**Week-1**

!pip install nltk

import nltk

from nltk.tokenize import word\_tokenize

from nltk.probability import FreqDist

from nltk.corpus import stopwords

nltk.download('punkt')

nltk.download('stopwords')

nltk.download('punkt\_tab')

def word\_analysis(text):

  words = word\_tokenize(text)

  words = [word.lower() for word in words]

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

  words = [word for word in words if word.isalnum() and word not in stop\_words]

  freq\_dist = FreqDist(words)

  print("Total words:",len(words))

  print("Unique words:",len(freq\_dist))

  print("Most common words:")

  print(freq\_dist.most\_common(10))

if \_\_name\_\_ == "\_\_main\_\_":

  text = "Natural Language Procesing (NLP) is a field of artificial intelligence that focuses on the interaction between computers and humans using natural lanuage. NLP enables computers to understand, interpret , and generate human -like text."

  word\_analysis(text)

**week-2**

import nltk

from nltk.tokenize import word\_tokenize

from nltk.corpus import stopwords

from nltk.probability import FreqDist

import random

nltk.download('punkt')

nltk.download('stopwords')

nltk.download('punkt\_tab')

def generate\_words(text, num\_words=10):

  words = word\_tokenize(text.lower())

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

  words = [word for word in words if word.isalnum() and word not in stop\_words]

  freq\_dist = FreqDist(words)

  generated\_words = []

  for \_ in range(num\_words):

    generated\_words.append(random.choice(list(freq\_dist.keys())))

  return generated\_words

if \_\_name\_\_ == "\_\_main\_\_":

  text = "Natural Language Processing (NLP) is a field of artificial intelligence that focuses on the interaction between computers and humans using natural language. NLP enables computers to understand, interpret, and generate human-like text."

  generated\_words = generate\_words(text, num\_words=5)

  print("Generated Words:", generated\_words)

**week-3**

import nltk

from nltk.tokenize import word\_tokenize

from nltk import pos\_tag

from nltk.corpus import wordnet

from nltk.stem import WordNetLemmatizer

from nltk.corpus import stopwords

nltk.download('punkt')

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

nltk.download('wordnet')

nltk.download('punkt\_tab')

nltk.download('stopwords')

def morphological\_analysis(text):

  words = word\_tokenize(text)

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

  words = [word for word in words if word.isalnum() and word not in stop\_words]

  pos\_tags = pos\_tag(words)

  lemmatizer = WordNetLemmatizer()

  lemmatized\_words = [lemmatizer.lemmatize(word, get\_wordnet\_pos(pos)) for word, pos in pos\_tags]

  print("Original words:",words)

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

def get\_wordnet\_pos(treebank\_tag):

   if treebank\_tag.startswith('J'):

     return 'a'

   elif treebank\_tag.startswith('V'):

     return 'v'

   elif treebank\_tag.startswith('N'):

     return 'n'

   elif treebank\_tag.startswith('R'):

     return 'r'

   else:

     return 'n'

if \_\_name\_\_=="\_\_main\_\_":

  text = input('enter the text:')

  morphological\_analysis(text)

  for word, pos in pos\_tag(word\_tokenize(text)):

    print(f"Word: {word}, POS Tag: {pos}, WordNet POS Tag: {get\_wordnet\_pos(pos)}")

**week-4**

!pip install nltk

import nltk

from nltk import ngrams

from collections import Counter

import re

def clean\_text(text):

  cleaned\_text=re.sub(r'[^a-zA-Z0-9\s]','',text).lower()

  return cleaned\_text

def ngram\_analysis(text,n):

  cleaned\_text=clean\_text(text)

  words=cleaned\_text.split()

  ngrams\_list=list(ngrams(words,n))

  ngrams\_count=Counter(ngrams\_list)

  return ngrams\_count

if \_\_name\_\_=="\_\_main\_\_":

  text=input("Enter the text")

  n=2

  result=ngram\_analysis(text,n)

  print(f"{n}-Gram Analysis:")

  for ngram,count in result.items():

    print(f"{ngram}:{count}times")

**week-5**

import nltk

from nltk import ngrams

from collections import Counter

import re

def preprocess\_text(text):

  text=re.sub(r'[^\w\s]'," ",text)

  text=text.lower()

  return text

def generate\_bigrams(tokens):

  return list(zip(tokens,tokens[1:]))

def calculate\_bigram\_probabilities(corpus):

  bigrams=generate\_bigrams(corpus)

  bigram\_counts=Counter(bigrams)

  vocabulary\_size=len(set(corpus))

  bigram\_probabilities={}

  for bigram in bigram\_counts:

              bigram\_probabilities[bigram]=(bigram\_counts[bigram]+1)//(corpus.count(bigram[0])+vocabulary\_size)

              return bigram\_probabilities

def bigram\_smoothing(text):

  preprocessed\_text=preprocess\_text(text)

  tokens=preprocessed\_text.split()

  bigram\_probabilities=calculate\_bigram\_probabilities(tokens)

  print("Bigram Probabilities:")

  for bigram,probability in bigram\_probabilities.items():

    print(f"{bigram}:{probability:.4f}")

if \_\_name\_\_=="\_\_main\_\_":

              text=input("Enter the text")

              bigram\_smoothing(text)

              preprocess\_text(text)

**week-6**

import numpy as np

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding, Dense, Reshape

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import skipgrams

corpus=["I am learning Advanced Datastructure"]

tokenizer=Tokenizer()

tokenizer.fit\_on\_texts(corpus)

total\_words=len(tokenizer.word\_index)+1

vocabulary\_size=total\_words

skip\_grams=[skipgrams(sequence,vocabulary\_size,window\_size=5) for sequence in tokenizer.texts\_to\_sequences(corpus)]

pairs,labels=skip\_grams[0][0],skip\_grams[0][1]

embedding\_dim=100

model=Sequential()

model.add(Embedding(input\_dim=total\_words,output\_dim=embedding\_dim,input\_length=1))

model.add(Reshape((embedding\_dim,)))

model.add(Dense(units=total\_words,activation='softmax'))

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

from tensorflow.keras.utils import to\_categorical

labels=to\_categorical(labels,num\_classes=total\_words)

model.fit(np.array(pairs)[:,0],labels, epochs=10,batch\_size=32)

word\_embeddings=model.get\_layer(index=0).get\_weights()[0]

for word,token in tokenizer.word\_index.items():

  print(f"{word}:{word\_embeddings[token]}")

**week-7**

import nltk

from nltk.corpus import treebank

from nltk.tag import hmm

from nltk.corpus import conll2000

# Download required NLTK resources

nltk.download('treebank')

nltk.download('conll2000')

# Example: Train a HMM POS tagger on the Treebank dataset

train\_sents = treebank.tagged\_sents()  # Sentences from the treebank corpus

test\_sents = treebank.tagged\_sents()[3000:]  # Test set, can adjust size

# Train a Hidden Markov Model tagger

trainer = hmm.HiddenMarkovModelTrainer()

tagger = trainer.train(train\_sents)

# Now, let's test the tagger

print("Accuracy on test set: ", tagger.evaluate(test\_sents))

# Example sentence for tagging

sentence = "This is a text sentence".split()

# Use the trained model to tag this sentence

tags = tagger.tag(sentence)

print("Tagged sentence: ", tags)

**week-8**

def viterbi(obs, states, start\_p, trans\_p, emit\_p):

    V = [{}]

    path = {}

    for state in states:

        V[0][state] = start\_p[state] \* emit\_p[state].get(obs[0], 0)

        path[state] = [state]

    for t in range(1, len(obs)):

        V.append({})

        new\_path = {}

        for curr\_state in states:

            (prob, prev\_state) = max(

                (V[t-1][y0] \* trans\_p[y0].get(curr\_state, 0) \* emit\_p[curr\_state].get(obs[t], 0), y0)

                for y0 in states

            )

            V[t][curr\_state] = prob

            new\_path[curr\_state] = path[prev\_state] + [curr\_state]

        path = new\_path

    n = len(obs) - 1

    (prob, state) = max((V[n][y], y) for y in states)

    return (prob, path[state])

**week-9**

import spacy

def pos\_tagger\_spacy(text):

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

doc = nlp(text)

tagged\_words = [(token.text,token.pos\_) for token in doc]

return tagged\_words

if \_\_name\_\_ == "\_\_main\_\_":

text = input("Enter the text")

tagged\_result = pos\_tagger\_spacy(text)

print("Input Text",text)

print("\n POS Tagged :")

for word, pos in tagged\_result:

print(f"{word}: {pos}")

**Week-10**

import spacy

# Load the spaCy model

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

# Sample text

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

# Step 1: Processing the text with spaCy

doc = nlp(text)

# Step 2: Chunking Up (extracting noun chunks)

print("Chunking Up: Extracting Noun Phrases")

for chunk in doc.noun\_chunks:

    print(f"Noun Phrase: {chunk.text}")

# Step 3: Chunking Down (breaking noun phrases into components)

print("\nChunking Down: Breaking down noun chunks into components")

for chunk in doc.noun\_chunks:

    print(f"\nNoun Phrase: {chunk.text}")

    for token in chunk:

        print(f"  Word: {token.text}, POS Tag: {token.pos\_}")

**week-11**

import nltk

from nltk import RegexpParser

from nltk.tokenize import word\_tokenize

from nltk.tag import pos\_tag

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

nltk.download('punkt')

nltk.download('punkt\_tab')

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

words = word\_tokenize(text)

tagged\_words = pos\_tag(words)

chunk\_grammar = r"""

  NP: {<DT>?<JJ>\*<NN>}

  PP: {<IN><NP>}

  VP: {<VB.\*><NP|PP>+$}

  CLAUSE: {<NP><VP>}

  """

chunk\_parser = RegexpParser(chunk\_grammar)

chunks = chunk\_parser.parse(tagged\_words)

print(chunks)