|  |  |
| --- | --- |
| **Department of Computer Engineering**  **Class: B.E. (Computer) (Div- B) (Sem-VIII)** | |
| **Subject: Computational Lab-II (NLP)(CSL804)** | |
| **Sr. No.** | **Title of Experiment** |
| **6.** | **To perform Chunking operation based on sample input text.** |

|  |
| --- |
| **Course Name:** Computational Lab-II(NLP) |
| **Course Code:**  CSL804 |
| **Experiment No.:** 06 |
| **Lab outcome:** Acquire practical knowledge within the chosen area of technology for project development. |
| **Name of Student:** Rahul S Mahanty |
| **Student Roll No.:**214 |
| **Year/Semester/Div:** B.E./VIII/B |

**Experiment No. 06**

**Aim:** To perform Chunking operation based on sample input text.

**Theory:**

Natural language processing deals with various text processing or phrases extraction process which may be in unstructured form of data. Hence, there is an need for an chunking operation over an particular input text.

**Chunking:**

Chunking is said to be a process of extracting phrases from unstructured text, which means analyzing a sentence to identify the constituents(Noun Groups, Verbs, verb groups, etc.) However, it does not specify their internal structure, nor their role in the main sentence. In general, Chunking means grouping of words/tokens into chunks.

It works on top of POS tagging. It uses POS-tags as input and provides chunks as output.

**Chunk extraction or partial parsing:**

It is defined as a process of meaningful extracting short phrases from the sentence (tagged with Part-of-Speech). Chunks are made up of words and the kinds of words are defined using the part-of- speech tags. One can even define a pattern or words that can’t be a part of chuck and such words are known as chinks. A Chunk Rule class specifies what words or patterns to include and exclude in a chunk.

**Chunk patterns :**

Chuck patterns are normal regular expressions which are modified and designed to match the part-of-speech tag designed to match sequences of part-of-speech tags. Angle brackets are used to specify an individual tag for example – to match a noun tag. One can define multiple tags in the same way.

**Types of Chunking:**

There are, broadly, two types of chunking:

a). Chunking up: b). Chunking down:

**a). Chunking up:**

Here, we don’t dive deep; instead, we are happy with just an overview of the information. It just helps us get a brief idea of the given data.

**b). Chunking down:**

Unlike the previous type of chunking, chunking down helps us get detailed information. So, if you just want an insight, consider “chunking up” otherwise prefer “chunking down”.

**Use of Chunking:**

We can have loads of descriptions or modifications around a particular word or the phrase of our interest, we use chunking to grab the required phrase alone, ignoring the rest around it. Hence, chunking paves a way to group the required phrases and exclude all the modifiers around them which are not necessary for our analysis. Summing up, chunking helps us extract the important words alone from lengthy descriptions. Thus, chunking is a step in information extraction.

Interestingly, this process of chunking in NLP is extended to various other applications; for instance, to group fruits of a specific category, say, fruits rich in proteins as a group, fruits rich in vitamins as another group, and so on. Besides, chunking can also be used to group similar cars, say, cars supporting auto-gear into one group and the others which support manual gear into another chunk and so on.

**Code #1 : Converting chunks to RegEx Pattern.**

# Laading Library

from nltk.chunk.regexp import tag\_pattern2re\_pattern

# Chunk Pattern to RegEx Pattern

print(&”Chunk Pattern :”, tag\_pattern2re\_pattern(‘<DT>?<NN.\*>’))

**Output :**

Chunk Pattern : ()?(<NN[^\{\}]\*)>)+

Curly Braces are used to specify a chunk like {} and to specify the chink pattern one can

just flip the braces }{. For a particular phrase type, these rules (chunk and a chink pattern)

can be combined into a grammer.

**Code #2 : Parsing the sentence with RegExParser.**

from nltk.chunk import RegexpParser

# Introducing the Pattern

chunker = RegexpParser(r’’’ NP:

{<DT><NN.\*><.\*>\*<NN.\*>}

}<VB.\*>{’’’)

chunker.parse([(“the”, “DT”), (“book”, “NN”), ( “has”,“VBZ”), (“many”, “JJ”), (“chapters”, “NNS”)])

**Output :**Tree(‘S’, [Tree(‘NP’, [(‘the’, ‘;DT’), (‘book’, ‘NN’)]), (‘has’,‘VBZ’), Tree(‘NP’, [(‘many’, ‘JJ’), (‘chapters’, ‘NNS’)])])

**Program:**

from nltk.tokenize import word\_tokenize

import nltk

text=word\_tokenize("Nothing is good or bad but thinking makes it so")

output=nltk.pos\_tag(text)

grammar=('''NP: {<DT>?<JJ>\*<NN>}''')

# Applying RegexpChunkParser

chunkParser=nltk.RegexpParser(grammar)

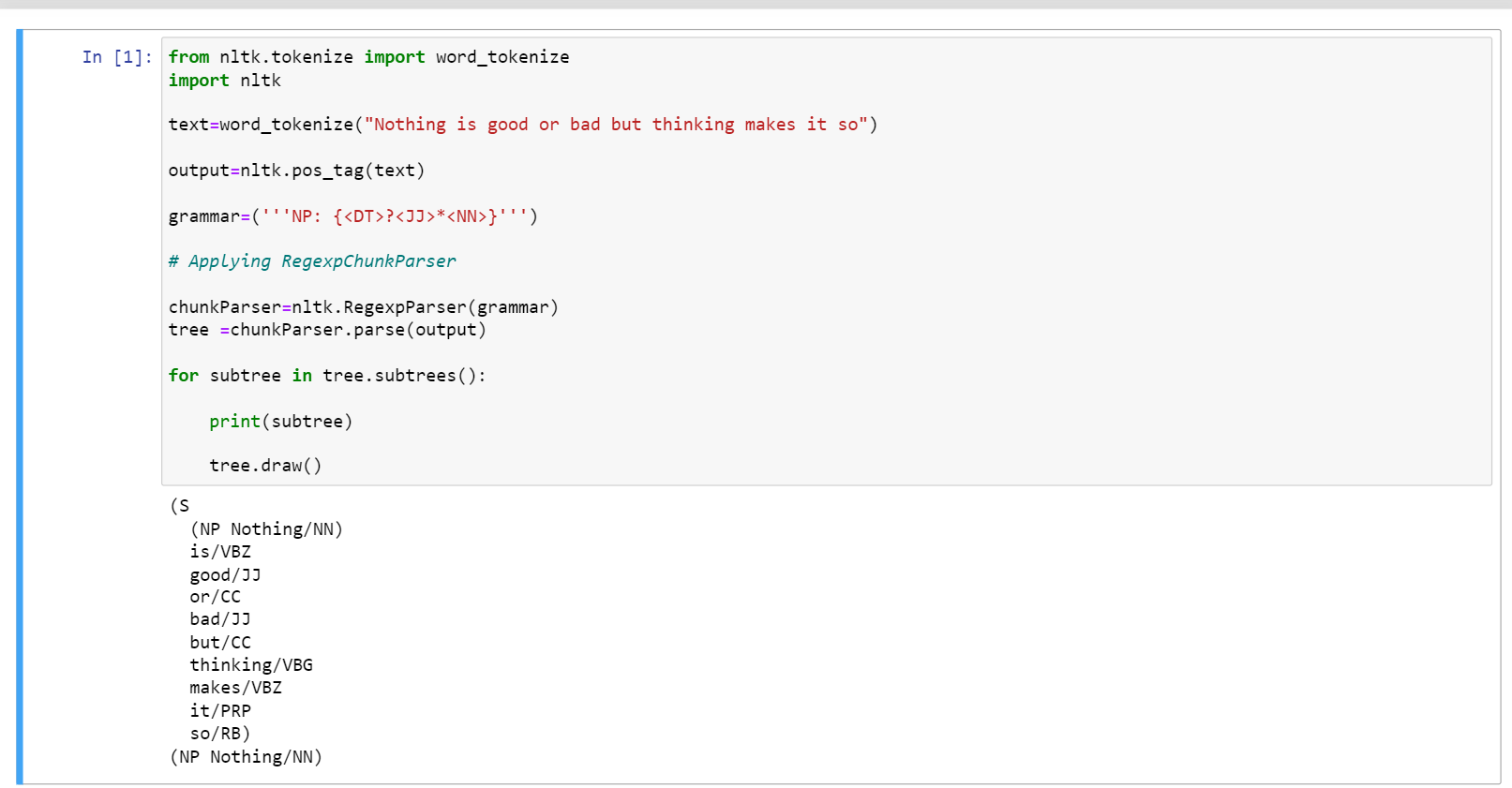
tree =chunkParser.parse(output)

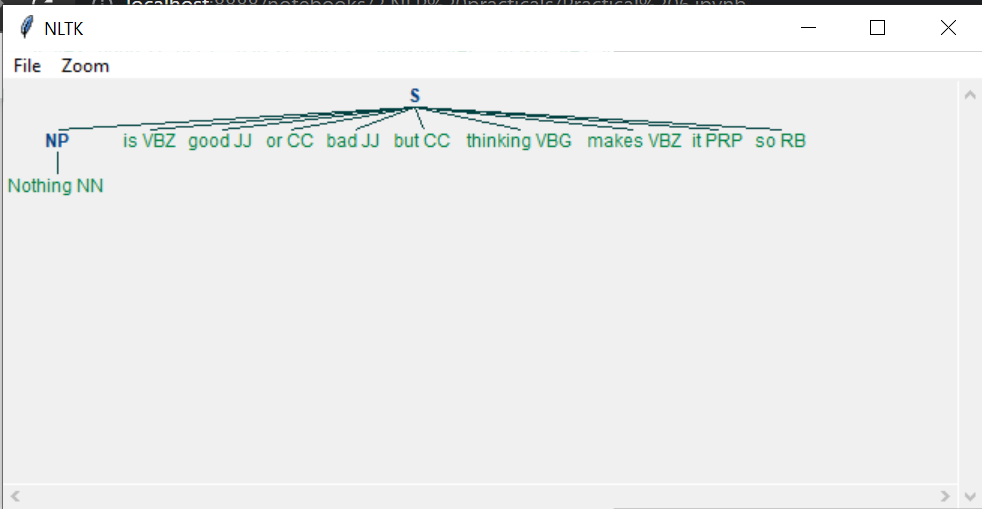
for subtree in tree.subtrees():

print(subtree)

tree.draw()

**Output:**

****

****

**Conclusion:** Therefore, we have successfully perform Chunking operation based on any sample input text.