AM604PC- Natural Language Processing Laboratory Manual

[ANURAG ENGINEERING COLLEGE]

[III B.Tech-CSE(AI& ML)- II -Sem]

Contents

NLTK: Natural Language Toolkit

The Natural Language Toolkit (NLTK) is a powerful Python library widely used in Natural Language Processing (NLP). It provides tools for text processing, corpus handling, linguistic analysis, and statistical modeling. NLTK is designed to be user-friendly, making it a popular choice for educational and research purposes in NLP.

Features of NLTK

Text Processing Tools

- Tokenization: NLTK can divide text into smaller units, such as words or sentences.
- Stemming: This feature reduces words to their root form, such as *running* to *run*.
- Lemmatization: Unlike stemming, lemmatization returns words to their dictionary form while considering their context.

Part-of-Speech (POS) Tagging

NLTK assigns grammatical categories to words in a sentence, such as nouns, verbs, and adjectives, aiding syntactic and semantic analysis.

Named Entity Recognition (NER)

This feature identifies entities such as persons, locations, and organizations within text. For example, in the sentence *"Steve Jobs founded Apple,"* Steve Jobs is tagged as a *Person*, and Apple as an *Organization*.

Syntactic Parsing

NLTK provides tools to analyze the grammatical structure of sentences using Context-Free Grammar (CFG) and Dependency Parsing.

Corpus Support

NLTK includes access to over 50 preloaded corpora such as the Brown Corpus and the Gutenberg Corpus, enabling easy experimentation with real-world text data.In *Natural Language Processing (NLP), corpus support refers to the ability of tools, frameworks, or libraries to provide access to collections of text data (known as corpora) for analysis, research, and application development. A corpus is a structured set of text, often annotated with metadata, that serves as a foundational resource for NLP tasks.*

WordNet Interface

WordNet is a semantic lexicon that allows exploration of relationships between words, such as synonyms, antonyms, hypernyms, and hyponyms.

Text Classification

NLTK supports text classification with both supervised and unsupervised learning methods, including algorithms such as Naive Bayes.

Collocations and Bigrams

The library can detect commonly co-occurring words, such as *New York*, using collocation and bigram techniques.

Language Modeling

It provides functionality for building unigram and bigram models for language analysis and generation.

Visualization

Visualization tools in NLTK allow for displaying parse trees and analyzing word frequency distributions.

Applications of NLTK

- Sentiment Analysis: Analyze the polarity of reviews or feedback.
- Document Classification: Categorize text documents, such as spam detection.
- Grammar and Syntax: Explore and implement parsing techniques.
- Chatbot Development: Build chatbots with tokenization, lemmatization, and classification.

NLTK is an indispensable tool for Natural Language Processing, offering a comprehensive suite of text processing, linguistic analysis, and visualization capabilities. Its ease of use and versatility make it an excellent choice for both educational and research purposes.

Tokenization

Program-1

Write a Python Program to perform Tokenization tasks on text

Tokenization in NLP

Tokenization is the process of breaking down a text into smaller units called *tokens*. These tokens can be words, phrases, or characters, depending on the application. For example, the sentence *"Natural Language Processing is fun"* can be tokenized into individual words: [*Natural*, *Language*, *Processing*, *is*, *fun*].

Following are the uses of Tokenization:

- Text Representation: Tokenization is a fundamental step in preparing text data for computational processing. Most NLP algorithms and models require input in the form of tokens.
- Feature Extraction: Tokens serve as features for models, enabling the representation of text in numerical forms such as vectors for further processing.
- Context Preservation: Tokenization ensures that the structure and meaning of text are preserved to an extent while breaking it into manageable parts.
- Language Understanding: It enables tasks like syntactic parsing, sentiment analysis, machine translation, and more by providing a manageable representation of the text.
- Efficiency: Working with tokens rather than raw text makes computations faster and more efficient, as the raw text might contain noise, punctuation, and other elements that could hinder analysis.

Tokenization is particularly useful in tasks such as search engines, chatbots, language modeling, and more, as it allows the system to process and understand language in a structured way.

Python Code

```
        import nltk
        from nltk.tokenize import word_tokenize, sent_tokenize

        # Download NLTK resources if not already downloaded
        nltk.download('punkt')

        def perform_tokenization(text):
        """
        Performs sentence and word tokenization on the input text.

        Args:
        text (str): The input text to be tokenized.

```

```
        Returns:
        tuple: A tuple containing a list of sentences and a list of words.
        """
        # Sentence tokenization
        sentences = sent_tokenize(text)

        # Word tokenization
        words = word_tokenize(text)

        return sentences, words

        def display_tokenization_results(sentences, words):
        """
        Displays the results of sentence and word tokenization.

        Args:
        sentences (list): List of tokenized sentences.
        words (list): List of tokenized words.
        """
        print("\nSentence Tokenization:")
        for i, sentence in enumerate(sentences, start=1):
        print(f"Sentence {i}: {sentence}")

        print("\nWord Tokenization:")
        print(words)

        def main():
        """
        Main function to demonstrate tokenization.
        """
        # Sample text for demonstration
        text = """Natural Language Processing is an amazing field of
           Artificial Intelligence.
        It enables machines to understand human language effectively and
           accurately."""

        print("Original Text:")
        print(text)

        # Perform tokenization
        sentences, words = perform_tokenization(text)

        # Display tokenization results
        display_tokenization_results(sentences, words)

        if __name__ == "__main__":
        main()
```

Listing 1.1: Word Tokenization using NLTK

Input:

```
Original Text:
Natural Language Processing is an amazing field of Artificial Intelligence.
It enables machines to understand human language effectively and accurately.
```

Output

```
Sentence Tokenization:
1: Natural Language Processing is an amazing field of Artificial Intelligence.
2: It enables machines to understand human language effectively and accurately.

Word Tokenization:
['Natural', 'Language', 'Processing', 'is', 'an', 'amazing', 'field', 'of',
'Artificial', 'Intelligence', '.', 'It', 'enables', 'machines', 'to', 'understand',
 'human', 'language', 'effectively', 'and', 'accurately', '.']
```

Viva Questions

1. What is tokenization in NLP?
2. What are the different types of tokenization?
3. How does whitespace-based tokenization differ from rule-based tokenization?
4. Why is tokenization important for text preprocessing?
5. What are some challenges in tokenization?

Stop Word Removal

Program-2

Write a Python program to perform stop word Removal tasks on text.

Stopwords

Stop word removal is a fundamental preprocessing step in Natural Language Processing (NLP). Stop words are common words (e.g., ”is”, ”and”, ”the”, ”a”, ”an”) that frequently occur in text but typically do not contribute to its meaning. Removing these words helps reduce the dimensionality of text data, streamline analysis, and minimize noise.

By eliminating stop words, NLP models can focus on the most relevant features, improving their performance and reducing computational overhead. This step enhances the efficiency and effectiveness of various NLP tasks. The NLTK library provides a predefined list of stop words for multiple languages, making it a convenient tool for implementing stop word removal.

Python Code

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords

# Download NLTK resources if not already downloaded
nltk.download('punkt')
nltk.download('stopwords')

def remove_stopwords(text):
"""
Removes stop words from the tokenized text.

Args:
text (str): The input text to be processed.

Returns:
list: A list of words after removing stop words.
"""
# Tokenize the text into words
tokens = word_tokenize(text)

# Get English stop words
stop_words = set(stopwords.words('english'))

# Remove stop words
```

```
    filtered_tokens = [word for word in tokens if word.lower() not in
        stop_words]

    return filtered_tokens

    def main():
    """
    Main function to demonstrate stop word removal.
    """
    # Sample text for demonstration
    text = """This is a simple example to demonstrate how stop word removal
        works in text processing.
    It removes commonly used words like 'is', 'a', 'to', and so on."""

    print("Original Text:")
    print(text)

    # Perform stop word removal
    filtered_words = remove_stopwords(text)

    # Display results
    print("\nText After Stop Word Removal:")
    print(filtered_words)

    if __name__ == "__main__":
    main()
```

Listing 2.1: Stop Word Removal using NLTK

Input:

```
This is a simple example to demonstrate how stop word removal works in text processing.
It removes commonly used words like 'is', 'a', 'to', and so on.
```

Output:

```
['simple', 'example', 'demonstrate', 'stop', 'word', 'removal',
'works', 'text', 'processing', '.', 'removes', 'commonly',
'used', 'words', 'like', ',', 'so', '.']
```

Viva Questions

1. What are stop words, and why are they removed in NLP tasks?
2. How does case sensitivity affect stop word removal?
3. What are the advantages and disadvantages of removing stop words in text processing?
4. Can you list some applications where stop word removal is particularly useful?

Word Analysis

Program-3

Write a Python Program for Word Analysis..

Word Analysis in NLP

Word analysis in Natural Language Processing (NLP) involves breaking down a given text into its fundamental components to extract meaningful insights. It helps in understanding word distribution, frequency, uniqueness, and statistical properties of words in a given text corpus. Word analysis forms the foundation for more advanced NLP techniques such as *Part-of-Speech (POS) tagging*, *Named Entity Recognition (NER)*, and *Topic Modeling*.

Key Steps in Word Analysis

1. Tokenization: Splitting the text into individual words or tokens.
2. Normalization: Converting words to lowercase and removing punctuation or special characters.
3. Frequency Analysis: Counting the occurrences of each word to identify the most commonly used words.
4. Unique Word Count: Determining how many distinct words are present in the text.
5. Statistical Analysis: Computing measures like average word length and word density.

Program

```
import nltk
from nltk.tokenize import word_tokenize
from collections import Counter
import string

nltk.download('punkt')

def word_analysis(text):
words = word_tokenize(text.lower())
words = [word for word in words if word.isalpha()]

word_freq = Counter(words)

unique_words = len(set(words))
avg_word_length = sum(len(word) for word in words) / len(words) if words
   else 0

print("\nWord Frequency Distribution:")
```

```
    for word, freq in word_freq.most_common(10):
    print(f"{word}: {freq}")

    print("\nBasic Text Statistics:")
    print(f"Total Words: {len(words)}")
    print(f"Unique Words: {unique_words}")
    print(f"Average Word Length: {avg_word_length:.2f}")

    if __name__ == "__main__":
    sample_text = "Natural Language Processing helps computers understand
       and generate human language."
    word_analysis(sample_text)
```

Output

```
Word Frequency Distribution:
language: 2
natural: 1
processing: 1
helps: 1
computers: 1
understand: 1
generate: 1
human: 1

Basic Text Statistics:
Total Words: 8
Unique Words: 8
Average Word Length: 6.12
```

Viva Questions

1. What is the significance of word frequency analysis in NLP?
2. How does tokenization work in NLTK?
3. Why do we remove punctuation and convert text to lowercase?
4. What is the importance of computing unique word count?
5. How can this technique be applied in real-world applications?

Word Generation

Program-4

Write a Python Program for Word Generation.
(using a bigram-based Markov model from a given text.)

Word Generation in NLP

Word generation is a fundamental concept in Natural Language Processing (NLP) that involves creating new text sequences by predicting the next word based on learned patterns from existing text. It is widely used in applications such as *chatbots, story generation, machine translation, and predictive text input*.

Bigram Markov Model

Here we perform word generation using bigram-based Markov model . A bigram is a sequence of two consecutive words, and a bigram language model calculates the probability of a word occurring given the previous word. The Markov model, a probabilistic model, assumes that the occurrence of a word depends only on the previous word, making it computationally efficient for text generation.

- Training Phase:
  - The model first processes a given text corpus and extracts bigrams (word pairs).
  - It builds a probability distribution where each word is mapped to the possible words that can follow it.
- Generation Phase:
  - The process starts with a given word (seed word).
  - The model probabilistically selects the next word based on the learned bigram probabilities.
  - This process continues until a desired number of words are generated or an end condition is met.

While bigram models provide a basic approach to text generation, more advanced techniques such as trigram models, n-gram smoothing, and deep learning models (e.g., *Recurrent Neural Networks (RNNs) and Transformers*) are used for generating more coherent and context-aware text.

Program

```python
import random
import nltk
from nltk.tokenize import word_tokenize
from collections import defaultdict

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

def train_bigram_model(text):
"""
Trains a simple bigram-based word generation model.

Args:
text (str): The input text.

Returns:
dict: A dictionary mapping words to their possible next words.
"""
words = word_tokenize(text.lower())
bigram_model = defaultdict(list)

for i in range(len(words) - 1):
bigram_model[words[i]].append(words[i + 1])
print(bigram_model)
return bigram_model

def generate_text(bigram_model, start_word, num_words=10):
"""
Generates a sequence of words based on a trained bigram model.

Args:
bigram_model (dict): Trained bigram word map.
start_word (str): The word to start generation.
num_words (int): Number of words to generate.

Returns:
str: Generated sentence.
"""
current_word = start_word.lower()
generated_words = [current_word]

for _ in range(num_words - 1):
if current_word in bigram_model:
next_word = random.choice(bigram_model[current_word])
generated_words.append(next_word)
current_word = next_word
else:
break  # Stop if no next word exists
return ' '.join(generated_words)

# Example Usage
if __name__ == "__main__":
sample_text = """John is a generous human being.
He loves to help others and always shares his knowledge.
Generosity and kindness are his true nature.
```

```
People respect him for his wisdom and humility."""

model = train_bigram_model(sample_text)
print("\nGenerated Text:")
print(generate_text(model, start_word="is", num_words=10))
```

Sample Output

```
 start_word: is
Generated Text:
is a generous human being . he loves to help


start_word: help
Generated Text:
help others and humility . people respect him for his
```

Viva Questions

1. What is a Markov model, and how is it used in word generation?
2. What is the role of bigrams in text prediction?
3. How does randomness influence generated text?
4. What are the advanced models used for text generation?
5. How does a Markov model differ from neural network-based text generation?

Word Sense Disambiguation

Program-5

create a sample list of atleast five words with ambiguous sense and write a pythong program to implement Word Sense Disambiguation

Word Sense Disambiguation

Word Sense Disambiguation (WSD) is a fundamental problem in Natural Language Processing (NLP) that involves determining the correct meaning of a word based on its surrounding context. Many words in English have multiple meanings (senses), and resolving their ambiguity is crucial for applications such as:

- Machine Translation (e.g., Google Translate needs to correctly translate polysemous words).
- Speech Recognition (e.g., "light" as brightness vs. "light" as not heavy).
- Chatbots and Virtual Assistants (e.g., distinguishing meanings of "bank" in finance vs. riverbank).
- Information Retrieval and Search Engines (e.g., distinguishing "rock" as a music genre vs. a physical object).

Lesk Algorithm

In this program, we implement WSD using the Lesk algorithm, a dictionary-based method that selects the meaning of a word by maximizing the overlap between its dictionary definition and the surrounding context in a sentence.

The Lesk Algorithm is a classic approach to WSD and works as follows:

1. For a given ambiguous word in a sentence, retrieve all its possible meanings (senses) from WordNet, which is a large lexical database of English that serves as a key resource in Natural Language Processing (NLP).
2. Compute the overlap between each sense's definition and the words in the given sentence.
3. Select the sense with the highest overlap as the correct meaning.

Though simple, the Lesk algorithm forms the basis for more advanced techniques like *supervised learning-based WSD*, *deep learning models (e.g., BERT)*, and *contextual embeddings*.

Sample List of Ambiguous Words

The following words with multiple meanings have been considered for WSD:

- Bank: (Financial institution / Riverbank)
- Bat: (Flying mammal / Equipment for playing sports)

- Crane: (A type of bird / A machine used for lifting heavy objects)
- Date: (A calendar day / A romantic meeting / A type of fruit)
- Light: (Not heavy / Illumination)
- Light: (Electromagnetic radiation / Not serious)
- Spring: (A season / A coiled object / A natural water source)
- Bass: (A type of fish / A low musical tone)
- Rock: (A stone / A genre of music)
- Watch: (A timepiece / The act of looking at something attentively)

Program

```
import nltk
from nltk.wsd import lesk
from nltk.tokenize import word_tokenize

# Download necessary resources
nltk.download('wordnet')
nltk.download('omw-1.4')
nltk.download('punkt')

# Sample list of ambiguous words and sentences
sample_sentences = {
    "bank": "I deposited money in the bank.",
    "bat": "A bat flew across the night sky.",
    "crane": "The crane lifted heavy steel beams at the construction site.",
    "date": "Our first date was at a coffee shop.",
    "light1": "This bag is very light to carry.",  # Light (Not heavy)
    "light2": "Please turn on the light in the room.",  # Light (Illumination)
    "spring": "The flowers bloom in spring.",
    "bass": "He caught a huge bass while fishing.",
    "rock": "She enjoys listening to rock music.",
    "watch": "I wear a watch on my left wrist."
}

def perform_wsd(word, sentence):
"""
Implements Word Sense Disambiguation (WSD) using the Lesk algorithm.

Args:
word (str): The ambiguous word for which WSD is performed.
sentence (str): The sentence containing the ambiguous word.

Returns:
str: The best-matching sense of the word based on context.
"""
tokens = word_tokenize(sentence)  # Tokenize the sentence
actual_word = word.replace("1", "").replace("2", "")  # Remove numbering from "light1", "light2"
```

```
    sense = lesk(tokens, actual_word)  # Apply the Lesk algorithm

    return sense.definition() if sense else "No suitable sense found"

    # Run WSD on all sample words
    for word, sentence in sample_sentences.items():
    sense = perform_wsd(word, sentence)
    print(f"Word: {word.replace('1', '').replace('2', '')}")  # Remove
       numbering from print output
    print(f"Sentence: {sentence}")
    print(f"Predicted Sense: {sense}\n")
```

Listing 5.1: Implementation of WSD using the Lesk Algorithm

Sample Output

```
Word: bank
Sentence: I deposited money in the bank.
Predicted Sense: a financial institution that accepts deposits and channels the money int

Word: bat
Sentence: A bat flew across the night sky.
Predicted Sense: nocturnal mouselike mammal with forelimbs modified to form membranous wi

Word: crane
Sentence: The crane lifted heavy steel beams at the construction site.
Predicted Sense: lifts and moves heavy objects; lifting tackle is suspended from a pivote

Word: date
Sentence: Our first date was at a coffee shop.
Predicted Sense: an appointment to meet at a specified time and place

Word: light
Sentence: This bag is very light to carry.
Predicted Sense: of comparatively little physical weight or density

Word: light
Sentence: Please turn on the light in the room.
Predicted Sense: (physics) electromagnetic radiation that can produce a visual sensation

Word: spring
Sentence: The flowers bloom in spring.
Predicted Sense: the season of growth

Word: bass
Sentence: He caught a huge bass while fishing.
Predicted Sense: the lean flesh of a saltwater fish

Word: rock
Sentence: She enjoys listening to rock music.
Predicted Sense: a genre of popular music originating in the 1950s

Word: watch
```

```
Sentence: I wear a watch on my left wrist.
Predicted Sense: a small portable timepiece
```

Viva Questions

1. What is Word Sense Disambiguation (WSD) and why is it important in NLP?
2. How does the Lesk algorithm determine the correct sense of a word?
3. What are some real-world applications of WSD?
4. What are the limitations of the Lesk algorithm?
5. How can supervised machine learning improve WSD performance?

Install NLTK Toolkit and Perform Stemming

Program-6

Install the NLTK toolkit and perform stemming on sample text.

NLTK (Natural Language Toolkit)

The Natural Language Toolkit (NLTK) is a widely used Python library for Natural Language Processing (NLP). It provides tools for text processing, tokenization, stemming, lemmatization, part-of-speech tagging, named entity recognition, and other linguistic operations. NLTK includes various corpora, lexical resources, and models that help perform complex NLP tasks.

Installing NLTK

To install NLTK, the following command is executed in the terminal or command prompt:

```
pip install nltk
```

Once installed, additional datasets and resources can be downloaded using:

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

This ensures that all necessary corpora, models, and tools required for NLP tasks are available.

Key Features of NLTK

- Tokenization: Splitting text into words or sentences.
- Stemming: Reducing words to their root form.
- Lemmatization: Converting words to their dictionary base form.
- POS Tagging: Assigning grammatical tags to words.
- Parsing & Syntax Analysis: Understanding sentence structure.
- Text Classification: Classifying text documents using machine learning techniques.
- Named Entity Recognition (NER): Identifying entities like names, locations, and organizations.

Stemming in NLP

Stemming is the process of reducing a word to its root or base form by removing suffixes. Unlike lemmatization, stemming does not necessarily produce a valid word but rather a root form that can be used for text analysis.

Various Stemming Algorithms

- Porter Stemmer:One of the most commonly used stemming algorithms, it applies a series of rules to iteratively remove common suffixes from English words.
- Lancaster Stemmer : A more aggressive stemming algorithm compared to Porter, often producing shorter root words.
- Snowball Stemmer: An improvement over the Porter stemmer that supports multiple languages and applies a more refined set of rules.
- Regexp Stemmer: Uses regular expressions to define stemming rules, allowing for customized word reductions based on specific patterns

Example of Stemming

Original Word	Stemmed Form
Running	Run
Studying	Study
Happily	Happi
Flying	Fly

Table 6.1: Examples of Stemming

Stemming is particularly useful in:

- Search engines for query optimization.
- Text normalization in NLP tasks.
- Reducing vocabulary size for machine learning models.

Python Implementation

Following Python program install NLTK and performs stemming using the Lancaster Stemmer .

```
# Import necessary libraries
import nltk
from nltk.stem import LancasterStemmer
from nltk.tokenize import word_tokenize

# Download required NLTK resources
nltk.download('punkt')

def perform_stemming(text):
"""
Performs stemming on the input text using the Lancaster Stemmer.

Args:
text (str): The input text to be stemmed.

Returns:
list: A list of stemmed words.
"""
# Initialize Lancaster Stemmer
stemmer = LancasterStemmer()
```

```
21
22      # Tokenize the text into words
23      words = word_tokenize(text)
24
25      # Apply stemming to each word
26      stemmed_words = [stemmer.stem(word) for word in words]
27
28      return stemmed_words
29
30      def main():
31      """
32      Main function to demonstrate stemming.
33      """
34      # Sample text for demonstration
35       text = 'Focusing is most important for reading writing
36                  understanding and for any skill development'
37
38      print("Original Text:")
39      print(text)
40
41      # Perform stemming
42      stemmed_text = perform_stemming(text)
43
44      # Display results
45      print("\nStemmed Text:")
46      print(stemmed_text)
47
48      if __name__ == "__main__":
49      main()
```

Listing 6.1: Installing NLTK and perform stemming

Output

```
Original Text:
Focusing is most important for reading writing
  understanding and for any skill development

Stemmed Text:
['focus', 'is', 'most', 'import', 'for', 'read', 'writ', 'understand',
  'and', 'for', 'any', 'skil', 'develop']
```

Viva Questions

1. What is stemming in NLP?
2. How does the Lancaster Stemmer algorithm work?
3. What is the difference between stemming and lemmatization?
4. Why do some words not retain their original form after stemming?
5. Name four stemming algorithms.

POS Tagging

Program-7:

Create Sample list of at least 10 words POS tagging and find the POS for any given word

Introduction to POS Tagging

Part-of-Speech (POS) tagging is a fundamental task in Natural Language Processing (NLP) that involves assigning grammatical categories (such as noun, verb, adjective, etc.) to words in a sentence. POS tagging helps in understanding sentence structure and meaning, making it essential for applications such as text processing, information retrieval, and machine translation.

Each word in a sentence belongs to a specific grammatical category based on its function. The commonly used POS tags in the Penn Treebank Tagset include:

- NN (Noun, singular) – e.g., *dog, car, tree*
- NNS (Noun, plural) – e.g., *dogs, cars, trees*
- VB (Verb, base form) – e.g., *run, eat, sleep*
- VBD (Verb, past tense) – e.g., *ran, ate, slept*
- VBG (Verb, gerund/present participle) – e.g., *running, eating, sleeping*
- JJ (Adjective) – e.g., *happy, blue, large*
- RB (Adverb) – e.g., *quickly, silently, well*
- IN (Preposition) – e.g., *in, on, at, by*
- DT (Determiner) – e.g., *the, a, an*
- PRP (Pronoun) – e.g., *he, she, they*

POS tagging is typically performed using rule-based, statistical, or machine learning-based approaches. The Natural Language Toolkit (NLTK) provides a built-in function for POS tagging using the `pos_tag()` function, which employs a pre-trained model.

Applications of POS Tagging

- Information Retrieval: Helps improve search engine results by understanding query intent.
- Machine Translation: Assists in translating words accurately by considering their POS.
- Text-to-Speech Systems: Improves pronunciation by distinguishing between homonyms (e.g., "lead" as a noun vs. verb).

- Grammar Checking: Identifies grammatical errors in writing tools.

POS tagging is an essential task in NLP that aids in text analysis by providing grammatical information. The use of NLTK's `pos_tag()` function simplifies the process, making it easier to analyze and understand sentence structures.

Implementation in Python

The following Python program performs POS tagging on a sample list of words and allows users to find the POS of any given word.

```
import nltk
from nltk import pos_tag
from nltk.tokenize import word_tokenize

# Download required resources
nltk.download('averaged_perceptron_tagger')
nltk.download('punkt')

def pos_tagging(words_list):
"""
Performs POS tagging on a list of words.
"""
return pos_tag(words_list)

def find_pos(word, tagged_list):
"""
Finds the POS tag for a given word.
"""
for w, pos in tagged_list:
if w.lower() == word.lower():
return pos
return "POS not found"

def main():
"""
Main function to demonstrate POS tagging and finding POS for a
   given word.
"""
# Sample list of words
words_list = ["The", "quick", "brown", "fox", "jumps", "over",
   "the", "lazy", "dog"]

# Perform POS tagging
tagged_words = pos_tagging(words_list)

# Display POS tagging results
print("POS tagging results:")
for word, pos in tagged_words:
print(f"{word}: {pos}")

# Take input from user
```

```
    search_word = input("\nEnter a word to find its POS: ")
    pos_result = find_pos(search_word, tagged_words)
    print(f"POS for '{search_word}': {pos_result}")

    if __name__ == "__main__":
    main()
```

Listing 7.1: POS Tagging using NLTK

Output

```
POS tagging results:
The: DT
quick: JJ
brown: JJ
fox: NN
jumps: VBZ
over: IN
the: DT
lazy: JJ
dog: NN

Enter a word to find its POS: fox
POS for 'fox': NN
```

Viva Questions

1. What is POS tagging, and why is it important in NLP?
2. Which Python library is commonly used for POS tagging?
3. What does the POS tag ‘NN’ represent? give an example?
4. How does POS tagging help in real-world applications like search engines or chatbots?
5. What does the POS tag ‘VBZ’ represent? give an example?

Morphological Analysis and N-Grams

Program-8:

Write a Python program to

- a) Perform Morphological Analysis using NLTK library
- b) Generate n-grams using NLTK N-Grams library
- c) Implement N-Grams Smoothing

Morphological Analysis

Morphological analysis involves breaking down words into their base forms, typically using:

- Stemming: Reducing words to their root form by removing affixes (e.g., *running* → *run*).
- Lemmatization: Mapping words to their dictionary base forms while considering context (e.g., *better* → *good*).

In this program, we use NLTK's WordNet Lemmatizer to perform lemmatization and analyze a given sentence.

N-Grams and Laplace Smoothing

An *n-gram* is a contiguous sequence of *n* items (words, characters, etc.) extracted from a text. Common examples include:

- Unigram (n=1): ["The", "quick", "brown", "fox"]
- Bigram (n=2): [("The", "quick"), ("quick", "brown")]
- Trigram (n=3): [("The", "quick", "brown"), ("quick", "brown", "fox")]

However, traditional n-gram models suffer from data sparsity—many word combinations may not appear in the training data. To handle this, Laplace (add-one) smoothing is applied, which assigns a small probability to unseen n-grams, preventing zero probabilities.

Implementation in Python

```
1
2         import nltk
3         from nltk.tokenize import word_tokenize
4         from nltk.stem import WordNetLemmatizer
5         from nltk.util import ngrams
6         from nltk.corpus import wordnet
7         from nltk import pos_tag
```

```
        from collections import Counter

        # Download required resources
        nltk.download('punkt')
        nltk.download('wordnet')
        nltk.download('averaged_perceptron_tagger')

        def get_wordnet_pos(word):
        """
        Converts POS tag to format compatible with WordNet Lemmatizer.
        """
        tag = 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)  # Default to NOUN if
           not found

        def morphological_analysis(sentence):
        """
        Performs morphological analysis on a sentence using
           lemmatization.
        """
        lemmatizer = WordNetLemmatizer()
        words = word_tokenize(sentence)
        lemmatized_words = [lemmatizer.lemmatize(word,
           get_wordnet_pos(word)) for word in words]
        return " ".join(lemmatized_words)

        def generate_ngrams(text, n):
        """
        Generates n-grams from a given text.
        """
        tokens = word_tokenize(text)
        return list(ngrams(tokens, n))

        def laplace_smoothing(ngrams_list):
        """
        Implements Laplace (add-one) smoothing for n-grams.
        """
        ngram_counts = Counter(ngrams_list)
        total_ngrams = sum(ngram_counts.values())
        vocab_size = len(set(ngrams_list))
        smoothed_probs = {ngram: (count + 1) / (total_ngrams +
           vocab_size) for ngram, count in ngram_counts.items()}
        return smoothed_probs, ngram_counts

        def main():
        """
        Main function to demonstrate morphological analysis, n-gram
           generation, and n-gram smoothing.
        """
        # Morphological Analysis
```

```
        sentence = "The running dogs are barking loudly."
        lemmatized_sentence = morphological_analysis(sentence)
        print(f"Original sentence: {sentence}")
        print(f"Morphological analysis (lemmatized):
           {lemmatized_sentence}")

        # N-gram Generation
        text = "The quick brown fox jumps over the lazy dog"
        n = 3  # Trigrams
        trigrams_list = generate_ngrams(text, n)
        print(f"\nGenerated {n}-grams:")
        print(trigrams_list)


        # N-gram Smoothing
        smoothed_probs, _ = laplace_smoothing(trigrams_list)
        print("\nN-gram probabilities after Laplace smoothing:")
        for ngram, prob in smoothed_probs.items():
        print(f"{ngram}: {prob:.4f}")

        if __name__ == "__main__":
        main()
```

Listing 8.1: Morphological Analysis and N-Grams

Output

```
Original sentence: The running dogs are barking loudly.
Morphological analysis (lemmatized): The run dog be barking loudly .

Generated 3-grams:
[('The', 'quick', 'brown'),
 ('quick', 'brown', 'fox'),
 ('brown', 'fox', 'jumps'),
 ('fox', 'jumps', 'over'),
  ('jumps', 'over', 'the'),
   ('over', 'the', 'lazy'),
   ('the', 'lazy', 'dog')]

N-gram probabilities after Laplace smoothing:
('The', 'quick', 'brown'): 0.1429
('quick', 'brown', 'fox'): 0.1429
('brown', 'fox', 'jumps'): 0.1429
('fox', 'jumps', 'over'): 0.1429
('jumps', 'over', 'the'): 0.1429
('over', 'the', 'lazy'): 0.1429
('the', 'lazy', 'dog'): 0.1429
```

Viva Questions

1. What is morphological analysis?
2. What is the difference between stemming and lemmatization?
3. What are n-grams, and how are they used in language modeling?
4. What is the purpose of Laplace smoothing in n-gram models?
5. How does tokenization impact n-gram generation?

Implementation of Porter Stemmer Algorithm for Stemming

Program-9

Write a Python program to implement Porter stemmer algorithm for stemming

Stemming

Stemming is a crucial preprocessing step in Natural Language Processing (NLP) that reduces words to their root or base form. It helps in text normalization, which improves the efficiency of text-based applications such as search engines, text classification, and sentiment analysis.

The Porter Stemming Algorithm, developed by Martin Porter in 1980, is one of the most widely used stemming techniques. It works through a series of heuristic rules to iteratively strip suffixes from words while maintaining the word's base meaning.

For example:

- `running` → `run`
- `flies` → `fli`
- `better` → `better` (unchanged, since it's already in its root form)

The algorithm applies the following five steps:

1. Removes plural and -ed, -ing suffixes.
2. Converts y to i if necessary.
3. Removes double suffixes (e.g., "-ization" → "-ize").
4. Removes derivational suffixes (e.g., "-full" → "-ful").
5. Handles exceptions and irregular cases.

Python Implementation

```
import nltk
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize

# Download required resources if not available
nltk.download('punkt')

def porter_stemming(text):
"""
Applies Porter Stemming Algorithm to each word in the input text.

Args:
```

```
text (str): Input sentence or paragraph.

Returns:
list: List of words after applying stemming.
"""
# Initialize Porter Stemmer
stemmer = PorterStemmer()

# Tokenize the input text
words = word_tokenize(text)

# Apply stemming to each token
stemmed_words = [stemmer.stem(word) for word in words]

return stemmed_words

def main():
# Example text
text = "Running jogging skipping  excercising workouts and dieting are good
    for health.."

# Apply stemming
stemmed_output = porter_stemming(text)

# Display original and stemmed text
print("Original Text:")
print(text)
print("\nStemmed Output:")
print(" ".join(stemmed_output))

if __name__ == "__main__":
main()
```

Listing 9.1: Porter Stemming Algorithm in Python

Output

```
Original Text:
Running jogging skipping  excercising workouts and dieting are good for health.

Stemmed Output:
run jog skip excercis workout and diet are good for health .
```

Viva Questions

1. What is stemming in NLP?
2. How does the Porter Stemming Algorithm work?
3. What is the difference between stemming and lemmatization?
4. Why is stemming useful in text processing?
5. What are the limitations of Porter Stemmer?

Conversion of Audio to Text and Text to Audio

Program-10:

Write a python program to

- Convert an audio file into text.
- Convert a text file into an audio file.
- (Using NLTK package)

Discription

Speech processing is a crucial component of NLP, allowing machines to understand spoken language and generate speech. There are two primary processes in this experiment:

Speech-to-Text (STT)

Speech recognition involves converting spoken language into text. This is achieved using libraries like:

- SpeechRecognition: A Python library that provides an interface to different speech recognition engines (Google Web Speech API, CMU Sphinx, etc.).
- NLTK (Natural Language Toolkit): While NLTK itself does not provide direct speech recognition, it can process the converted text for further linguistic analysis.

Text-to-Speech (TTS)

Text-to-Speech (TTS) synthesis converts written text into spoken language. This is used in applications like:

- Voice assistants (e.g., Alexa, Google Assistant)
- Audiobooks and accessibility services

For this experiment, we use:

- Google Text-to-Speech (gTTS): A Python library that allows converting text into speech using Google's TTS API.

Implementation

Prerequisites

Install the following libraries

- speechrecogntion
- gtts
- pydub
- ffmpeg with chocolatey

```
pip  install speechrecognition gtts nltk pydub
```

Python code

```

import speech_recognition as sr
from gtts import gTTS
import os

# Function to convert audio file to text
def audio_to_text(audio_file):
recognizer = sr.Recognizer()
with sr.AudioFile(audio_file) as source:
audio_data = recognizer.record(source)
try:
text = recognizer.recognize_google(audio_data)
print("Recognized Text:", text)
print("Speech-to-Text conversion successful")
return text
except sr.UnknownValueError:
print("Speech Recognition could not understand audio")
return None
except sr.RequestError:
print("Could not request results from Google Speech Recognition service")
return None

# Function to convert text to speech
def text_to_audio(text, output_file):
tts = gTTS(text=text, lang='en')
tts.save(output_file)
print("Text-to-Speech conversion successful. Saved as", output_file)

# Main Function
def main():
# Convert audio to text
audio_file = r"C:\Users\Wopea\Downloads\NLP LAB\abc.wav"  # Provide the path
    to your audio file
recognized_text = audio_to_text(audio_file)

if recognized_text:
# Convert text to audio
output_audio_file = "output_audio.mp3"
text_to_audio(recognized_text, output_audio_file)

if __name__ == "__main__":
main()
```

Listing 10.1: Audio to Text and Text to Audio Conversion

1. The program will process the given audio file (`sample_audio.wav`) and convert it into text.
2. The recognized text will be printed on the screen.
3. The text will then be converted back into an audio file (`output_audio.mp3`), which can be played to hear the generated speech.

Output

```
Recognized Text: this is Natural Language Processing lab at Anurag Engineering College
Speech-to-Text conversion successful
Text-to-Speech conversion successful.
 Saved as output_audio.mp3
```

Viva Questions

1. What is Speech Recognition, and how does it work?
2. What are the libraries used in this conversion?
3. How does Google Text-to-Speech (gTTS) work?
4. What are some real-world applications of text-to-speech and speech-to-text?