

Thadomal Shahani Engineering College
Bandra (W.), Mumbai - 400 050.

CERTIFICATE

Certify that Mr./Miss Harshik Bhagat
of Computer Department, Semester VII with
Roll No. 2103017 has completed a course of the necessary
experiments in the subject NLP under my
supervision in the **Thadomal Shahani Engineering College**
Laboratory in the year 2024 - 2025



Teacher In-Charge

Head of the Department

Date 15/10/24

Principal

CONTENTS

SR. NO.	EXPERIMENTS	PAGE NO.	DATE	TEACHERS SIGN.
1.	Study various application of NLP and NLP tools	3	15/7/24	
2.	Study various application of NLP and formulate prob. stat. for mini project	6	22/7/24	
3.	To implement NLP PreProc Task	9	29/7/24	
4.	To implement advance text pre-processing	13	5/8/24	16/10/24
5.	To implement advance text pre-processing shallow and deep parsing	18	12/8/24	
6.	To implement text processing	25	19/8/24	
7.	To implement morphological analysis	31	26/8/24	
8.	To implement part of speech tagging	34	2/9/24	
9.	To implement named entity	37	16/9/24	
10.	Advance NLP problem	45	23/9/24	
11.	Mini project	55	8/10/24	
12.	Written Assignment	65	14/10/24	

Experiment No: 1

Aim: Study various applications of NLP and NLP Tools.

Theory:

NLP (Natural Language Processing) is a field of Computer Science and a subfield of AI that aims to create computers which use natural language as input and/or output.

NLP also makes computers or intelligent systems perform useful tasks with the help of natural languages humans use.

NLP Libraries:

Some of the most prominent libraries used in NLP are:

1. NLTK (Natural Language Toolkit):

- a. One of the oldest and most well-known libraries for natural language processing in Python. It provides a wide range of tools for working with human language data, including text processing libraries, classification, tokenization, stemming, tagging, parsing, and semantic reasoning.
- b. Use Cases: It's often used for educational purposes and for projects that require basic text processing tasks like tokenization, stemming, and classification. It also includes a large collection of text corpora and lexical resources such as WordNet.
- c. Advantages:
 - i. Easy to use and well-documented.
 - ii. Provides a wide range of linguistic data and pre-built models.
- d. Limitations:
 - i. Can be slow with large datasets.
 - ii. Lacks the modern features and speed of more recent libraries like spaCy.

2. spaCy:

- a. spaCy is a popular and fast NLP library designed specifically for production use. It provides industrial-strength NLP tools with a focus on performance and efficiency, handling large volumes of text quickly and accurately.
- b. Use Cases: Ideal for projects that need to scale NLP tasks such as named entity recognition (NER), part-of-speech tagging, dependency parsing, and text classification. It's widely used in industry due to its speed and ease of use.
- c. Advantages:
 - i. Extremely fast and efficient.
 - ii. Supports deep learning integration.
 - iii. Pre-trained models for various languages.
- d. Limitations:
 - i. Less flexible than NLTK for low-level tasks.
 - ii. Larger memory footprint due to the focus on performance.

3. Hugging Face Transformers:

- a. Hugging Face's Transformers library is a state-of-the-art framework for natural language processing tasks based on transformer models like BERT, GPT, and

- T5. It provides pre-trained models and tools to fine-tune these models for various NLP tasks.
- b. Use Cases: Perfect for tasks like text generation, summarization, translation, question-answering, and sentiment analysis. Hugging Face has revolutionized NLP by making it easy to use powerful transformer models with minimal code.
 - c. Advantages:
 - i. Access to cutting-edge transformer models.
 - ii. Extensive model hub with pre-trained models.
 - iii. Easy integration with deep learning frameworks like TensorFlow and PyTorch.
 - d. Limitations:
 - i. Heavy on computational resources.
 - ii. Requires some understanding of deep learning concepts.

4. AllenNLP:

- a. AllenNLP is an open-source library built on top of PyTorch, developed by the Allen Institute for AI. It's focused on providing researchers and developers with easy access to high-level components for creating custom NLP models.
- b. Use Cases: Best suited for research purposes and developing custom NLP models. It's particularly useful for tasks like reading comprehension, semantic role labeling, and textual entailment.
- c. Advantages:
 - i. Research-focused with tools for easy experimentation.
 - ii. Modular and customizable for complex NLP tasks.
 - iii. Strong community support and well-maintained documentation.
- d. Limitations:
 - i. Steeper learning curve compared to other libraries.
 - ii. Primarily focused on research, so it may not be as optimized for production use.

Cloud-based NLP Platforms:

Cloud-based NLP platforms provide powerful tools for processing and understanding human language using AI and machine learning. These platforms enable developers and businesses to efficiently analyze text data, derive actionable insights, and enhance user experiences without the need for complex infrastructure.

1. Google Cloud NLP:

- a. Google Cloud NLP offers a comprehensive suite of tools for extracting insights from text data, including entity analysis, sentiment analysis, and syntax analysis.
- b. Features:
 - i. Entity Recognition: Identifies entities like people and locations.
 - ii. Sentiment Analysis: Assesses the sentiment of text (positive, negative, neutral).
 - iii. Syntax Analysis: Analyzes sentence structure and grammar.
 - iv. Content Classification: Categorizes text into predefined categories.

2. AWS Comprehend:

- a. AWS Comprehend provides a set of APIs for understanding text, including entity recognition, sentiment analysis, and key phrase extraction.
- b. Features:
 - i. Keyphrase Extraction: Identifies important phrases in text.
 - ii. Entity Recognition: Detects entities such as names and dates.
 - iii. Language Detection: Determines the language of the text.
 - iv. Sentiment Analysis: Analyzes and classifies text sentiment.
 - v. Topic Modeling: Groups similar content by topics.

3. Azure Text Analytics:

- a. Azure Text Analytics offers natural language understanding capabilities such as sentiment analysis, entity recognition, and key phrase extraction.
- b. Features:
 - i. Sentiment Analysis: Evaluates the sentiment of text.
 - ii. Key Phrase Extraction: Highlights important phrases.
 - iii. Named Entity Recognition: Detects entities like names and dates.
 - iv. Language Detection: Identifies the language of the text.
 - v. Opinion Mining: Extracts and associates opinions with specific aspects.

Experiment No: 2

Aim: Study various applications of NLP and formulate the problem statement for Mini Project.

Problem Statement for Mini Project (Spelling and Grammar Checker):

In the contemporary digital landscape, effective communication is paramount. However, challenges in spelling and grammar can hinder professionalism and clarity. This project develops a user-friendly spelling and grammar checker for improved written communication.

Theory:

Natural Language Processing (NLP) is a field of artificial intelligence that focuses on the interaction between computers and human language. It enables machines to understand, interpret, and respond to human language in a way that is both meaningful and useful.

Some of the most important applications of NLP are as follows:

1. Machine Translation:

- a. Machine translation is the automatic conversion of text from one language to another.
- b. It uses advanced algorithms to translate words, phrases, and entire documents while preserving meaning and context.
- c. Applications like Google Translate leverage NLP to provide real-time translation, making communication across languages seamless.
- d. Example: Google Translate can instantly translate a restaurant menu from French to English, allowing travelers to understand and order food in a foreign country without needing to know the language.

2. Text Categorization:

- a. Text Categorization (or text classification) is the process of assigning predefined labels or categories to text based on its content.
- b. This is an essential NLP application used to organize, filter, and manage large volumes of text data efficiently.
- c. The process involves analyzing the content of the text, identifying key themes, topics, or keywords, and then classifying the text into appropriate categories.
- d. Example: Gmail uses text categorization to automatically sort incoming emails into folders like "Primary," "Social," or "Promotions," helping users manage their inbox more efficiently.

3. Text Summarization:

- a. Text summarization is the process of creating a concise version of a longer text while retaining its key information.
- b. NLP techniques extract the most important sentences or phrases to generate summaries, which are useful in news aggregation, research, and content curation to quickly grasp the main points of lengthy documents.
- c. Example: News aggregators like Pocket use text summarization to generate brief summaries of articles, enabling readers to quickly understand the main points without reading the entire piece.

4. Chatbot Development:

- a. Chatbots are AI-driven applications that use NLP to simulate human conversation.
- b. They can understand and respond to user queries, provide customer support, and assist with tasks like booking appointments or answering frequently asked questions. NLP enables chatbots to interpret user intent and generate relevant responses, improving user interaction.
- c. Example: Many websites feature chatbots like those used by e-commerce platforms, where a chatbot can assist customers in finding products, answering questions, or processing returns, all through a conversational interface.

5. Plagiarism Detection:

- a. Plagiarism detection tools use NLP to compare text against a database of existing content to identify similarities.
- b. These tools are commonly used in academic and professional settings to ensure the originality of written work by detecting copied or paraphrased content.
- c. NLP algorithms analyze the structure, phrasing, and context to accurately flag potential plagiarism.
- d. Example: Turnitin, widely used in academic settings, scans student submissions for plagiarism by comparing them to a vast database of academic papers, articles, and websites, flagging any matches for review.

6. Spelling & Grammar Checkers:

- a. Analyze text to detect and correct errors related to spelling, punctuation, grammar, and style.
- b. These tools are widely used in word processors, email clients, and content creation platforms to improve the quality of written communication.
- c. Example: Grammarly is a popular writing assistant that uses NLP to check for spelling, grammar, punctuation, and style errors.

7. Sentiment/Opinion Analysis:

- a. Sentiment analysis (or opinion mining) involves analyzing text data to determine the sentiment or emotional tone expressed within it—whether it's positive, negative, or neutral.
- b. It is commonly used in customer feedback analysis, social media monitoring, and market research.
- c. Example: A company might use sentiment analysis to monitor public opinion about their brand on Twitter. By analyzing tweets containing their brand name, the NLP system can classify the sentiment of each tweet as positive, negative, or neutral.

8. Question Answering

- a. Question Answering (QA) systems are designed to automatically answer questions posed by users in natural language.
- b. They are widely used in search engines, customer support, educational tools, and AI-driven conversational agents.
- c. Example: When you ask a question in Google Search, such as "What is the capital of France?", the search engine uses NLP-based question answering

systems to retrieve and display the answer directly at the top of the search results as a featured snippet: "The capital of France is Paris."

9. Personal Assistants:

- a. Personal assistants like Siri, Alexa, and Google Assistant use NLP to understand and respond to user commands in natural language.
- b. They assist with tasks such as setting reminders, sending messages, searching the web, and controlling smart devices.
- c. Example: Amazon Alexa is a voice assistant that uses NLP to understand and complete tasks. For example, you can say "Alexa, set a timer for 10 minutes" or "What's the weather today?" and it provides an appropriate response.

10. Tutoring Systems

- a. Intelligent tutoring systems (ITS) use NLP to provide personalized instruction and feedback to learners.
- b. These systems can help with language learning, problem-solving exercises, and even complex subjects like mathematics or programming.
- c. Example: Duolingo is an intelligent language tutoring system that uses NLP to provide personalized language learning experiences.

Experiment No: 3

Aim: To implement NLP Preprocessing Tasks

Theory:

Text preprocessing in NLP is a critical step that involves transforming raw text into a format suitable for analysis and modeling. This process typically includes several tasks such as cleaning the text by removing HTML tags, special characters, and extra whitespaces, which helps in standardizing the text. Contractions are expanded to ensure that words are in their full form, making the text more readable and uniform. Text is then converted to lowercase to maintain consistency and make comparisons easier. Additionally, number words are converted to numeric form, and irrelevant stopwords are removed to focus on meaningful content. Phrase extraction and tokenization break down the text into manageable units, enabling better analysis. Preprocessing also includes script validation to ensure the text adheres to the expected format and language. These steps collectively enhance the quality and effectiveness of subsequent NLP tasks such as text classification, sentiment analysis, and information retrieval.

a. Removing HTML Tags, Extra Whitespaces, and Special Characters

Text data often contains HTML tags, unnecessary whitespaces, and special characters that do not contribute to the meaning of the text. Removing these elements is essential to clean the text and ensure that it is ready for further processing.

- In our code we use BeautifulSoup to parse the HTML and extract just the text, getting rid of any HTML tags. We use re.sub to get rid of unnecessary patterns.

b. Sentence Boundary Detection and Expanding Contractions

Identifying the boundaries of sentences within a text is crucial for tasks like sentiment analysis and machine translation. It involves detecting the end of one sentence and the beginning of another.

- Contractions (e.g., "don't," "can't") should be expanded (to "do not," "cannot") to standardize the text and improve consistency.

c. Lowercasing Text, Converting Number Words to Numeric Form, Removing Numbers and Stopwords

- **Lowercasing:** Converting text to lowercase helps in standardizing the text, as "Hello" and "hello" would be considered the same word.
- **Removing Numbers and Stopwords:** Numbers and stopwords (common words like "the," "is," "in") often do not contribute to the meaning of the text in specific NLP tasks and are removed to reduce noise.

d. Phrase Extraction, Tokenization, and Script Validation

- **Phrase Extraction:** Involves identifying and extracting meaningful phrases or sequences of words (e.g., "natural language processing").
- **Tokenization:** The process of splitting text into individual tokens (words or phrases), which are the basic units for further analysis.

- **Script Validation:** Ensures that the text is in the correct script or language, which is particularly important for multilingual NLP applications.

e. Handling Different Types of Textual Data and Visualizing Preprocessing Impacts

Different types of textual data (e.g., formal text, informal text, code) require different preprocessing techniques. Visualizing the impact of preprocessing helps understand how each step contributes to cleaning and preparing the text.

Code :

```
import re
import nltk
from bs4 import BeautifulSoup
from word2number import w2n
from nltk.corpus import stopwords
from contractions import fix # Using 'fix' method from the 'contractions' library
nltk.download('punkt')
nltk.download('stopwords')

text = """
<html><head><title>Sample Title</title></head>
<body><p>This is an example text. It contains HTML tags,
extra spaces, numbers like twenty-one, and special characters!
It also includes contractions like don't, can't.</p></body></html>
"""

# a. Removing HTML Tags, Extra Whitespaces, and Special Characters
def clean_text(text):
    # Remove HTML tags
    soup = BeautifulSoup(text, "html.parser")
    text = soup.get_text()

    # Remove special characters, keeping sentence-ending punctuation
    text = re.sub(r"[^a-zA-Z0-9\s\.\!?\?]", "", text)

    # Remove extra whitespaces
    text = re.sub(r"\s+", " ", text).strip()

    return text

cleaned_text = clean_text(text)
print("Cleaned Text:\n", cleaned_text)

# b. Sentence Boundary Detection and Expanding Contractions
def expand_contractions(text):
    # Expanding contractions using the 'fix' method from the contractions library
    return fix(text)

expanded_text = expand_contractions(cleaned_text)
```

```

sentences = nltk.sent_tokenize(expanded_text)

print("\nExpanded Text:\n", expanded_text)
print("\nSentences:\n", sentences)

# c. Lowercasing Text, Converting Number Words to Numeric Form, Removing Numbers
# and Stopwords
def preprocess_text(text):
    # Lowercase the text
    text = text.lower()

    # Convert number words to numeric form
    words = text.split()
    processed_words = []
    for word in words:
        try:
            processed_words.append(str(w2n.word_to_num(word)))
        except:
            processed_words.append(word)
    text = " ".join(processed_words)

    # Remove numbers
    text = re.sub(r'\d+', ' ', text)

    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    words = text.split()
    filtered_words = [word for word in words if word not in stop_words]

    return " ".join(filtered_words)

final_text = preprocess_text(expanded_text)
print("\nFinal Processed Text:\n", final_text)

# d. Phrase Extraction, Tokenization, and Script Validation
def phrase_extraction(text):
    # Using simple bi-grams for phrase extraction
    words = nltk.word_tokenize(text)
    phrases = list(nltk.bigrams(words))
    return phrases

phrases = phrase_extraction(final_text)
print("\nExtracted Phrases:\n", phrases)

```

Output:

Cleaned Text:

Sample Title This is an example text. It contains HTML tags extra spaces numbers like twentyone and special characters! It also includes contractions like dont cant.

Expanded Text:

Sample Title This is an example text. It contains HTML tags extra spaces numbers like twentyone and special characters! It also includes contractions like do not cannot.

Sentences:

['Sample Title This is an example text.', 'It contains HTML tags extra spaces numbers like twentyone and special characters!', 'It also includes contractions like do not cannot.']}

Final Processed Text:

sample title example text. contains html tags extra spaces numbers like twentyone special characters! also includes contractions like cannot.

Extracted Phrases:

[('sample', 'title'), ('title', 'example'), ('example', 'text'), ('text', '.'), ('.', 'contains'), ('contains', 'html'), ('html', 'tags'), ('tags', 'extra'), ('extra', 'spaces'), ('spaces', 'numbers'), ('numbers', 'like'), ('like', 'twentyone'), ('twentyone', 'special'), ('special', 'characters'), ('characters', '!'), ('!', 'also'), ('also', 'includes'), ('includes', 'contractions'), ('contractions', 'like'), ('like', 'can'), ('can', 'not'), ('not', '.')]

Experiment No: 4

Aim: To implement Advanced Text Pre-processing Techniques

Theory:

Advanced text pre-processing involves a series of sophisticated techniques used to clean and prepare text data for further analysis or machine learning tasks. Beyond basic pre-processing steps like tokenization, stemming, and stop word removal, advanced pre-processing methods address more complex issues in text data and improve the quality of the input for models.

Stop Word Removal:

Stop Words are the words that are highly occurring words in the document and to understand the document better we don't need them because they don't add any meaningful information.

Why should we remove stop words ?

So, there are two main reasons for that:

1. They **provide no meaningful information**, especially if we are building a text classification model. Therefore, we have to remove stopwords from our dataset.
2. As the frequency of stop words are too high, removing them from the corpus results in **much smaller data in terms of size**. Reduced size results in **faster computations on text data** and the text classification model need to deal with a **lesser number of features** resulting in a robust model.

Lemmatization

Lemmatization is a text preprocessing technique that groups together different forms of a word into a single base form, called a lemma. Lemmatization is the process of grouping together different inflected forms of the same word.

Lemmatization is used in natural language processing (NLP) and search engine optimization (SEO) to improve the accuracy and relevance of search results. It can help with:

- Information retrieval systems: When users search for a word, they may expect results that use any inflected form of that word.
- Machine learning algorithms: Lemmatization improves text processing.
- SEO: Lemmatization can improve the readability and clarity of content, and ensure that all variations of a word are treated as the same.

Lemmatization is more complex than stemming because it requires words to be categorized by both part of speech and inflected form.

Examples: Lemmatization:

Study → Study

Studying → Study

Studies → Study

Studied → Study

Studier → Study

Stemming:

Stemming is a text preprocessing technique in natural language processing (NLP). Specifically, it is the process of reducing inflected form of a word to one so-called “stem,” or root form, also known as a “lemma” in linguistics. Stemming is used to:

- Improve search accuracy: Stemming helps search engines find the correct meaning of words.
- Speed up processing time: Stemming removes different endings from words to find the most basic part, or "stem".
- Understand differences between words: Stemming helps identify differences between words.
- Simplify words: Stemming simplifies words.
- Reduce dimensionality: Stemming reduces dimensionality

Examples:

Study → Studi

Studying → Studi

Studies → Studi

Studied → Studi

Studier → Studier

Noise removal:

Removing special characters, punctuation, or non-alphabetic characters that do not add meaning to the text.

Multilingual Text Processing:

Multilingual NLP (Natural Language Processing) is a technology that incorporates linguistics, computer science, and artificial intelligence to process and analyze large amounts of natural human language in various contexts.

It is a branch of natural language processing (NLP) that uses artificial intelligence (AI) to enable machines to understand, interpret, and generate human language in multiple languages.

Entity Anonymization:

Entity anonymization involves replacing or redacting sensitive information (such as names, addresses, or other personal identifiers) with placeholders to protect privacy.

Entity Anonymization can be achieved in 2 ways:

- Regular Expression
- Spacy

Code:

```
import spacy
import re
import nltk
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
from transformers import pipeline
```

```

import presidio_anonymizer
#from presidio_anonymizer import AnalyzerEngine
from presidio_anonymizer import AnonymizerEngine

def remove_stopwords(text):

    nlp = spacy.load("en_core_web_sm")
    doc = nlp(text)

    filtered_words = [token.text for token in doc if not token.is_stop] # Remove
stopwords

    clean_text = ' '.join(filtered_words) # Join the filtered words to form a clean text
    return clean_text

def lemmatization(text):

    nlp = spacy.load('en_core_web_sm') # Load the spaCy English model
    doc = nlp(text)

    lemmatized_tokens = [token.lemma_ for token in doc] # Extract lemmatized
tokens
    lemmatized_text = ' '.join(lemmatized_tokens)

    #print("Original Text:", text)
    #print("Lemmatized Text:", lemmatized_text)

    return lemmatized_text

def stemming(sentence):

    ps = PorterStemmer()
    words = word_tokenize(sentence)

    for w in words:
        print(w, ":", ps.stem(w))

def remove_noise(text):

    text = text.lower()
    text = re.sub(r'\d+', '', text)
    text = re.sub(r'[^\w\s]', '', text)

    tokens = nltk.word_tokenize(text)
    cleaned_sentence = ' '.join(tokens)

    return cleaned_sentence

```

```

def multilingual_processing():
    # Use a smaller, lightweight model for translation
    translator = pipeline("translation", model="t5-small")

    # Translate English to French
    translated_text = translator("Translate English to French: Hello, how are you?")
    print("Translated to French:", translated_text[0]['translation_text'])

    # Create a text generation pipeline with GPT-2
    generator = pipeline("text-generation", model="gpt2") # or use another suitable
model

    # Generate text in English (as an example, since gpt-2 generates English text)
    generated_text = generator("Once upon a time", max_length=50)
    print("Generated text:", generated_text[0]['generated_text'])

def text_anonymity(text):
    # Define patterns for sensitive information
    email_pattern = r"\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b"
    phone_pattern = r"\b(?:\d{3}[-.\s]?)?\d{3}[-.\s]?\d{4}\b"

    # Replace sensitive information with placeholders
    anonymized_text = re.sub(email_pattern, '[EMAIL]', text)
    anonymized_text = re.sub(phone_pattern, '[PHONE]', anonymized_text)

    return anonymized_text

text = "The quick brown foxes are jumping over the lazy dogs."
stop_word_text = remove_stopwords(text)
lemmatized_text = lemmatization(stop_word_text)
print("Text: ", text)
print("Text after Stop words removal: ", stop_word_text)
print("Text after Lemmatization: ", lemmatized_text)

print()

sentence = "Programmers program with programming languages"
print("Sentence to be stemmed: ", sentence)
stemmed_text = stemming(sentence)

print()

noisy_text = "Hey!! @vivek What are you doing?"
after_noise_removal = remove_noise(noisy_text)
print("Text before Noise: ", noisy_text)
print("After Noise removal: ", after_noise_removal)

```

```
# multilingual_processing()
print()

anonymous_text = text = "My phone number is 123-456-7890 and my email is
example@example.com. Feel free to contact me!"

anonymized = text_anonymity(anonymous_text)
print("Anonymous Text: ", anonymous_text)
print("After Anonymized: ", anonymized)
```

OUTPUT:

```
C:\Users\Sanford\Python\python.exe C:/Users/Sanford/PycharmProjects/nlp_exp4/main.py
2024-09-04 00:39:22.603936: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see s
2024-09-04 00:39:24.206163: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see s
Text: The quick brown foxes are jumping over the lazy dogs.
Text after Stop words removal: quick brown foxes jumping lazy dogs .
Text after Lemmatization: quick brown fox jump lazy dog .

Sentence to be stemmed: Programmers program with programming languages
Programmers : programm
program : program
with : with
programming : program
languages : languag

Text before Noise: Hey!! @vivek What are you doing?
After Noise removal: hey vivek what are you doing

Anonymous Text: My phone number is 123-456-7890 and my email is example@example.com. Feel free to contact me!
After Anonymized: My phone number is [PHONE] and my email is [EMAIL]. Feel free to contact me!

Process finished with exit code 0
```

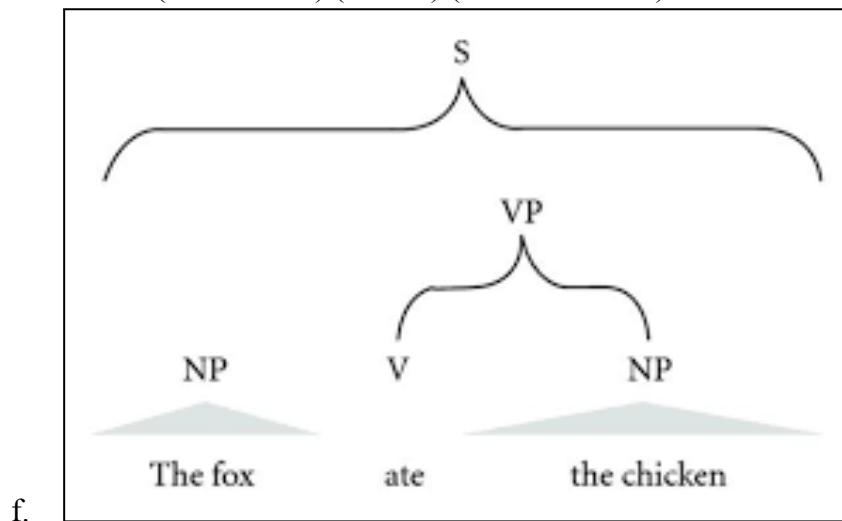
Experiment No: 5

Aim: To implement Shallow and Deep Parsing

Theory:

1. Shallow Parsing:

- a. Shallow Parsing is also known as Syntactic Chunking.
- b. It aims to identify syntactic phrases (chunks) within a sentence without analyzing the complete grammatical structure.
- c. It breaks sentences into chunks like noun phrases, verb phrases, etc., but doesn't provide detailed relations between them.
- d. Process:
 - i. Tokenization: Breaking the sentence into individual words.
 - ii. Chunking: Grouping tokens into syntactic chunks (e.g., noun phrases, verb phrases).
- e. Example:
 - i. Sentence: "The fox ate the chicken":
 - ii. Tokenization: { The, fox, ate, the, chicken }
 - iii. Chunking:
 - 1. Noun Phrase: The fox
 - 2. Verb Phrase: ate the chicken
 - 3. Noun Phrase: the chicken
 - iv. Shallow Parse:
(NP The fox) (VP ate) (NP the chicken)



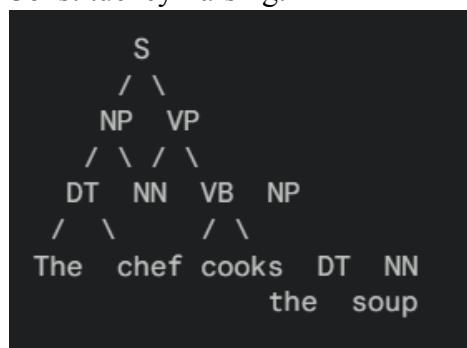
2. Chunking:

- a. Chunking is a core task in natural language processing (NLP), involving identifying syntactic phrases or chunks within a sentence.
- b. These chunks can be noun phrases (NPs), verb phrases (VPs), prepositional phrases (PPs), and other syntactic structures.

- c. Chunking Techniques:
 - i. Rule-Based Chunking: This approach relies on predefined rules or patterns to identify chunks. Rules can be based on part-of-speech (POS) tags, word patterns, or other syntactic features.
Example: A rule might state that a noun phrase consists of an optional determiner followed by one or more adjectives followed by a noun.
 - ii. Statistical Chunking: This approach uses machine learning techniques to learn chunk boundaries from labeled data. Statistical models, such as hidden Markov models (HMMs) or conditional random fields (CRFs), are trained on annotated corpora to identify chunk boundaries.
Example: A statistical chunker might use a CRF model to learn the probabilities of different chunk types based on features like POS tags and word sequences.
- d. Chunking Applications:
 - i. Information Extraction: Identifying named entities (e.g., people, organizations, locations) within text.
 - ii. Question Answering: Understanding the syntactic structure of questions to facilitate accurate responses.
 - iii. Text Summarization: Identifying key phrases and sentences to summarize a document.
 - iv. Machine Translation: Analyzing the syntactic structure of a sentence to aid in translation.

3. Deep Parsing:

- a. Deep parsing aims to construct a complete parse tree for a sentence, representing its grammatical structure in detail.
- b. It involves a detailed syntactic analysis of the sentence, where the complete grammatical structure, including dependencies and relationships between words, is represented.
- c. Process:
 - i. Constituency parsing: Breaking down the sentence into constituent phrases (e.g., noun phrases, verb phrases).
 - ii. Dependency parsing: Representing the syntactic dependencies between words in the sentence.
- d. Example:
 - i. Sentence: "The chef cooks the soup"
 - ii. Constituency Parsing:



This parse tree shows the hierarchical structure of the sentence, with the main constituents being a noun phrase (the subject), a verb phrase (the predicate), and another noun phrase (the object).

iii. Dependency Parsing:

```
cooks (VERB)
  /   \
chef (NOUN) soup (NOUN)
  |
det (DET)
  /
the (DET)
```

This dependency parse shows the relationships between words in the sentence, with the verb "cooks" as the root and the other words acting as its dependents.

4. CFG-based Parse Tree Generation:

- a. CFG-based Parse Tree Generation uses Context-Free Grammar (CFG) to represent the syntactic structure of a sentence.
- b. CFG defines production rules with terminals (words), non-terminals (syntactic categories), and a start symbol.
- c. Process:
 - i. Start with a sentence (e.g., "The cat runs").
 - ii. Apply grammar rules (e.g., $S \rightarrow NP\ VP$).
 - iii. Break down non-terminals into terminals recursively.
 - iv. Build a parse tree showing the sentence's structure.
- d. Example:
 - i. Sentence: "The cat runs":

```
S
/ \
NP  VP
/ \  |
Det N  V
the cat runs
```

- ii.
- e. It is used for syntax checking and NLP tasks, though ambiguity and complexity can arise.

Code / Output:

1. Shallow Parser:

```
import nltk
from nltk import pos_tag
from nltk import RegexpParser

# Download necessary NLTK data
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
```

```

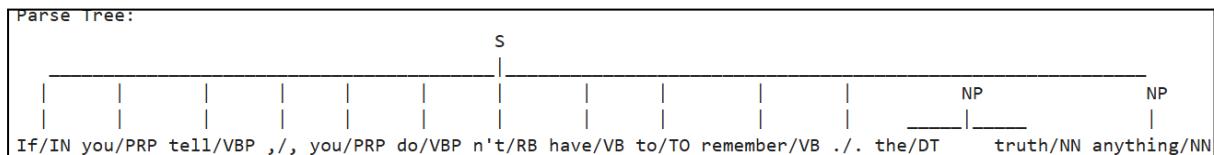
# Define the sentence
quote = "If you tell the truth, you don't have to remember anything."

# Tokenize and tag
tokens = nltk.word_tokenize(quote)
tags = nltk.pos_tag(tokens)

# Define the grammar and parse
grammar = "NP: {<DT>?<JJ>*<NN>+}"
cp = nltk.RegexpParser(grammar)
res = cp.parse(tags)

# Print the parse tree
print("Parse Tree:")
res.pretty_print()

```



2. Deep Parsing:

```

import nltk

# Define a simple CFG
grammar = nltk.CFG.fromstring("""
S -> NP VP
VP -> V NP | V NP PP
PP -> P NP
V -> "saw" | "ate" | "walked"
NP -> "John" | "Mary" | "Bob" | Det N | Det N PP
Det -> "a" | "an" | "the" | "my"
N -> "man" | "dog" | "cat" | "telescope" | "park"
P -> "in" | "on" | "by" | "with"
""")
```

```

# Define a sentence
sentence = "John saw the man in the park".split()
```

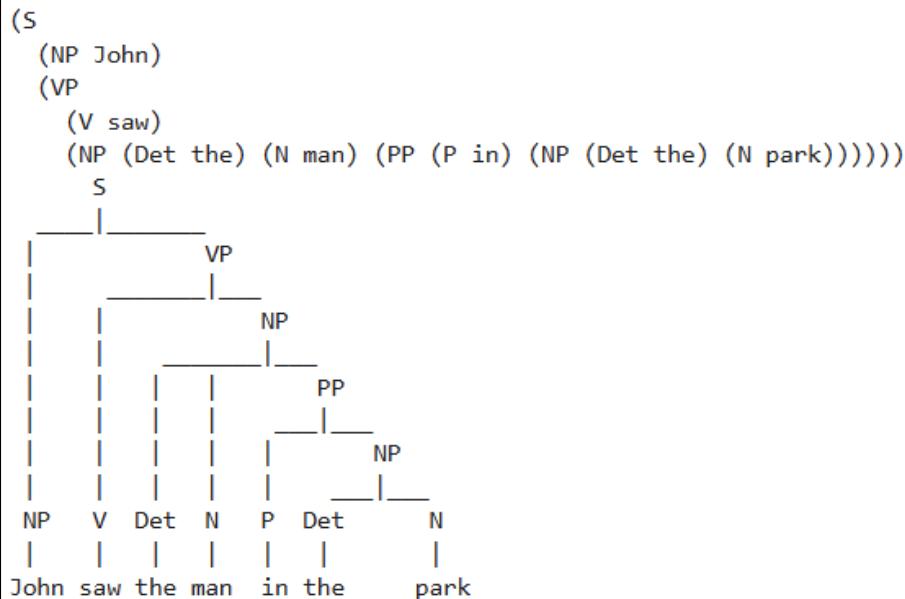
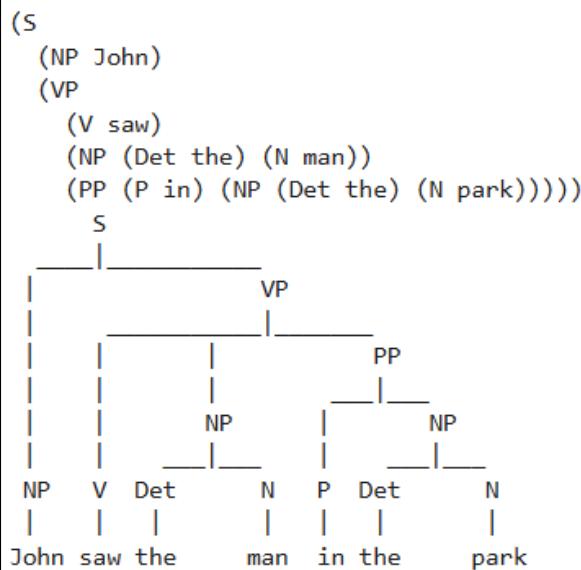
```

# Create a parser
parser = nltk.ChartParser(grammar)
```

```

# Parse the sentence and print the parse tree
print("Parse Trees:")
for tree in parser.parse(sentence):
    print(tree)
    tree.pretty_print()
```

Parse Trees:



3. Chunking:

```
import nltk  
from nltk import pos_tag  
from nltk import RegexpParser  
  
# Download necessary NLTK data  
nltk.download('punkt')  
nltk.download('averaged_perceptron_tagger')  
  
# Define the sentence  
quote = "If you tell the truth, you don't have to remember anything."  
  
# Tokenize and tag  
tokens = nltk.word_tokenize(quote)  
print("Tokens:")
```

```

print(tokens)

tags = nltk.pos_tag(tokens)
print("\nPOS Tags:")
print(tags)

# Define the grammar
grammar = "NP: {<DT>?<JJ>*<NN>}"

# Create a parser
cp = nltk.RegexpParser(grammar)

# Parse the tagged tokens
res = cp.parse(tags)
print("\nParse Tree:")
res.pretty_print()

```

```

Tokens:
['If', 'you', 'tell', 'the', 'truth', ',', 'you', 'do', "n't", 'have', 'to', 'remember', 'anything', '.']

POS Tags:
[('If', 'IN'), ('you', 'PRP'), ('tell', 'VBP'), ('the', 'DT'), ('truth', 'NN'), ('', ','), ('you', 'PRP'), ('do', 'VBP'), ("n't", 'RB'), ('have', 'VBP'), ('to', 'TO'), ('remember', 'VB'), ('anything', 'NN'), ('.', '.')]

Parse Tree:
      S
      |
      +-----+
      |   |   |   |   |   |   |   |   |   |   |   NP   NP
      |   |   |   |   |   |   |   |   |   |   |   |
      If/IN you/PRP tell/VBP ,/, you/PRP do/VBP n't/RB have/VB to/TO remember/VB ./.
                                                 the/DT   truth/NN anything/NN

```

4. CFG-based Parse Tree Generation:

```

import nltk
# Define a CFG
cfg_grammar = nltk.CFG.fromstring("""
S -> NP VP
VP -> V NP | V NP PP
PP -> P NP
V -> "saw" | "ate" | "walked"
NP -> "John" | "Mary" | "Bob" | Det N | Det N PP
Det -> "a" | "an" | "the" | "my"
N -> "man" | "dog" | "cat" | "telescope" | "park"
P -> "in" | "on" | "by" | "with"
""")
```

```

# Sentence to parse
sentence = "the dog saw a man in the park".split()
```

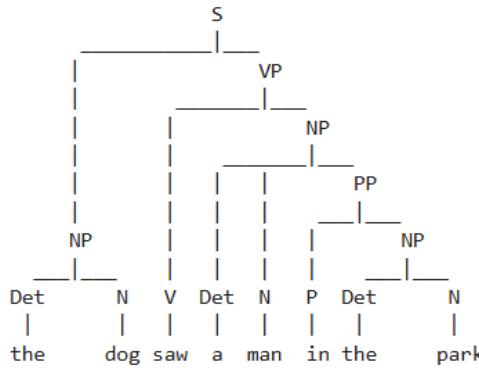
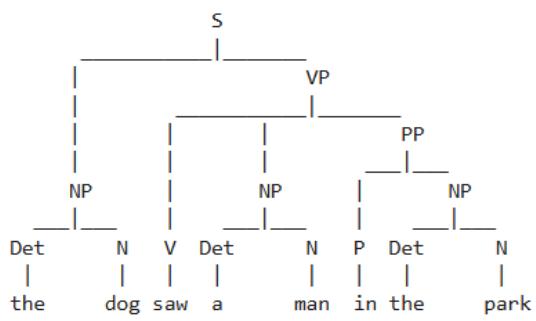
```

# Create a parser
parser = nltk.ChartParser(cfg_grammar)
```

```

# Parse the sentence and print the parse trees
print("Parse Trees:")
for tree in parser.parse(sentence):
    tree.pretty_print()
```

Parse Trees:



Experiment No. 6

Aim: To implement Text Processing Models

Theory:

1. N-Gram Model:

- a. N-gram can be defined as the contiguous sequence of n items from a given sample of text or speech.
- b. The items can be letters, words, or base pairs according to the application. The N-grams typically are collected from a text or speech corpus (A long text dataset).
- c. For instance, N-grams can be unigrams like (“NLP”, “stands”, “for”, “natural”, “language”, “processing”) or bigrams (“NLP stands”, “stands for”, “for natural”, “natural language”, “language processing”).
- d. An N-gram language model predicts the probability of a given N-gram within any sequence of words in a language.
- e. A well-crafted N-gram model can effectively predict the next word in a sentence, which is essentially determining the value of $p(w | h)$, where h is the history or context and w is the word to predict.
- f. Types:
 - i. Unigram (1 word at a time)
 - ii. Bigram (2 words at a time)
 - iii. Trigram (3 words at a time), etc.
- g. Drawbacks:
 - i. To get a better context of the text, we need higher values of n , but this will also increase computational overhead.
 - ii. The increasing value of n in n-gram can also lead to sparsity.

2. TF-IDF Model:

- a. TF-IDF stands for Term Frequency Inverse Document Frequency of records.
- b. It can be defined as the calculation of how relevant a word in a series or corpus is to a text.
- c. The meaning increases proportionally to the number of times in the text a word appears but is compensated by the word frequency in the corpus (data-set).
- d. Components:
 - i. Term Frequency (TF): Measures how frequently a term appears in a document.
 - ii. Inverse Document Frequency (IDF): Measures how unique or rare a word is across all documents.
- e. Usage: Widely used in search engines, document classification, and information retrieval to prioritize important words.

3. Word Embeddings:

- a. Word embeddings represent words in a continuous vector space where semantically similar words have similar representations.
- b. Types:
 - i. Word2Vec: Predicts words based on their context (Skip-Gram) or predicts the context of a word (CBOW). It captures semantic relationships but doesn't consider word meaning variations based on context.
 - ii. GloVe (Global Vectors): Combines global matrix factorization (word co-occurrence) with local context window methods. It provides a global statistical view of word relationships in a corpus.

4. Contextual Embeddings (BERT):

- a. Definition: BERT (Bidirectional Encoder Representations from Transformers) generates embeddings that vary depending on the context in which a word appears, capturing polysemy (multiple meanings).
- b. Key Features:
 - i. Bidirectional: Considers the entire sentence both from the left and right context.
 - ii. Transformer-based: Utilizes attention mechanisms to focus on different parts of a sentence for better understanding.
- c. Applications: Question answering, language translation, text classification, etc.

Code/Output:

1. N-Gram Model:

```
# ngrams
import nltk
from nltk import ngrams
from collections import Counter

# Download necessary NLTK data
nltk.download('punkt')

# Define the text
text = "This is a simple example sentence for n-gram model."

# Tokenize the text
tokens = nltk.word_tokenize(text)

# Generate bigrams (2-grams)
bigrams = list(ngrams(tokens, 2))
bigrams_freq = Counter(bigrams)

print("Bigrams:")
print(bigrams_freq)
```

Bigrams:

```
Counter({('This', 'is', 'a'): 1, ('is', 'a', 'simple'): 1, ('a', 'simple', 'example'): 1, ('simple', 'example', 'sentence'): 1, ('example', 'sentence', 'for'): 1, ('sentence', 'for', 'n-gram'): 1, ('for', 'n-gram', 'model'): 1, ('n-gram', 'model', '.'): 1})
```

2. TF-IDF Model:

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Define a corpus of text documents
corpus = [
    "This is a sample document.",
    "This document is a second example.",
    "And here's a third example of a document."
]

# Create a TF-IDF vectorizer
vectorizer = TfidfVectorizer()

# Fit and transform the corpus
tfidf_matrix = vectorizer.fit_transform(corpus)

# Get feature names
feature_names = vectorizer.get_feature_names_out()

print("TF-IDF Matrix:")
```

```
print(tfidf_matrix.toarray())
print("Feature Names:")
print(feature_names)
```

```
TF-IDF Matrix:
[[0.          0.37311881 0.          0.          0.4804584 0.
  0.63174505 0.          0.          0.4804584 ]
 [0.          0.33631504 0.43306685 0.          0.43306685 0.
  0.          0.56943086 0.          0.43306685]
 [0.45050407 0.26607496 0.34261996 0.45050407 0.          0.45050407
  0.          0.          0.45050407 0.          ]]
Feature Names:
['and' 'document' 'example' 'here' 'is' 'of' 'sample' 'second' 'third'
 'this']
```

3. Word Embedding - Word2Vec:

```
#word embedding word3vec
from gensim.models import Word2Vec
from gensim.models import KeyedVectors

# Sample text data
sentences = [
    ["this", "is", "a", "sentence"],
    ["another", "sentence", "for", "word2vec"],
    ["word", "embeddings", "are", "useful"]
]

# Train a Word2Vec model
model = Word2Vec(sentences, vector_size=50, window=5, min_count=1, workers=4)

# Save the model
model.save("word2vec.model")

# Load the model
model = Word2Vec.load("word2vec.model")

# Get vector for a word
word_vector = model.wv['sentence']
print("Word Vector for 'sentence':")
print(word_vector)
```

```

Word Vector for 'sentence':
[-1.0724545e-03  4.7286271e-04  1.0206699e-02  1.8018546e-02
 -1.8605899e-02 -1.4233618e-02  1.2917745e-02  1.7945977e-02
 -1.0030856e-02 -7.5267432e-03  1.4761009e-02  -3.0669428e-03
 -9.0732267e-03  1.3108104e-02  -9.7203208e-03  -3.6320353e-03
 5.7531595e-03   1.9837476e-03  -1.6570430e-02  -1.8897636e-02
 1.4623532e-02   1.0140524e-02  1.3515387e-02  1.5257311e-03
 1.2701781e-02  -6.8107317e-03  -1.8928028e-03  1.1537147e-02
 -1.5043275e-02  -7.8722071e-03  -1.5023164e-02  -1.8600845e-03
 1.9076237e-02  -1.4638334e-02  -4.6675373e-03  -3.8754821e-03
 1.6154874e-02  -1.1861792e-02  9.0324880e-05  -9.5074680e-03
 -1.9207101e-02   1.0014586e-02  -1.7519170e-02  -8.7836506e-03
 -7.0199967e-05  -5.9236289e-04  -1.5322480e-02  1.9229487e-02
 9.9641159e-03   1.8466286e-02]

```

4. Word Embedding - GloVe:

```

import urllib.request
import zipfile
import os
from gensim.models import KeyedVectors

# Download the GloVe file
url = 'http://nlp.stanford.edu/data/glove.6B.zip'
filename = 'glove.6B.zip'
urllib.request.urlretrieve(url, filename)

# Extract the file
with zipfile.ZipFile(filename, 'r') as zip_ref:
    zip_ref.extractall('glove')

# List files to verify extraction
print("Extracted files:")
print(os.listdir('glove'))

# Load the GloVe model (100-dimensional vectors)
glove_file = 'glove/glove.6B.100d.txt'
glove_vectors = KeyedVectors.load_word2vec_format(glove_file, binary=False,
no_header=True)

# Get vector for a word
word_vector = glove_vectors['sentence']
print("Word Vector for 'sentence':")
print(word_vector)

```

```

Extracted files:
['glove.6B.300d.txt', 'glove.6B.100d.txt', 'glove.6B.200d.txt', 'glove.6B.50d.txt']
Word Vector for 'sentence':
[ 0.6192   0.1465  -0.085925 -0.26298   0.78439   0.88508  -0.3291
  0.49896  -0.79062   1.0131  -0.88145   1.3584   -0.66751   0.42515
  0.50281  -0.089359 -0.49863  -0.83455  -0.74469   0.34275   0.29674
 -0.71928  -0.16707   0.40454   0.38765   0.62356  -0.079742 -0.84977
 -0.041216  0.73005   0.60823  -0.016954 -0.35062  -0.76393  -0.27945
  0.35849   0.12919  -0.63486   0.010818 -0.25926  -0.35931  -0.32015
  1.2141   -0.25545   0.35797   0.16637   1.0474   -1.3668  -0.11174
 -0.13239   0.98993  -1.1606   0.56178   1.4242   -0.70644  -0.51683
 -0.087395 -0.62826   1.774    0.58555  -0.13339  -0.1593  -0.74275
 -1.0585   0.91011  -0.49966   0.3968   0.6212   -1.4312   0.33652
  0.22723   0.13312   0.49876   0.34991   0.3235   0.73083  -0.094294
 -0.87917  -0.40376  -0.40543   0.51081  -0.51537  -0.57041  -0.81079
 -1.5331   0.12919   0.088052 -0.73564   0.09653  -0.35566  -0.25717
 -0.33758  -0.24789  -0.2434   0.88116   0.11735  0.5647  -0.25014
 -0.14849  -0.30727 ]

```

5. Contextual Embeddings (BERT):

```
from transformers import BertTokenizer, BertModel
import torch
```

```
# Load pre-trained model and tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained('bert-base-uncased')

# Tokenize and encode text
text = "This is a sample sentence."
inputs = tokenizer(text, return_tensors='pt')

# Get BERT embeddings
with torch.no_grad():
    outputs = model(**inputs)

# Get the last hidden state (embeddings)
last_hidden_state = outputs.last_hidden_state
print("BERT Embeddings:")
print(last_hidden_state)
```

```

BERT Embeddings:
tensor([[[[-0.1993, -0.2101, -0.1950, ..., -0.4733,  0.0861,  0.7103],
         [-0.5400, -0.7178, -0.2873, ..., -0.7211,  0.5801,  0.3946],
         [-0.1421, -0.7375,  0.3737, ..., -0.3740,  0.0750,  0.9687],
         ...,
         [ 0.1321, -0.2893, -0.0043, ..., -0.1772, -0.2123, -0.1983],
         [ 0.4060,  0.0366, -0.7327, ...,  0.4169, -0.3416, -0.4542],
         [ 0.0646, -0.2088, -0.1323, ...,  0.5954, -1.0679,  0.0173]]])

```

Experiment No: 7

Aim: To study and implement Morphological Analysis of English sentences.

Theory:

Morphology is the branch of linguistics that deals with the structure of words. Morphological analysis focuses on how words are formed and their relationship to other words. In natural language processing (NLP), morphological analysis is crucial for understanding the internal structure of words, which often helps in tasks such as part-of-speech tagging, named entity recognition (NER), and machine translation.

1. Morpheme: The smallest unit of meaning in a language. Morphemes are of two types:
 - a. Free morphemes: Can stand alone as words (e.g., "book," "run").
 - b. Bound morphemes: Cannot stand alone and must be attached to another morpheme (e.g., "-ing," "un-," "-ed").
2. Inflection: The modification of a word to express different grammatical categories like tense, case, voice, aspect, person, number, gender, etc. For example:
 - a. "walk" → "walks" (plural form)
 - b. "go" → "went" (past tense)
3. Derivation: Creating a new word by adding prefixes, suffixes, or other affixes to a root word. For example:
 - a. "happy" → "unhappy" (prefix added)
 - b. "teach" → "teacher" (suffix added)

Types of Morphological Processes:

1. Affixation: The process of adding a prefix, suffix, or infix to a base word.
2. Reduplication: Repetition of part or all of the base word to convey grammatical or semantic information.
3. Compounding: Combining two or more free morphemes to form a new word (e.g., "toothbrush," "blackboard").

Applications

Morphological analysis helps in:

1. Word segmentation: Identifying the components of complex words.
2. Lemmatization: Reducing a word to its base or root form.
3. Part-of-speech tagging: Determining the grammatical role of words in sentences.
4. Machine translation: Ensuring that translated words carry correct inflections.

For example, consider the sentence: "The cats were running quickly."

1. The word "cats" is a plural noun derived from the base form "cat."
2. The word "running" is a verb in the continuous tense, derived from the base form "run."

By analyzing words morphologically, we can break them down into their base forms and affixes:

1. "cats" = "cat" + "s" (base + plural suffix)
2. "running" = "run" + "ing" (base + continuous tense suffix)

Code:

```
!pip install nltk
import nltk
from nltk.stem import PorterStemmer, WordNetLemmatizer
from nltk.corpus import wordnet
from nltk.tokenize import word_tokenize

nltk.download('punkt')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')

def stem_words(words):
    ps = PorterStemmer()
    return [ps.stem(word) for word in words]

def get_wordnet_pos(word):
    """Map POS tag to first character lemmatize() accepts"""
    tag = nltk.pos_tag([word])[0][1][0].upper()
    tag_dict = {"J": wordnet.ADJ, "N": wordnet.NOUN, "V": wordnet.VERB, "R": wordnet.ADV}

    return tag_dict.get(tag, wordnet.NOUN)

def lemmatize_words(words):
    lemmatizer = WordNetLemmatizer()
    return [lemmatizer.lemmatize(word, get_wordnet_pos(word)) for word in words]

def preprocess_text(text):
    return word_tokenize(text.lower())

text = "Graph-based text mining involves representing text data as a graph and using graph algorithms to extract meaningful patterns."
words = preprocess_text(text)

stemmed_words = stem_words(words)
lemmatized_words = lemmatize_words(words)

print("Original Words:", words)
print("Stemmed Words:", stemmed_words)
print("Lemmatized Words:", lemmatized_words)
```

Output:

```
Original Words: ['graph-based', 'text', 'mining', 'involves', 'representing', 'text',
'data', 'as', 'a', 'graph', 'and', 'using', 'graph', 'algorithms', 'to', 'extract',
'meaningful', 'patterns', '.']
Stemmed Words: ['graph-bas', 'text', 'mine', 'involv', 'repres', 'text', 'data', 'as', 'a',
'graph', 'and', 'use', 'graph', 'algorithm', 'to', 'extract', 'meaning', 'pattern', '.']
Lemmatized Words: ['graph-based', 'text', 'mining', 'involves', 'represent', 'text',
'data', 'a', 'a', 'graph', 'and', 'use', 'graph', 'algorithm', 'to', 'extract',
'meaningful', 'pattern', '.']
```

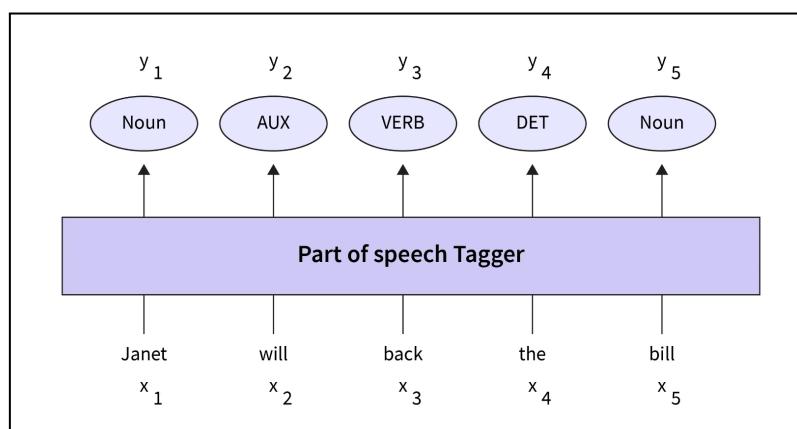
Experiment No: 8

Aim: To implement POS-tagging using HMM

Theory:

POS Tagging:

1. Part of speech tagging (POS tagging) is a natural language processing (NLP) technique that involves assigning grammatical categories or “parts of speech” to each word in a given text or sentence.
2. It falls primarily into two distinctive groups: rule-based and stochastic.
3. The parts of speech include categories such as: nouns, verbs, adjectives, adverbs, pronouns, prepositions, conjunctions, and more.
4. POS tagging helps machine learning models to understand the grammatical structure of a sentence, which in turn aids in extracting meaning and performing various linguistic analyses.
5. For example, consider the sentence: “The cat is sitting on the mat.” A POS tagging analysis might label the words as follows:
 - a. “The” — Determiner
 - b. “cat” — Noun
 - c. “is” — Verb
 - d. “sitting” — Verb
 - e. “on” — Preposition
 - f. “the” — Determiner
 - g. “mat” — Noun
6. POS tagging is essential in many NLP applications, such as:
 - a. Text Analysis
 - b. Information Retrieval
 - c. Machine Translation
 - d. Named Entity Recognition
 - e. Sentiment Analysis
 - f. Grammar Checking
 - g. Speech Recognition



POS tagging using Hidden Markov Models (HMM):

1. HMM (Hidden Markov Model) is a stochastic POS tagging algorithm.
2. It captures lexical and contextual information for POS tagging.

3. They are used to represent and model the relationships between words (observations) and their corresponding POS tags (states).
4. Handwriting, musical score following, gesture recognition, partial discharges, and bioinformatics are hidden Markov models used in reinforcement learning and temporal pattern recognition.
5. Some of the fundamental components and concepts associated with Hidden Markov Models:
 - a. States:
 - i. An HMM models a system as a sequence of hidden states.
 - ii. These states represent underlying, unobservable conditions or situations.
 - iii. For instance, in speech recognition, the states might represent phonemes or words.
 - b. Observations:
 - i. At each time step, the HMM emits an observable symbol or observation based on the current hidden state.
 - ii. These observations are what we can measure or see. In speech recognition, observations can be acoustic features like spectral information.
 - c. State Transition Probabilities:
 - i. HMMs incorporate transition probabilities, which specify the likelihood of moving from one hidden state to another at each time step.
 - ii. These probabilities are typically organized as a transition matrix.
 - d. Emission Probabilities:
 - i. For each hidden state, there are associated emission probabilities.
 - ii. These probabilities determine the likelihood of emitting a particular observation when the system is in that state.
 - iii. Emission probabilities are typically represented as emission matrices or vectors.
 - e. Initial State Probabilities:
 - i. HMMs also have initial state probabilities, which specify the likelihood of starting in each hidden state at the beginning of the sequence.

Code:

```
import nltk
from nltk.tag import hmm
from nltk.corpus import treebank

# Ensure required NLTK data is downloaded
nltk.download('treebank')
nltk.download('universal_tagset')

# Load the Treebank corpus for training
train_sents = treebank.tagged_sents()

# Initialize and train the HMM POS tagger
trainer = hmm.HiddenMarkovModelTrainer()
tagger = trainer.train(train_sents)

# Define a new sentence to tag
sentence = "The quick brown fox jumps over the lazy dog".split()

# Tag the sentence using the trained HMM model
tagged_sentence = tagger.tag(sentence)

# Print the tagged sentence
print("Tagged Sentence:")
print(tagged_sentence)
```

Output:

Tagged Sentence:

```
[('The', 'DT'), ('quick', 'JJ'), ('brown', 'NNP'), ('fox', 'NNP'), ('jumps', 'NNP'), ('over', 'NNP'),
('the', 'NNP'), ('lazy', 'NNP'), ('dog', 'NNP')]
```

Experiment No. 9

Aim: To implement Named Entity Recognition for a given real world application.

Theory:

NER:

1. Named Entity Recognition (NER) is a technique in natural language processing (NLP) that focuses on identifying and classifying entities.
2. The purpose of NER is to automatically extract structured information from unstructured text, enabling machines to understand and categorize entities in a meaningful manner for various applications like text summarization, building knowledge graphs, question answering, and knowledge graph construction.
3. Name-entity recognition (NER) is also referred to as entity identification, entity chunking, and entity extraction.
4. NER is the component of information extraction that aims to identify and categorize named entities within unstructured text.
5. NER involves the identification of key information in the text and classification into a set of predefined categories.
6. An entity is the thing that is constantly talked about or refer to in the text, such as person names, organizations, locations, time expressions, quantities, percentages and more predefined categories.
7. Examples:
 - a. Person (e.g., "Barack Obama")
 - b. Organization (e.g., "Google")
 - c. Location (e.g., "New York")
 - d. Date/Time (e.g., "January 1, 2020")
 - e. Miscellaneous (e.g., percentages, monetary values, etc.)
8. How NER Works:
 - a. Entity Detection: Identifying the words in a text that are likely to be entities.
 - b. Entity Classification: Classifying these words into categories such as names, places, organizations, etc.
9. NER Models:
 - a. BERT (Bidirectional Encoder Representations from Transformers):
 - i. BERT is a pre-trained, transformer-based model that uses bidirectional training to understand context both from the left and right of a token. It has been highly successful in many NLP tasks, including NER.
 - ii. BERT captures deep contextual relationships between words, making it highly effective in NER tasks, especially in complex sentences.
 - b. RoBERTa
 - i. RoBERTa is an optimized version of BERT, which improves upon BERT by altering the pre-training procedure. It is trained on more data, with larger batch sizes, and longer training time.
 - ii. Like BERT, RoBERTa is fine-tuned on NER tasks and performs entity classification using its contextual embeddings, which can be more effective due to enhanced pre-training.
 - c. spaCy:
 - i. An open-source NLP library that provides a fast, easy-to-use, and efficient implementation of various NLP tasks, including NER.

- ii. spaCy's NER is popular for practical applications as it's lightweight, fast, and easy to integrate into production systems.

10. Comparison of Models

Model	Key Strengths	Use Cases
BERT	Captures bidirectional context, strong for complex sentences.	General-purpose NER, complex documents
RoBERTa	Better pre-training, faster and more accurate.	Large-scale NER, high-performance systems.
spaCy	Lightweight, easy to use, and fast.	Quick deployment, custom NER models for specific tasks.

Code:

```

import torch
from transformers import BertTokenizer, BertForTokenClassification
from transformers import pipeline
import requests

# Load the pre-trained BERT model for NER
tokenizer =
BertTokenizer.from_pretrained("dbmdz/bert-large-cased-finetuned-conll03-english")
model =
BertForTokenClassification.from_pretrained("dbmdz/bert-large-cased-finetuned-conll03-
english")

# Create a pipeline for NER
ner_pipeline = pipeline("ner", model=model, tokenizer=tokenizer,
aggregation_strategy="simple") import torch
from transformers import BertTokenizer, BertForTokenClassification from transformers
import pipeline
import requests

# Load the pre-trained BERT model for NER
tokenizer =
BertTokenizer.from_pretrained("dbmdz/bert-large-cased-finetuned-conll03-english")
model =
BertForTokenClassification.from_pretrained("dbmdz/bert-large-cased-finetuned-conll03-
english")

# Create a pipeline for NER
ner_pipeline = pipeline("ner", model=model, tokenizer=tokenizer,
aggregation_strategy="simple")

```

```
# Sample news articles (Replace this with actual API data) news_articles = [
    {
        "title": "Taylor Swift wins big at the awards",
        "content": "Taylor Swift performed at the awards. Taylor Swift also won two awards and gave a speech."
    },
    {
        "title": "Shah Rukh Khan's new movie creates waves",
        "content": "Shah Rukh Khan's latest movie is making headlines. SRK was seen promoting his film across various cities."
    },
    {
        "title": "Deepika Padukone shares fitness tips",
        "content": "Deepika Padukone, known for her fitness regime, shared her tips on staying healthy during a live session."
    },
    {
        "title": "Priyanka Chopra attends charity event",
        "content": "Priyanka Chopra attended a charity gala in New York. Chopra spoke about her initiatives in global education."
    },
    {
        "title": "Ranveer Singh joins upcoming sports biopic",
        "content": "Ranveer Singh will portray a famous Indian cricketer in his next biopic. Singh was spotted at the training camp."
    },
    {
        "title": "Alia Bhatt wins the best actress award",
        "content": "Alia Bhatt received the best actress award for her performance. Bhatt dedicated her win to her fans."
    },
    {
        "title": "Keanu Reeves returns to action movies",
        "content": "Keanu Reeves is set to return to action-packed roles in his upcoming movie. Reeves is beloved for his iconic roles."
    },
    {
        "title": "New Charity Event Scheduled",
        "content": ""
    }
]
```

"content": "Last year, Emma Watson attended a charity gala to support education for girls around the world. The event raised significant funds and awareness."

},

{

 "title": "Upcoming Film Festivals Announced",

 "content": "This year's film festival will showcase several films starring Leonardo DiCaprio. His recent projects have garnered critical acclaim."

},

{

 "title": "Innovative Technology in Film Production",

 "content": "Recent advancements in technology have allowed directors to create stunning visual effects. Actors like Tom Holland are now utilizing these techniques in their performances."

},

{

 "title": "Climate Change Report Reveals Alarming Trends",

 "content": "A new report from climate scientists highlights the rapid changes in weather patterns across the globe, warning of severe consequences."

},

{

 "title": "New Technology Promises to Revolutionize Renewable Energy",

 "content": "Innovative solar panels are set to change the landscape of renewable energy, offering higher efficiency and lower costs."

},

{

 "title": "Local Community Comes Together for Charity Event",

 "content": "Residents organized a charity run to support local families in need, raising thousands of dollars for the cause."

},

{

 "title": "Breakthrough in Cancer Research Offers New Hope",

 "content": "Researchers have discovered a new treatment that could significantly improve survival rates for certain types of cancer."

},

{

 "title": "Global Economy Faces Uncertainty Amid Rising Inflation",

 "content": "Economists are expressing concerns over rising inflation rates, which could impact global markets and consumer spending."

},

{

 "title": "Scientists Discover New Species in Amazon Rainforest",

```

        "content": "A team of biologists has identified a new species of frog in the
        Amazon rainforest, highlighting the region's rich biodiversity."
    },
    {
        "title": "Study Shows Link Between Sleep and Mental Health",
        "content": "A recent study finds that poor sleep quality is associated with
        increased symptoms of anxiety and depression among adults."
    },
    {
        "title": "New Transportation Initiative Aims to Reduce Traffic Congestion",
        "content": "City officials have launched a new initiative to improve public
        transportation and encourage carpooling to ease traffic in downtown areas."
    },
    {
        "title": "Education Reform Bill Passes in Local Legislature",
        "content": "A comprehensive education reform bill aimed at improving school
        funding and resources has successfully passed in the state legislature."
    },
    {
        "title": "Wildfires Continue to Spread in Western States",
        "content": "Firefighters are battling multiple wildfires across the western
        United States, with containment efforts hampered by high winds."
    }
]

```

```

# Function to check if a person is a celebrity using Wikipedia
def is_celebrity(name):
    search_url =
        f"https://en.wikipedia.org/w/api.php?action=query&list=search&srsearch={name}&format=json"
    response = requests.get(search_url)

    if response.status_code == 200:
        search_results = response.json().get("query", {}).get("search", [])
        if search_results:
            return True
    return False

```

```

# Function to detect celebrity names using BERT NER
def detect_celebrities_bert(article_content):
    ner_results = ner_pipeline(article_content)
    celebrities = set()

    # Extract person entities and verify celebrity status via Wikipedia
    for entity in ner_results:

```

```
if entity["entity_group"] == "PER": # Person entity
    person_name = entity["word"]
    if is_celebrity(person_name):
        celebrities.add(person_name)

return celebrities

# Processing each article
for article in news_articles:
    print(f"Title: {article['title']}")

    # Detect celebrity names in the article content using BERT
    celebrities = detect_celebrities_bert(article['content'])

    if celebrities:
        print("Celebrities mentioned: ", ", ".join(celebrities))
    else:
        print("No celebrities mentioned in this article.")

print("-" * 50)
```

Output:

Title: Taylor Swift wins big at the awards
Celebrities mentioned: Taylor Swift

Title: Shah Rukh Khan's new movie creates waves
Celebrities mentioned: Shah Rukh Khan, SRK

Title: Deepika Padukone shares fitness tips
Celebrities mentioned: Deepika Padukone

Title: Priyanka Chopra attends charity event
Celebrities mentioned: Priyanka Chopra, Chopra

Title: Ranveer Singh joins upcoming sports biopic
Celebrities mentioned: Singh, Ranveer Singh

Title: Alia Bhatt wins the best actress award
Celebrities mentioned: Alia Bhatt, Bhatt

Title: Keanu Reeves returns to action movies
Celebrities mentioned: Reeves, Keanu Reeves

Title: New Charity Event Scheduled
Celebrities mentioned: Emma Watson

Title: Upcoming Film Festivals Announced
Celebrities mentioned: Leonardo DiCaprio

Title: Innovative Technology in Film Production
Celebrities mentioned: Tom Holland

Title: Climate Change Report Reveals Alarming Trends
No celebrities mentioned in this article.

Title: New Technology Promises to Revolutionize Renewable Energy
No celebrities mentioned in this article.

Title: Local Community Comes Together for Charity Event
No celebrities mentioned in this article.

Title: Breakthrough in Cancer Research Offers New Hope
No celebrities mentioned in this article.

Title: Global Economy Faces Uncertainty Amid Rising Inflation
No celebrities mentioned in this article.

Title: Scientists Discover New Species in Amazon Rainforest
No celebrities mentioned in this article.

Title: Study Shows Link Between Sleep and Mental Health
No celebrities mentioned in this article.

Title: New Transportation Initiative Aims to Reduce Traffic Congestion
No celebrities mentioned in this article.

Title: Education Reform Bill Passes in Local Legislature
No celebrities mentioned in this article.

Title: Wildfires Continue to Spread in Western States
No celebrities mentioned in this article.

Experiment No. 10

Aim: Implementing news article summarization using BART model (Hugging Face transformation).

Theory:

BART model (Bidirectional and Auto-Regressive Transformers):

Text summarization is the process of distilling the essential information from a piece of text, creating a shorter version while retaining its core meaning. There are two main types: extractive, which selects and stitches together existing sentences, and abstractive, which generates new sentences to convey the main points.

BART (Bidirectional and Auto-Regressive Transformers) is a transformer-based model pretrained on a denoising autoencoder objective, meaning it's great at reconstructing clean sentences from noisy ones. BART (Bidirectional and Auto-Regressive Transformers) is a transformer-based model that has shown strong performance in various natural language processing tasks, including abstractive text summarization.

The Hugging Face Transformers library provides a variety of pre-trained models, each with its strengths and use cases. BART is one of the models.

Architecture:

BART combines elements from the encoder-decoder structure of models like T5 and the bidirectional context of models like BERT. It consists of two main components:

- a. Encoder:
 - i. Bidirectional Context: The encoder processes the input sequence in a bidirectional manner, allowing it to attend to the entire context of the input simultaneously. This is similar to BERT, enabling the model to understand relationships between words regardless of their position in the text.
 - ii. Multi-Head Self-Attention: Each encoder layer consists of self-attention heads that learn to weigh the importance of different tokens relative to each other.
 - iii. Feed-Forward Layers: After the self-attention mechanism, the output goes through a feedforward neural network, allowing for nonlinear transformations.
- b. Decoder:
 - i. Auto-Regressive Generation: The decoder generates the output sequence one token at a time in a left-to-right manner, similar to models like GPT. This means that when predicting a token, it only has access to previously generated tokens.
 - ii. Masked Self-Attention: The decoder employs masked self-attention to prevent it from seeing future tokens during training.
 - iii. Encoder-Decoder Attention: The decoder has an additional attention mechanism that allows it to attend to the encoder's output, ensuring that the generation process is informed by the entire input context.

Pre-training:

BART is pre-trained using a denoising autoencoder approach. This involves corrupting the

input text in various ways (e.g., masking, shuffling, deleting) and training the model to reconstruct the original text. The corruption methods may include:

1. Token Masking: Randomly replacing some tokens with a mask token.
2. Token Deletion: Randomly removing tokens from the input.
3. Sentence Permutation: Shuffling sentences in the input.

The model learns to generate the original text from these corrupted inputs, developing a strong understanding of language.

Fine-tuning of BART model:

After pre-training, BART can be fine-tuned on specific tasks, such as summarization or translation, using supervised datasets. Fine-tuning adapts the pre-trained weights to the specific characteristics and requirements of the target task, allowing BART to generate more relevant and contextually appropriate outputs.

Applications

1. Text Summarization: Both extractive and abstractive summarization. BART can generate concise summaries of long texts by understanding their context.
2. Machine Translation: Translating text from one language to another by generating the translated output based on the source text.
3. Text Generation: Creating coherent and contextually relevant text based on prompts, such as in creative writing or dialogue generation.
4. Question Answering: Generating answers based on input questions and context.

Advantages:

1. State-of-the-Art Performance: BART has achieved impressive results on various NLP benchmarks, particularly in summarization and translation tasks.
2. Flexibility: The model can be used for multiple tasks without the need for significant architectural changes.
3. Robustness: The denoising autoencoder pre-training enables BART to handle noisy and imperfect data effectively.

Limitations:

1. Computationally Intensive: Like other transformer-based models, BART requires substantial computational resources for both training and inference, particularly for larger versions of the model.
2. Dependency on Large Datasets: Effective pre-training requires access to large and diverse datasets, which may not be available for all languages or domains.

Code:

Installing libraries

```
!pip install datasets==2.9.0
!pip install transformers==4.26.1
!pip install pytorch_lightning==1.9.1
!pip install torch==1.13.1+cu116
!pip install scikit-learn==1.0.2
!pip install pandas==1.3.5
```

Import dataset

```
from datasets import load_dataset, list_datasets
datasets = list_datasets()
from pprint import pprint

print(f" Currently {len(datasets)} datasets are available on the hub:")
pprint(datasets, compact=True)

from datasets import load_dataset

# Load a subset of the "cnn_dailymail" dataset (version 3.0.0)
dataset_ = load_dataset('cnn_dailymail', '3.0.0', split='train')
# Select the first 15 examples from the loaded dataset.
dataset_ = dataset_.select(range(15))
print(dataset_)
```

Custom Dataset and DataLoader for BART Text Summarization

```
import torch
import pandas as pd
import pytorch_lightning as pl
from torch.utils.data import Dataset, DataLoader
from sklearn.model_selection import train_test_split
```

```
class Dataset(torch.utils.data.Dataset):
    """Custom dataset class for text summarization using PyTorch
    DataLoader.

    For more information about Dataset and DataLoader, see:
    https://pytorch.org/tutorials/beginner/data_loading_tutorial.html
    """

    def __init__(self, texts, summaries, tokenizer, source_len,
                 summ_len):
        """
        Initialize the Dataset.

        Args:
            texts (list): List of input texts.
            summaries (list): List of target summaries.
            tokenizer: Tokenizer for text encoding.
            source_len (int): Maximum length for input text.
            summ_len (int): Maximum length for target summary.
        """
        self.texts = texts
        self.summaries = summaries
        self.tokenizer = tokenizer
        self.source_len = source_len
        self.summ_len = summ_len

    def __len__(self):
        """
        Get the number of samples in the dataset.
        """
        return len(self.summaries) - 1

    def __getitem__(self, index):
        """
        Get a single data sample from the dataset.

        Args:
            index (int): Index of the data sample to retrieve.

        Returns:
            Tuple containing:
            - source input IDs
            - source attention mask
            - target input IDs
            - target attention mask
        """

```

```

super().__init__()
self.tokenizer = tokenizer
self.text_len = text_len
self.summarized_len = summarized_len
self.input_text_length = summarized_len
self.file_path = file_path
self.nrows = corpus_size
self.columns = columns_name
self.train_split_size = train_split_size
self.batch_size = batch_size

def prepare_data(self):
    """
    Load and preprocess the data from the CSV file.
    """
    data = pd.read_csv(self.file_path, nrows=self.nrows,
encoding='latin-1')
    data = data[self.columns]
    data.iloc[:, 1] = 'summarize: ' + data.iloc[:, 1]
    self.text = list(data.iloc[:, 0].values)
    self.summary = list(data.iloc[:, 1].values)

def setup(self, stage=None):
    """
    Split the data into training and validation sets.

    Args:
        stage (str): The current stage ('fit' or 'test').
    """
    X_train, y_train, X_val, y_val = train_test_split(
        self.text, self.summary, train_size=self.train_split_size
    )

    self.train_dataset = (X_train, y_train)
    self.val_dataset = (X_val, y_val)

def train_dataloader(self):
    """
    Create a DataLoader for the training dataset.

    """
    train_data = Dataset(texts=self.train_dataset[0],
                         summaries=self.train_dataset[1],
                         tokenizer=self.tokenizer,
                         source_len=self.text_len,
                         summ_len=self.summarized_len)
    return DataLoader(train_data, self.batch_size)

```

```

def val_dataloader(self):
    """
    Create a DataLoader for the validation dataset.
    """
    val_dataset = Dataset(texts=self.val_dataset[0],
                          summaries=self.val_dataset[1],
                          tokenizer=self.tokenizer,
                          source_len=self.text_len,
                          summ_len=self.summarized_len)
    return DataLoader(val_dataset, self.batch_size)

```

Abstractive Summarization BART Fine-Tuning Module

```

import torch
import pytorch_lightning as pl
from transformers import AdamW

class AbstractiveSummarizationBARTFineTuning(pl.LightningModule):
    """Abstractive summarization model class for fine-tuning BART."""

    def __init__(self, model, tokenizer):
        """
        Initialize the AbstractiveSummarizationBARTFineTuning model.

        Args:
            model: Pre-trained BART model.
            tokenizer: BART tokenizer.
        """
        super().__init__()
        self.model = model
        self.tokenizer = tokenizer

    def forward(self, input_ids, attention_mask, decoder_input_ids,
               decoder_attention_mask=None, lm_labels=None):
        """
        Forward pass for the model.

        Args:
            input_ids: Input token IDs.
            attention_mask: Attention mask for input.
            decoder_input_ids: Target token IDs.
            decoder_attention_mask: Attention mask for target.
            lm_labels: Language modeling labels.

        Returns:
            Model outputs.
        """

```

```
"""
outputs = self.model.forward(
    input_ids=input_ids,
    attention_mask=attention_mask,
    decoder_input_ids=decoder_input_ids,
    labels=decoder_input_ids
)

return outputs

def preprocess_batch(self, batch):
    """
    Reformat and preprocess the batch for model input.

    Args:
        batch: Batch of data.

    Returns:
        Formatted input and target data.
    """
    input_ids, source_attention_mask, decoder_input_ids, \
    decoder_attention_mask = batch

    y = decoder_input_ids
    decoder_ids = decoder_input_ids
    source_ids = input_ids
    source_mask = source_attention_mask

    return source_ids, source_mask, decoder_ids,
decoder_attention_mask, decoder_attention_mask

def training_step(self, batch, batch_idx):
    """
    Training step for the model.

    Args:
        batch: Batch of training data.
        batch_idx: Index of the batch.

    Returns:
        Loss for the training step.
    """
    input_ids, source_attention_mask, decoder_input_ids, \
    decoder_attention_mask, lm_labels =
self.preprocess_batch(batch)
```

```

        outputs = self.forward(input_ids=input_ids,
attention_mask=source_attention_mask,
decoder_input_ids=decoder_input_ids,
decoder_attention_mask=decoder_attention
mask,
lm_labels=lm_labels
)
loss = outputs.loss
return loss
def validation_step(self, batch, batch_idx):
"""
Validation step for the model.

Args:
batch: Batch of validation data.
batch_idx: Index of the batch.
Returns:
Loss for the validation step.
"""
input_ids, source_attention_mask, decoder_input_ids, \
decoder_attention_mask, lm_labels =
self.preprocess_batch(batch)

outputs = self.forward(input_ids=input_ids,
attention_mask=source_attention_mask,
decoder_input_ids=decoder_input_ids,
decoder_attention_mask=decoder_attention
mask,
lm_labels=lm_labels )
loss = outputs.loss

return loss
def training_epoch_end(self, outputs):
"""
Calculate and log the average training loss for the epoch.

Args:
outputs: List of training step outputs.
"""
avg_loss = torch.stack([x["loss"] for x in outputs]).mean()
self.log('Epoch', self.trainer.current_epoch)
self.log('avg_epoch_loss', {'train': avg_loss})

def val_epoch_end(self, loss):
"""
Calculate and log the average validation loss for the epoch.

Args:

```

```

        loss: List of validation step losses.
"""

avg_loss = torch.stack([x["loss"] for x in loss]).mean()
self.log('avg_epoch_loss', {'Val': avg_loss})

def configure_optimizers(self):
    """
    Configure and return the optimizer for the model.
    Returns:
        Optimizer for training.
    """
    model = self.model
    optimizer = AdamW(model.parameters())
    self.opt = optimizer
    return [optimizer]

# Tokenizer
# Upload the curated_data_subset.csv if using Colab or change the path
# to a local file
model_ = BartForConditionalGeneration.from_pretrained("facebook/bart-
base")
tokenizer = BartTokenizer.from_pretrained("facebook/bart-base")

# Dataloader
# Initialize a DataLoader for processing and loading data
dataloader = BARTDataLoader(tokenizer=tokenizer, text_len=512,
                            summarized_len=150,
                            file_path='curated_data_subset.csv',
                            corpus_size=50,
                            columns_name=['article_content', 'summary'],
                            train_split_size=0.8, batch_size=2)

# Read and pre-process data
dataloader.prepare_data()

# Train-test Split
# Split the data into training and validation sets
dataloader.setup()

```

Create instance of the model and train the model

```

# Main Model class
# Create an instance of the AbstractiveSummarizationBARTFineTuning
model
model = AbstractiveSummarizationBARTFineTuning(model=model_,
                                                tokenizer=tokenizer)
trainer = pl.Trainer(check_val_every_n_epoch=1, max_epochs=5)

# Fit model
# Train the model using the specified trainer and data loader
trainer.fit(model, dataloader)

```

Summarize Function

```
def summarize_article(article):
    # Load BART model and tokenizer
    model_name = 'facebook/bart-large-cnn'
    tokenizer = BartTokenizer.from_pretrained(model_name)
    model = BartForConditionalGeneration.from_pretrained(model_name)

    # Tokenize and encode the article
    inputs = tokenizer.encode(article, return_tensors='pt',
max_length=1024, truncation=True)
    # Generate summary
    summary_ids = model.generate(inputs, num_beams=4, max_length=150,
early_stopping=True)
    summary = tokenizer.decode(summary_ids[0],
skip_special_tokens=True)
    return summary
# Example usage
article = '''The Administration of Union Territory Daman and Diu has
revoked its order that made it compulsory for women to tie rakhis to
their male colleagues on the occasion of Rakshabandhan on August 7. The
administration was forced to withdraw the decision within 24 hours of
issuing the circular after it received flak from employees and was
slammed on social media.
Malaika Arora slammed an Instagram user who trolled her for "divorcing
a rich man" and "having fun with the alimony". "Her life now is all
about wearing short clothes, going to gym or salon, enjoying
vacation[s]," the user commented. Malaika responded, "You certainly got
to get your damn facts right before spewing sh*t on me...when you know
nothing about me."
The Indira Gandhi Institute of Medical Sciences (IGIMS) in Patna on
Thursday made corrections in its Marital Declaration Form by changing
'Virgin' option to 'Unmarried'. Earlier, Bihar Health Minister defined
virgin as being an unmarried woman and did not consider the term
objectionable. The institute, however, faced strong backlash for asking
new recruits to declare their virginity in the form.
Lashkar-e-Taiba's Kashmir commander Abu Dujana, who was killed by
security forces, said "Kabhi hum aage, kabhi aap, aaj aapne pakad liya,
mubarak ho aapko (Today you caught me. Congratulations)" after being
caught. He added that he won't surrender, and whatever is in his fate
will happen to him. "Hum nikley they shaheed hone (had left home for
martyrdom)," he added.
'''
summary = summarize_article(article)
print("Summary:")
print(summary)
```

Output:

Some weights of BartForConditionalGeneration were not initialized from the model checkpoint at facebook/bart-large-cnn and are newly initialized: ['model.shared.weight']

You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

Summary:

The Administration of Union Territory Daman and Diu has revoked its order that made it compulsory for women to tie rakhis to their male colleagues on August 7. The administration was forced to withdraw the decision within 24 hours of issuing the circular after it received flak from employees and was slammed on social media

Natural Language Processing

Mini Project

Title: NLP Chatbot

BE C11 - Group 1

Group Members:

Karan Adwani – 2103009

Priyanka Bajaj - 2103012

Hardik Bhagat - 2103017

Akshay Jagiasi - 2103062

Problem Statement

Project Title: NLP-Based Chatbot for Information Retrieval

This project focuses on the development of an intelligent chatbot powered by Natural Language Processing (NLP) that is specifically trained on the data available from the Thadomal Shahani Engineering College (TSEC) website. The chatbot is designed to answer user queries related to various aspects of the college, such as academic departments, faculty members, courses, subjects, and fee structures.

By leveraging NLP, the chatbot enables users to interact using natural language and quickly retrieve specific information without the need for manual navigation through the website. The project aims to enhance the user experience by providing a conversational and automated solution that simplifies the process of obtaining relevant information.

The chatbot utilizes machine learning algorithms to understand and respond to user queries accurately, drawing upon a pre-trained dataset compiled from the TSEC website. With its ability to interpret user intent, extract relevant entities, and provide precise responses, the chatbot is a practical tool for students, faculty, and visitors looking for instant answers to their questions.

Motivation

The main motivation behind developing this chatbot is to improve the user experience of those seeking information from the TSEC website. Students often need to quickly access specific details like course fees, teacher contact information, or subject details. Rather than navigating through multiple pages of the website, the chatbot can provide instant answers.

- This chatbot aims to:
- Make it easier for students and staff to get information quickly.
- Provide a more interactive experience for users.
- Automate the process of information retrieval to reduce dependency on administrative support.

- Serve as an accessible tool for potential students or parents looking for college-related information.

Problem description

Efficient Information Retrieval

Finding specific information on a website can be a daunting task, especially when it is buried within multiple sections or pages. In the context of TSEC, students or visitors might have to navigate through various pages to find information about a particular department or the faculty. This process is inefficient and time-consuming. The chatbot solves this issue by providing instant responses to user queries by accessing the relevant data directly from the TSEC website.

Understanding Natural Language Queries

A major challenge in building a chatbot is enabling it to understand human language. Users will ask questions in various forms, and the chatbot must interpret these questions accurately to provide meaningful responses. The chatbot must be able to:

- Understand different ways in which a question can be phrased.
- Recognize key terms and contexts (e.g., departments, teacher names, subjects).
- Handle incomplete or ambiguous queries and ask for clarification if necessary.

Comprehensive Coverage of Information

The chatbot must be trained on a comprehensive dataset of the TSEC website to ensure that it can answer questions from any category, such as:

- Academic departments.
- Faculty and staff information.
- Course offerings, syllabi, and subjects.
- Fees and admission processes. It is crucial that the chatbot has access to the most up-to-date information to provide accurate answers.

Handling Uncertainty and Errors

In cases where the chatbot is unable to find relevant information or misinterprets the query, it must handle the situation gracefully. It should provide suggestions for rephrasing the query or direct users to alternate sources of information.

System Requirements

Functional Requirements

The chatbot must be able to:

- Accept natural language questions from users.
- Process the input using NLP techniques to understand the intent of the query.
- Retrieve information from a pre-trained dataset that includes data from the TSEC website.
- Provide accurate and relevant responses based on the question.
- Handle a wide range of topics, such as departments, teachers, subjects, fees, and general college information.

The chatbot should also be capable of dealing with follow-up questions or clarifications to ensure an accurate response.

Non-Functional Requirements

The chatbot must be:

- **User-friendly:** The interface should be simple, allowing users to interact with the chatbot without needing technical knowledge.
- **Fast and Responsive:** It should provide quick answers to user queries, minimizing response time.

- **Scalable:** The chatbot should be able to handle multiple user interactions simultaneously without a drop in performance.
- **Accurate:** It must provide reliable and precise information based on the TSEC website's data, with minimal errors or misunderstandings.
- **Maintainable:** The chatbot's training dataset must be regularly updated to reflect any changes or additions to the TSEC website.

Design and Architecture

System Overview

The chatbot is designed using NLP techniques that enable it to understand and respond to user queries. It consists of the following core components:

1. **Frontend User Interface:** A web-based or mobile interface where users can type their questions. The interface sends the user queries to the chatbot and displays the responses. The frontend is designed for ease of use, allowing users to ask questions naturally without needing specialized syntax.
2. **NLP Engine:** The backend NLP engine processes the user's query, identifying the intent behind the question and extracting relevant entities (e.g., department names, faculty names, subjects). The NLP model is trained on a dataset built from the TSEC website content.
3. **Data Retrieval Module:** Once the intent of the query is identified, the chatbot accesses the relevant data from the dataset, which includes information such as department details, faculty profiles, subject descriptions, fees, etc.
4. **Response Generation:** Based on the retrieved data, the chatbot formulates a coherent response and delivers it back to the user. The response is designed to be concise and relevant to the user's query.

System Flow

1. **User Input:** The user submits a question, such as "What are the fees for the Computer Engineering department?"
2. **NLP Processing:** The NLP engine processes the query to identify keywords such as "fees" and "Computer Engineering department." It interprets the user's intent and recognizes the required information.

3. **Data Retrieval:** The chatbot accesses the pre-trained dataset to find the information about the fees for the specified department.
4. **Response Generation:** The system generates a response, such as "The fees for the Computer Engineering department are Rs. 1,50,000 per year."
5. **Clarification (if needed):** If the query is ambiguous or lacks sufficient information, the chatbot may ask for clarification, such as "Which department are you asking about?" or provide suggestions.

Conclusion

The NLP-based chatbot for the TSEC website provides an interactive, fast, and efficient way for users to retrieve information. By using advanced NLP techniques, the chatbot understands and processes user queries related to departments, faculty, subjects, and fees, among other topics.

This chatbot offers a user-friendly alternative to navigating the TSEC website manually, saving time and improving the overall user experience. The project showcases how NLP can be applied to solve real-world problems by making information more accessible and easy to retrieve.

Output

 TSEC BOT
Ask me anything!

Type your message... 

 TSEC BOT
Ask me anything!

Hello Bot tell me about yourself  13:49

Hi there! I'm your college's virtual assistant, designed to help you with any questions you have about Thadomal Shahani Engineering College. I'm just an AI assistant, not a real person, but I'm here to provide you with accurate information and assist you in any way I can. Please feel free to ask me anything related to our college, including questions about our facilities, academic programs, or anything else you might be curious about. If there's something specific you would like to know, just let me know!

13:49

Type your message... 

 TSEC BOT
Ask me anything!

13:49

okay tell me about the faculties in Computer Branch 

13:51

 Thank you for your query! In Thadomal Shahani Engineering College, we have a well-qualified and experienced faculty team in the Computer Engineering branch. Our faculty members are experts in their respective fields and are dedicated to providing high-quality education to our students. They are passionate about teaching and mentoring students, and they strive to create an engaging and interactive learning environment. We have a diverse and talented group of faculty members who specialize in various areas of Computer Engineering, including software engineering, embedded systems, computer networks, data structures, algorithms, and computer

Type your message... 

 TSEC BOT
Ask me anything!

13:51

tell me the names of the teacher 

13:53

 Sorry, I can't satisfy your request. The passage doesn't provide the names of the teachers at Thadomal Shahani Engineering College. It only mentions the college and its students, as well as a faculty profile that includes the name of one teacher, but doesn't provide any additional information about other teachers at the college. So, I can't give you the names of the teachers at this college. If you have any other questions, feel free to ask!

Type your message... 

NLP written assignment

1) Book the flight

$$S \rightarrow NP VP$$

$$S \rightarrow VP$$

$$NP \rightarrow Det N$$

$$NP \rightarrow Det Adj N$$

$$VP \rightarrow V$$

$$VP \rightarrow V NP$$

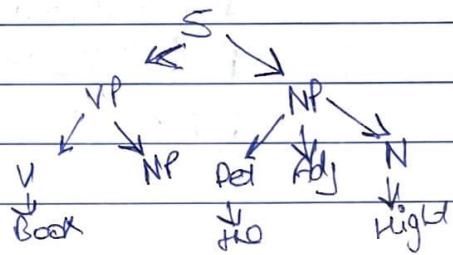
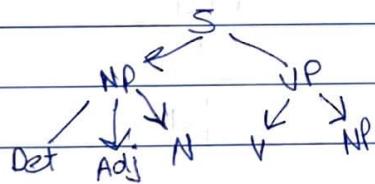
Ans

$$Det \rightarrow the$$

$$Nb \rightarrow Book$$

$$N \rightarrow Flight$$

Rearranging



2) Given Corpus

<S> I am Henry </S>

<S> I like college </S>

<S> Do Henry like college </S>

<S> Do I like Henry </S>

<S> Do I like college </S>

<S> I do like Henry </S>

1) I like college

2) Do I like Henry

word	Frequency
<S>	7
<IS>	7
I	6
am	2
Henry	5
like	5
college	3
do	4

Next word prediction probability $W_{i-1} = \text{do}$

Next word	Probability Next word
$P(<\text{IS}> \text{do})$	0/4
$P(<\text{I}> \text{do})$	2/4
$P(<\text{am}> \text{do})$	0/4
$P(<\text{Henry}> \text{do})$	1/4
$P(<\text{like}> \text{do})$	1/4
$P(<\text{college}> \text{do})$	0/4
$P(\text{do} \text{do})$	0/4

Next word prediction	Probability $W_i = 1 = \text{Henry}$
Next word	Probability next word = $\frac{N}{6}$
$P(<\text{IS}> \text{Henry})$	3/5
$P(<\text{I}> \text{Henry})$	1/5
$P(<\text{am}> \text{Henry})$	0
$P(<\text{Henry}> \text{Henry})$	0
$P(<\text{like}> \text{Henry})$	1/5
$P(<\text{college}> \text{Henry})$	0
$P(<\text{DO}> \text{Henry})$	0

<15> Do I like

use Trigram

$$P(<1\text{ like}>) = \frac{1}{3}$$

Next word $w_{i-2} = \text{I}$ and $w_{i-1} = \text{like}$

Next word

$$P(<15> | \text{I like})$$

$$\frac{1}{3}$$

$$P(<\text{I}> | \text{I like})$$

$$\frac{1}{3}$$

$$P(<\text{am}> | \text{I like})$$

$$\frac{1}{3}$$

$$P(<\text{Henry}> | \text{I like})$$

$$\frac{1}{3}$$

$$P(<\text{like}> | \text{I like})$$

$$\frac{1}{3}$$

$$P(<\text{college}> | \text{I like})$$

$$\frac{1}{3}$$

$$P(<\text{do}> | \text{I like})$$

$$\frac{1}{3}$$

Probability Next word = count($w_{i-2} w_{i-1}$)

4) <S>> Do I like college?

use bigram

$$w_0, w_1 = \text{I}$$

$$w_{i-2} = \text{like}$$

$$w_{i-1} = \text{college}$$

Next word

Probability = $\frac{\text{count}(w_0, w_1, w_{i-2}, w_{i-1})}{\text{count}(w_0, w_1, w_{i-2}, w_{i-1})}$

$$P(<15> | \text{I like college})$$

$$\frac{1}{2}$$

$$P(<\text{I}> | \text{I like college})$$

$$\frac{1}{2}$$

$$P(<\text{am}> | \text{I like college})$$

$$\frac{1}{2}$$

$$P(<\text{Henry}> | \text{I like college})$$

$$\frac{1}{2}$$

$$P(<\text{like}> | \text{I like college})$$

$$\frac{1}{2}$$

$$P(<\text{college}> | \text{I like college})$$

$$\frac{1}{2}$$

$$P(<\text{do}> | \text{I like college})$$

$$\frac{1}{2}$$

1) $\langle s \rangle$ I like college $\langle i \rangle$

$$= P(I | \langle s \rangle) \times P(\text{like}(I)) \times P(\text{college} | \text{like}) + P(\langle s \rangle | \text{college})$$

$$= \frac{3}{7} \times \frac{3}{6} \times \frac{3}{5} + \frac{3}{3} = \frac{57}{70} = 0.13$$

2) $\langle s \rangle$ Do I like Henry $\langle i \rangle$

$$P(\text{do} | \langle s \rangle) \times P(I | \text{do}) \times P(\text{like}(I)) + P(\text{Henry} | \text{like})$$

$$+ P(\langle s \rangle | \text{Henry})$$

$$= \frac{3}{7} + \frac{2}{4} \times \frac{3}{6} \times \frac{2}{5} + \frac{3}{3} = \frac{9}{350} = 0.0257$$

Q3

$\langle s \rangle$ the student pass the test $\langle i \rangle$

$\langle s \rangle$ the student with for the pass $\langle i \rangle$

$\langle s \rangle$ teacher's test student $\langle i \rangle$

Bigramics are

(the | $\langle s \rangle$), (the | student), (student | pass), (pass | like),
 (the | test), (test | $\langle s \rangle$, $\langle s \rangle$), (the | student), (student | test),
 (test | like), (like | fail), (fail | test), (test | $\langle s \rangle$),
 (teacher | test), (test | student), (student | $\langle s \rangle$)

$$P(\text{the} | \text{student}) = P(\text{student} | \text{the}) / P(\text{student})$$

$$P(\text{the} | \langle s \rangle) = P(\langle s \rangle | \text{the}) / P(\langle s \rangle)$$

$$= 0$$

$$= 2/3$$

$$P(\text{student} | \text{the}) = P(\text{the} | \text{student}) / P(\text{the})$$

$$= 2/4$$

$$P(<IS> | test) = P(test | <S>) / P(test)$$

$$= 1/2$$

$$P(\text{student} | \text{test}) \text{ count}(\text{test}, \text{student}) / \text{count}(\text{test})$$

$$= 1/2$$

Spelling correction using N-gram is a simple & effective approach to address misspelled words in a text. N-grams are used to identify & suggest correction for misspelled words by comparing them to correctly spelled words for language model.

1) Building an n-gram model : First we need to arrange a corpus of text with correctly spelled words to build an n-gram language model.

2) Tokenization into N-grams : The text in the corpus is tokenized into N-grams (typically bigrams i.e. trigrams).

3) Counting N-grams : For each N-gram, you count the number of times it appears in the corpus.

4) Spelling correction : To correct a misspelled word, you tokenize the word into N-grams & compare these N-grams to N-grams in your language model & calculate the likelihood probability. For each N-gram from misspelled, you consider possible correct alternative.

Q4

$\angle S \rightarrow$ the IDT students / NN Pass / v the DT test / NN < 15>
 $\angle S \rightarrow$ the IDT students / NN wait / v hit / p the IDT Pass / NN < 15>
 $\angle S \rightarrow$ teachers / N test / v students / NN < 15>

Envision probability

chwords		Noun	Verb	Model	preposition	DT
The	student	0	Pass	0	4/4=1	10
Student	Pass	0.6	0	0	0	0
Pass	Wast	0	0	1/3=0.3	0	0
for	test	0	0	0	0	0
teachers	test	0.2	0	1/3=0.3	0	0
		0.2	0	0	0	0

N		N	M	P	<15>
2	0	0	0	0	0
0	3	0	0	0	3
0	0	0	0	0	0
0	0	0	0	0	0

N		N	M	P	<15>
0.3	0.3	0	0.6	0	0
0.5	0.5	0	0	0	0.5
0.3	0.3	0.3	0.3	0.3	0.3
0	0	0	0	0	0

S

In the two sentences different meanings arise from different type ambiguity

a) Time flies like an arrow.

The ambiguity in this sentence is an example of structural ambiguity. It can be represented in two different ways based on structural grouping of data.

i) Time flies [i.e. insects known as flies that crawl over time] are similar to an arrow. In this like an arrow is similar, indicating comparison between time flies & arrow.

ii) Time (the concept of time passes quickly in manner similar to an arrow "like an arrow") is simple indicating comparison between speed of time & arrow.

b) "He crushed his key to my heart."

i) Semantic ambiguity the word crushed is the source of semantic ambiguity. It can be interpreted in two different ways:

- literally as in physically crushing into pieces

- figuratively emotional impact (crushed) it is

emotional impact (crushed)

literally as in physically crushing into pieces

literally as in physically crushing into pieces

6

a) Important to Bill

→ Important is an Adjective

→ to Bill → prepositional phrase, ∞ [B]

The entire phrase is therefore an adjective phrase

b) "hotkey up the bee?"

"Looked" is a verb, ∞ [verb]

"Up the bee" is a prepositional phrase

∴ the entire segment is a verb phrase.

N: Noun [Martin, Justin, will, spot, full]

M: Model [I we, will]

V: Verb [was, spot; put]

S: Statement [Justin will spot Martin]

T: Transiting words [I, we, will, spot]

<E> Martin Justin we watch will <F>

<S> spot will watch Martin <E>

<S> will Justin spot Martin <E>

<S> Martin will spot <E>

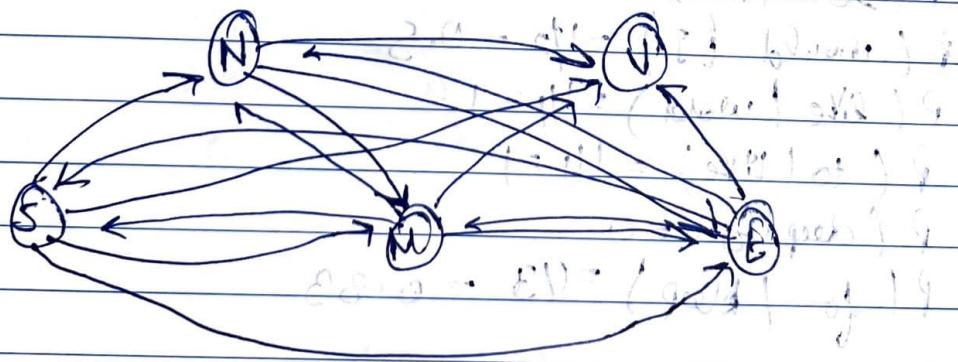
Emission Probability Matrix

	N	M	S	G
Martin	4/10	0	0	0
Justin	2/10	0	0	0
Lam	0	1/3	0	0
watch	0	0	2/4	0
will	2/10	2/3	0	0
Spot	2/10	0	1/4	0
Pat	0	0	1/4	0

Transition probability matrix

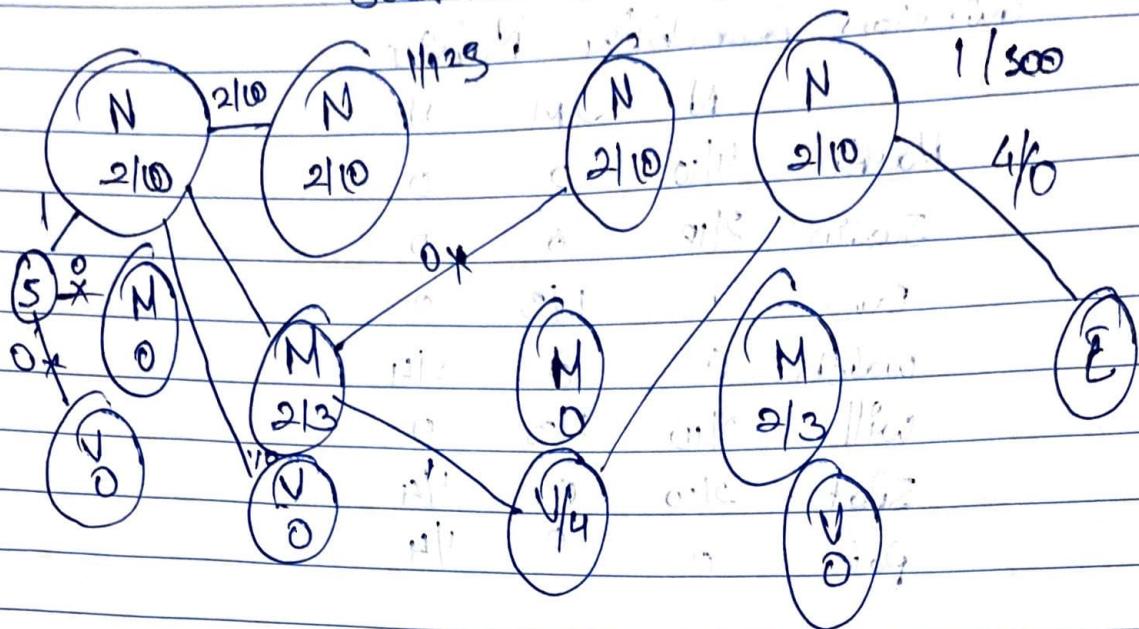
	N	M	S	G
N	0	0	1/3	0
M	2/10	3/10	1/0	4/10
S	4/10	0	0	0

Transition Probability



Martin Justin Lam can watch will spot pat

Justin will spot - will



$S \rightarrow N \rightarrow M \rightarrow V \rightarrow N \rightarrow E$

Q8

$\langle S \rangle$ I tell you to sleep $\langle S \rangle$ rest $\langle S \rangle$

$\langle S \rangle$ I would like to sleep for an hour $\langle S \rangle$

$\langle S \rangle$ Sleep helps one to relax $\langle S \rangle$

$$P(\langle S \rangle) = \text{count } \langle S \rangle / \text{count } S = 2/3 = 0.66$$

$$P(\text{tell} | I) = 1/2 = 0.5$$

$$P(\text{you} | \text{tell}) = 1/1 = 1$$

$$P(\text{rest} | \text{nd}) = 1/1 = 1$$

$$P(\text{would} | I) = 1/2 = 0.5$$

$$P(\text{like} | \text{would}) = 1/1 = 1$$

$$P(\text{to} | \text{like}) = 1/1 = 1$$

$$P(\text{sleep} | G) = 0$$

$$P(\text{for} | \text{sleep}) = 1/3 = 0.33$$

To do many things and not get tired

$$\begin{aligned}
 P(\text{an} | \text{For}) &= 1/1 = 1 \\
 P(\text{hour} | \text{an}) &= 1/1 = 1 \\
 P(\text{c/s} | \text{hour}) &= 1/1 = 1 \\
 P(\text{helps} | \text{sleep}) &= 1/3 = 0.3 \\
 P(\text{one} | \text{helps}) &= 1/1 = 1 \\
 P(\text{to} | \text{one}) &= 1/1 = 1 \\
 P(\text{relax} | \text{to}) &= 1/3 = 0.33 \\
 P(\text{c/s} | \text{relax}) &= 1/1 = 1
 \end{aligned}$$

Highest $\Rightarrow P(\text{sleep} | \text{to}) = 0.33$
 \therefore word is sleep

Q9) Emission probability:

words	Noun	Model	Verb
Martin	4/9	0	0
Sustin	2/9	0	0
an	0	1/4	0
watch	0	0	2/4
with	1/9	3/4	0
Spot	2/9	0	1/4
path	0	0	1/4

(Q11)

$\leftarrow S \rightarrow$ Mary Jane can see will $\leftarrow S \rightarrow$

$\leftarrow S \rightarrow$ Spot will see Morley $\leftarrow b \rightarrow$

$\leftarrow S \rightarrow$ will Jane spot Morley $\leftarrow S \rightarrow$

$\leftarrow S \rightarrow$ Morley will pat spot $\leftarrow S \rightarrow$

Emission probability Matrix

	N	M	V
Mary	4/9	0	0
Jane	2/9	0	0
Can	0	1/4	0
See	0	0	2/4
will	1/9	3/4	0
spot	2/9	0	1/4
pat	0	0	1/4

Transmission probability Matrix

$N \rightarrow M \rightarrow V \leftarrow S \rightarrow$

$\leftarrow S \rightarrow 3/4 \quad 1/4 \quad 0 \quad 0$

$N \quad 1/9 \quad 3/9 \quad 1/9 \quad 4/9$

$M \quad 1/4 \quad 0 \quad 3/4 \quad 0$

$V \quad 4/4 \quad 0 \quad 0 \quad 0$

$v_{11} \circledcirc N \quad v_{21} \circledcirc N \quad v_{31} \circledcirc N \quad v_{41} \circledcirc N$

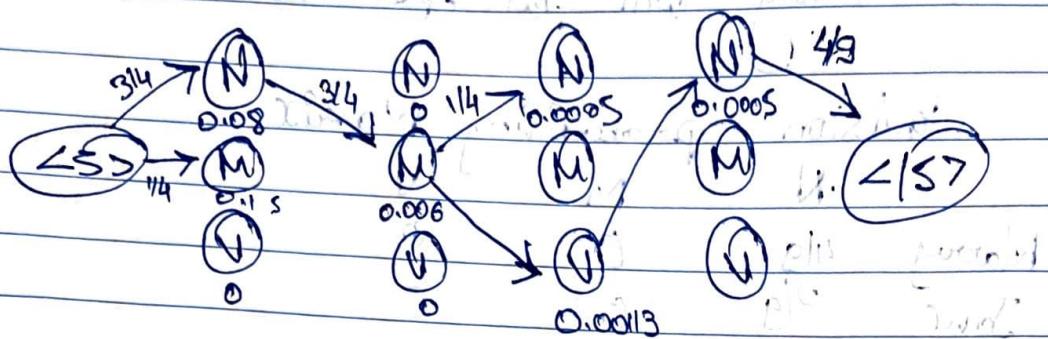
$\leftarrow S \rightarrow \quad v_{12} \circledcirc M \quad v_{22} \circledcirc M \quad v_{32} \circledcirc M \quad v_{42} \circledcirc M \quad \leftarrow E \rightarrow$

$v_{13} \circledcirc V \quad v_{23} \circledcirc N \quad v_{33} \circledcirc N \quad v_{43} \circledcirc V$

will can spot - Morley

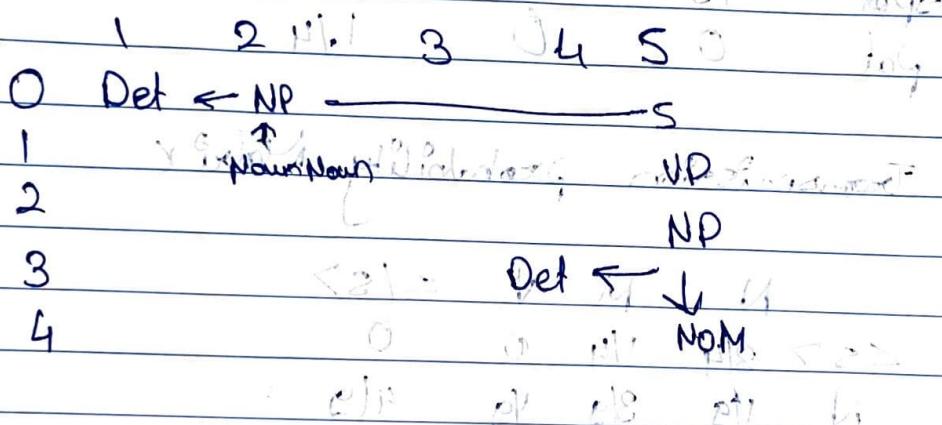
Q12)

Using the transmission and emission probabilities from the previous numerals. Find the

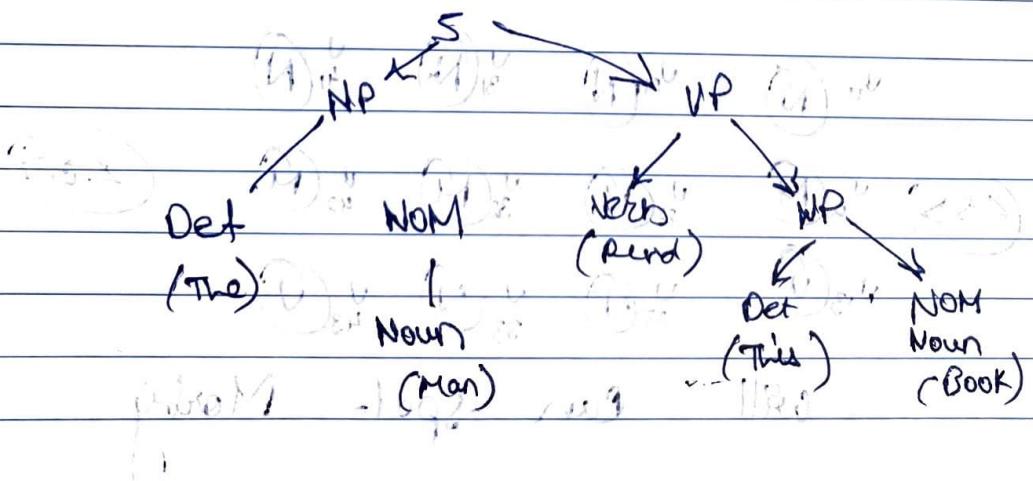


Q13)

O The man 2 reading this book.



This can be converted to tree as follow.



- Q14) i) Write, right \rightarrow They are homophones, which mean they sound the same but have unrelated meaning & spelling
- ii) Big, large \rightarrow These words are synonic. They both mean having a greater size.
- iii) Dark, light \rightarrow These words are antonym. Dark is opposite of light.
- iv) Car, Vehicle \rightarrow These words have hyponym, homonym relationship. A car is a specific type of vehicle.

Q15)

\rightarrow Ans is c
 $(19|20); d_1 \dots d_7 \dots [0|1-9])(0-27) - ([0-2][1-9] / 3[0-1])$
This regular exp is similar but further specifies that digits can range from 0-31, which words are positive numbers.

Q16) Hoboym \rightarrow Hyponym

"[nunion" [w.] is a hyponym of the wheels in sentence'

Hyponym - Hyponym

In second sentence 'drives car' [car] is a hyponym of companies.

26/15