_reviews.empty:
        print(f"No reviews with rating greater than or equal to {min_rating}.")
    else:
        print(filtered_reviews)
```

```
# Add some sample reviews
add_review("Alice", 5, "Excellent product!")
add_review("Bob", 4, "Very good, but could be improved.")
add_review("Charlie", 3, "It's okay, nothing special.")
add_review("David", 5, "Highly recommend it!")
# Display all reviews
display_reviews()
# Get average rating
print(f"Average Rating: {average_rating()}")
# Filter reviews with rating >= 4
filter_reviews_by_rating(4)
```

Output:

```
Review added by Alice!
Review added by Bob!
Review added by Charlie!
Review added by David!
      User Rating                          Comment
0    Alice     5                Excellent product!
1      Bob     4  Very good, but could be improved.
2  Charlie     3       It's okay, nothing special.
3    David     5              Highly recommend it!
Average Rating: 4.25
    User Rating                          Comment
0  Alice     5                Excellent product!
1    Bob     4  Very good, but could be improved.
3  David     5              Highly recommend it!
```

b. Create a chatbot for HITS.

Code:

```
import nltk

from nltk.chat.util import Chat, reflections

# Ensure that you have downloaded the necessary NLTK data

nltk.download('punkt')

# Define pairs of patterns and responses

patterns_responses = [

    (r"hi|hello|hey", ["Hello! How can I assist you with the HITS today?"]),

    (r"what is HITS?", ["HITS stands for Humanitarian Information Tracking System. It's used to track and manage humanitarian activities. How can I help you with it?"]),

    (r"how do I create an incident report?", ["To create an incident report, please provide details such as the incident description, location, date, and severity level."]),

    (r"what are the severity levels?", ["The severity levels are: Low, Medium, High, and Critical. Please choose the appropriate level for the incident."]),

    (r"how do I track an incident?", ["You can track an incident by entering the incident ID provided when the report was created."]),

    (r"incident status for (\d+)", ["Checking the status for incident ID %1..."]),

    (r"what is my status with incident (\d+)", ["Let me check the current status of incident ID %1."]),

    (r"how can I get a report of incidents?", ["You can generate a report by filtering incidents based on date, location, and severity. Would you like to filter by those parameters?"]),

    (r"thank you|thanks", ["You're welcome! If you need further assistance, just let me know."]),

    (r"bye|exit", ["Goodbye! Feel free to contact me anytime if you need help."]),

    (r"(.*)", ["Sorry, I didn't quite understand that. Could you please rephrase your question?"]),

]

# Create a chatbot instance

def chat():

    print("Hello! I'm your HITS assistant. Type 'bye' to exit.")

    chatbot = Chat(patterns_responses, reflections)

    chatbot.converse()


if __name__ == "__main__":
```

chat()

Output:

```
[nltk_data] Downloading package punkt to C:\Users\HP.DESKTOP-
[nltk_data]     JO7A8MB.000\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

```
Hello! I'm your HITS assistant. Type 'bye' to exit.
>hi
Hello! How can I assist you with the HITS today?
>what is HITS?
HITS stands for Humanitarian Information Tracking System. It's used to track and manage humanitarian activities. How can I he
lp you with it?
>how do I create an incident report?
To create an incident report, please provide details such as the incident description, location, date, and severity level.
>what are the severity levels?
The severity levels are: Low, Medium, High, and Critical. Please choose the appropriate level for the incident.
>how do I track an incident?
You can track an incident by entering the incident ID provided when the report was created.
>incident status for (\d+)
Sorry, I didn't quite understand that. Could you please rephrase your question?
>what is my status with incident
Sorry, I didn't quite understand that. Could you please rephrase your question?

>
```

Practical No:-6

Aim:- Perform Sentiment Analysis

Code:-

```
import nltk
from nltk.sentiment import SentimentIntensityAnalyzer
tweets = [
    "I love this product! It's amazing.","This is the worst experience I've ever had.","I'm not sure how I feel about this."]
nltk.download('vader_lexicon')
def analyze_sentiment_vader(tweets):
    sia = SentimentIntensityAnalyzer()
    for tweet in tweets:
        scores = sia.polarity_scores(tweet)
        print(f'Tweet: {tweet}')
        print(f'Sentiment Scores: {scores}\n')
analyze_sentiment_vader(tweets)
```

Output:-

```
Tweet: I love this product! It's amazing.
Sentiment Scores: {'neg': 0.0, 'neu': 0.266, 'pos': 0.734, 'compound': 0.8516}

Tweet: This is the worst experience I've ever had.
Sentiment Scores: {'neg': 0.369, 'neu': 0.631, 'pos': 0.0, 'compound': -0.6249}

Tweet: I'm not sure how I feel about this.
Sentiment Scores: {'neg': 0.246, 'neu': 0.754, 'pos': 0.0, 'compound': -0.2411}

[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
```

Practical No:-7

Aim:- Develop a project on Social Media Platform Classification.

Code:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report
import nltk
from nltk.corpus import stopwords
import string
nltk.download('stopwords')
data = {
   'post': ["Just had an amazing meal at the new restaurant! #yum", "Check out my new photo at the beach! #vacation #beach", "Got a new promotion at work! Feeling great #success" ,"What an insightful article on climate change #environment", "Loving the new series on Netflix! #bingewatching" ,"Just finished my workout! Time to relax #fitness"],
   'platform': ['Twitter',  # Short, informal, hashtags common
            'Instagram', # Photo-based, casual language, hashtags
      'Facebook',  # Longer posts, mixed content
      'Twitter',  # Short posts, hashtag-driven
      'Instagram', # Photo-based, informal hashtags
      'Facebook'   # Mixed content, longer posts]}
df = pd.DataFrame(data)
def preprocess_text(text):
   text = text.lower()
   text = ''.join([char for char in text if char not in string.punctuation])
   stop_words = set(stopwords.words('english'))
   text = ' '.join([word for word in text.split() if word not in stop_words])
   return tex
```

```
df['clean_post'] = df['post'].apply(preprocess_text)

vectorizer = TfidfVectorizer()

X = vectorizer.fit_transform(df['clean_post'])

y = df['platform']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

model = MultinomialNB()

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print(f"Accuracy: {accuracy_score(y_test, y_pred)}")

print("Classification Report:")

print(classification_report(y_test, y_pred))

new_posts = ["Had a great time watching the new movie #moviebuff" , "Just posted a new picture from my hike #nature"]

new_posts_clean = [preprocess_text(post) for post in new_posts]

new_posts_vectorized = vectorizer.transform(new_posts_clean)

new_predictions = model.predict(new_posts_vectorized)

for post, prediction in zip(new_posts, new_predictions):

    print(f"Post: {post}\nPredicted Platform: {prediction}\n")
```

Output:

```
Accuracy: 0.0
Classification Report:
              precision    recall  f1-score   support

    Facebook       0.00      0.00      0.00       0.0
   Instagram       0.00      0.00      0.00       1.0
     Twitter       0.00      0.00      0.00       1.0

    accuracy                           0.00       2.0
   macro avg       0.00      0.00      0.00       2.0
weighted avg       0.00      0.00      0.00       2.0

Post: Had a great time watching the new movie #moviebuff
Predicted Platform: Facebook

Post: Just posted a new picture from my hike #nature
Predicted Platform: Facebook

[nltk_data] Downloading package stopwords to C:\Users\HP.DESKTOP-
[nltk_data]     JO7A8MB.000\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1248: UndefinedMetricWarning: Precision and F-s
core are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1248: UndefinedMetricWarning: Recall and F-scor
e are ill-defined and being set to 0.0 in labels with no true samples. Use `zero_division` parameter to control this behavio
r.
  _warn_prf(average, modifier, msg_start, len(result))
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1248: UndefinedMetricWarning: Precision and F-s
```