INDEX

VASAVI COLLEGE OF ENGINEERING (AUTONOMOUS)
IBRAHIMBAGH, HYDERABAD – 500031

Institute Vision

Striving for a symbiosis of technological excellence and human values.

Institute Mission

To arm young brains with competitive technology and nurture holistic development of the individuals for a better tomorrow.

VASAVI COLLEGE OF ENGINEERING (AUTONOMOUS)
IBRAHIMBAGH, HYDERABAD – 500031

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Department Vision

To be a center for academic excellence in the field of Computer Science and Engineering education to enable graduates to be ethical and competent professionals.

Department Mission

To enable students to develop logic and problem solving approach that will help build their careers in the innovative field of computing and provide creative solutions for the benefit of society.

VASAVI COLLEGE OF ENGINEERING (AUTONOMOUS)
IBRAHIMBAGH, HYDERABAD – 500031

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

PEO's, PO's and PSO's

PROGRAM EDUCATIONAL OBJECTIVES (PEOs):

Graduates should be able to utilize the knowledge gained from their academic program to:

PEO 1: Solve problems in a modern technological society as valuable and productive engineers.

PEO 2: Function and communicate effectively, both individually and within multidisciplinary teams.

PEO 3: Be sensitive to the consequences of their work, both ethically and professionally, for productive professional careers.

PEO 4: Continue the process of life-long learning.

PROGRAM OUTCOMES (POs):

Engineering Graduates will be able to:

PO1: Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2: Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3: Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate

consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4: Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6: The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7: Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

P10: Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

P11: Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

P12: Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES (PSOs):

PSO1: Graduates will have knowledge of programming and designing to develop solutions for engineering problems.

PSO2: Graduates will be able to demonstrate an understanding of system architecture, information management and networking.

PSO3: Graduates will possess knowledge of applied areas of computer science and engineering and execute them appropriately.

VASAVI COLLEGE OF ENGINEERING (Autonomous)
IBRAHIMBAGH, HYDERABAD – 500 031

Department of Computer Science & Engineering

UI21PE751CS Natural Language Processing Lab

B.E VII SEMESTER

Course Outcomes mapping with Blooms Taxonomy

S.NO	Course Outcomes	BTL
1	Implement the word tokenization for NLP pre-processing	3
2	Implement the N-gram and Neural language models	3
3	Implement parsing algorithms for grammar checking	3
4	Implement solution for sentiment analysis	3
5	Implement the NLP task namely document classification	3

VASAVI COLLEGE OF ENGINEERING (Autonomous)
IBRAHIMBAGH, HYDERABAD – 500 031

Department of Computer Science & Engineering

UI21PE751CS Natural Language Processing Lab

B.E VII SEMESTER

CO- PO Mapping

Course Code	Course Outcomes	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	Implement the word tokenization for NLP pre-processing				3	3							
CO2	Implement the N-gram and Neural language models				3	3							
CO3	Implement parsing algorithms for grammar checking				3	3							
CO4	Implement solution for sentiment analysis				3	3							
CO5	Implement the NLP task namely document classification				3	3							
	Average				3	3							

VASAVI COLLEGE OF ENGINEERING (Autonomous)
IBRAHIMBAGH, HYDERABAD – 500 031

Department of Computer Science & Engineering

UI21PE751CS Natural Language Processing Lab

B.E VII SEMESTER

CO-PSO Mapping

Course Code	**Course Outcomes**	**PSO1**	**PSO2**	**PSO3**
CO1	Implement the word tokenization for NLP pre-processing			3
CO2	Implement the N-gram and Neural language models			3
CO3	Implement parsing algorithms for grammar checking			3
CO4	Implement solution for sentiment analysis			3
CO5	Implement the NLP task namely document classification			3
	Average			3

VASAVI COLLEGE OF ENGINEERING (Autonomous)
IBRAHIMBAGH, HYDERABAD – 500 031

Department of Computer Science & Engineering

UI21PE751CS Natural Language Processing Lab

B.E VII SEMESTER

CO-PO Mapping Justification

UI21PE751CS	**CO1**	**PO4**	**3**	PO4 expects the students to have research-based knowledge and use research methods for analysing and interpreting data and to provide valid conclusions.
		PO5	**3**	PO5 expects the students to be able to predict and model complex engineering activities with an understanding of the limitations
	CO2	**PO4**	**3**	PO4 expects the students to have research-based knowledge and use research methods for analysing and interpreting data and to provide valid conclusions.
		PO5	**3**	PO5 expects the students to be able to predict and model complex engineering activities with an understanding of the limitations
	CO3	**PO4**	**3**	PO4 expects the students to have research-based knowledge and use research methods for analysing and interpreting data and to provide valid conclusions.
		PO5	**3**	PO5 expects the students to be able to predict and model complex engineering activities with an understanding of the limitations
	CO4	**PO4**	**3**	PO4 expects the students to have research-based knowledge and use research methods for analysing and interpreting data and to provide valid conclusions.
		PO5	**3**	PO5 expects the students to be able to predict and model complex engineering activities with an understanding of the limitations
	CO5	**PO4**	**3**	PO4 expects the students to have research-based knowledge and use research methods for analysing and interpreting data and to provide valid conclusions.
		PO5	**3**	PO5 expects the students to be able to predict and model complex engineering activities with an understanding of the limitations

CO-PSO Mapping Justification

UI21PE751CS	**CO1**	**PSO3**	**3**	PSO3 expects the students to be able to gain the knowledge of computer science and engineering in the areas of Data Analytics and apply them.
	CO2	**PSO1**	**3**	PSO3 expects the students to be able to gain the knowledge of computer science and engineering in the areas of Data Analytics and apply them.
	CO3	**PSO1**	**3**	PSO3 expects the students to be able to gain the knowledge of computer science and engineering in the areas of Data Analytics and apply them.
	CO4	**PSO1**	**3**	PSO3 expects the students to be able to gain the knowledge of computer science and engineering in the areas of Data Analytics and apply them.
	CO5	**PSO1**	**3**	PSO3 expects the students to be able to gain the knowledge of computer science and engineering in the areas of Data Analytics and apply them.

VASAVI COLLEGE OF ENGINEERING (Autonomous)
IBRAHIMBAGH, HYDERABAD – 500 031

Department of Computer Science & Engineering

UI21PE751CS NATURAL LANGUAGE PROCESSING LAB

SYLLABUS FOR B.E VII SEMESTER

Course Objectives	Course Outcomes
Students should be able to	At the end of the course, students will be able to
1. Apply dynamic programming design strategy for NLP tasks . 2. Design Feedforward and Recurrent Neural Networks for performing Language Modelling	1. Implement the word tokenization for NLP pre-processing 2. Implement the N-gram and Neural language models. 3. Implement parsing algorithms for grammar checking 4. Implement solution for sentiment analysis 5. Implement the NLP task namely document classification

1. Implementation of Tokenization using UNIX commands.
2. Implement a word tokenization using regular expressions.
3. Implement Minimum Edit Distance(MED) algorithm for spelling correction.
4. Implement n-gram language model.
5. Implement Naïve Bayes classification for sentiment analysis.
6. Implement POS tagging using HMM.
7. Implementation of Implement CKY parsing algorithm.
8. Implement PCKY parsing algorithm.
9. Implementation of Computing cosine similarity between the words using term document matrix and term-term matrix.
10. Implementation of TF-IDF matrix for the given document set.
11. Implementation of Language Model Using Feed forward Neural Network.
12. Implement Language Model Using RNN.
13. Perform Text Analytics

VASAVI COLLEGE OF ENGINEERING (Autonomous)
IBRAHIMBAGH, HYDERABAD – 500 031

Department of Computer Science & Engineering

UI21PE751CS Natural Language Processing Lab

B.E VII SEMESTER

Experiments to CO, PO, PSO Mapping

S. No.	Name of the Experiment	CO	PO	PSO
1	Implementation of Tokenization using UNIX commands	1	4,5	3
2	Implement a word tokenization using regular expressions	1	4,5	3
3	Implement Minimum Edit Distance(MED) algorithm for spelling correction	1	4,5	3
4	Implement n-gram language model	1	4,5	3
5	Implement Naïve Bayes classification for sentiment analysis	3	4,5	3
6	Implement POS tagging using HMM	2	4,5	3
7	Implementation of Implement CKY parsing algorithm	5	4,5	3
8	Implement PCKY parsing algorithm	5	4,5	3
9	Implementation of Computing cosine similarity between the words using term document matrix and term-term matrix	2	4,5	3
10	Implementation of TF-IDF matrix for the given document set	2	4,5	3
11	Implementation of Language Model Using Feed forward Neural Network	4	4,5	3
12	Implement Language Model Using RNN	4	4,5	3

13	**Implement Perform Text Analytics**	**3**	**4,5**	**3**

VASAVI COLLEGE OF ENGINEERING (Autonomous)
IBRAHIMBAGH, HYDERABAD – 500 031

Department of Computer Science & Engineering

UI21PE751CS Natural Language Processing Lab

B.E VII SEMESTER

Introduction to the lab

Natural Language Processing (NLP) is a field of artificial intelligence concerned with the interaction between computers and human language. It involves the development of algorithms and models that enable computers to understand, interpret, and generate human language in a way that is meaningful. NLP encompasses a wide range of tasks and applications, including:

1. **Text Classification and Categorization**: Assigning labels or categories to text based on its content (e.g., sentiment analysis, topic classification).

2. **Named Entity Recognition (NER)**: Identifying and categorizing entities (such as names of people, organizations, locations) in text.

3. **Text Generation**: Creating coherent and contextually relevant text based on given input (e.g., chatbots, language translation).

4. **Machine Translation**: Translating text from one language to another automatically (e.g., Google Translate).

5. **Sentiment Analysis**: Determining the sentiment expressed in a piece of text (positive, negative, neutral).

6. **Speech Recognition**: Converting spoken language into text (e.g., virtual assistants like Siri or Alexa).

7. **Information Retrieval**: Finding relevant information in a large collection of text (e.g., search engines).

EXPERIMENT-1 : Implementation of Tokenization using UNIX commands

Certainly! Word tokenization is a fundamental task in Natural Language Processing (NLP) that involves breaking down a text into individual words or tokens. We can create a simple program using Unix utilities like sed, tr, grep, and awk to perform basic word tokenization. Here's a step-by-step example of how you can achieve

1. Create a text file text1.txt containing text of your choiceAns:echo "Thisisanexampletextfile.">text1.txt

2. Use tr utility to replace every occurrence of a given character to another given characterAns:tr'o"x'<example.txt(HelloWorld!) >HellxWxrld!

3. What are s and c options of tr command?Ans:echo "aaabbbccc" |tr-s'a'Output:abbbc

-s option (squeeze-repeats): This option is used to squeeze (reduce) repeated characters in the input to a singlecharacter.

-coption(complement): Thisoptionisusedtocomplement thesetofcharacters.Whenused,itreplacescharactersnotlistedinthefirstsetwiththe correspondingcharacterfromthesecond set

echo"abc123" |tr -c'a-z"X'-->Output:XXX123

4. Whatistheoutputofthefollowingcommand? tra-zA-Z<text1.txt Ans:Thecommandtra-zA-Z<text1.txt willtransformalllowercaseletterstouppercaseinthecontentofthefiletext1.txt.The outputwillbe displayed onthe terminal.

5. Create a text file text2.txt containing any text (one word per line) sort the words in text2 file using “sort”command Ans:echo-e"banana\napple\norange\nkiwi\ngrape">text2.txtsorttext2.txt

6. Sort and display unique lines in the text2.txt fileAns:sort -utext2.txt

Output:

apple

banana

grape

kiwi

orange

7. Sort and display unique lines in text2.txt file such that each word is preceded with frequency count of the word in text2.txt file

Ans: sort text2.txt | uniq -c

Output:

1apple

2banana

1 grape

3 kiwi

1 orange

8. Obtain and display the tokens in text1.txt file

Ans: tr -sc 'A-Za-z' '\n' < text1.txt | grep -v '^$' (input: This is an example text file.)

Output:

This is an

example text

file

9. Display the tokens in sorted order

Ans: tr -sc 'A-Za-z' '\n' < text1.txt | grep -v '^$' | sort

Output:

This an

example file

is text

10.Display the unique tokens in sorted order

Ans: tr -sc 'A-Za-z' '\n' < text1.txt | grep -v '^$' | sort | uniq

Output:

This an

example file

is text

EXPERIMENT-2 : Implement a word tokenization using regular expressions

To tokenize words using the regular expression tokenizer provided by NLTK (Natural Language Toolkit) in Python, you'll first need to ensure NLTK is installed (pip install nltk). Then, you can use the RegexpTokenizer class from NLTK to define and apply custom tokenization rules based on regular expressions. Here's how you can create a Python program to tokenize words using NLTK's regular expression tokenizer

```
fromnltk.tokenizeimportRegexpTokenizer

s = "Good muffins cost $3.88\nin New York.Please buy me\ntwoof

them.\n\nThanks."tokenizer =RegexpTokenizer(r'\w+|\$[\d\.]+|\S+')

print(tokenizer.tokenize(s))
```

Output:

['Good','muffins','cost','$3.88', 'in','New','York','.','Please','buy','me','two','of','them','.','Thanks','.']

```
import nltknltk.download('punkt')
nltk.download('averaged_perceptron_t

agger')nltk.download('maxent_ne_chu

nker')nltk.download('words')

sentence="""Ateighto'clockonThursdaymorning...Arthurdidn't

feelverygood."""tokens= nltk.word_tokenize(sentence)

print(tokens)
```

Output:

['At','eight',"o'clock",'on','Thursday','morning','....','Arthur','did',"n't",'feel','very','good','.']

```
text = "This is V.C.E C.S.E I am a Student . I paid a fees of 13.0"

print(re.findall("(?:[A-Z]\.)+[A-Z]",text)) # ([A-Z]\.)+ add "?:" before this for concatenation text =
"That U.S.A poster-print costs $12.40... which is 3.45."

print(re.split(" ",text)) #regex for hypened-words

print(re.findall("\w+-\w+",text))

print(re.findall("(?:\w+-\w+)",text))

 from nltk.corpus import stopwords

print(word_tokenize(text))
```

```
li = []
for w in word_tokenize(text):
if w not in stopwords.words('english'): li.append(w)
li
```

Output:

```
['V.C.E', 'C.S.E']
['That', '', 'U.S.A', 'poster-print', 'costs', '$12.40...', 'which', 'is', '3.45.'] ['poster-print']
['poster-print']
['That', 'U.S.A', 'poster-print', 'costs', '$', '12.40', '...', 'which', 'is', '3.45', '.']
['That', 'U.S.A', 'poster-print', 'costs', '$', '12.40', '...', '3.45', '.']
```

EXPERIMENT - 3: Implement Minimum Edit Distance(MED) algorithm for spelling correction

To compute the Minimum Edit Distance (also known as Levenshtein distance) using NLTK in Python, you can use the edit_distance function provided by NLTK's edit_distance module. This function calculates the minimum number of single-character edits (insertions, deletions, substitutions) required to change one word into another. Here's a Python program to demonstrate how to compute the Minimum Edit Distance using NLTK

```
from re import M

#source = input('Enter the string given: ') source = 'kitten'

#target = input('Enter the string to convert to: ') target = 'sitting'

m = len(source) n = len(target)

dp = [[0 for i in range(m+1)] for j in range(n+1)] print(dp)

for i in range(m+1):

dp[0][i] = i

for j in range(n+1):

dp[j][0] = j print(dp)

for i in range(1, n+1):

for j in range(1, m+1):

if source[j-1] == target[i-1]:

cost = 0 else:

cost = 1

dp[i][j] = min(dp[i-1][j]+1, dp[i][j-1]+1, dp[i-1][j-1]+cost) print(dp)

print('Edit Distance: ', dp[n][m])
```

Output:

```
[[0,0,0,0,0,0,0],[0,0,0,0,0,0,0],[0,0,0,0,0,0,0],
[[0,1,2,3,4,5,6],[1,0,0,0,0,0,0],[2,0,0,0,0,0,0],
[[0,1,2,3,4,5,6],[1,1,2,3,4,5,6],[2,2,1,2,3,4,5],
Edit Distance: 3
```

```
import nltk

def find_minimum_edit_distance(word1

   word2):

distance = nltk.edit_distance(word1,

   word2)returndistance
```

```
#      Example
usageword1=
"kitten"word2
="sitting"
min_edit_dist
ance=find_mi
nimum_edit_
distance(word
1,word2)

print(f"Theminimumeditdistancebetween'{word1}'and'{word2}'is:{min_edit_distance}")
```

Output:

TheminimumeditdistancebetweenKittenandSittingis:3

EXPERIMENT-4 : Implement n-gram language model

To create an N-gram language model using NLTK in Python, we'll use NLTK's ngrams function to generate n-grams from a text corpus. An N-gram language model predicts the probability of the next word in a sequence given the previous (n-1) words. Here's a Python program that demonstrates how to build an N-gram language model using NLTK

```
import nltk nltk.download('brown')

nltk.download('punkt')

from nltk.corpus import brown

from nltk.util import bigrams

from nltk.lm.preprocessing import pad_both_ends, padded_everygram_pipeline

from nltk.lm import MLE
```

```
corpus = brown.sents(categories="news")

test_sentence = ['There', "wasn't", 'a', 'bit', 'of', 'trouble', 'in', 'Texas']

test_sentence_bigrams = list(bigrams(pad_both_ends(test_sentence, n=2)))
print(test_sentence_bigrams)

train_data, vocab = padded_everygram_pipeline(2, corpus)

lm = MLE(2) lm.fit(train_data, vocab)

print("Number of words in vocabulary is:", len(lm.vocab))

 print(lm.counts)

prob = 1

 for t in test_sentence_bigrams:

            score = lm.score(t[1], [t[0]])

             print(score)

             prob *= score

print(prob)
```

OutPut:-

```
[nltk_data] Downloading package brown to
[nltk_data]     /Users/varshuraodugyala/nltk_data...
[nltk_data]   Package brown is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     /Users/varshuraodugyala/nltk_data...
[nltk_data]   Package punkt is already up-to-date!

[('<s>', 'There'), ('There', "wasn't"), ("wasn't", 'a'), ('a', 'bit'), ('bit', 'of'), ('of', 'trouble'), ('trouble', 'in'),
('in', 'Texas'), ('Texas', '</s>')]
```

```
Number of words in vocabulary is: 14397
<NgramCounter with 2 ngram orders and 214977 ngrams>
0.011464417045208739
0.017241379310344827
0.3333333333333333
0.002508780732563974
0.2857142857142857
0.001053001053001053
0.375
0.001584786053882726
0.125
3.6943506664397765e-15



0.125
```

EXPERIMENT-5 : Implement Naïve Bayes classification for sentiment analysis.

To implement a Naive Bayes classifier for sentiment analysis using NLTK (Natural Language Toolkit) in Python, we'll use a dataset of movie reviews as an example. We'll preprocess the data, extract features, train the classifier, and then evaluate its performance.

```
import nltk nltk.download('movie_reviews')

from nltk.corpus import movie_reviews import random

documents = [(list(movie_reviews.words(fileid)), category) for category in
movie_reviews.categories() for fileid in movie_reviews.fileids(category)]

random.shuffle(documents)

all_words = nltk.FreqDist(w.lower() for w in movie_reviews.words())

word_features = list(all_words)[:2000]

def document_features(document):

        document_words = set(document)

         features = {}

        for word in word_features:

                features[f'contains({word})'] = (word in document_words)
```

```
        return features
featuresets = [(document_features(d), c) for (d, c) in documents]
train_set, test_set = featuresets[100:], featuresets[:100]
classifier = nltk.NaiveBayesClassifier.train(train_set)
confusion_matrix = {"tp": 0, "tn": 0, "fp": 0, "fn": 0}
for i in range(len(test_set)):
        predicted = classifier.classify(test_set[i][0])
         actual = test_set[i][1] print(predicted, actual)
        if predicted == 'pos' and actual == 'pos':
                confusion_matrix['tp'] += 1
        elif predicted == 'neg' and actual == 'neg':
                confusion_matrix['tn'] += 1
        elif predicted == 'pos' and actual == 'neg':
                confusion_matrix['fp'] += 1
        else:
                confusion_matrix['fn'] += 1
print("\nConfusion Matrix:")
 print("True Positives (TP): ", confusion_matrix["tp"])
print("True Negatives (TN): ", confusion_matrix["tn"])
 print("False Positives (FP): ", confusion_matrix["fp"])
print("False Negatives (FN): ", confusion_matrix["fn"])
```

Output:

```
[nltk_data] Downloading package movie_reviews to
[nltk_data]     /Users/varshuraodugyala/nltk_data...
[nltk_data]   Package movie_reviews is already up-to-date!

pos pos
neg neg
pos pos
neg neg
neg pos
```

neg neg
pos pos
neg neg
neg neg
neg neg
pos pos
pos pos
pos pos
neg neg
pos pos
neg neg
neg neg
neg neg
pos pos
neg pos
pos pos
neg neg
pos pos
pos pos
pos pos
pos pos
neg pos
neg neg
neg neg
pos pos
pos neg
neg pos
neg pos
neg neg
neg neg
neg neg
neg pos
pos pos
pos neg
neg neg
neg pos
pos pos
pos neg
pos pos
neg neg
neg neg
pos pos
pos pos
neg neg
pos pos
neg neg
pos pos
pos pos
pos neg
neg neg
pos pos
pos pos
neg neg
neg neg
neg neg
neg neg
neg neg
pos pos
pos pos
neg neg
neg neg
pos neg

```
neg neg
pos neg
neg neg
neg neg
pos pos
pos neg
neg neg
pos pos
neg neg
pos pos
pos pos
neg neg
neg neg
neg neg
neg neg
pos pos
neg neg

Confusion Matrix:
True Positives (TP):  33
True Negatives (TN):  44
False Positives (FP):  13
False Negatives (FN):  10
```

EXPERIMENT-6 : Implement POS tagging using HMM

To implement a Hidden Markov Model (HMM) Part-of-Speech (POS) tagger for the Brown corpus using NLTK (version 3.x), we'll utilize NLTK's Hidden Markov Model Tagger class. However, as of NLTK version 3.x, direct support for HMM-based POS tagging using Hidden Markov Model Tagger is not available. Instead, NLTK provides an interface for using third-party implementations like the hmm_tagger from the nltk.tag.hmm module.

6.a)

```
import nltk

from nltk.corpus import brown

from nltk.tag import hmm

nltk.download('punkt')
```

```
sentences = brown.tagged_sents()

trainer=hmm.HiddenMarkovModelTrainer()

tagger = trainer.train(sentences)

text= " this is a sample sentence for POS tagging in python"

words=nltk.word_tokenize(text)

tags=tagger.tag(words)

for word,tag in tags:print(f"{word}: {tag}")
```

Output :

```
this:DT

is:BEZ

a:AT

sample:NN

sentence:NN

for:IN

POS:AT

tagging:AT

in:AT

python:AT
```

6.b)

```
import nltk

from nltk.corpus import brown
```

```
from nltk.tag import hmm

nltk.download('brown')

brown_tagged_sentences = brown.tagged_sents(categories='news')

size=int(len(brown_tagged_sentences)*0.9)

train_sentences = brown_tagged_sentences[:size]

test_sentences = brown_tagged_sentences[:size]

trainer = hmm.HiddenMarkovModelTrainer()

tagger=trainer.train(train_sentences)

print(tagger.accuracy(test_sentences))
```

Output :

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

[nltk_data] Package brown is already up-to-date!

0.98089945979386
```

EXPERIMENT-7 : Implementation of Implement CKY parsing algorithm

The CKY (Cocke-Kasami-Younger) algorithm is a parsing algorithm for context-free grammars typically used in natural language processing. Below is an implementation of the CKY parser in Python

```
def print_chart(chart, n):
        for i in range(n):
                for j in range(n+1):
                        print(chart[i][j], end="\t")
                print()
```

```
def CKY_PARSE(words, grammar):

        n = len(words)

       table = [[set() for _ in range(n+1)] for _ in range(n+1)]

# Fill diagonal with preterminal rules
for i in range(n):
   word = words[i]
   for lhs, rhs in grammar:
      if rhs == (word,):
         table[i][i+1].add(lhs)

# Fill upper cells using binary rules
for length in range(2, n+1):
   for i in range(n - length + 1):
      j = i + length
      for k in range(i+1, j):
         for lhs, rhs in grammar:
            if len(rhs) == 2:
               if rhs[0] in table[i][k] and rhs[1] in table[k][j]:
                  table[i][j].add(lhs)
return table


sentence = "the dog chased the cat" words = sentence.split()

n = len(words)

grammar = [ ('S', ('NP', 'VP')), ('NP', ('DET', 'NOMINAL')), ('VP', ('VERB', 'NP')), ('NOMINAL',
('cat',)), ('NOMINAL', ('dog',)), ('VERB', ('chased',)), ('DET', ('the',)) ]

chart = CKY_PARSE(words, grammar)

print(words, "\n")

 print_chart(chart, n)

start_symbol = 'S'

 if start_symbol in chart[0][n]:

        print("The sentence is grammatically correct.")

else:

        print("The sentence is not grammatically correct.")
```

Output:-

['the', 'dog', 'chased', 'the','cat']

set() {'DET'} {'NP'} set() set() {'S'}

set() set() {'NOMINAL'} set() set() set()

set() set() set() {'VERB'} set() set()

set() set() set() set() {'DET'} {'NP'}

set() set() set() set() set() {'NOMINAL'}

set() set() set() set() set() set()

The sentence is grammatically correct.

EXPERIMENT-8 : Implement PCKY parsing algorithm

The Probabilistic CKY (PCKY) parser is an extension of the CKY parser that incorporates probabilities for parsing probabilistic context-free grammars (PCFGs). Below is an implementation of a PCKY parser in Python

```
def print_chart(chart, n, non_terminals):
    for p in range(n+1):
        for q in range(n+1):
        print('[', p, ',', q, ']:', end=" ")
        for nt in non_terminals:
            if chart[p][q][nt] > 0:
                print('{', nt, ':', chart[p][q][nt], '}', end="")
```

```
            print()

      print()

def PCKY_PARSE(words, grammar, non_terminals):

      n = len(words)

      print(words, "\n")

      table = [[dict() for _ in range(n+1)] for _ in range(n+1)]
      for i in range(n+1):
            for j in range(n+1):
                  for nt in non_terminals:
                        table[i][j][nt] = 0.0

      for j in range(1, n+1):

           for lhs, rhs, pr in grammar:

                  if rhs == (words[j-1],):

                      table[j-1][j][lhs] = pr


            # Fill upper cells
            for i in range(j-2, -1, -1):
                  for k in range(i+1, j):
                        for lhs, rhs, pr in grammar:
                              if len(rhs) == 2 and table[i][k][rhs[0]] > 0
and table[k][j][rhs[1]] > 0:
                                    prob = pr *
table[i][k][rhs[0]]*table[k][j][rhs[1]]
                                    if table[i][j][lhs] < prob:
                                          table[i][j][lhs] = prob
return table


sentence = "the flight includes a meal"

words = sentence.split()

 n = len(words)

grammar = [ ('S', ('NP', 'VP'), 0.80), ('NP', ('DET', 'NOMINAL'), 0.30),
('VP', ('VERB', 'NP'), 0.20), ('NOMINAL', ('meal',), 0.01), ('NOMINAL',
```

```
('flight',), 0.02), ('VERB', ('includes',), 0.05), ('DET', ('the',),
0.40), ('DET', ('a',), 0.40) ]

non_terminals = ['S', 'NP', 'VP', 'DET', 'NOMINAL', 'VERB']

chart = PCKY_PARSE(words, grammar, non_terminals)

print_chart(chart, n, non_terminals)

start_symbol = 'S'

 if chart[0][n]['S'] > 0:

      print("The sentence is grammatically correct.")

 else:

      print("The sentence is not grammatically correct.")
```

OUTPUT:

['the', 'flight', 'includes', 'a', 'meal'] [0 , 0] :

[0 , 1] : { DET : 0.4 }

[0 , 2] : { NP : 0.0024 } [0 , 3] :

[0 , 4] :

[0 , 5] : { S : 2.3040000000000003e-08 }

[1 , 0] :

[1 , 1] :

[1 , 2] : { NOMINAL : 0.02 } [1 , 3] :

[1 , 4] :

[1 , 5] :

[2 , 0] :

[2 , 1] :

[2 , 2] :

[2 , 3] : { VERB : 0.05 } [2 , 4] :

[2 , 5] : { VP : 1.2000000000000002e-05 } [3 , 0] :

[3 , 1] :

[3 , 2] :

[3 , 3] :

[3 , 4] : { DET : 0.4 }

[3 , 5] : { NP : 0.0012 } [4 , 0] :

[4 , 1] :

[4 , 2] :

[4 , 3] :

[4 , 4] :

[4 , 5] : { NOMINAL : 0.01 } [5 , 0] :

[5 , 1] :

[5 , 2] :

[5 , 3] :

[5 , 4] :

[5 , 5] :

The sentence is grammatically correct.

EXPERIMENT-9: Implementation of Computing cosine similarity between the words using term document matrix and term-term matrix

Cosine similarity is a metric used to measure how similar two vectors are, regardless of their size. It is often used in natural language processing and information retrieval to assess the similarity between documents.

```
import nltk
 import random
 import math
from nltk.corpus import brown, stopwords
nltk.download('brown')
 nltk.download('stopwords')
def extract_words(document):
        all_terms_list = brown.words(fileids=document)
        only_words_list = [w.lower() for w in all_terms_list if w.isalpha()]
        stopwords_list = stopwords.words('english')
        final_terms_list = [w for w in only_words_list if w not in stopwords_list]
         return final_terms_list
def freq(word, document):
         d_terms = extract_words(document)
```

```
        fdist = nltk.FreqDist(d_terms)
        return fdist[word]
doc_names = ['ca01', 'ca02', 'ca03', 'ca04']
vocab = set()
for doc in doc_names:
        vocab.update(extract_words(doc))
vocab_len = len(vocab)
 print("Length of vocabulary =", vocab_len)
word1 = list(vocab)[random.randint(0, len(vocab)-1)]
 word2 = list(vocab)[random.randint(0, len(vocab)-1)]
 print("word-1:", word1)
print("word-2:", word2)
word1_vector = [freq(word1, doc) for doc in doc_names]
 word2_vector = [freq(word2, doc) for doc in doc_names]
print("word-1-vector:", word1_vector)
print("word-2-vector:", word2_vector)
dot_product = sum(word1_vector[i] * word2_vector[i] for i in range(len(doc_names)))
vector1_len = math.sqrt(sum(w2 for w in word1_vector))
vector2_len = math.sqrt(sum(w2 for w in word2_vector))
cos_theta = dot_product / (vector1_len * vector2_len) if vector1_len*vector2_len != 0 else 0
print(f"cos_theta({word1}, {word2}) =", cos_theta)
```

OUTPUT:

length of vocabulary = 2023

word-1: ignored

word-2: interest

word-1-vector: [0, 0, 1, 0]

word-2-vector: [2, 0, 0, 0]

cos_theta(ignored , interest) = 0.0

EXPERIMENT-10: Implementation of TF-IDF matrix for the given document set

To calculate cosine similarity using TF-IDF (Term Frequency-Inverse Document Frequency), you can use the Scikit-learn library, which provides tools for computing TF-IDF vectors and then calculating cosine similarity between them.

```
import nltk
import random
import math
 from nltk.corpus import brown, stopwords
nltk.download('brown')
nltk.download('stopwords')
doc_names = ['ca01', 'ca02', 'ca03', 'ca04']
def extract_words(document):
        all_terms_list = brown.words(fileids=document)
        only_words_list = [w.lower() for w in all_terms_list if w.isalpha()]
        stopwords_list = stopwords.words('english')
```

```
        final_terms_list = [w for w in only_words_list if w not in stopwords_list]
        return final_terms_list
def term_freq(word, document):
        d_terms=extract_words(document)
        fdist = nltk.FreqDist(d_terms)
        return math.log10(fdist[word] + 1)
def idf(word):
        df=0 for doc in doc_names:
                if word in extract_words(doc):
                        df += 1
        return math.log10(len(doc_names) / df) if df != 0 else 0
vocab = set()
for doc in doc_names:
        vocab.update(extract_words(doc))
 vocab_len = len(vocab)
 print("Length of vocabulary =", vocab_len)
word1 = list(vocab)[random.randint(0, len(vocab)-1)]
 word2 = list(vocab)[random.randint(0, len(vocab)-1)]
print("word-1:", word1)
print("word-2:", word2)
word1_vector = [term_freq(word1, doc) * idf(word1) for doc in doc_names]
 word2_vector = [term_freq(word2, doc) * idf(word2) for doc in doc_names]
print("word-1-vector:", word1_vector)
print("word-2-vector:", word2_vector)
dot_product = sum(word1_vector[i] * word2_vector[i] for i in range(len(doc_names)))
 vector1_len = math.sqrt(sum(w2 for w in word1_vector))
vector2_len = math.sqrt(sum(w2 for w in word2_vector))
```

```
cos_theta = dot_product / (vector1_len * vector2_len) if vector1_len * vector2_len != 0 else 0
print(f"cos_theta({word1}, {word2}) =", cos_theta)
```

OUTPUT:

```
Length of vocabulary = 2023
word-1: opelika
word-2: compulsory
word-1-vector: [0.1812381165789131, 0.0, 0.0, 0.0]
word-2-vector: [0.0, 0.0, 0.1812381165789131, 0.0]
cos_theta(opelika, compulsory) = 0.0
```

EXPERIMENT-11: Language Model Using Feed forward Neural Network

Language modeling using a Feed-Forward Neural Network (FFNN) can be implemented with libraries like Tensor Flow or PyTorch. In this example, I'll use Tensor Flow to create a simple language model. The model will be trained to predict the next word in a sequence of words.

```
import tensorflow as tf

from tensorflow.keras.preprocessing.text import Tokenizer

 from tensorflow.keras.preprocessing.sequence import pad_sequences

 import numpy as np

corpus = [ 'This is a simple example', 'Language modeling is interesting', 'Neural networks are powerful', 'Feed-forward networks are common in natural language processing' ]

tokenizer = Tokenizer() tokenizer.fit_on_texts(corpus)

total_words = len(tokenizer.word_index) + 1

input_sequences = []

 for line in corpus:

        token_list = tokenizer.texts_to_sequences([line])[0]

       for i in range(1, len(token_list)):

              n_gram_sequence = token_list[:i+1]
```

```
            input_sequences.append(n_gram_sequence)

max_sequence_length = max([len(x) for x in input_sequences])

input_sequences = pad_sequences(input_sequences, maxlen=max_sequence_length, padding='pre')

X, y = input_sequences[:, :-1], input_sequences[:, -1]

y = tf.keras.utils.to_categorical(y, num_classes=total_words)

model = tf.keras.Sequential([ tf.keras.layers.Embedding(total_words, 50,
input_length=max_sequence_length-1), tf.keras.layers.LSTM(100),
tf.keras.layers.Dense(total_words, activation='softmax') ])

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

model.fit(X, y, epochs=100, verbose=1)

seed_text = "Neural networks" next_words = 7 # Number of words to generate

for _ in range(next_words):

    token_list = tokenizer.texts_to_sequences([seed_text])[0]

    token_list = pad_sequences([token_list], maxlen=max_sequence_length-1, padding='pre')
    predicted_index = np.argmax(model.predict(token_list, verbose=0), axis=-1)[0]

    output_word = ""
    for word, index in tokenizer.word_index.items():
        if index == predicted_index:
            output_word = word
            break

    seed_text += " " + output_word


print(seed_text)
```

OUTPUT:

```
Epoch 1/1001/1 [==============================] - ETA: 0s - loss: 2.8825 -

accuracy:0.1250
```

1/1 [==========] - 1s 897ms/step - loss: 2.8825 - accuracy: 0.1250
Epoch 100/100 1/1 [==========] - ETA: 0s - loss: 0.0587 -

accuracy:1.0000

1/1 [==========] - 0s 8ms/step - loss: 0.0587 - accuracy: 1.0000
Neural networks are powerful in Natural Language Processing.

EXPERIMENT-12: Implement Language Model Using RNN

Language modeling using a Recurrent Neural Network (RNN) can be implemented effectively using libraries like TensorFlow. Here's a step-by-step guide to building a language model using an RNN

```
import tensorflow as tf

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad_sequences

import numpy as np

# Toy dataset corpus = ['This is a simple example',

                        'Language modeling is interesting',

                         'Neural networks are powerful',

                        'Recurrent neural networks capture sequences well'] # Tokenize the text

tokenizer = Tokenizer()

tokenizer.fit_on_texts(corpus)

total_words = len(tokenizer.word_index) + 1

 # Create input sequences and labels

input_sequences = []

for line in corpus:
```

```
        token_list = tokenizer.texts_to_sequences([line])[0]
        for i in range(1, len(token_list)):
                n_gram_sequence = token_list[:i+1]
                input_sequences.append(n_gram_sequence)
max_sequence_length = max([len(x) for x in input_sequences])
input_sequences = pad_sequences(input_sequences, maxlen=max_sequence_length, padding='pre')
X, y = input_sequences[:, :-1], input_sequences[:, -1]
y = tf.keras.utils.to_categorical(y, num_classes=total_words) # Build the model
model = tf.keras.Sequential([
        tf.keras.layers.Embedding(total_words, 50,input_length=max_sequence_length-1),
        tf.keras.layers.LSTM(100),
        tf.keras.layers.Dense(total_words, activation='softmax')])
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(X, y, epochs=100, verbose=1)
# Generate text using the trained model
seed_text = "Recurrent neural networks" next_words = 5
for _ in range(next_words):
        token_list = tokenizer.texts_to_sequences([seed_text])[0]
        token_list = pad_sequences([token_list], maxlen=max_sequence_length-1, padding='pre')
        predicted = np.argmax(model.predict(token_list), axis=-1)
        output_word = ""
        for word, index in tokenizer.word_index.items():
                if index == predicted:
                        output_word = word
                        break
        seed_text += " " + output_word
  print(seed_text)
```

OUTPUT:

Epoch 1/100 1/1 [==============================] - ETA: 0s - loss: 2.8342 - accuracy:0.0667

1/1 [==============================] - 1s 1s/step - loss: 2.8342 - accuracy: 0.0667
Epoch 100/100 1/1 [==============================] - ETA: 0s - loss: 0.3300 - accuracy:1.0000

1/1 [==============================] - 0s 0s/step - loss: 0.3300 - accuracy: 1.0000

1/1 [==============================] - 0s 0s/step - loss: 0.3300 - accuracy: 1.0000
1/1 [==============================] - ETA:0s

1/1 [==============================] - 0s 336ms/step
1/1 [==============================] - ETA:
0s
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - ETA:
0s
1/1 [==============================] - 0s 8ms/step
1/1 [==============================] - ETA:
0s
1/1 [==============================] - 0s 16ms/step
Recurrent neural networks capture sequences well well well

EXPERIMENT-13: Implement Perform Text Analytics

Text classification is a process in natural language processing (NLP) where text is categorized into predefined classes or labels.

Sentiment Analysis: Determining whether a piece of text expresses positive, negative, or neutral sentiment.

```
from sklearn.feature_extraction.text import CountVectorizer

from sklearn.naive_bayes import MultinomialNB


# Sample data

documents = ["I love programming.", "Python is great for NLP.", "Text analytics is fun!"]

labels = [1, 1, 0]  # 1 for positive, 0 for neutral/negative


# Vectorization

vectorizer = CountVectorizer()

X = vectorizer.fit_transform(documents)
```

```
# Model training
clf = MultinomialNB()
clf.fit(X, labels)

# Prediction
test_docs = ["I enjoy coding.", "NLP is interesting."]
X_test = vectorizer.transform(test_docs)
predictions = clf.predict(X_test)
print(predictions)
```

OutPut:

[1 1]

ADDITIONAL PROGRAMS

1)AMBIGUITY

Python program:

```
pip install nltk
from nltk import CFG, ChartParser

# ------------------------------
# Example 1: Lexical Ambiguity
# ------------------------------
word = "bank"
meanings = [
    "A financial institution where people deposit and withdraw money",
    "The land alongside a river",
    "To tilt an airplane while turning"
]

print("LEXICAL AMBIGUITY:")
print(f"The word '{word}' has {len(meanings)} meanings:")
for i, meaning in enumerate(meanings, 1):
    print(f"{i}. {meaning}")

print("\n" + "-"*50 + "\n")

# ------------------------------
# Example 2: Syntactic Ambiguity
```

```
# ------------------------------
# Grammar to parse "I saw the man with a telescope"
grammar = CFG.fromstring("""
    S -> NP VP
    VP -> V NP | VP PP
    PP -> P NP
    NP -> Det N | NP PP | 'I'
    V -> 'saw'
    Det -> 'the' | 'a'
    N -> 'man' | 'telescope'
    P -> 'with'
""")

sentence = ['I', 'saw', 'the', 'man', 'with', 'a', 'telescope']
parser = ChartParser(grammar)

print("SYNTACTIC AMBIGUITY:")
print("Sentence: 'I saw the man with a telescope'")
print("Possible parses:\n")

for tree in parser.parse(sentence):
    print(tree)          # Print the parse tree in text form
    tree.pretty_print()  # Pretty print the tree
    print()
```

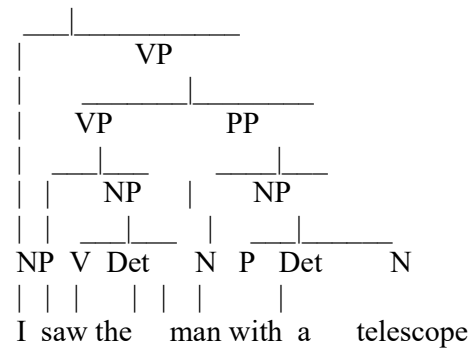
Output:

LEXICAL AMBIGUITY:

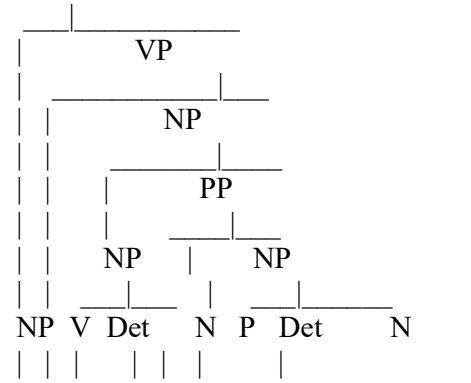
The word 'bank' has 3 meanings:
1. A financial institution where people deposit and withdraw money
2. The land alongside a river
3. To tilt an airplane while turning

SYNTACTIC AMBIGUITY:
Sentence: 'I saw the man with a telescope'
Possible parses:

```
(S
  (NP I)
  (VP
    (VP (V saw) (NP (Det the) (N man)))
    (PP (P with) (NP (Det a) (N telescope)))))
```




```
(S
  (NP I)
  (VP
    (V saw)
    (NP
      (NP (Det the) (N man))
      (PP (P with) (NP (Det a) (N telescope))))))
```



I saw the man with a telescope

2)Text classification program:

pip install scikit-learn

import spacy from spacy import displacy

nlp = spacy.load("en_core_web_sm")

text = """ Apple Inc. is looking at buying U.K. startup for $1 billion. Tim Cook, the CEO of Apple, said this deal will be completed by next month. Barack Obama was the 44th president of the United States. """

doc = nlp(text)

for ent in doc.ents: print(f"Text: {ent.text} | Label: {ent.label_}")

displacy.render(doc, style="ent", jupyter=True)

OUTPUT:

Text: Apple Inc. | Label: ORG
Text: U.K. | Label: GPE
Text: $1 billion | Label: MONEY
Text: Tim Cook | Label: PERSON
Text: Apple | Label: ORG
Text: next month | Label: DATE
Text: Barack Obama | Label: PERSON
Text: 44th | Label: ORDINAL
Text: the United States | Label: GPE

Apple Inc. **ORG** is looking at buying U.K. **GPE** startup for $1 billion **MONEY** .
Tim Cook **PERSON** , the CEO of Apple **ORG** , said this deal will be completed by next month **DATE** .
Barack Obama **PERSON** was the 44th **ORDINAL** president of the United States **GPE** .

3)Smoothing

pip install sentence-transformers scikit-learn

from sentence_transformers import SentenceTransformer from sklearn.metrics.pairwise import cosine_similarity import numpy as np

model = SentenceTransformer('all-MiniLM-L6-v2') # small, fast, good accuracy

sentences = ["I love playing football.", "Soccer is my favorite sport.", "The weather is sunny today.", "It is raining outside."]

sentence_vectors = model.encode(sentences)

print("Sentence embedding shape:", sentence_vectors.shape)

similarity_matrix = cosine_similarity(sentence_vectors)

print("\nCosine Similarity Matrix:\n") print(similarity_matrix)

most_similar_idx = np.argmax(similarity_matrix[0][1:]) + 1 # ignore self-similarity

print(f"\nSentence most similar to '{sentences[0]}' is '{sentences[most_similar_idx]}'")

OUTPUT:

Sentence embedding shape: (4, 384)

Cosine Similarity Matrix:

```
[[ 1.          0.65746427 -0.02304317  0.01971502]
 [ 0.65746427  1.0000002   0.05700713  0.04840674]
 [-0.02304317  0.05700713  1.0000001   0.46222505]
 [ 0.01971502  0.04840674  0.46222505  0.99999976]]
```

Sentence most similar to 'I love playing football.' is 'Soccer is my favorite sport.'

4)Text summarization:

pip install transformers torch

from transformers import pipeline

```
summarizer = pipeline("summarization", model="t5-small") # or "facebook/bart-large-cnn"

text = """ Natural Language Processing (NLP) is a field of artificial intelligence that focuses on the interaction between computers and humans using natural language. The ultimate objective of NLP is to read, decipher, understand, and make sense of human languages in a manner that is valuable. Many challenges in NLP involve natural language understanding, natural language generation, sentiment analysis, text summarization, machine translation, and question answering. """

summary = summarizer(text, max_length=50, min_length=25, do_sample=False)

print("Original Text:\n", text) print("\nSummarized Text:\n", summary[0]['summary_text'])

print("\nSummarized Text:\n", summary[0]['summary_text'])
```

OUTPUT:

Original Text:

Natural Language Processing (NLP) is a field of artificial intelligence that focuses on the interaction between computers and humans using natural language.
The ultimate objective of NLP is to read, decipher, understand, and make sense of human languages in a manner that is valuable.
Many challenges in NLP involve natural language understanding, natural language generation, sentiment analysis, text summarization, machine translation, and question answering.

Summarized Text:
natural language processing (NLP) is a field of artificial intelligence that focuses on the interaction between computers and humans using natural language . many challenges in NLP involve natural language understanding, natural language generation, sentiment analysis, text summarization, machine translation .

5)RECOMMENDATION SYSTEM:

```
from sklearn.feature_extraction.text

import TfidfVectorizer from sklearn.metrics.pairwise

import cosine_similarity
```

```
movies = [ {"title": "The Matrix", "description": "A hacker learns about the true nature of reality and fights against controllers."}, {"title": "Inception", "description": "A thief enters dreams to plant an idea into a CEO's mind."}, {"title": "Interstellar", "description": "Explorers travel through a wormhole in space to ensure humanity's survival."}, {"title": "The Lord of the Rings", "description": "A hobbit journeys to destroy a powerful ring and save Middle-earth."}, {"title": "The Social Network", "description": "The story of the founding of Facebook and the legal battles that followed."} ]

descriptions = [movie['description'] for movie in movies]

vectorizer = TfidfVectorizer(stop_words='english') tfidf_matrix =
vectorizer.fit_transform(descriptions)

similarity_matrix = cosine_similarity(tfidf_matrix)

def recommend(title, movies=movies, similarity_matrix=similarity_matrix, top_n=3): # Find index of the movie

idx = next(i for i, m in enumerate(movies) if m['title'] == title)

# Get similarity scores for this movie
sim_scores = list(enumerate(similarity_matrix[idx]))

# Sort by similarity and get top_n (excluding the movie itself)
sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
sim_scores = sim_scores[1:top_n+1]

# Print recommendations
print(f"Movies similar to '{title}':")
for i, score in sim_scores:
   print(f"- {movies[i]['title']} (Similarity: {score:.2f})")
 recommend("Inception")
```

Output:

Movies similar to 'Inception':
- The Matrix (Similarity: 0.00)
- Interstellar (Similarity: 0.00)
- The Lord of the Rings (Similarity: 0.00)

6) Encoder decoder machine translation:

```
import numpy as np
```

```
 from tensorflow.keras.models import Model

from tensorflow.keras.layers import Input, LSTM, Embedding, Dense

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad_sequences

input_texts = ["hello", "how are you", "i am fine", "thank you", "good morning"]

target_texts = [" bonjour ", " comment ça va ", " je vais bien ", " merci ", " bonjour "]

input_tokenizer = Tokenizer()

input_tokenizer.fit_on_texts(input_texts)

input_sequences = input_tokenizer.texts_to_sequences(input_texts)

max_encoder_seq_length = max(len(seq) for seq in input_sequences)

num_encoder_tokens = len(input_tokenizer.word_index) + 1

 encoder_input_data = pad_sequences(input_sequences, maxlen=max_encoder_seq_length,
padding='post')

target_tokenizer = Tokenizer(filters='')

target_tokenizer.fit_on_texts(target_texts)

target_sequences = target_tokenizer.texts_to_sequences(target_texts)

max_decoder_seq_length = max(len(seq) for seq in target_sequences)

num_decoder_tokens = len(target_tokenizer.word_index) + 1

decoder_input_data = pad_sequences([seq[:-1] for seq in target_sequences],
maxlen=max_decoder_seq_length-1, padding='post')

 decoder_target_data = pad_sequences([seq[1:] for seq in target_sequences],
maxlen=max_decoder_seq_length-1, padding='post')

decoder_target_data_oh = np.zeros((len(input_texts), max_decoder_seq_length-1,
num_decoder_tokens), dtype='float32')

for i, seq in enumerate(decoder_target_data):

        for t, word_idx in enumerate(seq):

                if word_idx > 0:
```

```
decoder_target_data_oh[i, t, word_idx] = 1.0

latent_dim = 32

embedding_dim = 32

encoder_inputs = Input(shape=(max_encoder_seq_length,))

enc_emb = Embedding(num_encoder_tokens, embedding_dim,
input_length=max_encoder_seq_length)(encoder_inputs)

 encoder_lstm = LSTM(latent_dim, return_state=True)

encoder_outputs, state_h, state_c = encoder_lstm(enc_emb) encoder_states = [state_h, state_c]

decoder_inputs = Input(shape=(max_decoder_seq_length-1,))

dec_emb = Embedding(num_decoder_tokens, embedding_dim,
input_length=max_decoder_seq_length-1)(decoder_inputs)

decoder_lstm = LSTM(latent_dim, return_sequences=True, return_state=True)

decoder_outputs, _, _ = decoder_lstm(dec_emb, initial_state=encoder_states)

decoder_dense = Dense(num_decoder_tokens, activation='softmax')

decoder_outputs = decoder_dense(decoder_outputs)

model = Model([encoder_inputs, decoder_inputs], decoder_outputs)

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()

model.fit([encoder_input_data, decoder_input_data], decoder_target_data_oh, batch_size=2,
epochs=50, verbose=1)
```

OUTPUT:

Model: "functional"

```
┌─────────────────────────────┬──────────────────────────┬─────────────┬──────────────────────────┐
│ Layer (type)                │ Output Shape             │ Param #     │ Connected to             │
├─────────────────────────────┼──────────────────────────┼─────────────┼──────────────────────────┤
│ input_layer_1               │ (None, 3)                │           0 │ -                        │
```

(InputLayer)			
input_layer_2 (InputLayer)	(None, 4)	0	-
embedding (Embedding)	(None, 3, 32)	352	input_layer_1[0]...
embedding_1 (Embedding)	(None, 4, 32)	352	input_layer_2[0]...
lstm (LSTM)	[(None, 32), (None, 32), (None, 32)]	8,320	embedding[0][0]
lstm_1 (LSTM)	[(None, 4, 32), (None, 32), (None, 32)]	8,320	embedding_1[0][0]... lstm[0][1], lstm[0][2]
dense_1 (Dense)	(None, 4, 11)	363	lstm_1[0][0]

Total params: 17,707 (69.17 KB)

Trainable params: 17,707 (69.17 KB)

Non-trainable params: 0 (0.00 B)

Epoch 1/50
3/3 **2s** 7ms/step - accuracy: 0.1500 - loss: 1.6760
Epoch 2/50
3/3 **0s** 6ms/step - accuracy: 0.2500 - loss: 1.6684
Epoch 3/50
3/3 **0s** 5ms/step - accuracy: 0.2500 - loss: 1.6608
Epoch 4/50
3/3 **0s** 6ms/step - accuracy: 0.2500 - loss: 1.6526
Epoch 5/50
3/3 **0s** 5ms/step - accuracy: 0.2500 - loss: 1.6446
Epoch 6/50
3/3 **0s** 6ms/step - accuracy: 0.2500 - loss: 1.6355
Epoch 7/50
3/3 **0s** 6ms/step - accuracy: 0.2500 - loss: 1.6247
Epoch 8/50

3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 5ms/step - accuracy: 0.2500 - loss: 1.6127
Epoch 9/50
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 6ms/step - accuracy: 0.2500 - loss: 1.5992
Epoch 10/50
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 5ms/step - accuracy: 0.2500 - loss: 1.5839
Epoch 11/50
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 6ms/step - accuracy: 0.2500 - loss: 1.5649
Epoch 12/50
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 6ms/step - accuracy: 0.2500 - loss: 1.5423
Epoch 13/50
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 5ms/step - accuracy: 0.2500 - loss: 1.5158
Epoch 14/50
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 7ms/step - accuracy: 0.2500 - loss: 1.4838
Epoch 15/50
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 5ms/step - accuracy: 0.2500 - loss: 1.4497
Epoch 16/50
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 6ms/step - accuracy: 0.2500 - loss: 1.4098
Epoch 17/50
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 5ms/step - accuracy: 0.2500 - loss: 1.3674
Epoch 18/50
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 6ms/step - accuracy: 0.2500 - loss: 1.3398
Epoch 19/50
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 6ms/step - accuracy: 0.2500 - loss: 1.3221
Epoch 20/50
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 5ms/step - accuracy: 0.2500 - loss: 1.3089
Epoch 21/50
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 6ms/step - accuracy: 0.2500 - loss: 1.2993
Epoch 22/50
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 6ms/step - accuracy: 0.2500 - loss: 1.2899
Epoch 23/50
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 5ms/step - accuracy: 0.2500 - loss: 1.2804
Epoch 24/50
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 6ms/step - accuracy: 0.2500 - loss: 1.2668
Epoch 25/50
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 5ms/step - accuracy: 0.2500 - loss: 1.2504
Epoch 26/50
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 5ms/step - accuracy: 0.2500 - loss: 1.2313
Epoch 27/50
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 6ms/step - accuracy: 0.2500 - loss: 1.2096
Epoch 28/50
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 5ms/step - accuracy: 0.2500 - loss: 1.1874
Epoch 29/50
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 6ms/step - accuracy: 0.2500 - loss: 1.1644
Epoch 30/50

3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 6ms/step - accuracy: 0.2500 - loss: 1.1467
Epoch 31/50
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 5ms/step - accuracy: 0.2500 - loss: 1.1232
Epoch 32/50
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 6ms/step - accuracy: 0.2500 - loss: 1.1045
Epoch 33/50
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 6ms/step - accuracy: 0.2500 - loss: 1.0826
Epoch 34/50
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 6ms/step - accuracy: 0.2500 - loss: 1.0600
Epoch 35/50
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 7ms/step - accuracy: 0.2500 - loss: 1.0368
Epoch 36/50
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 7ms/step - accuracy: 0.2500 - loss: 1.0128
Epoch 37/50
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 5ms/step - accuracy: 0.3000 - loss: 0.9888
Epoch 38/50
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 6ms/step - accuracy: 0.3500 - loss: 0.9661
Epoch 39/50
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 8ms/step - accuracy: 0.3500 - loss: 0.9444
Epoch 40/50
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 6ms/step - accuracy: 0.3500 - loss: 0.9232
Epoch 41/50
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 6ms/step - accuracy: 0.3500 - loss: 0.9039
Epoch 42/50
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 6ms/step - accuracy: 0.3500 - loss: 0.8851
Epoch 43/50
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 6ms/step - accuracy: 0.3500 - loss: 0.8670
Epoch 44/50
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 5ms/step - accuracy: 0.4000 - loss: 0.8505
Epoch 45/50
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 6ms/step - accuracy: 0.4000 - loss: 0.8341
Epoch 46/50
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 5ms/step - accuracy: 0.4000 - loss: 0.8168
Epoch 47/50
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 5ms/step - accuracy: 0.4500 - loss: 0.8035
Epoch 48/50
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 5ms/step - accuracy: 0.5000 - loss: 0.7849
Epoch 49/50
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 6ms/step - accuracy: 0.5000 - loss: 0.7692
Epoch 50/50
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 5ms/step - accuracy: 0.5000 - loss: 0.7505

<keras.src.callbacks.history.History at 0x14ea50860>

7)Named entity recognition:

```
pip install spacy

!python -m spacy download en_core_web_sm

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

import spacy

nlp = spacy.load("en_core_web_sm")

text = """Apple is planning to open a new office in Hyderabad, India by 2026. Tim Cook, the CEO of Apple, met with Narendra Modi to discuss investment opportunities."""

doc = nlp(text)

print("Named Entities, their labels, and positions:\n")

for ent in doc.ents:

        print(f"{ent.text:<25} | {ent.label_:<10} | Start: {ent.start_char}, End: {ent.end_char}")

print("\nEntity Label Explanation:") print(spacy.explain("GPE")) print(spacy.explain("ORG")) print(spacy.explain("PERSON"))
```

Output:

Named Entities, their labels, and positions:

```
Apple                     | ORG        | Start: 0, End: 5
Hyderabad                 | GPE        | Start: 42, End: 51
India                     | GPE        | Start: 53, End: 58
2026                      | DATE       | Start: 62, End: 66
Tim Cook                  | PERSON     | Start: 80, End: 88
Apple                     | ORG        | Start: 101, End: 106
Narendra Modi             | PERSON     | Start: 117, End: 130

Entity Label Explanation:
Countries, cities, states
Companies, agencies, institutions, etc.
People, including fictional
```

8)Word to vec:

```
pip install gensim
pip install nltk
import nltk
nltk.download('punkt')
import nltk
nltk.download("punkt_tab")

from gensim.models import Word2Vec import nltk

nltk.download('punkt')

text = """Natural Language Processing is a field of Artificial Intelligence. It deals with how computers understand human language. Word embeddings like Word2Vec capture semantic meaning of words."""

sentences = nltk.sent_tokenize(text) data = [nltk.word_tokenize(sentence.lower()) for sentence in sentences]

model = Word2Vec(sentences=data, vector_size=50, window=5, min_count=1, sg=1)

print("Vector representation for 'language':\n", model.wv['language'])

print("\nWords most similar to 'language':") print(model.wv.most_similar('language'))

print("\nSimilarity between 'language' and 'computers':", model.wv.similarity('language', 'computers'))
```

Output:

```
Vector representation for 'language':
 [-0.01631313  0.00898634 -0.00828216  0.00164936  0.01698417 -0.00893383
  0.00903378 -0.01356238 -0.00709291  0.01878958 -0.00315817  0.00064451
 -0.00826876 -0.01536609 -0.00303185  0.0049505  -0.00176973  0.01108032
 -0.00550026  0.00451065  0.01092105  0.01671072 -0.00290536 -0.01840301
  0.00874378  0.00115624  0.01488257 -0.00161443 -0.00528963 -0.01750265
 -0.00171541  0.00567086  0.01080243  0.01410966 -0.01140788  0.00371402
  0.01217452 -0.00960064 -0.00619748  0.01360284  0.00326445  0.00038038
  0.00693825  0.00044166  0.01925861  0.01011735 -0.01783446 -0.01409558
  0.00180856  0.01277417]

Words most similar to 'language':
[('understand', 0.23005631566047668), ('embeddings', 0.16102485358715057), ('word',
0.14890338480472565), ('deals', 0.12482792884111404), ('with', 0.08072882890701294), ('is',
0.07400191575288773), ('word2vec', 0.055481769144535065), ('.', 0.04347098991274834),
```

('processing', 0.018909117206931114), ('how', 0.01197921484708786)]

Similarity between 'language' and 'computers': -0.034299165

9)word disambiguity:

```
import nltk
from nltk.corpus import wordnet as wn
from nltk.wsd import lesk

# Download resources (only the first time)
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('omw-1.4')

# Example sentences with the ambiguous word "bank"
sentences = [
    "He deposited money in the bank.",
    "The fisherman sat on the bank of the river."
]

for sent in sentences:
    tokens = nltk.word_tokenize(sent)
    # Apply Lesk algorithm for disambiguation
    sense = lesk(tokens, "bank")
    print(f"Sentence: {sent}")
    print(f"Predicted Sense: {sense}")
    print(f"Definition: {sense.definition() if sense else 'No definition found'}\n")
```

Output:

Sentence: He deposited money in the bank.
Predicted Sense: Synset('savings_bank.n.02')
Definition: a container (usually with a slot in the top) for keeping money at home

Sentence: The fisherman sat on the bank of the river.
Predicted Sense: Synset('bank.v.07')
Definition: cover with ashes so to control the rate of burning

10)sentence to vec:

```
from sklearn.feature_extraction.text import TfidfVectorizer

 from sklearn.metrics.pairwise import cosine_similarity

sentences = [ "I love playing football.", "Soccer is my favorite sport.", "The Eiffel Tower is in Paris." ]

vectorizer = TfidfVectorizer()

sentence_vectors = vectorizer.fit_transform(sentences)

print("Vector shape:", sentence_vectors.shape)

similarity = cosine_similarity(sentence_vectors[0], sentence_vectors[1])

print("Similarity between first two sentences:", similarity[0][0])
```

Output:

```
Vector shape: (3, 13)
Similarity between first two sentences: 0.0
```

11)TEXT GENERATION WITH LSTM:

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Embedding

# Sample text (toy dataset)
```

```
text = "lstm is a type of recurrent neural network. it is used in natural language processing."
chars = sorted(list(set(text)))
char_to_idx = {c: i for i, c in enumerate(chars)}
idx_to_char = {i: c for i, c in enumerate(chars)}

# Convert text to integers
encoded = [char_to_idx[c] for c in text]

# Prepare sequences
seq_length = 10
x = []
y = []
for i in range(len(encoded) - seq_length):
x.append(encoded[i:i+seq_length])
y.append(encoded[i+seq_length])
x = np.array(x)
y = np.array(y)

# Build model
model = Sequential()
model.add(Embedding(len(chars), 50, input_length=seq_length))
model.add(LSTM(100))
model.add(Dense(len(chars), activation='softmax'))

model.compile(loss='sparse_categorical_crossentropy', optimizer='adam')

# Train
model.fit(x, y, epochs=50, verbose=1)

# Generate text
seed = "lstm is a "
seed_encoded = [char_to_idx[c] for c in seed]
for _ in range(50):
x_pred = np.array(seed_encoded[-seq_length:]).reshape(1, seq_length)
pred = np.argmax(model.predict(x_pred, verbose=0))
seed_encoded.append(pred)

generated_text = "".join(idx_to_char[i] for i in seed_encoded)
print("Generated text:")
print(generated_text)
```

OUTPUT:

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/embedding.py:97: UserWarning: Argument
`input_length` is deprecated. Just remove it.
  warnings.warn(
```

```
Epoch 1/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 4s 50ms/step - loss: 3.0434
Epoch 2/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - loss: 3.0330
Epoch 3/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 3.0179
Epoch 4/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 3.0033
Epoch 5/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - loss: 2.9773
Epoch 6/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step - loss: 2.9452
Epoch 7/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step - loss: 2.8873
Epoch 8/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step - loss: 2.8286
Epoch 9/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step - loss: 2.7777
Epoch 10/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step - loss: 2.8105
Epoch 11/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 2.7791
Epoch 12/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 2.7369
Epoch 13/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step - loss: 2.7485
Epoch 14/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - loss: 2.7649
Epoch 15/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - loss: 2.7455
Epoch 16/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step - loss: 2.6912
Epoch 17/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - loss: 2.7197
Epoch 18/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - loss: 2.7014
Epoch 19/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 2.6688
Epoch 20/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step - loss: 2.6845
Epoch 21/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 2.6241
Epoch 22/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 2.6552
Epoch 23/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step - loss: 2.5843
Epoch 24/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step - loss: 2.6730
Epoch 25/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step - loss: 2.5956
Epoch 26/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 2.5933
Epoch 27/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 2.5536
```

```
Epoch 28/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 2.4900
Epoch 29/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 2.5217
Epoch 30/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 2.4694
Epoch 31/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 2.4195
Epoch 32/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 2.4173
Epoch 33/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 2.4048
Epoch 34/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 2.3628
Epoch 35/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 2.3708
Epoch 36/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 2.3163
Epoch 37/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 2.2657
Epoch 38/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 2.2793
Epoch 39/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 2.2479
Epoch 40/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 2.1738
Epoch 41/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 2.1553
Epoch 42/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 2.1356
Epoch 43/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 2.1616
Epoch 44/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 2.1351
Epoch 45/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 2.0126
Epoch 46/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 1.9456
Epoch 47/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 1.9893
Epoch 48/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 1.9486
Epoch 49/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 1.8525
Epoch 50/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 1.8525
Generated text:
lstm is a yye    preeerer nnnenaalllggggppeesssiiiiiintnntuaa
```

11) SENTIMENT ANALYSIS with LSTM:

```
import tensorflow as tf
from tensorflow.keras.datasets import imdb
```

```
from tensorflow.keras.preprocessing import sequence
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense

# Load IMDB dataset (only top 10,000 words)
max_features = 10000
maxlen = 200 # cut texts after 200 words
batch_size = 32

print("Loading data...")
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)
print(len(x_train), "train sequences")
print(len(x_test), "test sequences")

# Pad sequences to equal length
x_train = sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = sequence.pad_sequences(x_test, maxlen=maxlen)

# Build LSTM model
model = Sequential()
model.add(Embedding(max_features, 128))
model.add(LSTM(128, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation="sigmoid"))

# Compile & Train
model.compile(loss="binary_crossentropy", optimizer="adam", metrics=["accuracy"])
print("Training...")
model.fit(x_train, y_train, batch_size=batch_size, epochs=2, validation_data=(x_test, y_test))

# Evaluate
score, acc = model.evaluate(x_test, y_test, batch_size=batch_size)
print("Test score:", score)
print("Test accuracy:", acc)
```

OUTPUT:

Loading data... 25000 train sequences 25000 test sequences Training... Epoch 1/2 **782/782** ━━━━━━━━━━━━━━━━━━━━ **558s** 709ms/step - accuracy: 0.7112 - loss: 0.5426 - val_accuracy: 0.8265 - val_loss: 0.3970 Epoch 2/2 **782/782** ━━━━━━━━━━━━━━━━━━━━ **555s** 710ms/step - accuracy: 0.8339 - loss: 0.3815 - val_accuracy: 0.8284 - val_loss: 0.3968 **782/782** ━━━━━━━━━━━━━━━━━━━━ **92s** 118ms/step - accuracy: 0.8264 - loss: 0.4007

Test score: 0.39676380157470703

Test accuracy: 0.8284000158309937