**MASTER OF SCIENCE (INFORMATION TECHNOLOGY)**

**PART – II, SEMESTER – IV** **2023 – 2024**

**PRACTICAL JOURNAL**

**PAPER I: BLOCKCHAIN**

**PAPER II: NATURAL LANGUAGE PROCESSING**

**PAPER III: DEEP LEARNING**

**SUBMITTED BY**

**Payal Vikas Gharge**

**SEAT NO: 4047A008**

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**KELKAR EDUCATIONS TRUST’S**

**Vinayak Ganesh Vaze College of Arts, Science And**

**Commerce (Autonomous)**

**Mithagar Road,**

**Mulund East,**

**Mumbai - 400081**

# BLOCKCHAIN

**INDEX**

|  |  |
| --- | --- |
| **Practical**  **No** | **Title** |
| **1** | Demonstrate the use of different types of variables in solidity |
| **2** | Write a solidity program to demonstrate relational operators. |
| **3** | Write a Solidity program to print sum of 10 numbers using for loop. |
| **4** | Write a Solidity program to print sum of 10 numbers using while loop. |
| **5** | Write a Solidity program to print sum of 10 numbers using do while. |
| **6** | Write a Solidity program to check if number is even or odd. |
| **7** | Write a Solidity program to use string. |
| **8** | Demonstrate the use of array. Also find sum of array |
| **9** | Write a Solidity program to use enumeration. |
| **10** | Write a Solidity program to use arithmetic operations. |
| **11** | Create a book structure, assign values and display the values using Solidity. |
| **12** | Write a solidity program to create view and pure function. |
| **13** | Write a solidity program to implement function overloading. |
| **14** | Write a solidity program to use mathematical function. |
| **15** | Write a solidity program to use cryptographic function. |
| **16** | Write a Python program to create a simple client class that generates the private and public keys by using the built-in Python RSA algorithm and test it. |
| **17** | Write a Python program to create a transaction class to send and receive money and test it. |
| **18** | Write a Python program to create a mining function and test it. |
| **19** | Demonstrate the use of Bitcoin Core API. |

Practical 1 : Demonstrate the use of different types of variables in solidity

Code :

pragma solidity ^0.5.0; contract Pract1{ int x=15; //state var int public y=10;//global function getValue(int z) public{ y=y+z;

} function show() public view returns

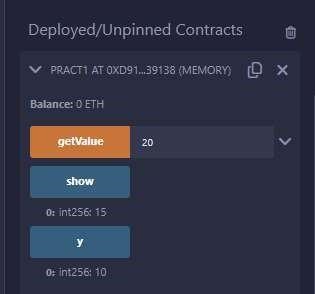
(int)

{ return x;

}

}

Output :



Practical 2 : Write a solidity program to demonstrate relational operators.

pragma solidity ^0.5.0; contract Pract2{ bool public a=true; bool public b=false; bool public r1or=a||b;

bool public r2and=a&&b; bool

public r3not=!b;

}

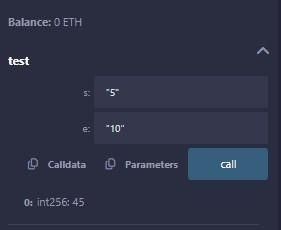
Output :



Practical 3 : Write a Solidity program to print sum of 10 numbers using for loop

|  |
| --- |
| pragma solidity ^0.5.0; contract Pract3{ function test(int s, int e) public view returns(int)  { int i;  int sum=0; for(i=s;i<=e;i++)  {  sum+=i; //sum=sum+i;  } return  sum;  }  } |

Output :

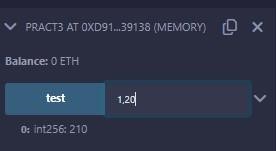


3 |

Practical 4 : Write a Solidity program to print sum of 10 numbers using while loop.

|  |
| --- |
| pragma solidity ^0.5.0; contract Pract3{ function test(int s, int e) public view returns(int)  { int i;  int sum=0; i=s; while(i<=e)  {  sum+=i; //sum=sum+i; i++; } return  sum;  }  } |

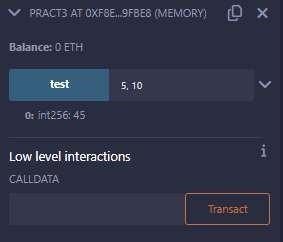
Output :



Practical 5 : Write a Solidity program to print sum of 10 numbers using do while.

|  |
| --- |
| pragma solidity ^0.5.0; contract Pract3{ function test(int s, int e) public view returns(int)  { int i;  int sum=0; i=s; do  {  sum+=i; //sum=sum+i; i++;  }while(i<=e); return sum;  }  } |

Output :

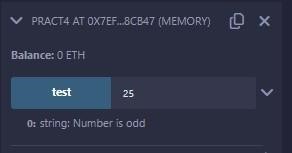


Practical 6 : Write a Solidity program to check if number is even or odd.

Code :

|  |
| --- |
| pragma solidity ^0.5.0; contract Pract4{  function test(int x) public view returns(string memory)  {  if(x%2==0) return "Number is even"; else return "Number is odd";  }  } |

Output :

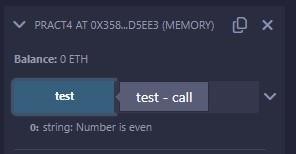


Practical 7 : Write a Solidity program to use string.

Code :

|  |
| --- |
| pragma solidity ^0.5.0; contract Pract4{  function test(int x) public view returns(string memory)  {  if(x%2==0) return "Number is even"; else return "Number is odd";  }  } |

Output :



Practical 8 : Demonstrate the use of array. Also find sum of array.

Code: contract Types {

|  |
| --- |
| uint[5] data; constructor() public  { data = [uint(10), 20, 30, 40, 50];  }  function array\_example() public view returns (uint,uint) {    return (data[0],data[4]);  }  function array\_example2() public view returns (uint [5] memory) {    return data;  }  } |

Output :

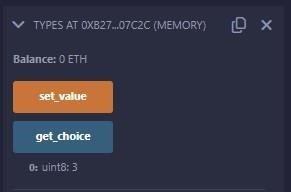


Practical 9 : Write a Solidity program to use enumeration.

Code :

|  |
| --- |
| pragma solidity ^0.5.0; contract Types { enum week\_days  {  Monday,Tuesday,Wednesday,Thursday,Friday,Saturday,  Sunday  }  week\_days choice; function set\_value() public { choice = week\_days.Thursday;  }  function get\_choice() public view returns (week\_days) { return choice;  }  } |

Output :



Practical 10 : Write a Solidity program to use arithmetic operations

Code :

pragma solidity ^0.5.0; // Creating a contract contract SolidityTest { // Initializing variables uint16 public a = 20; uint16 public b = 10;

// Initializing a variable // with sum uint public sum = a + b; // Initializing a variable // with the difference uint public diff = a - b;

// Initializing a variable // with product uint public mul = a \* b;

// Initializing a variable // with quotient uint public div = a / b;

// Initializing a variable // with modulus uint public mod = a % b;

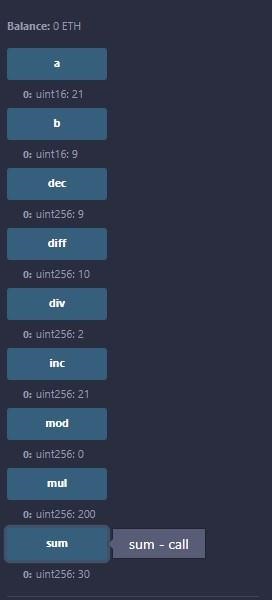
// Initializing a variable // decrement value uint public dec = --b;

// Initializing a variable //

with increment value uint public inc = ++a;

}

Output :

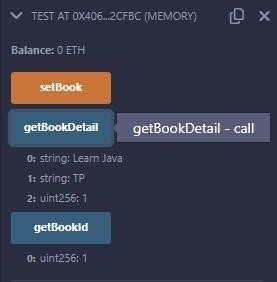


Practical 11 : Create a book structure, assign values and display the values using Solidity.

Code :

|  |
| --- |
| pragma solidity ^0.5.0; contract test { struct Book { string title; string author; uint book\_id;  }  Book book;    function setBook() public { book  = Book('Learn Java', 'TP', 1);  }  function getBookId() public view returns (uint) { return book.book\_id;  }  function getBookDetail() public view returns (string memory, string memory,uint) { return (book.title, book.author, book.book\_id); }    } |

Output :



Practical 12 : Write a solidity program to create view and pure function.

Code :

|  |
| --- |
| pragma solidity ^0.5.0; contract Test { int public x=10; //global int y=90;//state function f1() public returns(int){ //read and update is allowed x=100; return x;  }  function f2() public view returns(int){  // x=100; //erro beacuse x is global/state  //we can access but we cannot update state or global variable int view function return x;  }  function f3() public pure returns(int){  //we cannot access or update state or global variable in pure function int z=80; return z;  }    } |

Output :

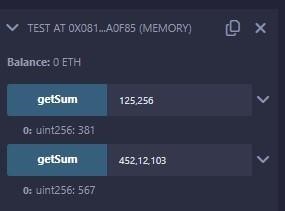


Practical 13 : Write a solidity program to implement function overloading.

Code :

|  |
| --- |
| pragma solidity ^0.5.0; contract Test { function getSum(uint a, uint b) public pure returns(uint){ return a + b;  }  function getSum(uint a, uint b, uint c ) public pure returns(uint){ return a + b + c;  }  } |

Output :



Practical 14 : Write a solidity program to use mathematical function.

Code :

pragma solidity ^0.5.0;

contract Test { function callAddMod() public pure returns(uint){ return addmod(4, 5, 3);

}

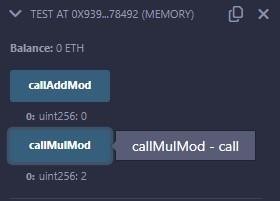
function callMulMod() public pure returns(uint){ return mulmod(4,

5, 3);

}

}

Output :



Practical 15 : Write a solidity program to use cryptographic function.

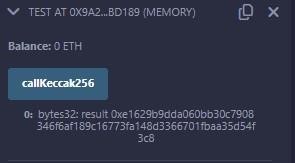
Code :

pragma solidity ^0.5.0; contract Test { function callKeccak256() public pure returns(bytes32 result){ return keccak256("ABC");

}

}

Output :



Practical 16 : Write a Python program to create a simple client class that generates the private and public keys by using the built-in Python RSA algorithm and test it.

Code :

!pip install crypto !pip install pycrypto !pip install pycryptodome import hashlib import random import string import json import binascii import numpy as np import pandas as pd

import logging import datetime import collections

from Crypto.PublicKey import RSA from Crypto import Random from

Crypto.Cipher import PKCS1\_v1\_5

class Client: def \_\_init\_\_(self):

random = Random.new().read self.\_private\_key = RSA.generate(1024, random) self.\_public\_key = self.\_private\_key.publickey() self.\_signer = PKCS1\_v1\_5.new(self.\_private\_key) @property

def identity(self):

return

binascii.hexlify(self.\_public\_key.exportKey(format='DER')).decode('ascii')

Dinesh = Client() print ("sender ",Dinesh.identity)

Output :



Practical 17 : Write a Python program to create a transaction class to send and receive money and test it.

Code :

!pip install crypto !pip install pycrypto !pip install pycryptodome import hashlib import random import binascii import datetime import collections

from Crypto.PublicKey import RSA from Crypto import Random from Crypto.Cipher import PKCS1\_v1\_5 from collections import OrderedDict import Crypto import

Crypto.Random from Crypto.Hash import

SHA from Crypto.Signature import PKCS1\_v1\_5

class Client: def \_\_init\_\_(self):

random = Random.new().read self.\_private\_key = RSA.generate(1024, random) self.\_public\_key = self.\_private\_key.publickey() self.\_signer =

PKCS1\_v1\_5.new(self.\_private\_key)

@property

def identity(self):

return binascii.hexlify(self.\_public\_key.exportKey(format='DER')).decode('ascii')

class Transaction: def \_\_init\_\_(self, sender, recipient, value):

self.sender = sender self.recipient = recipient self.value

= value self.time = datetime.datetime.now()

def to\_dict(self): if self.sender == "Genesis":

identity = "Genesis"

else:

identity = self.sender.identity

return collections.OrderedDict({

'sender': identity,

'recipient': self.recipient,

'value': self.value,

'time' : self.time})

def sign\_transaction(self):

private\_key = self.sender.\_private\_key

signer = PKCS1\_v1\_5.new(private\_key)

h = SHA.new(str(self.to\_dict()).encode('utf8'))

return binascii.hexlify(signer.sign(h)).decode('ascii')

def display\_transaction(transaction):

#for transaction in transactions:

dict = transaction.to\_dict() print ("sender: " + dict['sender'])

print ('-----')

print ("recipient: " + dict['recipient'])

print ('-----')

print ("value: " + str(dict['value']))

print ('-----')

print ("time: " + str(dict['time']))

print ('-----')

transactions = []

Dinesh = Client()

Ramesh = Client()

t1 = Transaction( Dinesh,

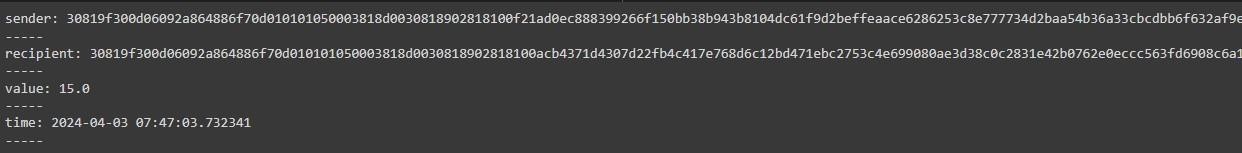
Ramesh.identity,

15.0

)

t1.sign\_transaction() display\_transaction (t1) Output

:



21

Practical 18 : Write a Python program to create a mining function and

test it.

Code :

import hashlib

def sha256(message):

return hashlib.sha256(message.encode('ascii')).hexdigest()

def mine(message, difficulty=1):

assert difficulty >= 1

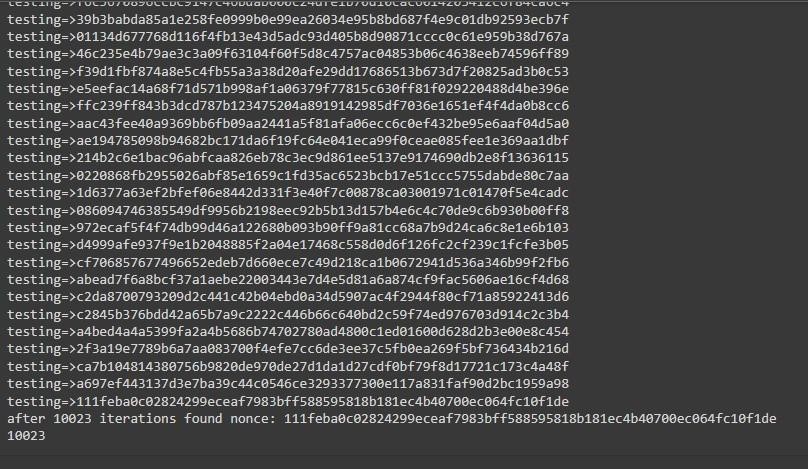
#if(difficulty <1):

# return

#'1'\*3=> '111' prefix = '1' \* difficulty print("prefix",prefix) for i in range(100000):

digest = sha256(str(hash(message)) + str(i)) print("testing=>"+digest) if digest.startswith(prefix): print ("after " + str(i) + " iterations found nonce: "+ digest) return i #i= nonce value

mine ("test message",3) Output :



Practical 19 : Demonstrate the use of Bitcoin Core API.

Code :

#pip install bitcoinlib from

bitcoinlib.wallets import Wallet w = Wallet.create('Wallet3') key1 =

w.get\_key() print(key1.address)

w.scan() print(w.info())

Output :



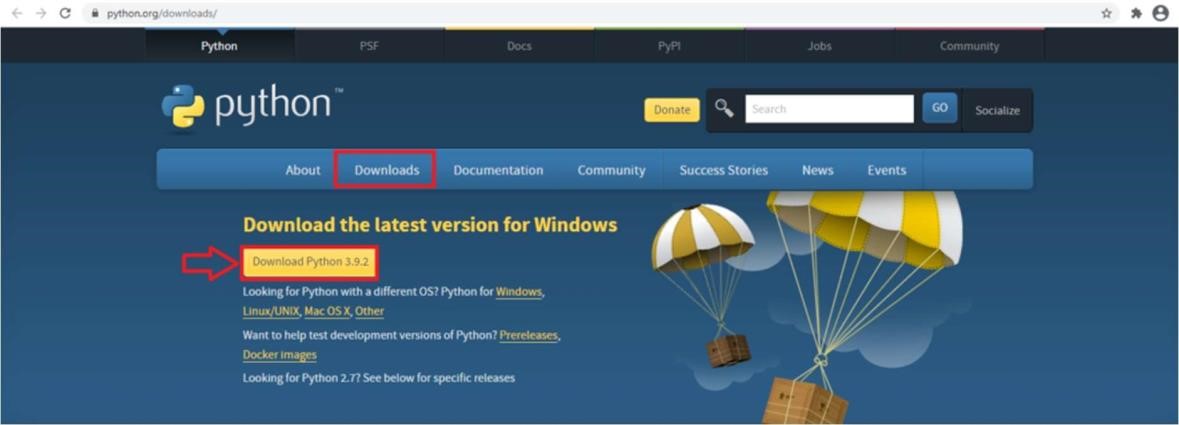
# Natural Language Processing

Practical 1 :

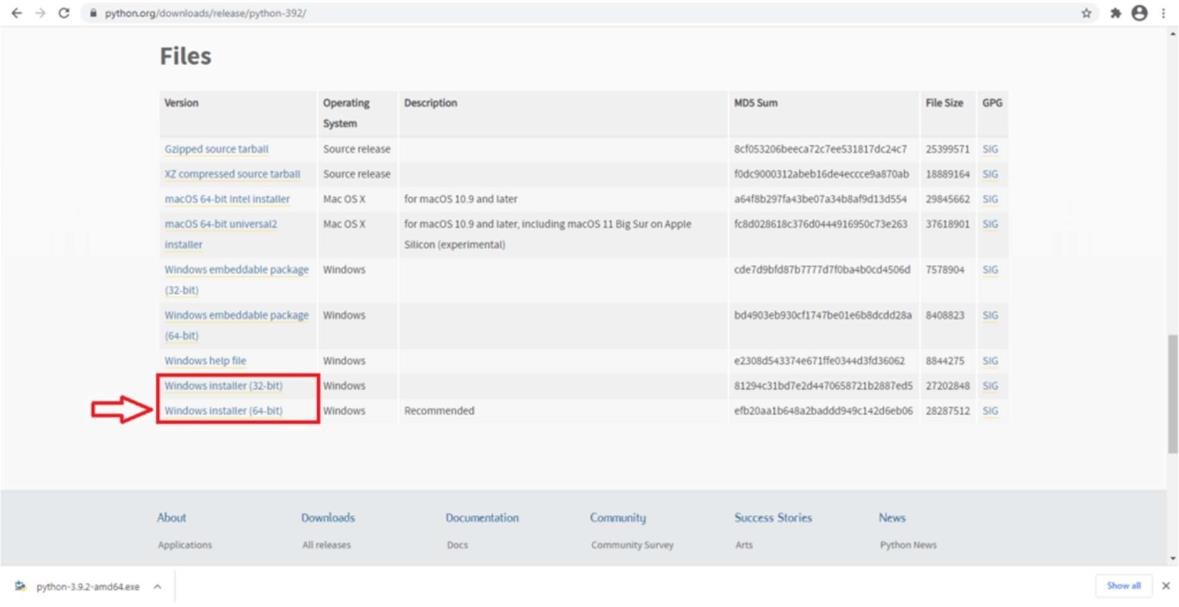
a) Install NLTK

Python 3.9.2 Installation on Windows

Step 1) Go to link https://www.python.org/downloads/, and select the latest version for windows.

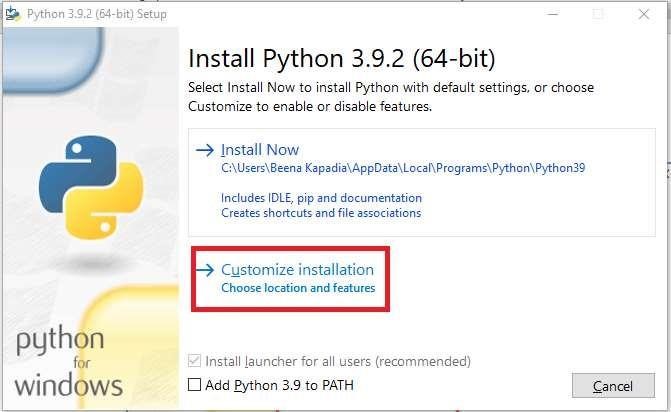


Note: If you don't want to download the latest version, you can visit the download tab and see all releases.

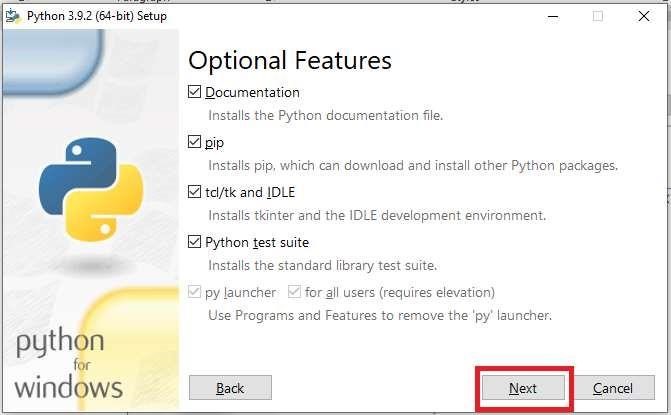


Step 2) Click on the Windows installer (64 bit)

Step 3)Select Customize Installation

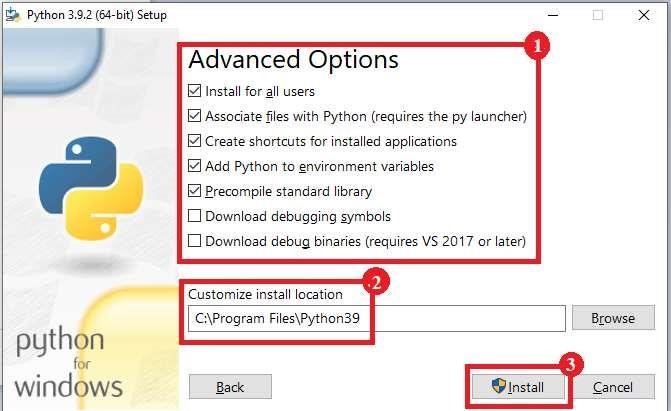


Step 4) Click NEXT



Step 5) In next screen

1. Select the advanced options
2. Give a Custom install location. Keep the default folder as c:\Program files\Python39
3. Click Install



Step 6) Click Close button once install is done.

Step 7) open command prompt window and run the following commands:

C:\Users\Beena Kapadia>pip install --upgrade pip

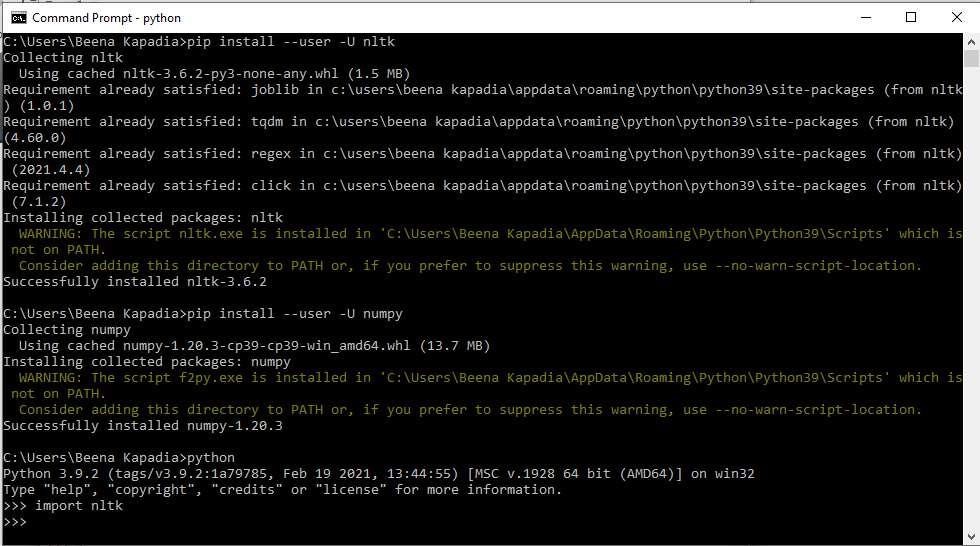
C:\Users\Beena Kapadia>pip install --user -U nltk

C:\Users\Beena Kapadia>>pip install --user -U numpy

C:\Users\Beena Kapadia>python

>>> import nltk

>>>



b. Convert the given text to speech.

Text-to-speech (TTS) is a technology that converts written text into spoken words. It is commonly used in various applications such as accessibility tools for visually impaired individuals, virtual assistants, automated customer service systems, and in-car navigation systems.

Source code:

# text to speech

# for google colab put ! at the start of pip.

# pip install gtts

# pip install playsound

from playsound import playsound

# import required for textto speech conversion

from gtts import gTTS mytext = "Welcome to Natural Language programming" language = "en" myobj = gTTS(text=mytext, lang=language, slow=False) myobj.save("myfile.mp3")

playsound("myfile.mp3")

Output: welcomeNLP.mp3 audio file is getting created and it plays the file with playsound() method, while running the program.

c) Convert audio file Speech to Text.

Speech-to-text technology converts spoken language from audio files into written text. It involves preprocessing the audio, extracting features, using an acoustic model to recognize speech sounds, incorporating a language model for context, and decoding to generate the transcribed text. Advanced techniques like deep learning models have significantly improved the accuracy and efficiency of speech recognition systems.

Source code:

Note: required to store the input file "male.wav" in the current folder before running the program.

#pip3 install SpeechRecognitionpydub

# for google colab put ! at the start of pip.

import speech\_recognition as sr

filename = "male.wav"

# initialize the recognizer

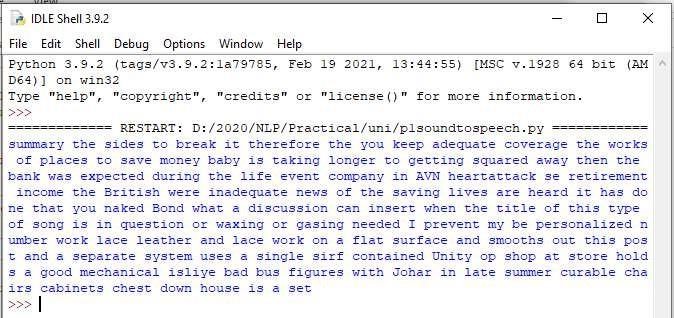
r = sr.Recognizer()

# open the file with sr.AudioFile(filename) as source: # listen for the data (load audio to memory) audio\_data = r.record(source) # recognize (convert from speech to text) text = r.recognize\_google(audio\_data) print(text)

Input:

male.wav (any wav file)

Output:



## Practical 2

A corpus refers to a large and structured collection of texts, spoken or written, that is used for linguistic analysis, language modeling, and machine learning tasks. It may include a wide range of text sources such as books, articles, transcripts, social media posts, and more, often organized and annotated for specific research or application purposes. Corpora play a crucial role in natural language processing (NLP) tasks like sentiment analysis, machine translation, named entity recognition, and speech recognition, providing valuable data for training and evaluating language models and algorithms.

a. Study of various Corpus – Brown, Inaugural, Reuters, udhr with various methods like

fields, raw, words, sents, categories.

source code:

'''NLTK includes a small selection of texts from the Project brown electronic text archive, which contains some 25,000 free electronic books, hosted at http://www.brown.org/. We begin by getting the Python interpreter to load the NLTK package, then ask to see nltk.corpus.brown.fileids(), the file identifiers in this corpus:'''

import nltk

from nltk.corpus import brown print ('File ids of

brown corpus\n',brown.fileids())

'''Let’s pick out the first of these texts — Emma by Jane Austen — and give it a short name, emma, then find out how many words it contains:''' ca01 = brown.words('ca01')

# display first few words print('\nca01 has following words:\n',ca01)

# total number of words in ca01

print('\nca01 has',len(ca01),'words')

#categories or files

print ('\n\nCategories or file in brown corpus:\n')

print (brown.categories())

'''display other information about each text, by looping over all the values of fileid corresponding to the brown file identifiers listed earlier and then computing statistics for each text.'''

print ('\n\nStatistics for each text:\n')

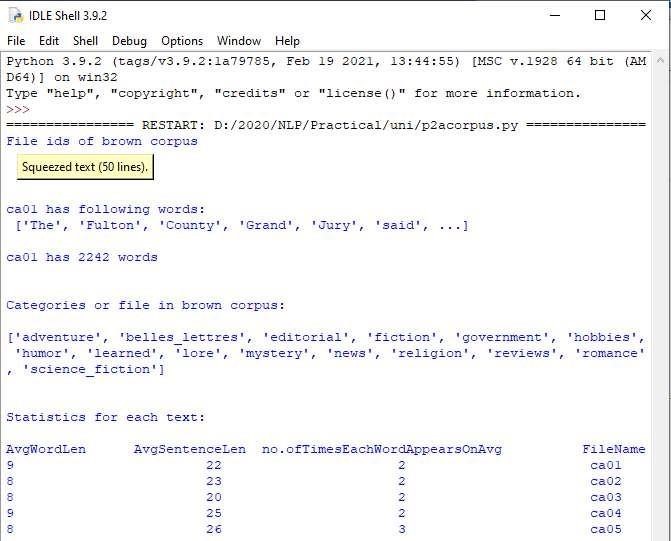
print ('AvgWordLen\tAvgSentenceLen\tno.ofTimesEachWordAppearsOnAvg\t\tFileName') for fileid in brown.fileids():

num\_chars = len(brown.raw(fileid)) num\_words = len(brown.words(fileid)) num\_sents = len(brown.sents(fileid))

num\_vocab = len(set([w.lower() for w in brown.words(fileid)])) print (int(num\_chars/num\_words),'\t\t\t', int(num\_words/num\_sents),'\t\t\t',

int(num\_words/num\_vocab),'\t\t\t', fileid)

output:



b. Create and use your own corpora(plaintext, categorical)

source code:

'''NLTK includes a small selection of texts from the Project filelist electronic text archive, which contains some 25,000 free electronic books, hosted at http://www.filelist.org/. We begin by getting the Python interpreter to load the NLTK package, then ask to see nltk.corpus.filelist.fileids(), the file identifiers in this corpus:''' Plaintext Corpora:

import nltk from nltk.corpus import

PlaintextCorpusReader

corpus\_root = 'D:/2020/NLP/Practical/uni'

filelist = PlaintextCorpusReader(corpus\_root,

'.\*') print ('\n File list: \n') print (filelist.fileids()) print (filelist.root)

'''display other information about each text, by looping over all the values of fileid corresponding to the filelist file identifiers listed earlier and then computing statistics for each text.'''

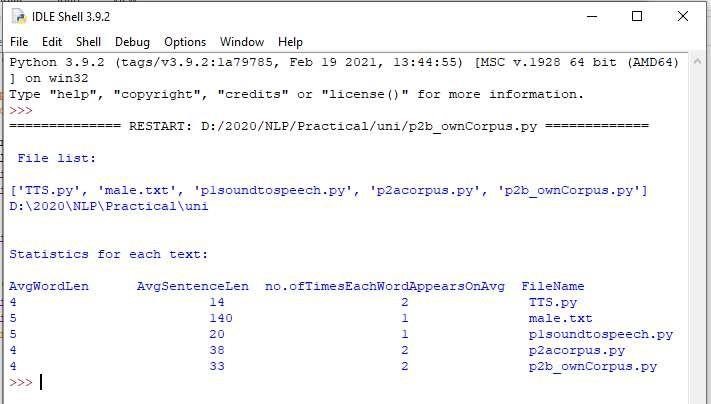
print ('\n\nStatistics for each text:\n')

print ('AvgWordLen\tAvgSentenceLen\tno.ofTimesEachWordAppearsOnAvg\tFileName') for fileid in filelist.fileids():

num\_chars = len(filelist.raw(fileid)) num\_words = len(filelist.words(fileid)) num\_sents = len(filelist.sents(fileid))

num\_vocab = len(set([w.lower() for w in filelist.words(fileid)])) print (int(num\_chars/num\_words),'\t\t\t', int(num\_words/num\_sents),'\t\t\t', int(num\_words/num\_vocab),'\t\t', fileid)

output:



Categorial Corpora:

from nltk.corpus.reader import CategorizedPlaintextCorpusReader

mycat = CategorizedPlaintextCorpusReader(

'C:\\MYCorpus\_cat', r'sample.\*\.txt', cat\_pattern = r'.\*?\_(one|two).\*') print ("Categorize : ", mycat.categories())

print ("\nOne : ", mycat.fileids(categories =['one']))

print ("\nTwo : ", mycat.fileids(categories =['two']))

mycat.words(categories='one')

print ('Avg Word Len\tAvg Sentence Len\t No of Times Each Word Appears On Avg\t FileName') for fileid in mycat.fileids():

num\_chars = len(mycat.raw(fileid)) num\_words =

len(mycat.words(fileid)) num\_sents = len(mycat.sents(fileid)) num\_vocab = len(set([w.lower() for w in mycat.words(fileid)]))

print (int(num\_chars/num\_words),'\t\t\t', int(num\_words/num\_sents), '\t\t\t',

int(num\_words/num\_vocab), '\t\t\t\t', fileid)

c. Study Conditional frequency distributions source code:

#process a sequence of pairs text = ['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', ...] pairs = [('news', 'The'), ('news', 'Fulton'), ('news', 'County'), ...] import nltk from nltk.corpus import brown fd = nltk.ConditionalFreqDist( (genre, word)

for genre in brown.categories()

for word in brown.words(categories=genre))

genre\_word = [(genre, word)

for genre in ['news', 'romance']

for word in brown.words(categories=genre)]

print(len(genre\_word))

print(genre\_word[:4])

print(genre\_word[-4:])

cfd = nltk.ConditionalFreqDist(genre\_word)

print(cfd)

print(cfd.conditions())

print(cfd['news'])

print(cfd['romance']) print(list(cfd['romance'])) from nltk.corpus import inaugural cfd = nltk.ConditionalFreqDist(

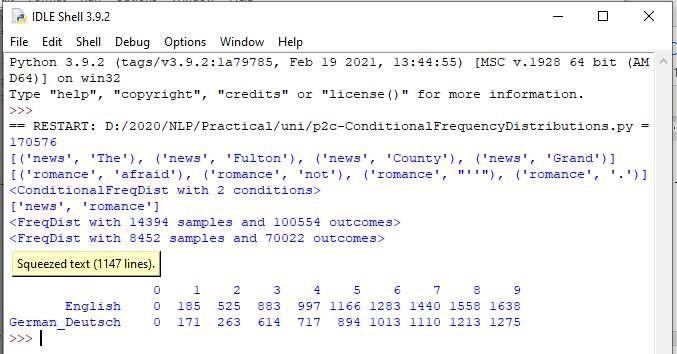
(target, fileid[:4]) for fileid in inaugural.fileids() for w in inaugural.words(fileid) for target in ['america', 'citizen']

if w.lower().startswith(target))

from nltk.corpus import udhr languages = ['Chickasaw', 'English', 'German\_Deutsch', 'Greenlandic\_Inuktikut', 'Hungarian\_Magyar', 'Ibibio\_Efik'] cfd = nltk.ConditionalFreqDist( (lang, len(word)) for lang in languages for word in udhr.words(lang + 'Latin1'))

cfd.tabulate(conditions=['English',

'German\_Deutsch'], samples=range(10), cumulative=True) output:



d. Study of tagged corpora with methods like tagged\_sents, tagged\_words.

A tagged corpus, also known as an annotated corpus or labeled corpus, is a type of corpus where each word or token in the text is associated with specific linguistic annotations or tags. These tags provide information about the grammatical structure, part of speech (POS), syntactic relationships, named entities, sentiment, or other linguistic features of the text.

Source code:

#study of tagged corpora – tagged words nltk.corpus.brown.tagged\_words() from nltk.corpus import brown brown\_news\_tagged = brown.tagged\_words(categories='news', tagset='universal') tag\_fd = nltk.FreqDist(tag for (word, tag) in brown\_news\_tagged) tag\_fd.keys()

#Tagged Sentences

brown\_tagged\_sents = brown.tagged\_sents(categories='news')

brown\_sents = brown.sents(categories='news') print(brown\_sents)

print(brown\_tagged\_sents)

e. Write a program to find the most frequent noun tags.

Code:

import nltk

from collections import defaultdict text = nltk.word\_tokenize("Nick likes to play

football. Nick does not like to play cricket.") tagged = nltk.pos\_tag(text) print(tagged)

# checking if it is a noun or not addNounWords = [] count=0 for words in tagged: val = tagged[count][1] if(val == 'NN' or val == 'NNS' or val == 'NNPS' or val == 'NNP'): addNounWords.append(tagged[count][0]) count+=1

print (addNounWords)

temp = defaultdict(int)

# memoizing count for sub in addNounWords: for wrd in sub.split():

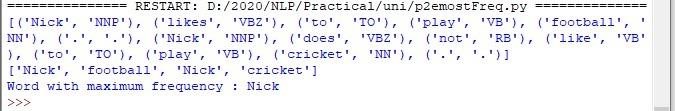
temp[wrd] += 1

# getting max frequency

res = max(temp, key=temp.get)

# printing result print("Word with maximum frequency : " + str(res))

output:



1. Map Words to Properties Using Python Dictionaries code:

#creating and printing a dictionay by mapping word with its properties thisdict = {

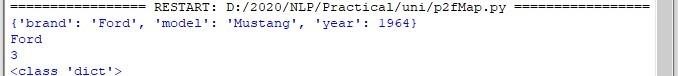
"brand": "Ford",

"model": "Mustang",

"year": 1964 } print(thisdict) print(thisdict["brand"] ) print(len(thisdict))

print(type(thisdict))

output:



1. Study i) DefaultTagger, ii) Regular expression tagger, iii) UnigramTagger

i) DefaultTagger code: import nltk from nltk.tag import

DefaultTagger exptagger =

DefaultTagger('NN') from nltk.corpus import treebank testsentences = treebank.tagged\_sents() [1000:]

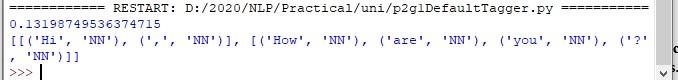
print(exptagger.evaluate (testsentences))

#Tagging a list of sentences import nltk

from nltk.tag import DefaultTagger

exptagger = DefaultTagger('NN') print(exptagger.tag\_sents([['Hi', ','], ['How', 'are', 'you', '?']]))

output



ii) Regular expression tagger, code: from nltk.corpus import brown from nltk.tag

import RegexpTagger test\_sent = brown.sents(categories='news')[0]

regexp\_tagger = RegexpTagger(

[(r'^-?[0-9]+(.[0-9]+)?$', 'CD'), # cardinal numbers

(r'(The|the|A|a|An|an)$', 'AT'), # articles

(r'.\*able$', 'JJ'), # adjectives

(r'.\*ness$', 'NN'), # nouns formed from adjectives

(r'.\*ly$', 'RB'), # adverbs

(r'.\*s$', 'NNS'), # plural nouns

(r'.\*ing$', 'VBG'), # gerunds

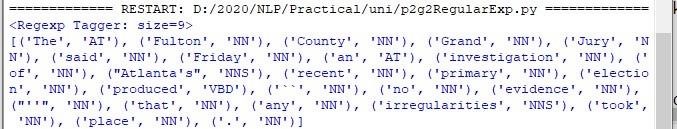
(r'.\*ed$', 'VBD'), # past tense verbs

(r'.\*', 'NN') # nouns (default)

])

print(regexp\_tagger)

print(regexp\_tagger.tag(test\_sent)) output:

 iii) UnigramTagger code: # Loading Libraries from nltk.tag import UnigramTagger

from nltk.corpus import treebank

# Training using first 10 tagged sentences of the treebank corpus as data.

# Using data train\_sents = treebank.tagged\_sents()[:10]

# Initializing tagger = UnigramTagger(train\_sents)

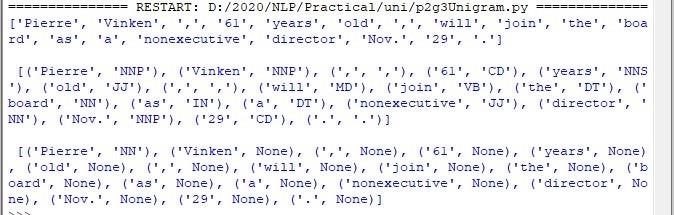
# Lets see the first sentence # (of the treebank corpus) as list print(treebank.sents()[0])

print('\n',tagger.tag(treebank.sents()[0])

)

#Finding the tagged results after training. tagger.tag(treebank.sents()[0])

#Overriding the context model tagger = UnigramTagger(model ={'Pierre': 'NN'}) print('\n',tagger.tag(treebank.sents()[0])) output:



h. Find different words from a given plain text without any space by comparing this text with a given corpus of words. Also find the score of words.

Question:

Initialize the hash tag test data or URL test data and convert to plain text without any space.. Read a text file of different words and compare the plain text data with the words exist in that text file and find out different words available in that plain text. Also find out how many

words could be found. (for example, text = "#whatismyname" or text =

www.whatismyname.com. Convert that to plain text without space as: whatismyname and read text file as words.txt. Now compare plain text with words given in a file and find the words form the plain text and the count of words which could be found) Source code:

from \_\_future\_\_ import with\_statement #with statement for reading file

import re # Regular expression

words = [] # corpus file words

testword = [] # test words

ans = [] # words matches with corpus

print("MENU")

print("-----------") print(" 1 . Hash

tag segmentation ") print(" 2 . URL segmentation ") print("enter the input choice for performing word segmentation")

choice = int(input())

if choice == 1: text = "#whatismyname" # hash tag test data to

segment print("input with HashTag",text)

pattern=re.compile("[^\w']") a = pattern.sub('', text) elif choice == 2: text = "www.whatismyname.com" # url test data to segment print("input with

URL",text) a=re.split('\s|(?<!\d)[,.](?!\d)', text) splitwords = ["www","com","in"] # remove the words which is containg in the list a ="".join([each for each in a if each not in splitwords]) else: print("wrong choice...try again")

print(a)

for each in a: testword.append(each) #test word test\_lenth =

len(testword) # lenth of the test data

# Reading the corpus with open('words.txt', 'r') as f: lines = f.readlines() words =[(e.strip()) for e in lines]

def Seg(a,lenth): ans=[] for k in range(0,lenth+1): # this loop checks char by char in the corpus

if a[0:k] in words:

print(a[0:k],"-appears in the corpus") ans.append(a[0:k]) break

if ans != []: g = max(ans,key=len) return g

test\_tot\_itr = 0 #each iteration value

answer = [] # Store the each word contains the corpus

Score = 0 # initial value for score

N = 37 # total no of corpus M = 0 C = 0 while test\_tot\_itr<test\_lenth:

ans\_words =

Seg(a,test\_lenth) if ans\_words != 0: test\_itr = len(ans\_words) answer.append(ans\_words)

a = a[test\_itr:test\_lenth]

test\_tot\_itr += test\_itr

Aft\_Seg = " ".join([each for each in answer]) # print segmented words in the list print("output") print("---------") print(Aft\_Seg) # print After

segmentation the input

# Calculating Score C = len(answer) score = C \* N / N # Calculate the score

print("Score",score)

Input:

Words.txt --------------

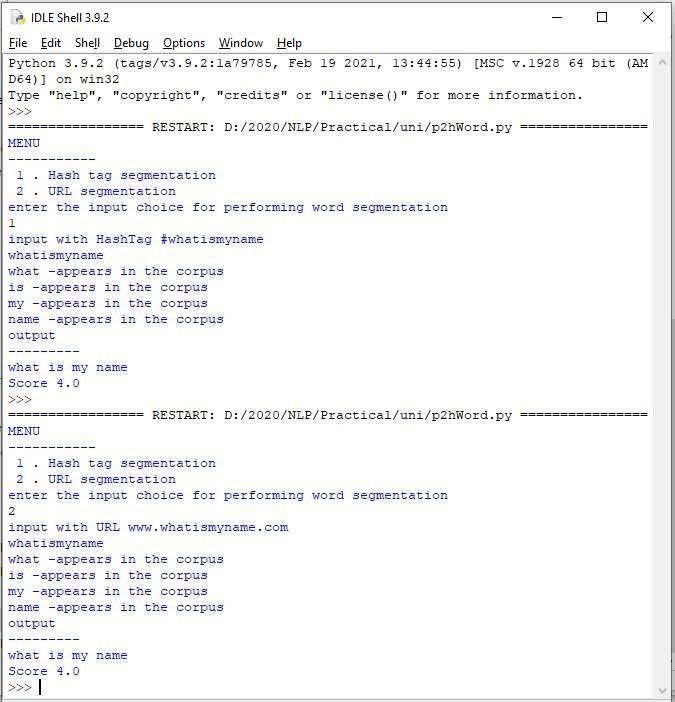
check domains domain honesty big hour rocks follow name back cheap social being media human 30 current seconds rates earth ought this to is

go insane down it apple time

what let is us my go

name

Output:



Practical 3 :

a. Study of Wordnet Dictionary with methods as synsets, definitions, examples, antonyms

WordNet is a lexical database of English words and their semantic relationships. It's organized as a network of synsets (sets of synonyms), where each synset represents a distinct concept or meaning. WordNet includes nouns, verbs, adjectives, and adverbs, with links between related synsets such as hypernyms (superordinates), hyponyms (subordinates), meronyms (part-whole relationships), and antonyms.

Source code:

'''WordNet provides synsets which is the collection of synonym words also called “lemmas”''' import nltk

from nltk.corpus import wordnet

print(wordnet.synsets("computer"))

# definition and example of the word ‘computer’

print(wordnet.synset("computer.n.01").definition())

#examples print("Examples:", wordnet.synset("computer.n.01").examples())

#get Antonyms print(wordnet.lemma('buy.v.01.buy').antonyms())

output:



b. Study lemmas, hyponyms, hypernyms.

A lemma refers to the base or canonical form of a word. Hyponyms are words or concepts that are more specific or subordinate to a broader category or concept, known as a hypernym. Hypernyms are words or concepts that are more general or encompassing, and they serve as the superordinate or parent categories in a semantic hierarchy.

Source code:

import nltk

from nltk.corpus import wordnet

print(wordnet.synsets("computer"))

print(wordnet.synset("computer.n.01").lemma\_names()) #all lemmas for each synset. for e in wordnet.synsets("computer"):

print(f'{e} --> {e.lemma\_names()}')

#print all lemmas for a given synset print(wordnet.synset('computer.n.01').lemmas())

#get the synset corresponding to lemma print(wordnet.lemma('computer.n.01.computing\_device').synset())

#Get the name of the lemma print(wordnet.lemma('computer.n.01.computing\_device').name())

#Hyponyms give abstract concepts of the word that are much more specific

#the list of hyponyms words of the computer

syn = wordnet.synset('computer.n.01')

print(syn.hyponyms)

print([lemma.name() for synset in syn.hyponyms() for lemma in synset.lemmas()])

#the semantic similarity in WordNet vehicle = wordnet.synset('vehicle.n.01')

car = wordnet.synset('car.n.01')

print(car.lowest\_common\_hypernyms(vehicle))

Output:



c. Write a program using python to find synonym and antonym of word "active" using Wordnet.

Source code:

from nltk.corpus import wordnet

synonyms = []

antonyms = []

for syn in wordnet.synsets("active"): for l in syn.lemmas(): synonyms.append(l.name())

if l.antonyms(): antonyms.append(l.antonyms()[0].name())

print(set(synonyms))

print(set(antonyms))

d. Compare two nouns

source code:

import nltk from nltk.corpus

import wordnet

syn1 = wordnet.synsets('football')

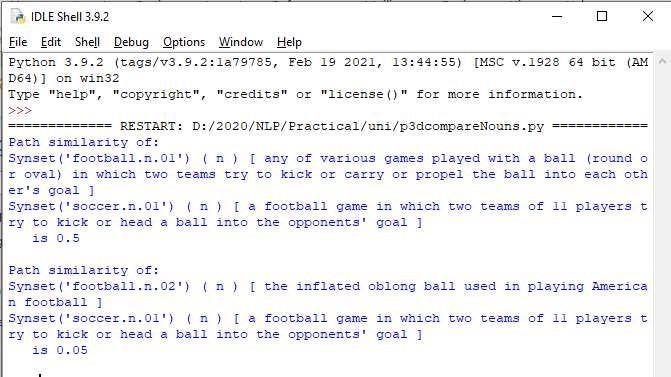
syn2 = wordnet.synsets('soccer')

# A word may have multiple synsets, so need to compare each synset of word1 with synset of word2 for s1 in syn1: for s2 in syn2: print("Path similarity of: ")

print(s1, '(', s1.pos(), ')', '[', s1.definition(), ']')

print(s2, '(', s2.pos(), ')', '[', s2.definition(), ']')

print(" is", s1.path\_similarity(s2)) print() output:



e. Handling stopword:

Stopwords are commonly used words in natural language that are often filtered out during text preprocessing in natural language processing (NLP) tasks. These words are generally considered to be non-informative or redundant in the context of text analysis and can be removed to focus on more meaningful content.

i) Using nltk Adding or Removing Stop Words in NLTK's Default Stop Word List

code: import nltk from

nltk.corpus import stopwords nltk.download('stopwords')

from nltk.tokenize import word\_tokenize

text = "Yashesh likes to play football, however he is not too fond of tennis."

text\_tokens = word\_tokenize(text)

tokens\_without\_sw = [word for word in text\_tokens if not word in stopwords.words()]

print(tokens\_without\_sw)

#add the word play to the NLTK stop word collection all\_stopwords = stopwords.words('english') all\_stopwords.append('play')

text\_tokens = word\_tokenize(text) tokens\_without\_sw = [word for word in text\_tokens if not word in all\_stopwords]

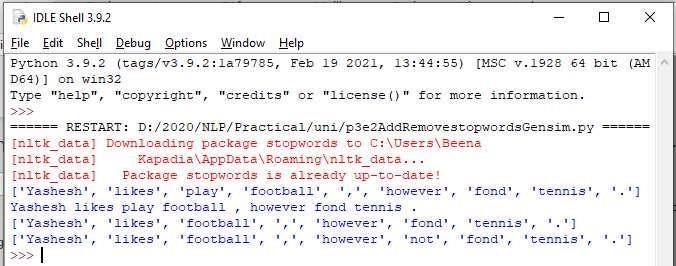
print(tokens\_without\_sw)

#remove ‘not’ from stop word collection all\_stopwords.remove('not')

text\_tokens = word\_tokenize(text) tokens\_without\_sw = [word for word in text\_tokens if not word in all\_stopwords]

print(tokens\_without\_sw)

output



ii) Using Gensim Adding and Removing Stop Words in Default Gensim Stop Words List

code:

#pip install gensim import gensim

from gensim.parsing.preprocessing import remove\_stopwords

text = "Yashesh likes to play football, however he is not too fond of tennis."

filtered\_sentence = remove\_stopwords(text)

print(filtered\_sentence)

all\_stopwords = gensim.parsing.preprocessing.STOPWORDS

print(all\_stopwords)

'''The following script adds likes and play to the list of stop words in Gensim:'''

from gensim.parsing.preprocessing import STOPWORDS

all\_stopwords\_gensim = STOPWORDS.union(set(['likes', 'play']))

text = "Yashesh likes to play football, however he is not too fond of tennis." text\_tokens = word\_tokenize(text) tokens\_without\_sw = [word for word in text\_tokens if not word in all\_stopwords\_gensim]

print(tokens\_without\_sw)

'''Output:

['Yashesh', 'football', ',', 'fond', 'tennis', '.']

The following script removes the word "not" from the set of stop words in Gensim:'''

from gensim.parsing.preprocessing import STOPWORDS

all\_stopwords\_gensim = STOPWORDS sw\_list = {"not"}

all\_stopwords\_gensim = STOPWORDS.difference(sw\_list)

text = "Yashesh likes to play football, however he is not too fond of tennis." text\_tokens = word\_tokenize(text) tokens\_without\_sw = [word for word in text\_tokens if not word in all\_stopwords\_gensim]

print(tokens\_without\_sw)

output Microsoft Visual C++ 14.0 is required. Get it with "Build Tools for Visual Studio": https://visualstudio.microsoft.com/downloads/

iii) Using Spacy Adding and Removing Stop Words in Default Spacy Stop Words List

code:

#pip install spacy

#python -m spacy download en\_core\_web\_sm

#python -m spacy download en

import spacy import nltk from

nltk.tokenize import word\_tokenize

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

#add the word play to the NLTK stop word collection all\_stopwords = sp.Defaults.stop\_words all\_stopwords.add("play")

text = "Yashesh likes to play football, however he is not too fond of tennis."

text\_tokens = word\_tokenize(text)

tokens\_without\_sw = [word for word in text\_tokens if not word in all\_stopwords]

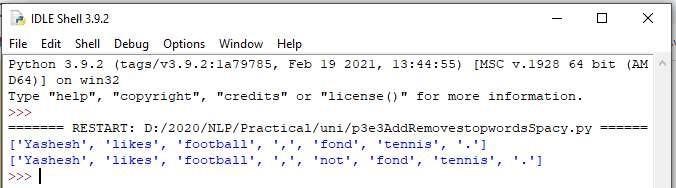
print(tokens\_without\_sw)

#remove 'not' from stop word collection all\_stopwords.remove('not')

tokens\_without\_sw = [word for word in text\_tokens if not word in all\_stopwords]

print(tokens\_without\_sw)

output:



## Practical 4 : Text Tokenization

Tokenization is the process of breaking down a text or a corpus into smaller units called tokens. These tokens can be words, phrases, sentences, or even individual characters, depending on the level of granularity required for analysis. Tokenization is a fundamental step in natural language processing (NLP) and text analysis tasks.

1. Tokenization using Python’s split() function code: text = """ This tool is an a beta stage. Alexa developers can use Get Metrics API to

seamlessly analyse metric. It also supports custom skill model, prebuilt Flash Briefing model, and the Smart Home Skill API. You can use this tool for creation of monitors, alarms, and dashboards that spotlight changes. The release of these three tools will enable developers to create visual rich skills for Alexa devices with screens. Amazon describes these tools as the collection of tech and tools for creating visually rich and interactive voice experiences. """ data = text.split('.') for i in data: print (i)

output:



1. Tokenization using Regular Expressions (RegEx)

code:

import nltk

# import RegexpTokenizer() method from nltk

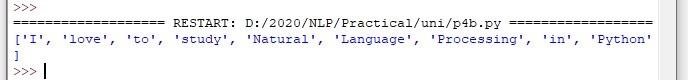
from nltk.tokenize import RegexpTokenizer

# Create a reference variable for Class RegexpTokenizer tk = RegexpTokenizer('\s+', gaps = True)

# Create a string input str = "I love to study Natural Language Processing in Python"

# Use tokenize method tokens = tk.tokenize(str) print(tokens)

output:



1. Tokenization using NLTK

code: import nltk from nltk.tokenize import

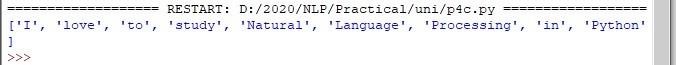
word\_tokenize

# Create a string input str = "I love to study Natural Language Processing in Python"

# Use tokenize method

print(word\_tokenize(str))

output:



1. Tokenization using the spaCylibrary

code: import spacy nlp =

spacy.blank("en")

# Create a string input str = "I love to study Natural Language Processing in Python"

# Create an instance of document;

# doc object is a container for a sequence of Token objects.

doc = nlp(str)

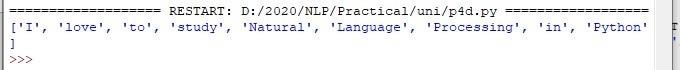
# Read the words; Print the words

#

words = [word.text for word in doc]

print(words)

output:



1. Tokenization using Keras

code:

#pip install keras #pip install tensorflow import keras from keras.preprocessing.text import text\_to\_word\_sequence

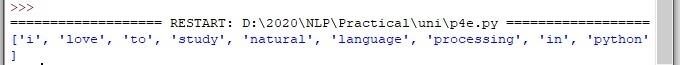
# Create a string input str = "I love to study Natural Language Processing in Python"

# tokenizing the text

tokens = text\_to\_word\_sequence(str)

print(tokens)

output:



1. Tokenization using Gensim

code:

#pip install gensim

from gensim.utils import tokenize

# Create a string input str = "I love to study Natural Language Processing in Python"

# tokenizing the text

list(tokenize(str))

output: Microsoft Visual C++ 14.0 is required. Get it with "Build Tools for Visual Studio": https://visualstudio.microsoft.com/downloads/

## Practical 5 : Import NLP Libraries for Indian Languages and perform

Note: Execute this practical in https://colab.research.google.com/

a) word tokenization in Hindi Source

code:

!pip install torch==1.3.1+cpu -f https://download.pytorch.org/whl/torch\_stable.html

pip install torch==1.3.1+cpu -f https://download.pytorch.org/whl/torch\_stable.html

pip install inltk

!pip install inltk

!pip install tornado==4.5.3

from inltk.inltk import setup

setup(‘hi’)

from inltk.inltk import tokenize

hindi\_text = “”” ा कृ िातकभ ष सीखन ब ितदलचहै।“””

# tokenize(input text, language code) tokenize(hindi\_text, “hi”)

output

[‘▁ ा कृ िातक’, ‘▁भ ष ’, ‘▁सीखन ’, ‘▁ब त’, ‘▁िादलच’, ‘▁है’, ‘।‘]

b) Generate similar sentences from a given Hindi text input Source

code:

!pip install torch==1.3.1+cpu -f https://download.pytorch.org/whl/torch\_stable.html

!pip install inltk

!pip install tornado==4.5.3

from inltk.inltk import setup

setup(‘hi’)

from inltk.inltk import get\_similar\_sentences

# get similar sentences to the one given in hindi output = get\_similar\_sentences(‘मआजब तखुश ा ’, 5, ‘hi’) print(output)

Output:

[‘मआजकलब तखुश ा ’, ‘मआजअ िाधकखुश ा ’, ‘मअभीब तखुश ा ’, ‘मवतम नब तखुश ा ’,

‘मवतम नब तखुश ा ’]

c) Identify the Indian language of a text Source code:

!pip install torch==1.3.1+cpu -f https://download.pytorch.org/whl/torch\_stable.html

!pip install inltk

!pip install tornado==4.5.3

from inltk.inltk import setup

setup(‘gu’)

from inltk.inltk import identify\_language #Identify the Lnaguage of given text

identify\_language(‘બીનાકાપિડયા’)

Output:

Gujarati

Practical 6 : Illustrate part of speech tagging.

Part-of-speech (POS) tagging, also known as grammatical tagging or word-category disambiguation, is a process in natural language processing (NLP) that assigns grammatical tags to words in a text based on their role and function within a sentence. These tags represent the syntactic category of each word, such as noun, verb, adjective, adverb, pronoun, conjunction, preposition, etc.

POS Tagging, chunking and NER:

a) sentence tokenization, word tokenization, Part of speech Tagging and chunking of user

defined text. Source code: import nltk from nltk import tokenize nltk.download('punkt') from nltk import tag from nltk import chunk nltk.download('averaged\_perceptron\_tagger') nltk.download('maxent\_ne\_chunker') nltk.download('words')

para = "Hello! My name is Beena Kapadia. Today you'll be learning NLTK." sents = tokenize.sent\_tokenize(para) print("\nsentence tokenization\n===================\n",sents)

# word tokenization print("\nword tokenization\n===================\n") for index in range(len(sents)): words = tokenize.word\_tokenize(sents[index])

print(words)

# POS Tagging

tagged\_words = [] for index

in range(len(sents)):

tagged\_words.append(tag.pos\_tag(words))

print("\nPOS Tagging\n===========\n",tagged\_words)

# chunking

tree = [] for index in

range(len(sents)):

tree.append(chunk.ne\_chunk(tagged\_words[index] )) print("\nchunking\n========\n") print(tree)

Output:

sentence tokenization

===================

['Hello!', 'My name is Beena Kapadia.', "Today you'll be learning NLTK."]

word tokenization

===================

['Hello', '!']

['My', 'name', 'is', 'Beena', 'Kapadia', '.']

['Today', 'you', "'ll", 'be', 'learning', 'NLTK', '.']

POS Tagging

===========

[[('Today', 'NN'), ('you', 'PRP'), ("'ll", 'MD'), ('be', 'VB'), ('learning', 'VBG'), ('NLTK', 'NNP'),

('.', '.')], [('Today', 'NN'), ('you', 'PRP'), ("'ll", 'MD'), ('be', 'VB'), ('learning', 'VBG'), ('NLTK', 'NNP'), ('.', '.')], [('Today', 'NN'), ('you', 'PRP'), ("'ll", 'MD'), ('be', 'VB'), ('learning', 'VBG'),

('NLTK', 'NNP'), ('.', '.')]]

chunking

========

[Tree('S', [('Today', 'NN'), ('you', 'PRP'), ("'ll", 'MD'), ('be', 'VB'), ('learning', 'VBG'),

Tree('ORGANIZATION', [('NLTK', 'NNP')]), ('.', '.')]), Tree('S', [('Today', 'NN'), ('you', 'PRP'),

("'ll", 'MD'), ('be', 'VB'), ('learning', 'VBG'), Tree('ORGANIZATION', [('NLTK', 'NNP')]), ('.',

'.')]), Tree('S', [('Today', 'NN'), ('you', 'PRP'), ("'ll", 'MD'), ('be', 'VB'), ('learning', 'VBG'), Tree('ORGANIZATION', [('NLTK', 'NNP')]), ('.', '.')])]

b) Named Entity recognition using user defined text.

Source code:

!pip install -U spacy

!python -m spacy download en\_core\_web\_sm

import spacy

# Load English tokenizer, tagger, parser and NER nlp = spacy.load("en\_core\_web\_sm")

# Process whole documents text = ("When Sebastian Thrun started working on self-driving cars at " "Google in 2007, few people outside of the company took him "

"seriously. “I can tell you very senior CEOs of major American "

"car companies would shake my hand and turn away because I wasn’t "

"worth talking to,” said Thrun, in an interview with Recode earlier "

"this week.")

doc = nlp(text)

# Analyse syntax

print("Noun phrases:", [chunk.text for chunk in doc.noun\_chunks]) print("Verbs:", [token.lemma\_ for token in doc if token.pos\_ == "VERB"]) Output:

Noun phrases: ['Sebastian Thrun', 'self-driving cars', 'Google', 'few people', 'the company', 'him', 'I', 'you', 'very senior CEOs', 'major American car companies', 'my hand', 'I', 'Thrun', 'an interview', 'Recode']

Verbs: ['start', 'work', 'drive', 'take', 'tell', 'shake', 'turn', 'be', 'talk', 'say']

c) Named Entity recognition with diagram using NLTK corpus – treebank. Source

code:

Note: It runs on Python IDLE

import nltk nltk.download('treebank')

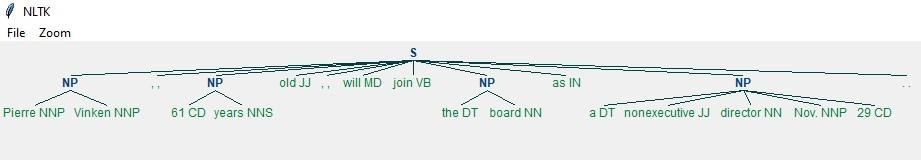
from nltk.corpus import treebank\_chunk

treebank\_chunk.tagged\_sents()[0]

treebank\_chunk.chunked\_sents()[0]

treebank\_chunk.chunked\_sents()[0].draw()

Output:



## Practical 7 : Finite state automata

In NLTK (Natural Language Toolkit), grammar can be defined using context-free grammars (CFGs). A context-free grammar consists of a set of production rules that describe how symbols (such as words or phrases) in a language can be combined to form valid sentences. Each production rule specifies a symbol on the left-hand side and a sequence of symbols (including terminals and non-terminals) on the right-hand side.

a) Define grammar using nltk. Analyze a sentence using the same.

Code: import nltk from nltk import

tokenize grammar1 = nltk.CFG.fromstring("""

S -> VP

VP -> VP NP

NP ->Det NP

Det -> 'that'

NP -> singular Noun

NP -> 'flight'

VP -> 'Book'

""")

sentence = "Book that flight"

for index in range(len(sentence)): all\_tokens = tokenize.word\_tokenize(sentence)

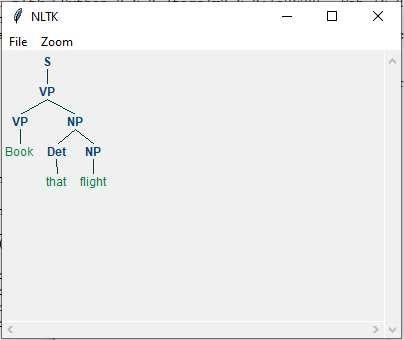
print(all\_tokens)

parser = nltk.ChartParser(grammar1)

for tree in parser.parse(all\_tokens): print(tree)

tree.draw()

output:



b) Accept the input string with Regular expression of Finite Automaton: 101+.

Source code: def FA(s):

#if the length is less than 3 then it can't be accepted, Therefore end the process. if len(s)<3: return "Rejected"

#first three characters are fixed. Therefore, checking them using index if s[0]=='1': if s[1]=='0': if s[2]=='1':

# After index 2 only "1" can appear. Therefore break the process if any other character is detected for i in range(3,len(s)): if s[i]!='1': return "Rejected" return "Accepted" # if all 4 nested if true return "Rejected" # else of 3rd if return "Rejected" # else of 2nd if return "Rejected" # else of 1st if

inputs=['1','10101','101','10111','01010','100','','10111101','1011111'

] for i in inputs: print(FA(i))

Output:

Rejected

Rejected

Accepted

Accepted

Rejected

Rejected

Rejected

Rejected

Accepted

c) Accept the input string with Regular expression of FA: (a+b)\*bba.

Code: def FA(s): size=0

#scan complete string and make sure that it contains only 'a' & 'b' for i in s: if i=='a' or i=='b': size+=1

else: return "Rejected"

#After checking that it contains only 'a' & 'b' #check it's length it should be 3 atleast if size>=3: #check the last 3 elements if s[size3]=='b': if s[size-

2]=='b': if s[size-1]=='a':

return "Accepted" # if all 4 if

true return "Rejected" # else of 4th if return "Rejected" # else of 3rd if return "Rejected" # else of 2nd if

return "Rejected" # else of 1st if

inputs=['bba', 'ababbba', 'abba','abb',

'baba','bbb',''] for i in inputs: print(FA(i))

output: Rejected

Rejected

Accepted

Accepted

Rejected

Rejected

Rejected

Rejected

Accepted

d) Implementation of Deductive Chart Parsing using context free grammar and a givensentence.

Deductive chart parsing is a technique used in natural language processing (NLP) to analyze the syntactic structure of sentences based on a context-free grammar (CFG). It involves building a parse chart that systematically explores possible parse trees according to the grammar rules, using deductive reasoning to determine valid parses.

Source code: import nltk from nltk import tokenize grammar1 = nltk.CFG.fromstring("""

S -> NP VP

PP -> P NP

NP -> Det N | Det N PP | 'I'

VP -> V NP | VP PP

Det -> 'a' | 'my'

N -> 'bird' | 'balcony'

V -> 'saw'

P -> 'in'

""")

sentence = "I saw a bird in my balcony"

for index in range(len(sentence)): all\_tokens = tokenize.word\_tokenize(sentence)

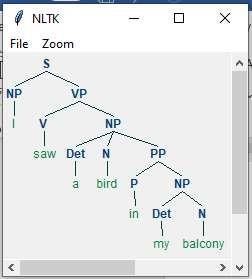
print(all\_tokens)

# all\_tokens = ['I', 'saw', 'a', 'bird', 'in', 'my', 'balcony'] parser = nltk.ChartParser(grammar1)

for tree in parser.parse(all\_tokens): print(tree)

tree.draw()

output:



Practical 8 : Study PorterStemmer, LancasterStemmer, RegexpStemmer, Snowball Stemmer,

Study WordNetLemmatizer

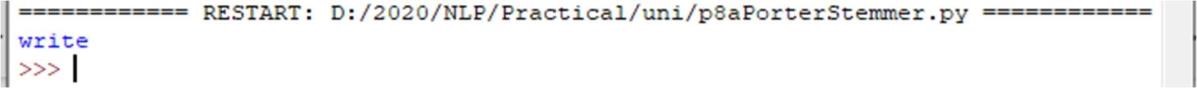
PorterStemmer is a stemming algorithm that reduces words to their base or root form by removing common suffixes, often used in information retrieval and text mining tasks. LancasterStemmer is another stemming algorithm that is more aggressive than PorterStemmer, capable of producing shorter stems but may result in over-stemming in certain cases. RegexpStemmer is a customizable stemming algorithm in NLTK that allows users to specify their own regular expressions for stemming words based on specific patterns. Snowball Stemmer, also known as Porter2 Stemmer, is an improved version of

PorterStemmer with support for multiple languages and more accurate stemming rules. WordNetLemmatizer is a lemmatization tool in NLTK that maps words to their base or dictionary forms (lemmas) based on WordNet's lexical database, useful for NLP tasks requiring accurate word normalization.

Code:

#PorterStemmer import nltk

from nltk.stem import PorterStemmer word\_stemmer = PorterStemmer() print(word\_stemmer.stem('writing')) Output:

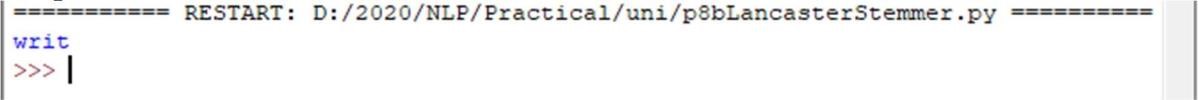


#LancasterStemmer

import nltk from nltk.stem import

LancasterStemmer Lanc\_stemmer = LancasterStemmer()

print(Lanc\_stemmer.stem('writing')) Output:



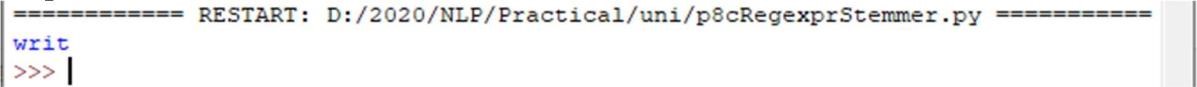
#RegexpStemmer import nltk

from nltk.stem import RegexpStemmer

Reg\_stemmer = RegexpStemmer('ing$|s$|e$|able$', min=4)

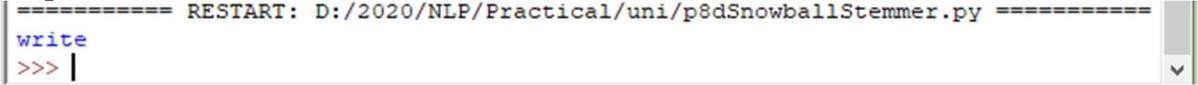
print(Reg\_stemmer.stem('writing'))

output



#SnowballStemmer import nltk from nltk.stem import SnowballStemmer english\_stemmer = SnowballStemmer('english') print(english\_stemmer.stem ('writing'))

output



#WordNetLemmatizer from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

print("word :\tlemma") print("rocks :",

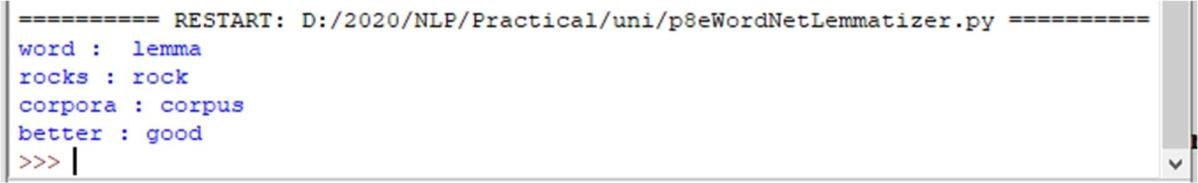
lemmatizer.lemmatize("rocks"))

print("corpora :", lemmatizer.lemmatize("corpora"))

# a denotes adjective in "pos"

print("better :", lemmatizer.lemmatize("better", pos ="a"))

Output:



## Practical 9 : Implement Naive Bayes classifier

Naive Bayes classifier is a probabilistic machine learning model commonly used in natural language processing (NLP) for tasks such as text classification, sentiment analysis, spam filtering, and document categorization.

The Naive Bayes classifier is based on Bayes' theorem, which calculates the probability of a hypothesis given the evidence. In NLP, the hypothesis could be the class or category of a document (e.g., positive or negative sentiment), and the evidence could be the words or features in the document.

Code:

#pip install pandas

#pip install sklearn

import pandas as pd

import numpy as np

sms\_data = pd.read\_csv("spam.csv", encoding='latin-1')

import re

import nltk

from nltk.corpus import stopwords from nltk.stem.porter import PorterStemmer

stemming = PorterStemmer() corpus = [] for i in range (0,len(sms\_data)):

s1 = re.sub('[^a-zA-Z]',repl = ' ',string = sms\_data['v2'][i]) s1.lower() s1 =

s1.split() s1 = [stemming.stem(word) for word in s1 if word not in

set(stopwords.words('english'))] s1 = ' '.join(s1)

corpus.append(s1)

from sklearn.feature\_extraction.text import CountVectorizer

countvectorizer =CountVectorizer()

1. = countvectorizer.fit\_transform(corpus).toarray() print(x)

1. = sms\_data['v1'].values

print(y)

from sklearn.model\_selection import train\_test\_split x\_train,x\_test,y\_train,y\_test = train\_test\_split(x,y,test\_size = 0.3, stratify=y,random\_state=2) #Multinomial Naïve Bayes. from sklearn.naive\_bayes import MultinomialNB multinomialnb = MultinomialNB()

multinomialnb.fit(x\_train,y\_train)

# Predicting on test data:

y\_pred = multinomialnb.predict(x\_test)

print(y\_pred)

#Results of our Models

from sklearn.metrics import classification\_report, confusion\_matrix

from sklearn.metrics import accuracy\_score

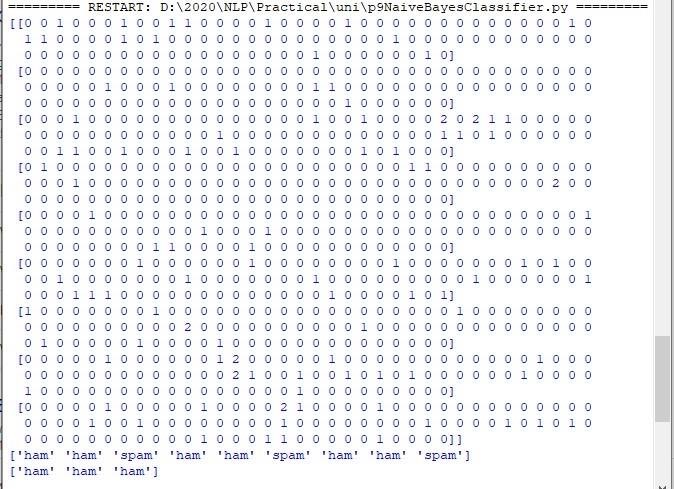
print(classification\_report(y\_test,y\_pred))

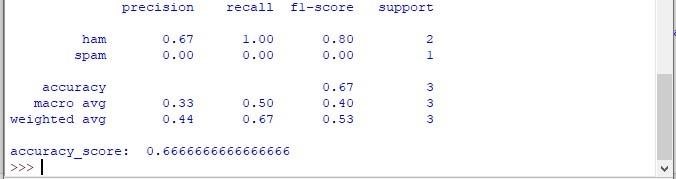
print("accuracy\_score: ",accuracy\_score(y\_test,y\_pred))

input:

spam.csv file from github

output:





## Practical 10 : Speech Tagging

Speech tagging, also known as part-of-speech tagging or POS tagging, is the process of assigning grammatical tags (parts of speech) to words in a text based on their syntactic roles and relationships within sentences. These tags indicate the word's category, such as noun, verb, adjective, adverb, pronoun, conjunction, preposition, etc.

Speech tagging is a fundamental task in natural language processing (NLP) and text analysis, as it helps in understanding the grammatical structure of sentences and extracting meaningful information from text data. It is used in various NLP applications, including information retrieval, text classification, sentiment analysis, machine translation, and named entity recognition.

i.Speech tagging using spacy

code import

spacy

sp = spacy.load('en\_core\_web\_sm') sen = sp(u"I like to play

football. I hated it in my childhood though") print(sen.text) print(sen[7].pos\_) print(sen[7].tag\_) print(spacy.explain(sen[7].tag\_)) for word in sen: print(f'{word.text:{12}} {word.pos\_:{10}} {word.tag\_:{8}} {spacy.explain(word.tag\_)}')

sen = sp(u'Can you google it?')

word = sen[2]

print(f'{word.text:{12}} {word.pos\_:{10}} {word.tag\_:{8}}

{spacy.explain(word.tag\_)}') sen = sp(u'Can you search it on google?') word = sen[5]

print(f'{word.text:{12}} {word.pos\_:{10}} {word.tag\_:{8}} {spacy.explain(word.tag\_)}')

#Finding the Number of POS Tags sen = sp(u"I like to play football. I hated it in my childhood though")

num\_pos = sen.count\_by(spacy.attrs.POS)

num\_pos

for k,v in sorted(num\_pos.items()): print(f'{k}. {sen.vocab[k].text:{8}}: {v}')

#Visualizing Parts of Speech Tags

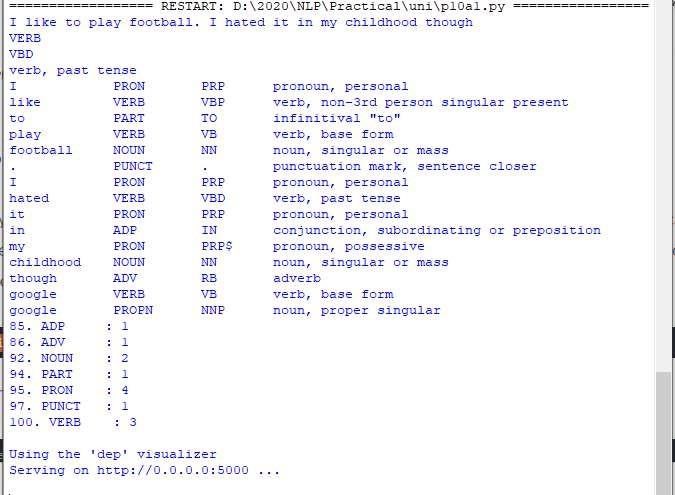
from spacy import displacy

sen = sp(u"I like to play football. I hated it in my childhood though")

displacy.serve(sen, style='dep', options={'distance': 120})

output

:



To view the dependency tree, type the following address in your browser: http://127.0.0.1:5000/. You will see the following dependency tree:

ii. Speech tagging using nktl

code: import

nltk

from nltk.corpus import state\_union

from nltk.tokenize import PunktSentenceTokenizer

#create our training and testing data:

train\_text = state\_union.raw("2005-GWBush.txt")

sample\_text = state\_union.raw("2006-GWBush.txt")

#train the Punkt tokenizer like:

custom\_sent\_tokenizer = PunktSentenceTokenizer(train\_text)

# tokenize:

tokenized = custom\_sent\_tokenizer.tokenize(sample\_text)

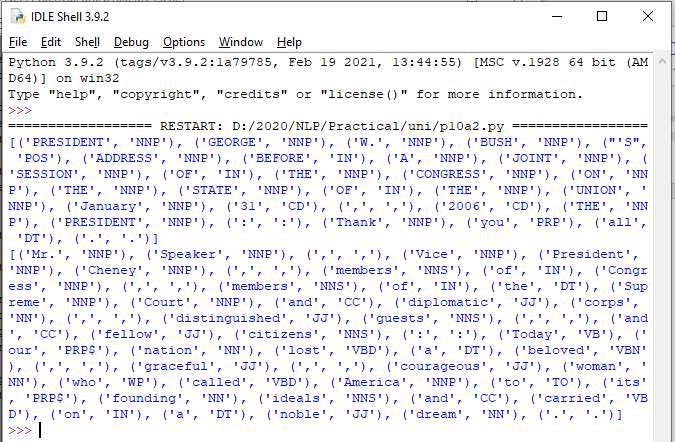
def process\_content(): try: for i in tokenized[:2]: words = nltk.word\_tokenize(i) tagged = nltk.pos\_tag(words) print(tagged)

except Exception as e:

print(str(e))

process\_content()

output:



b. Statistical parsing:

Statistical parsing is a computational linguistic technique that uses probabilistic models to analyze the syntactic structure of natural language sentences. It involves training statistical models on annotated corpora, such as treebanks, to learn the probabilities of different syntactic structures and parse trees.

Treebanks are collections of sentences annotated with their syntactic parse trees, where each node represents a grammatical constituent and edges denote syntactic relationships.

i. Usage of Give and Gave in the Penn Treebank sample Source

code:

#probabilitistic parser

#Usage of Give and Gave in the Penn Treebank sample

import nltk import

nltk.parse.viterbi

import nltk.parse.pchart

def give(t): return t.label() == 'VP' and len(t) > 2 and t[1].label() ==

'NP'\ and (t[2].label() == 'PP-DTV' or t[2].label() ==

'NP')\ and ('give' in t[0].leaves() or 'gave' in t[0].leaves())

def sent(t): return ' '.join(token for token in t.leaves() if token[0] not in '\*-0')

def print\_node(t, width): output = "%s %s: %s / %s: %s" %\

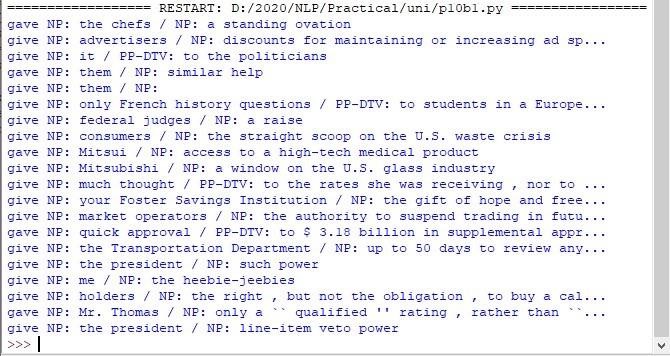
(sent(t[0]), t[1].label(), sent(t[1]), t[2].label(), sent(t[2])) if len(output) > width: output = output[:width] + "..."

print (output)

for tree in nltk.corpus.treebank.parsed\_sents(): for t in tree.subtrees(give):

print\_node(t, 72)

Output:

 ii. probabilistic parser Source code: import nltk from nltk import PCFG

grammar = PCFG.fromstring('''

NP -> NNS [0.5] | JJ NNS [0.3] | NP CC NP [0.2]

NNS -> "men" [0.1] | "women" [0.2] | "children" [0.3] | NNS CC NNS [0.4] JJ -> "old" [0.4] | "young" [0.6]

CC -> "and" [0.9] | "or" [0.1]

''')

print(grammar)

viterbi\_parser = nltk.ViterbiParser(grammar)

token = "old men and women".split()

obj = viterbi\_parser.parse(token)

print("Output: ")

for x in obj:

print(x)

Output:



c. Malt parsing:

Parse a sentence and draw a tree using malt parsing.

Note: 1) Java should be installed.

1. maltparser-1.7.2 zip file should be copied in C:\Users\Beena

Kapadia\AppData\Local\Programs\Python\Python39 folder and should be extracted in the same folder.

1. engmalt.linear-1.7.mco file should be copied to C:\Users\Beena Kapadia\AppData\Local\Programs\Python\Python39 folder Source code:

# copy maltparser-1.7.2(unzipped version) and engmalt.linear-1.7.mco files to C:\Users\Beena

Kapadia\AppData\Local\Programs\Python\Python39 folder # java should be installed

# environment variables should be set - MALT\_PARSER - C:\Users\Beena

Kapadia\AppData\Local\Programs\Python\Python39\maltparser-1.7.2 and MALT\_MODEL - C:\Users\Beena Kapadia\AppData\Local\Programs\Python\Python39\engmalt.linear-1.7.mco

from nltk.parse import malt mp = malt.MaltParser('maltparser-1.7.2', 'engmalt.linear-1.7.mco')#file t = mp.parse\_one('I saw a bird from my window.'.split()).tree() print(t) t.draw()

Output:

(saw I (bird a (from (window. my))))



Practical 11 :

a) Multiword Expressions in NLP

A multiword expression (MWE) refers to a sequence of words that functions as a single semantic unit, often carrying a meaning that cannot be inferred from the individual words alone. MWEs are prevalent in natural language and include phrases such as "kick the bucket" (meaning to die), "take it easy" (meaning to relax), "spill the beans" (meaning to reveal a secret), and "break a leg" (meaning good luck).

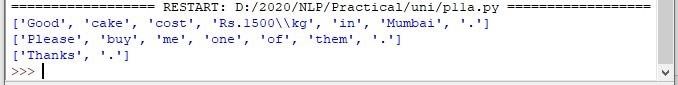
MWEs can be idiomatic, where the meaning of the expression is different from the literal meanings of its component words, or they can be compositional, where the meaning can be inferred from the individual words but the combination is still considered a single unit.

Source code:

# Multiword Expressions in NLP

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

word\_tokenize s = '''Good cake cost Rs.1500\kg in Mumbai. Please buy me one of them.\n\nThanks.''' mwe = MWETokenizer([('New', 'York'), ('Hong', 'Kong')], separator='\_') for sent in sent\_tokenize(s): print(mwe.tokenize(word\_tokenize(sent))) Output:



b) Normalized Web Distance and Word Similarity

Web distance refers to measuring the semantic relatedness between words or phrases by analyzing their co-occurrence patterns in a large corpus of web data, such as search engine results or web pages. This approach leverages the vast amount of textual information available on the web to calculate similarity or distance metrics, aiding tasks like information retrieval, text classification, and sentiment analysis.

Word similarity, on the other hand, focuses on quantifying the semantic closeness between individual words based on their meaning and context, using methods like distributional semantic models, word embeddings, and semantic networks to capture semantic relationships within language.

Source code:

# Normalized Web Distance and Word Similarity

#convert

#Reliance supermarket

#Reliance hypermarket

#Reliance

#Reliance

#Reliance downtown

#Relianc market

#Mumbai

#Mumbai Hyper

#Mumbai dxb

#mumbai airport

#k.m trading

#KM Trading

#KM trade

#K.M. Trading

#KM.Trading

#into

#Reliance

#Reliance

#Reliance

#Reliance

#Reliance

#Reliance

#Mumbai

#Mumbai

#Mumbai

#Mumbai

#KM Trading

#KM Trading

#KM Trading

#KM Trading

#KM Trading

import numpy as np

import re

import textdistance # pip install textdistance # we

will need scikit-learn>=0.21 import sklearn #pip install sklearn from sklearn.cluster import

AgglomerativeClustering

texts = [

'Reliance supermarket', 'Reliance hypermarket', 'Reliance', 'Reliance', 'Reliance downtown', 'Relianc market',

'Mumbai', 'Mumbai Hyper', 'Mumbai dxb', 'mumbai airport',

'k.m trading', 'KM Trading', 'KM trade', 'K.M. Trading', 'KM.Trading'

]

def normalize(text):

""" Keep only lower-cased text and numbers""" return re.sub('[^a-z0-9]+', ' ', text.lower())

def group\_texts(texts, threshold=0.4):

""" Replace each text with the representative of its cluster""" normalized\_texts = np.array([normalize(text) for text in texts]) distances = 1 - np.array([

[textdistance.jaro\_winkler(one, another) for one in normalized\_texts] for another in normalized\_texts

])

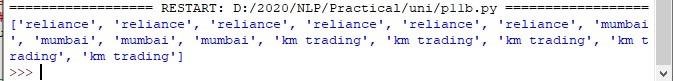
clustering = AgglomerativeClustering( distance\_threshold=threshold, #

this parameter needs to be tuned carefully affinity="precomputed", linkage="complete", n\_clusters=None

).fit(distances) centers = dict() for cluster\_id in set(clustering.labels\_): index = clustering.labels\_ == cluster\_id centrality = distances[:, index][index].sum(axis=1) centers[cluster\_id] = normalized\_texts[index][centrality.argmin()] return [centers[i] for i in clustering.labels\_]

print(group\_texts(texts))

Output:



Source code:

#Word Sense Disambiguation from nltk.corpus import wordnet as wn

def get\_first\_sense(word, pos=None): if pos: synsets = wn.synsets(word,pos)

else: synsets = wn.synsets(word)

return synsets[0]

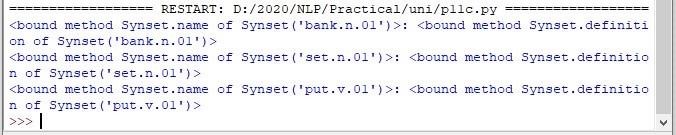
best\_synset = get\_first\_sense('bank') print ('%s: %s' % (best\_synset.name, best\_synset.definition)) best\_synset =

get\_first\_sense('set','n') print ('%s: %s' %

(best\_synset.name, best\_synset.definition)) best\_synset =

get\_first\_sense('set','v') print ('%s: %s' %

(best\_synset.name, best\_synset.definition)) Output:



c. Word Sense Disambiguation

Word Sense Disambiguation (WSD) is a natural language processing task that aims to determine the correct meaning or sense of a word within a given context. Many words in natural language have multiple meanings, known as senses, and WSD algorithms are designed to identify the most appropriate sense of a word based on its usage in a specific sentence or document.

Code:- import codecs

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

from nltk.corpus import stopwords, wordnet

from nltk.stem import WordNetLemmatizer, PorterStemmer

def simpleFilter(sentence):

filtered\_sent = [] lemmatizer = WordNetLemmatizer()

stop\_words = set(stopwords.words("english")) words = word\_tokenize(sentence)

for w in words: if w

not in stop\_words: filtered\_sent.append(lemmatizer.lemmatize(w))

return filtered\_sent

def simlilarityCheck(word1, word2):

word1 = word1 + ".n.01" word2 = word2 + ".n.01" try: w1 = wordnet.synset(word1)

w2 = wordnet.synset(word2)

return w1.wup\_similarity(w2)

except:

return 0

def synonymsCreator(word): synonyms = []

for syn in wordnet.synsets(word):

for i in syn.lemmas():

synonyms.append(i.name())

return synonyms

# Remove Stop Words . Word Stemming . Return new tokenised list. def filteredSentence(sentence):

filtered\_sent = [] lemmatizer = WordNetLemmatizer() #lemmatizes the words ps = PorterStemmer() #stemmer stems the root of the word.

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

words = word\_tokenize(sentence)

for w in words: if w not in stop\_words:

filtered\_sent.append(lemmatizer.lemmatize(ps.stem(w))) for i in synonymsCreator(w): filtered\_sent.append(i)

return filtered\_sent

if \_\_name\_\_ == '\_\_main\_\_':

cricfile = codecs.open("E:\\Department\\MscIT\_Sem\_III\_and sem I nov

2020\\NLP\\cricketbat.txt", 'r', "utf-8")

sent2 = cricfile.read().lower() vampirefile = codecs.open("E:\\Department\\MscIT\_Sem\_III\_and sem I nov

2020\\NLP\\vampirebat.txt", 'r', 'utf-8') sent1 = vampirefile.read().lower() sent3 = "start"

while(sent3 != "end"):

sent3 = input("Enter Query: ").lower()

filtered\_sent1 = []

filtered\_sent2 = []

filtered\_sent3 = []

counter1 = 0

counter2 = 0 sent31\_similarity = 0

sent32\_similarity = 0

filtered\_sent1 = simpleFilter(sent1) filtered\_sent2 = simpleFilter(sent2)

filtered\_sent3 = simpleFilter(sent3)

for i in filtered\_sent3:

for j in filtered\_sent1:

counter1 = counter1 + 1 sent31\_similarity =

sent31\_similarity + simlilarityCheck(i,j)

for j in filtered\_sent2:

counter2 = counter2 + 1 sent32\_similarity =

sent32\_similarity + simlilarityCheck(i,j)

filtered\_sent1 = []

filtered\_sent2 = []

filtered\_sent3 = []

filtered\_sent1 = filteredSentence(sent1) filtered\_sent2 = filteredSentence(sent2)

filtered\_sent3 = filteredSentence(sent3)

sent1\_count = 0

sent2\_count = 0

for i in filtered\_sent3:

for j in filtered\_sent1:

if(i==j): sent1\_count = sent1\_count + 1 for j in filtered\_sent2: if(i==j): sent2\_count = sent2\_count + 1

if((sent1\_count + sent31\_similarity)>(sent2\_count+sent32\_similarity)):

print ("Mammal Bat")

else: print ("Cricket Bat")

Output:-

#-----------------------------------------------

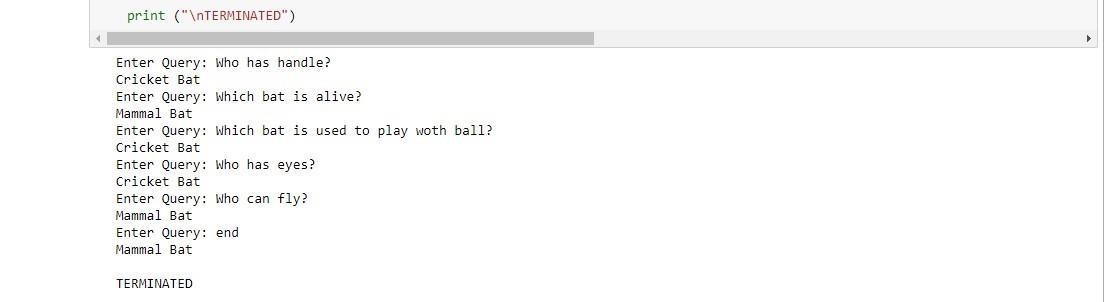
#Sentence1: the river bank has water in it and it has fishes trees . lots of water is stored in the banks. boats float in it and animals come and drink water from it.

#sentence2: the commercial banks are used for finance. all the financial matters are managed by financial banks and they have lots of money, user accounts like salary account and savings account, current account. money can also be withdrawn from this bank.

#query: from which bank should i withdraw money.

#sen1: any of various nocturnal flying mammals of the order Chiroptera, having membranous wings that extend from the forelimbs to the hind limbs or tail and anatomical adaptations for echolocation, by which they navigate and hunt prey.

#sen 2: a cricket wooden bat is used for playing criket. it is rectangular in shape and has handle and is made of wood or plastic and is used by cricket players. print ("\nTERMINATED")



# Deep Learning

**INDEX**

|  |  |
| --- | --- |
| **Practical**  **No** | **Title** |
| **1** | **Performing matrix multiplication and finding eigen vectors and eigen values using TensorFlow** |
| **2** | **Solving XOR problem using deep feed forward network.** |
| **3** | **Implementing deep neural network for performing binary classification task.** |
| **4** | **Using deep feed forward network with two hidden layers for performing multiclass classification and predicting the class.** |
| **5** | **Using a deep feed forward network with two hidden layers for performing linear regression and predicting values.** |
| **6** | **Evaluating feed forward deep network for multiclass Classification using KFold cross-validation.** |
| **7** | **Implementing regularization to avoid overfitting in binary classification.** |

Practical No: 1

Aim: Performing matrix multiplication and finding eigen vectors and eigen values using TensorFlow.

import tensorflow as tf

print("Matrix Multiplication Demo")

x=tf.constant([1,2,3,4,5,6],shape=[2,3])

print(x) y=tf.constant([7,8,9,10,11,12],shape=[3,2])

print(y)

z=tf.matmul(x,y)

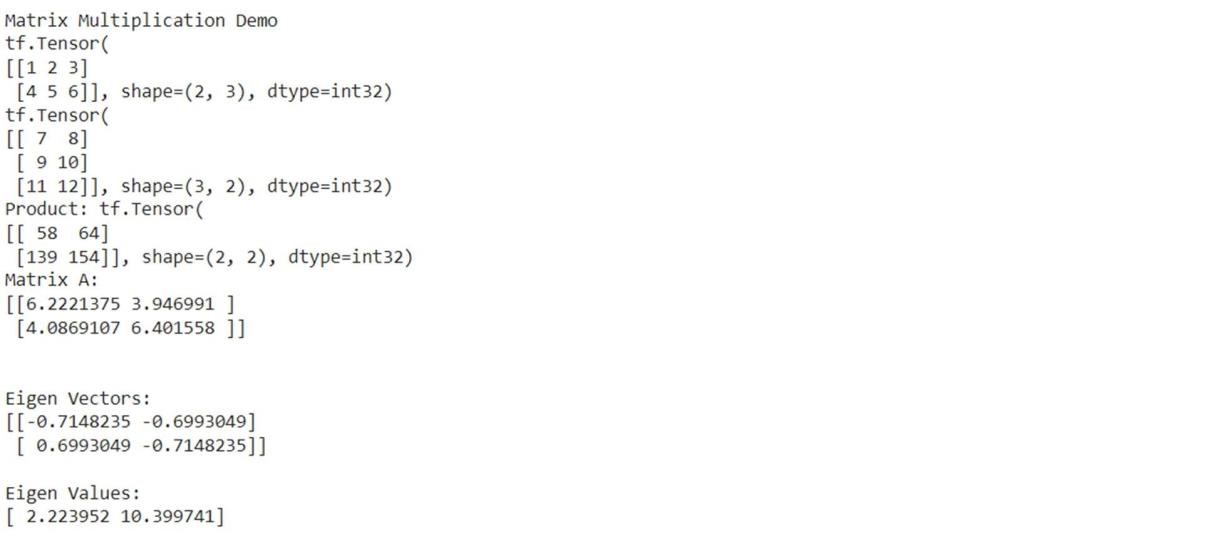
print("Product:",z)

e\_matrix\_A=tf.random.uniform([2,2],minval=3,maxval=10,dtype=tf.float32,name="matrixA")

print("Matrix A:\n{}\n\n".format(e\_matrix\_A))eigen\_values\_A,eigen\_vectors\_A=tf.linalg.eigh(e\_matrix\_A)

print("Eigen Vectors:\n{}\n\nEigen Values:\n{}\n".format(eigen\_vectors\_A,eigen\_values\_A))

OUTPUT:



Deep Learning 4047A008

Practical No: 2

Aim: Solving XOR problem using deep feed forward network.

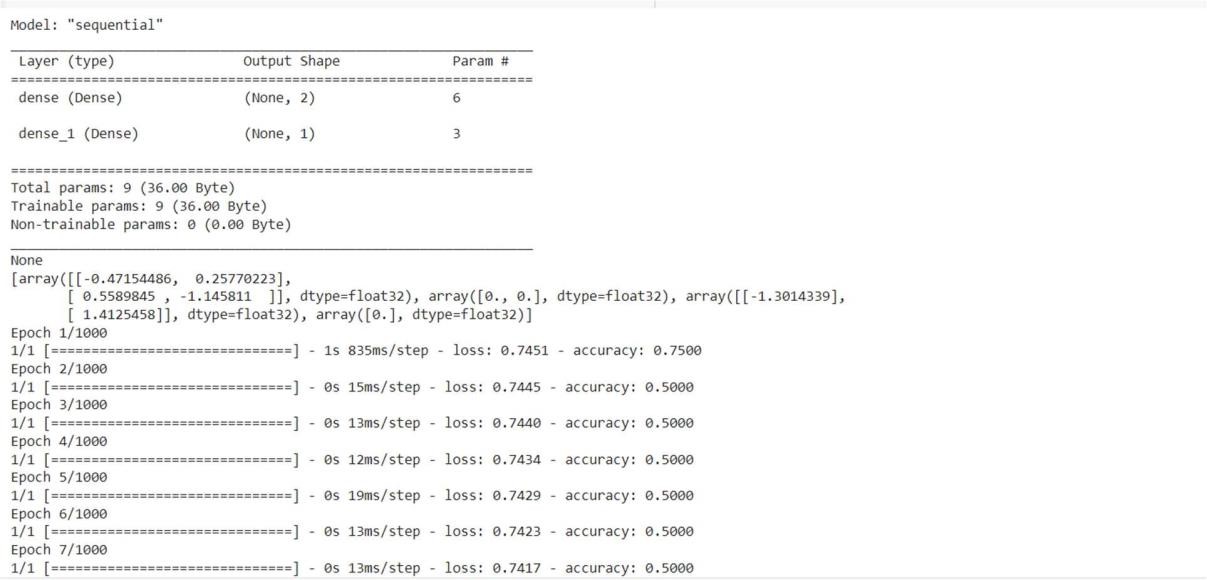
import numpy as np from keras.layers import Dense from keras.models import Sequential model=Sequential()

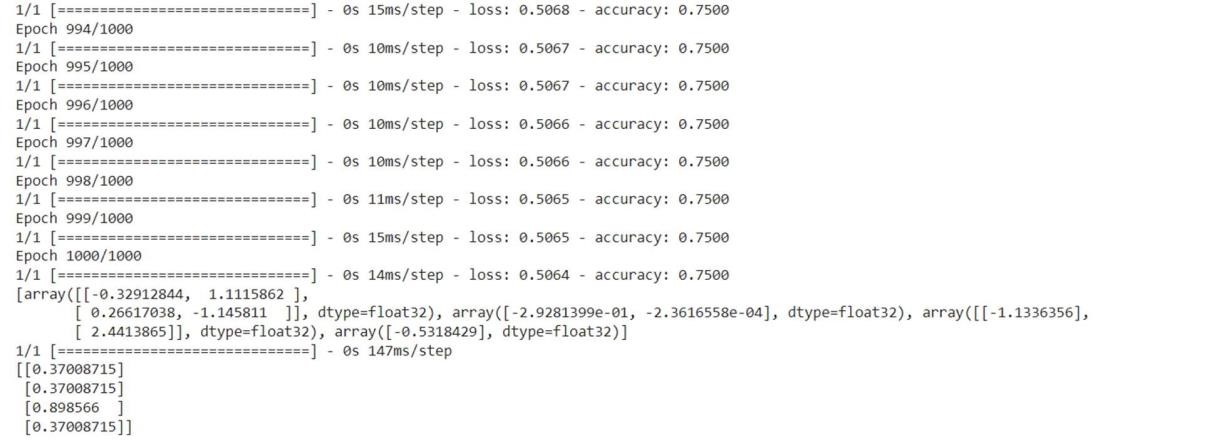
model.add(Dense(units=2,activation='relu',input\_dim=2)) model.add(Dense(units=1,activation='sigmoid'))

model.compile(loss='binary\_crossentropy',optimizer='adam',metrics=['accuracy']) print(model.summary()) print(model.get\_weights())

X=np.array([[0.,0.],[0.,1.],[1.,0.],[1.,1.]]) Y=np.array([0.,1.,1.,0.]) model.fit(X,Y,epochs=1000,batch\_size=4) print(model.get\_weights()) print(model.predict(X,batch\_size=4))

OUTPUT:





Deep Learning 4047A008

Practical No: 3

Aim: Implementing deep neural network for performing classification task. Problem statement: the given dataset comprises of health information about diabetic women patient. we need to create deep feed forward network that will classify women suffering from diabetes mellitus as 1. Code import numpy as np from keras.layers import Dense from keras.models import Sequential dataset =np.loadtxt('pima-indians-diabetes.csv',delimiter=',')

X=dataset[:,0:8]

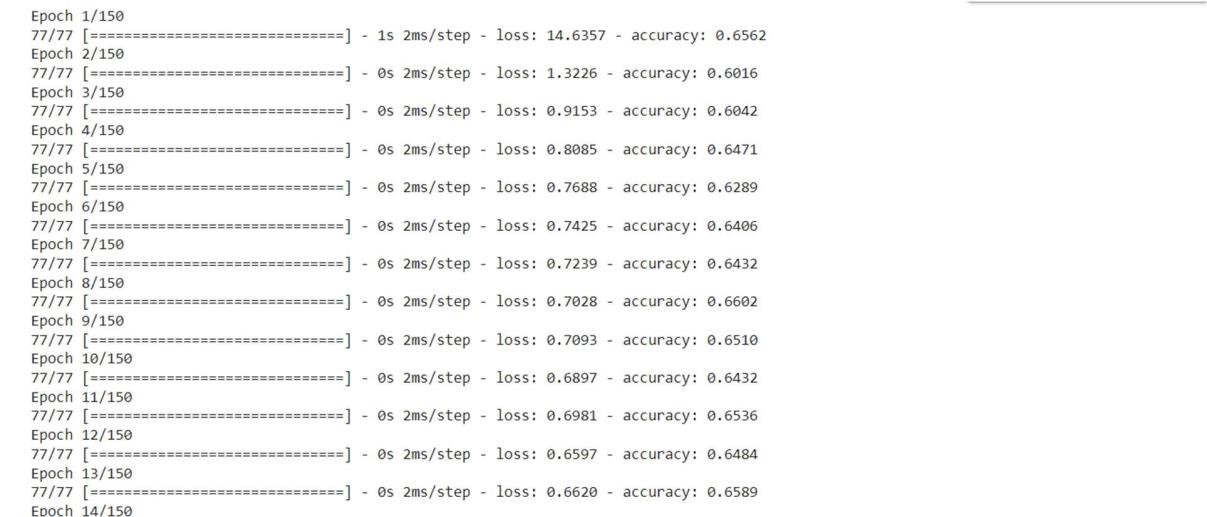
Y=dataset[:,8] ### Creating model:

model=Sequential() model.add(Dense(units=12,activation='relu',input\_dim=8 )) model.add(Dense(units=8,activation='relu')) model.add(Dense(1,activation='sigmoid'))

### Compiling and fitting model:

model.compile(loss='binary\_crossentropy',optimizer='adam',metrics=['accuracy' ]) model.fit(X,Y,epochs=150,batch\_size=10) \_,accuracy=model.evaluate(X,Y) print('Accuracy of model is',(accuracy\*100))

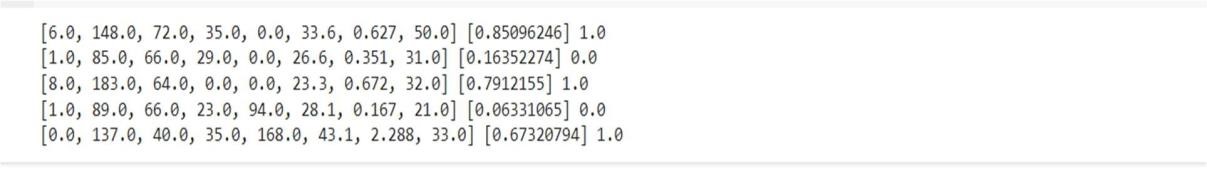
### Using model for prediction class: prediction=model.predict(X) for i in range(5):print(X[i].tolist(),prediction[i],Y[i])



Evaluating the accuracy:



Using model for prediction class:



Deep Learning 4047A008

Practical No: 4

Aim: Using deep feed forward network with two hidden layers for performing multiclass classification and predicting the class.

from keras.models import Sequential from keras.layers import Dense from sklearn.datasets import make\_blobs from sklearn.preprocessing import MinMaxScaler

X,Y=make\_blobs(n\_samples=100,centers=2,n\_features=2,random\_state=1

)

scalar=MinMaxScaler()

scalar.fit(X)

X=scalar.transform(X)

model=Sequential() model.add(Dense(4,input\_dim=2,activation='relu')) model.add(Dense(4,activation='relu')) model.add(Dense(1,activation='sigmoid'))

model.compile(loss='binary\_crossentropy',optimizer='adam') model.fit(X,Y,epochs=500)

Xnew,Yreal=make\_blobs(n\_samples=3,centers=2,n\_features=2,random\_state=1)

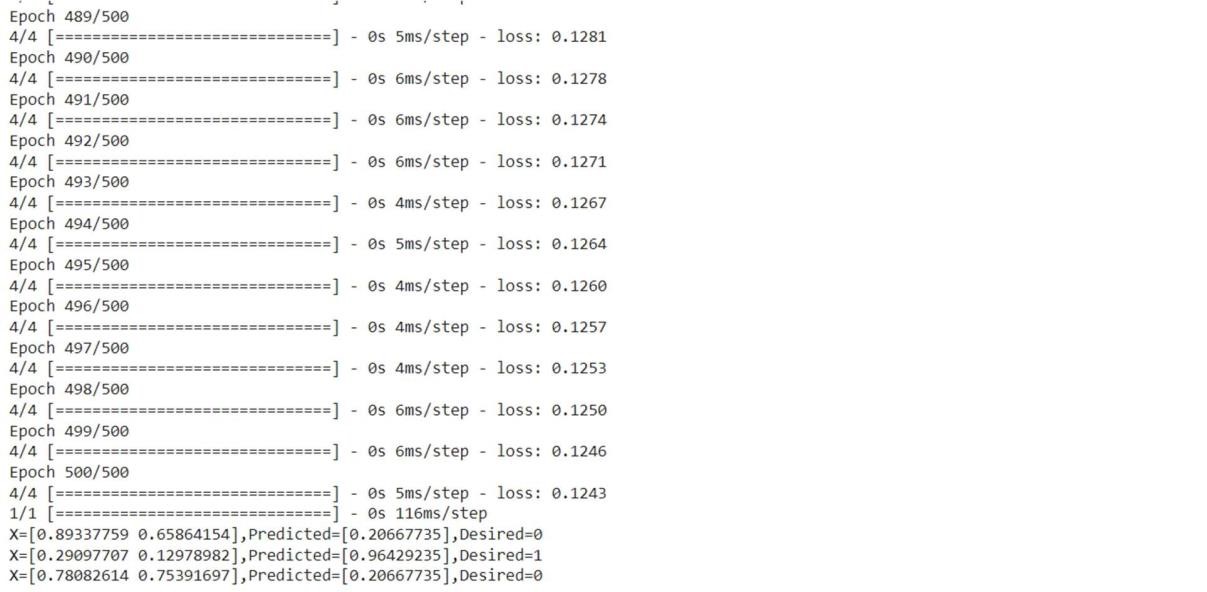
Xnew=scalar.transform(Xnew)

Ynew=model.predict(Xnew) for

i in range(len(Xnew)):

print("X=%s,Predicted=%s,Desired=%s"%(Xnew[i],Ynew[i],Yreal[i]))

OUTPUT:





Deep Learning 4047A008

Practical No: 5

Aim: Using a deep field forward network with two hidden layers for performing linear regression and predicting values.

from keras.models import Sequential from keras.layers import Dense from sklearn.datasets import make\_regression from sklearn.preprocessing import

MinMaxScaler

X,Y=make\_regression(n\_samples=100,n\_features=2,noise=0.1,random\_state= 1) scalarX,scalarY=MinMaxScaler(),MinMaxScaler() scalarX.fit(X)  scalarY.fit(Y.reshape(100,1))

X=scalarX.transform(X)

Y=scalarY.transform(Y.reshape(100,1)) model=Sequential() model.add(Dense(4,input\_dim=2,activation='relu')) model.add(Dense(4,activation='relu')) model.add(Dense(1,activation='sigmoid')) model.compile(loss='mse',optimizer='adam') model.fit(X,Y,epochs=1000,verbose=0)

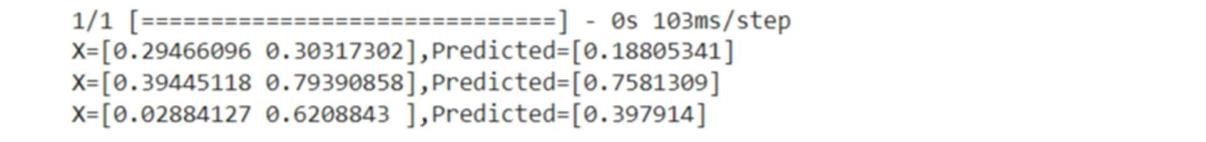
Xnew,a=make\_regression(n\_samples=3,n\_features=2,noise=0.1,random\_state=1)

Xnew=scalarX.transform(Xnew)

Ynew=model.predict(Xnew) for i in range(len(Xnew)):

print("X=%s,Predicted=%s"%(Xnew[i],Ynew[i]))

OUTPUT:



Practical No: 6

Aim: Evaluating feed forward deep network for multiclass Classification using KFold cross-validation.

#loading libraries import pandas from keras.models import Sequential from keras.layers import Dense from keras.wrappers.scikit\_learn import KerasClassifier from keras.utils import np\_utils from sklearn.model\_selection import cross\_val\_score from sklearn.model\_selection import KFold from sklearn.preprocessing import LabelEncoder

#loading dataset

df=pandas.read\_csv('Flower.csv',header=None) print(df)

#splitting dataset into input and output variables

X = df.iloc[:,0:4].astype(float) y=df.iloc[:,4]

#print(X)

#print(y)

#encoding string output into numeric output encoder=LabelEncoder() encoder.fit(y) encoded\_y=encoder.transform(y) print(encoded\_y) dummy\_Y=np\_utils.to\_categorical(encoded\_y

) print(dummy\_Y) def baseline\_model():

# create model model = Sequential() model.add(Dense(8, input\_dim=4, activation='relu')) model.add(Dense(3, activation='softmax'))

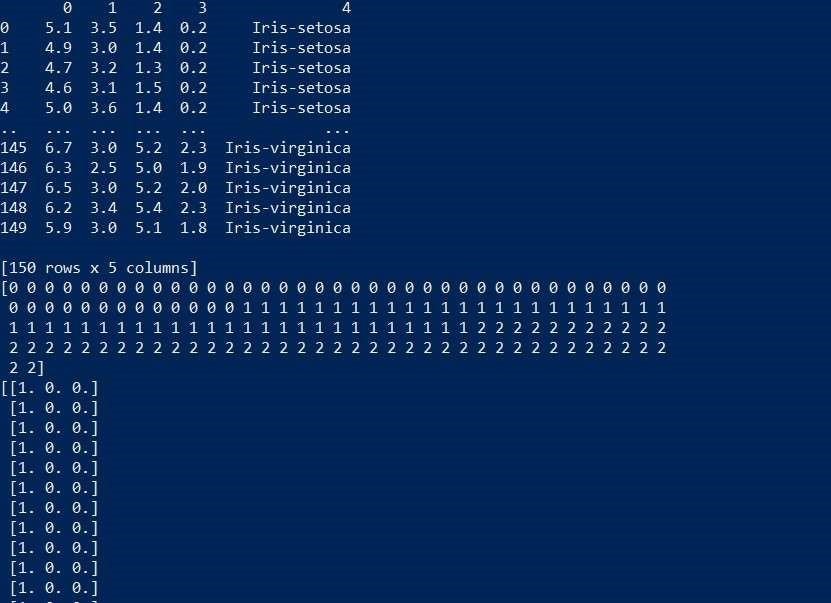
# Compile model

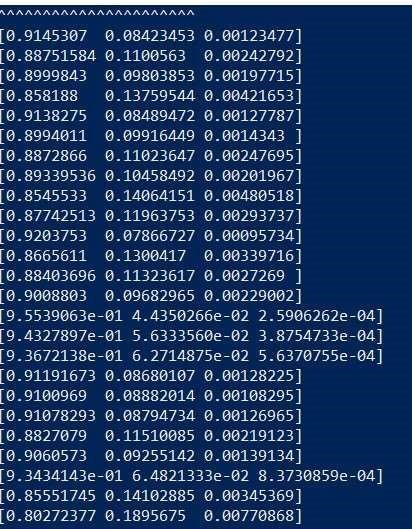
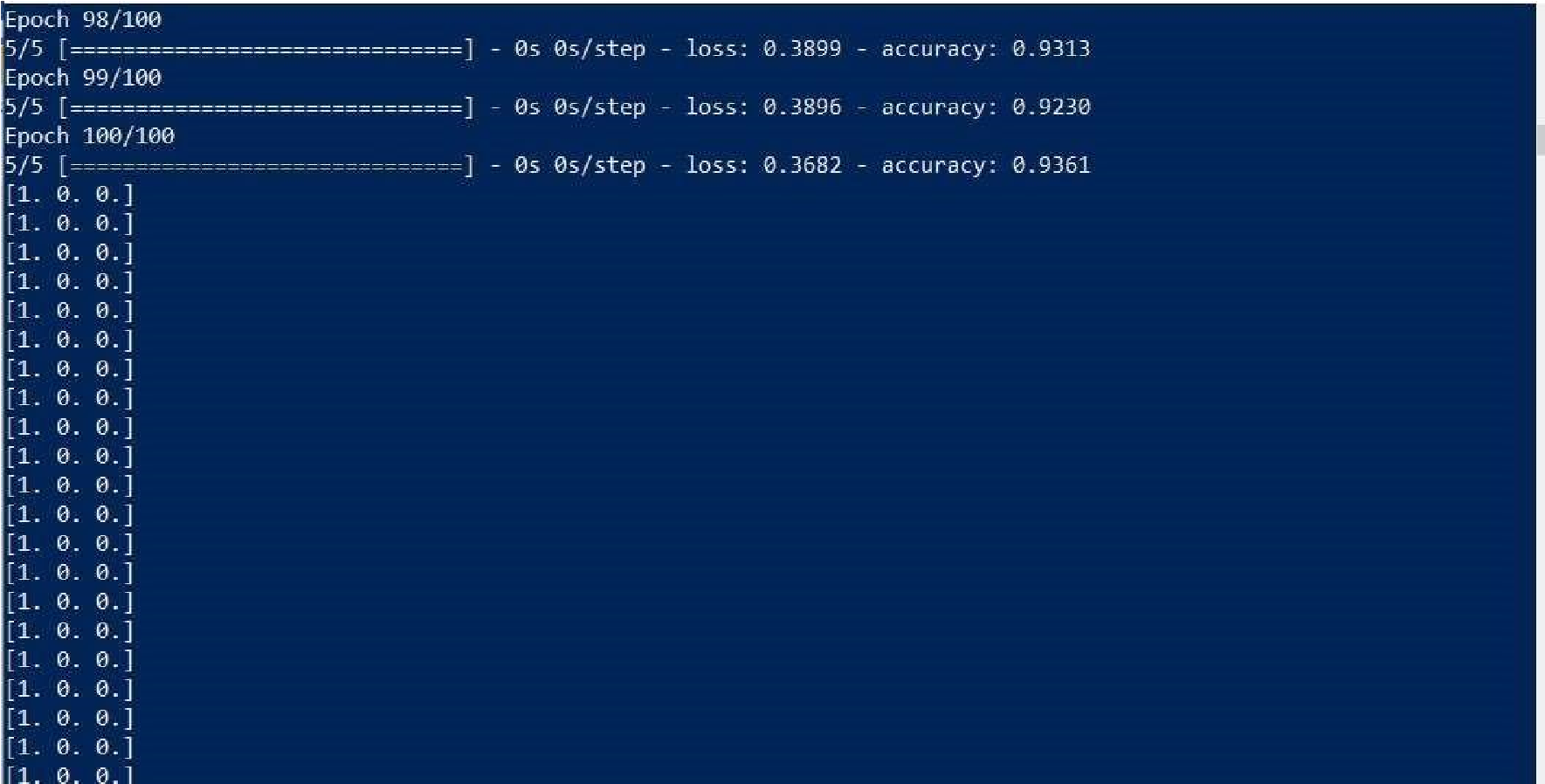
model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy']) return model estimator=baseline\_model() estimator.fit(X,dummy\_Y,epochs=100,shuffle=True

) action=estimator.predict(X) for i in range(25):

print(dummy\_Y[i]) print('^^^^^^^^^^^^^^^^^^^^^^') for i in range(25): print(action[i])

OUTPUT:





Code 2: import pandas from keras.models import Sequential from keras.layers import Dense from keras.wrappers.scikit\_learn import KerasClassifier from keras.utils import np\_utils from sklearn.model\_selection import cross\_val\_score from sklearn.model\_selection import KFold from sklearn.preprocessing import LabelEncoder dataset=pandas.read\_csv("Flower.csv",header=None) dataset1=dataset.values

X=dataset1[:,0:4].astype(float) Y=dataset1[:,4] print(Y) encoder=LabelEncoder() encoder.fit(Y) encoder\_Y=encoder.transform(Y) print(encoder\_Y) dummy\_Y=np\_utils.to\_categorical(encoder\_Y

) print(dummy\_Y) def baseline\_model():

model=Sequential() model.add(Dense(8,input\_dim=4,activation='relu')) model.add(Dense(3,activation='softmax'))

model.compile(loss='categorical\_crossentropy',optimizer='adam',metrics=['accuracy']) return model estimator=KerasClassifier(build\_fn=baseline\_model,epochs=100,batch\_size=

5) kfold = KFold(n\_splits=10, shuffle=True) results = cross\_val\_score(estimator, X, dummy\_Y, cv=kfold) print("Baseline: %.2f%% (%.2f%%)" % (results.mean()\*100, results.std()\*100))



(Changing neuron) model.add(Dense(10,input\_dim=4,activation='relu'))



## Practical No : 7

Aim: implementing regularization to avoid overfitting in binary classification.

from matplotlib import pyplot from sklearn.datasets import make\_moons from keras.models import Sequential from keras.layers import Dense

X,Y=make\_moons(n\_samples=100,noise=0.2,random\_state=1

) n\_train=30 trainX,testX=X[:n\_train,:],X[n\_train:] trainY,testY=Y[:n\_train],Y[n\_train:]

#print(trainX)

#print(trainY) #print(testX) #print(testY) model=Sequential() model.add(Dense(500,input\_dim=2,activation='relu')

)

model.add(Dense(1,activation='sigmoid')) model.compile(loss='binary\_crossentropy',optimizer='adam',metrics=['accuracy' ]) history=model.fit(trainX,trainY,validation\_data=(testX,testY),epochs=4000) pyplot.plot(history.history['accuracy'],label='train') pyplot.plot(history.history['val\_accuracy'],label='test') pyplot.legend() pyplot.show()

OUTPUT:





The above code and resultant graph demonstrate overfitting with accuracy of testing data less than accuracy of training data also the accuracy of testing data increases once and then start decreases gradually.to solve this problem we can use regularization

Hence, we will add two lines in the above code as highlighted below to implement l2 regularization with alpha=0.001

from matplotlib import pyplot from sklearn.datasets import make\_moons from keras.models import Sequential from keras.layers import Dense from keras.regularizers import l2

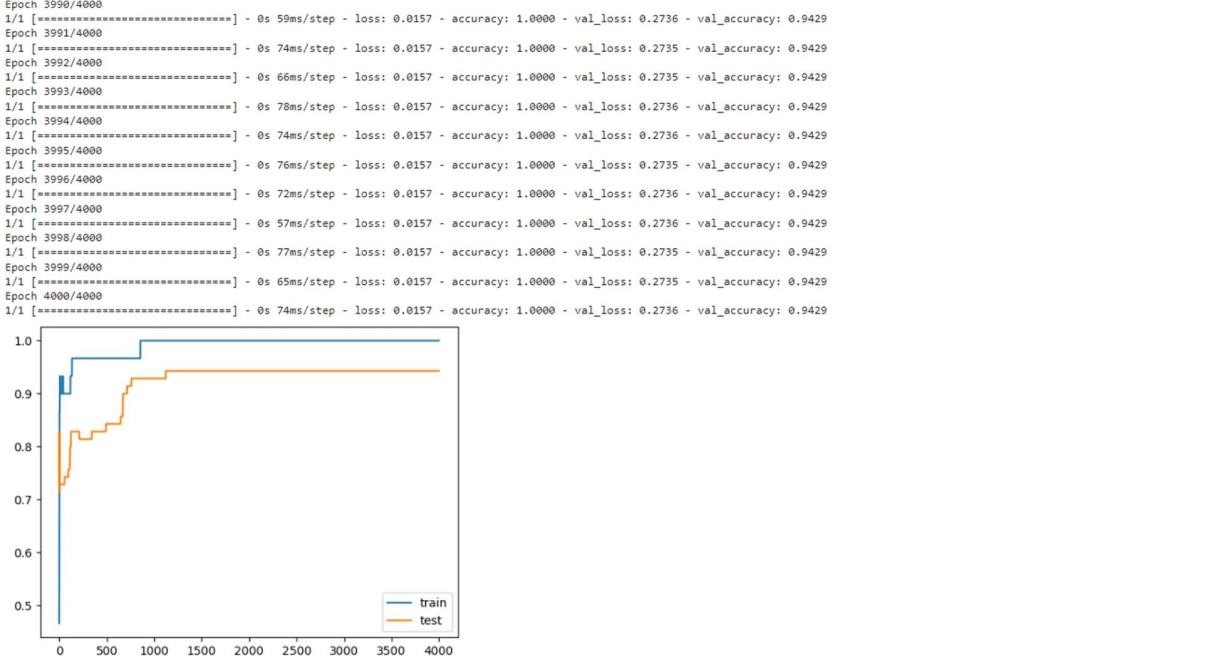
X,Y=make\_moons(n\_samples=100,noise=0.2,random\_state=1) n\_train=30 trainX,testX=X[:n\_train,:],X[n\_train:] trainY,testY=Y[:n\_train],Y[n\_train:]

#print(trainX)

#print(trainY)

#print(testX) #print(testY) model=Sequential() model.add(Dense(500,input\_dim=2,activation='relu',kernel\_regularizer=l2(0.001))) model.add(Dense(1,activation='sigmoid')) model.compile(loss='binary\_crossentropy',optimizer='adam',metrics=['accuracy']) history=model.fit(trainX,trainY,validation\_data=(testX,testY),epochs=4000) pyplot.plot(history.history['accuracy'],label='train') pyplot.plot(history.history['val\_accuracy'],label='test') pyplot.legend()

pyplot.show()



By replacing l2 regularizer with l1 regularizer at the same learning rate 0.001 we get the following output.



By applying l1 and l2 regularizer we can observe the following changes in accuracy of both trainig and testing data. The changes in code are also highlighted.

from matplotlib import pyplot from sklearn.datasets import make\_moons from keras.models import Sequential from keras.layers import Dense from keras.regularizers import l1\_l2

X,Y=make\_moons(n\_samples=100,noise=0.2,random\_state=1) n\_train=30 trainX,testX=X[:n\_train,:],X[n\_train:] trainY,testY=Y[:n\_train],Y[n\_train:]

#print(trainX)

#print(trainY)

#print(testX) #print(testY) model=Sequential() model.add(Dense(500,input\_dim=2,activation='relu',kernel\_regularizer=l1\_l2(l1=0.001,l2=0.001))

)

model.add(Dense(1,activation='sigmoid')) model.compile(loss='binary\_crossentropy',optimizer='adam',metrics=['accuracy' ]) history=model.fit(trainX,trainY,validation\_data=(testX,testY),epochs=4000) pyplot.plot(history.history['accuracy'],label='train') pyplot.plot(history.history['val\_accuracy'],label='test') pyplot.legend() pyplot.show()

OUTPUT:

