Bansilal Ramnath Agarwal Charitable Trust's

Vishwakarma Institute of Information Technology

Department of
Artificial Intelligence and Data Science

Student Name: Siddhesh Dilip Khairnar		
Class: TY	Division: B	Roll No: 372028
Semester: 6th	Academic Year: 2023-24	
Subject Name & Code: Natural Language Processing		
Title of Assignment: Perform chunking to detect noun, adjective,adverb and verb phrases from given sentence.		
Date of Performance: 09-02-2024	Date of Submission: 16-03-2024	

Assignment 5

Aim:

Perform chunking to detect noun, adjective, adverb and verb phrases from given sentence.

Software Requirements:

Google Collab

Background Theory:

	Shallow Parsing	***Deep Parsing***
Technique	Identifies and extracts chunks based on part-of-speech tags	Analyzes full syntactic structure of a sentence
Scope	Identifies only parts of a sentence	Analyzes the full sentence
Grammar	Simple and less complex grammar	More complex grammar
Resource Usage	Requires less processing power and resources	Requires more processing power and resources
Speed	Faster than deep parsing	Slower than shallow parsing

Application	Named entity recognition, sentiment analysis, text classification	Machine translation, text-to-speech synthesis, semantic analysis
Accuracy	Provides less detailed and accurate information compared to deep parsing	Provides more detailed and accurate information about the syntactic structure and meaning of a sentence
Suitability	Useful for tasks where speed and efficiency are important	Useful for tasks that require high accuracy and detailed analysis

Shallow Parsing Example:

Consider the sentence: "The cat sat on the mat."

Shallow parsing, also known as chunking, would identify the chunks based on part-of-speech tags, such as:

- The cat (noun phrase)
- sat (verb)
- on the mat (prepositional phrase)

This is a faster and less resource-intensive approach to sentence analysis, and is commonly used in applications such as named entity recognition, sentiment analysis, and text classification.

Deep Parsing Example:

Consider the sentence: "The cat saw the bird in the tree and chased it away."

Deep parsing analyzes the full syntactic structure of the sentence, identifying the subject (cat), object (bird), and verb (chased), as well as the relationship between them.

A deep parser would be able to understand the meaning of the sentence and identify that the cat saw the bird in the tree and then chased it away.

This is a more accurate and detailed approach to sentence analysis, but it requires more processing power and resources, and is typically used in applications that require high accuracy and detailed analysis, such as machine translation, text-to-speech synthesis, and semantic analysis.

Code:

1] Customized grammar to define noun, adjective, adverb, verb phrases and perform chunking based on it on sample sentences:

```
import nltk

sample_sentences = [
  "The quick brown fox jumps over the lazy dog.",
  "I always wake up early in the morning.",
  "She sings beautifully in the shower.",
  "They danced all night at the party.",
  "He carefully placed the fragile vase on the shelf."
]


grammar_rules = r"""
  NP: {<DT>?<JJ>*<NN.*>+}
  ADJP: {<RB>*<JJ.*>+}
  ADVP: {<RB.*>}
  VP: {<VB.*><RP>?<DT>?<JJ.*>*<NN.*>*<IN>?<PRP.*>*<RB>*}
"""


def chunk_sentence(sentence):
  tokens = nltk.word_tokenize(sentence)
```

```
  tagged = nltk.pos_tag(tokens)
  cp = nltk.RegexpParser(grammar_rules)
  tree = cp.parse(tagged)
  return tree


for sentence in sample_sentences:
  print(chunk_sentence(sentence))
```

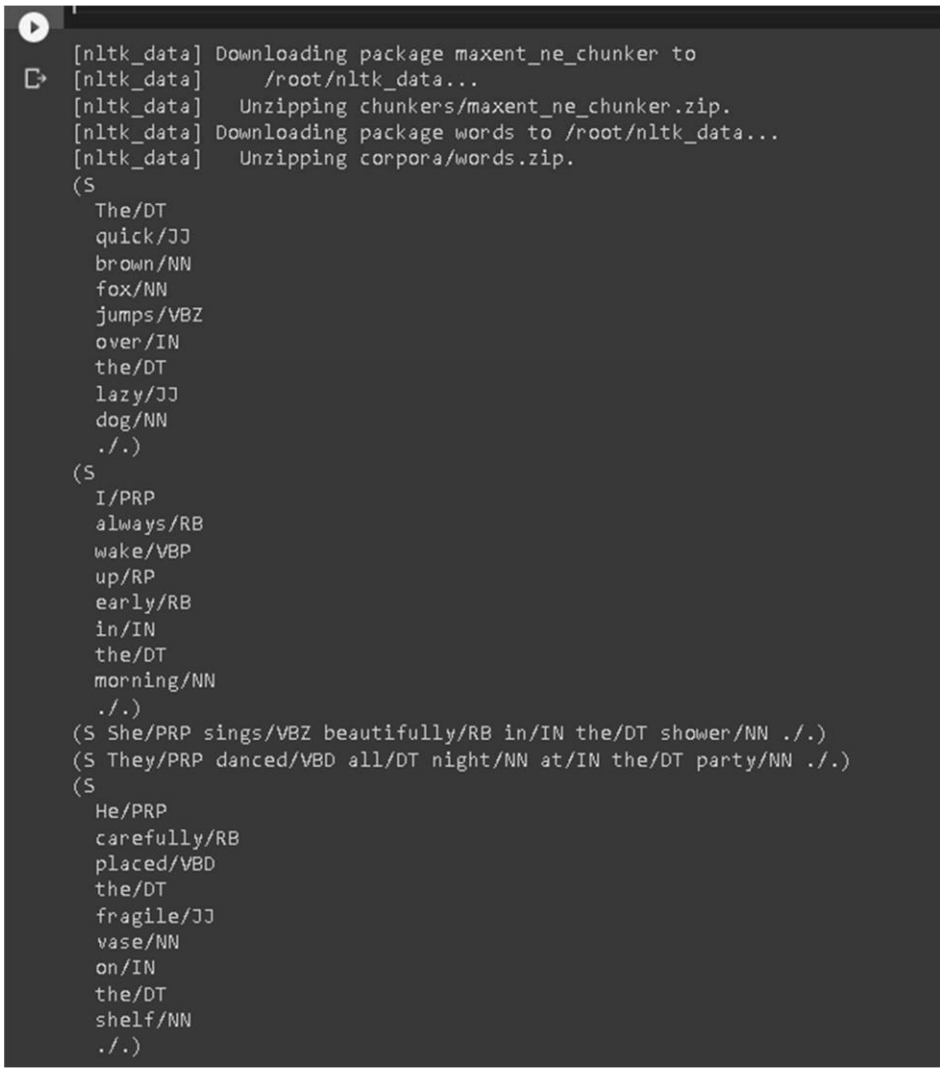
Output:

```
(S
  (NP The/DT quick/JJ brown/NN fox/NN)
  (VP jumps/VBZ over/IN)
  (NP the/DT lazy/JJ dog/NN)
  ./.)
(S
  I/PRP
  (ADVP always/RB)
  (VP wake/VBP up/RP)
  (ADVP early/RB)
  in/IN
  (NP the/DT morning/NN)
  ./.)
(S
  She/PRP
  (VP sings/VBZ)
  (ADVP beautifully/RB)
  in/IN
  (NP the/DT shower/NN)
  ./.)
(S
  They/PRP
  (VP danced/VBD)
  (NP all/DT night/NN)
  at/IN
  (NP the/DT party/NN)
  ./.)
(S
  He/PRP
  (ADVP carefully/RB)
  (VP placed/VBD)
  (NP the/DT fragile/JJ vase/NN)
  on/IN
  (NP the/DT shelf/NN)
  ./.)
```

2] Perform chunking using inbuilt functions:

```
import nltk
nltk.download('maxent_ne_chunker')
nltk.download('words')

sample_sentences = [
  "The quick brown fox jumps over the lazy dog.",
  "I always wake up early in the morning.",
  "She sings beautifully in the shower.",
  "They danced all night at the party.",
  "He carefully placed the fragile vase on the shelf."
]


def chunk_sentence(sentences):
  tokens = nltk.word_tokenize(sentences)
  tagged = nltk.pos_tag(tokens)
  chunked = nltk.ne_chunk(tagged)
  return chunked



for i in sample_sentences:
  print(chunk_sentence(i))
```

Output:



```
[nltk_data] Downloading package maxent_ne_chunker to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping chunkers/maxent_ne_chunker.zip.
[nltk_data] Downloading package words to /root/nltk_data...
[nltk_data]   Unzipping corpora/words.zip.
(S
  The/DT
  quick/JJ
  brown/NN
  fox/NN
  jumps/VBZ
  over/IN
  the/DT
  lazy/JJ
  dog/NN
  ./.)
(S
  I/PRP
  always/RB
  wake/VBP
  up/RP
  early/RB
  in/IN
  the/DT
  morning/NN
  ./.)
(S She/PRP sings/VBZ beautifully/RB in/IN the/DT shower/NN ./.)
(S They/PRP danced/VBD all/DT night/NN at/IN the/DT party/NN ./.)
(S
  He/PRP
  carefully/RB
  placed/VBD
  the/DT
  fragile/JJ
  vase/NN
  on/IN
  the/DT
  shelf/NN
  ./.)
```

Comparing both result:

	Inbuilt Functions	***Customized Grammar rules***
Speed and Efficiency	Generally faster and efficient	May require more time and expertise
Accuracy	May not always provide the desired level of accuracy and customization	Can provide more accurate and specific chunking results
Flexibility	May not always provide the desired level of customization	Can be tailored to the specific needs of an application
Ease of Use	Convenient and easy to use	Requires expertise in creating and tuning rules
Suitable for	Suitable for general-purpose applications or tasks that don't require high customization	Suitable for specific domains or tasks that require high accuracy and customization

Conclusion:

In above assignment, we successfully performed chunking to detect noun, adjective, adverb etc. phrases. We find all these phrases using 2 different technique using inbuilt chunk functions and using customized grammar rules chunking, and last we compare both the result. Inbuilt function gives result fast but not always true, but in case of customized rules result calculating time may be more than inbuilt function, but it produced most accurate result every time.