Now that we know the parts of speech, we can do what is called chunking, and group words into hopefully meaningful chunks. One of the main goals of chunking is to group into what are known as "noun phrases." These are phrases of one or more words that contain a noun, maybe some descriptive words, maybe a verb, and maybe something like an adverb. The idea is to group nouns with the words that are in relation to them.

In order to chunk, we combine the part of speech tags with [regular expressions](https://pythonprogramming.net/regular-expressions-regex-tutorial-python-3/). Mainly from regular expressions, we are going to utilize the following:

+ = match 1 or more

? = match 0 or 1 repetitions.

\* = match 0 or MORE repetitions

. = Any character except a new line

The last things to note is that the part of speech tags are denoted with the "<" and ">" and we can also place regular expressions within the tags themselves, so account for things like "all nouns" (<N.\*>)

import nltk

from nltk.corpus import state\_union

from nltk.tokenize import PunktSentenceTokenizer

train\_text = state\_union.raw("2005-GWBush.txt")

sample\_text = state\_union.raw("2006-GWBush.txt")

custom\_sent\_tokenizer = PunktSentenceTokenizer(train\_text)

tokenized = custom\_sent\_tokenizer.tokenize(sample\_text)

def process\_content():

try:

for i in tokenized:

words = nltk.word\_tokenize(i)

tagged = nltk.pos\_tag(words)

chunkGram = r"""Chunk: {<RB.?>\*<VB.?>\*<NNP>+<NN>?}"""

chunkParser = nltk.RegexpParser(chunkGram)

chunked = chunkParser.parse(tagged)

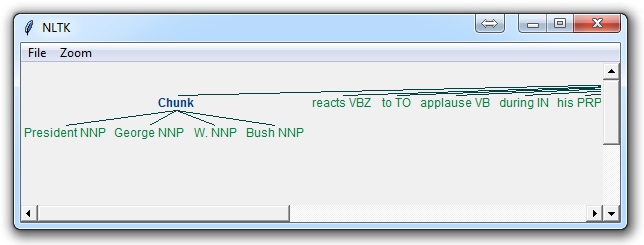
chunked.draw()

except Exception as e:

print(str(e))

process\_content()

The result of this is something like:



The main line here in question is:

chunkGram = r"""Chunk: {<RB.?>\*<VB.?>\*<NNP>+<NN>?}"""

This line, broken down:

<RB.?>\* = "0 or more of any tense of adverb," followed by:

<VB.?>\* = "0 or more of any tense of verb," followed by:

<NNP>+ = "One or more proper nouns," followed by

<NN>? = "zero or one singular noun."

Try playing around with combinations to group various instances until you feel comfortable with chunking.

Not covered in the video, but also a reasonable task is to actually access the chunks specifically. This is something rarely talked about, but can be an essential step depending on what you're doing. Say you print the chunks out, you are going to see output like:

(S

(Chunk PRESIDENT/NNP GEORGE/NNP W./NNP BUSH/NNP)

'S/POS

(Chunk

ADDRESS/NNP

BEFORE/NNP

A/NNP

JOINT/NNP

SESSION/NNP

OF/NNP

THE/NNP

CONGRESS/NNP

ON/NNP

THE/NNP

STATE/NNP

OF/NNP

THE/NNP

UNION/NNP

January/NNP)

31/CD

,/,

2006/CD

THE/DT

(Chunk PRESIDENT/NNP)

:/:

(Chunk Thank/NNP)

you/PRP

all/DT

./.)

Cool, that helps us visually, but what if we want to access this data via our program? Well, what is happening here is our "chunked" variable is an NLTK tree. Each "chunk" and "non chunk" is a "subtree" of the tree. We can reference these by doing something like chunked.subtrees. We can then iterate through these subtrees like so:

for subtree in chunked.subtrees():

print(subtree)

Next, we might be only interested in getting just the chunks, ignoring the rest. We can use the filter parameter in the chunked.subtrees() call.

for subtree in chunked.subtrees(filter=lambda t: t.label() == 'Chunk'):

print(subtree)

Now, we're filtering to only show the subtrees with the label of "Chunk." Keep in mind, this isn't "Chunk" as in the NLTK chunk attribute... this is "Chunk" literally because that's the label we gave it here: chunkGram = r"""Chunk: {<RB.?>\*<VB.?>\*<NNP>+<NN>?}"""

Had we said instead something like chunkGram = r"""Pythons: {<RB.?>\*<VB.?>\*<NNP>+<NN>?}""", then we would filter by the label of "Pythons." The result here should be something like:

-

(Chunk PRESIDENT/NNP GEORGE/NNP W./NNP BUSH/NNP)

(Chunk

ADDRESS/NNP

BEFORE/NNP

A/NNP

JOINT/NNP

SESSION/NNP

OF/NNP

THE/NNP

CONGRESS/NNP

ON/NNP

THE/NNP

STATE/NNP

OF/NNP

THE/NNP

UNION/NNP

January/NNP)

(Chunk PRESIDENT/NNP)

(Chunk Thank/NNP)

Full code for this would be:

import nltk

from nltk.corpus import state\_union

from nltk.tokenize import PunktSentenceTokenizer

train\_text = state\_union.raw("2005-GWBush.txt")

sample\_text = state\_union.raw("2006-GWBush.txt")

custom\_sent\_tokenizer = PunktSentenceTokenizer(train\_text)

tokenized = custom\_sent\_tokenizer.tokenize(sample\_text)

def process\_content():

try:

for i in tokenized:

words = nltk.word\_tokenize(i)

tagged = nltk.pos\_tag(words)

chunkGram = r"""Chunk: {<RB.?>\*<VB.?>\*<NNP>+<NN>?}"""

chunkParser = nltk.RegexpParser(chunkGram)

chunked = chunkParser.parse(tagged)

print(chunked)

for subtree in chunked.subtrees(filter=lambda t: t.label() == 'Chunk'):

print(subtree)

chunked.draw()

except Exception as e:

print(str(e))

process\_content()