Context-Free Grammar

Submitted By

Sanjana Rajagopala

SUID - 607219462

DESCRIPTION OF APPROACH AND RESULTS

STEP 1: STORAGE OF WORDS IN BUFFER

As a first step in constructing the CFG for any corpus, we begin with defining the required sentence in our corpus and storing them in string buffers. This serves as the training corpus for our grammar.

STEP 2: POS TAGGING

After the construction of input sentences, we need to identify the parts of speech of the words. This step is necessary because it helps in recognition of the suitable POS tag of each word token so that the grammar can be built accordingly. To do so, each of the sentences are tokenized into a list and further, each word in the sentence is tokenized. Considering the high performance and maximum entropy classifier of the Stanford POS tagger, I used it to tag the given corpus. This step serves as the foundation for defining the CFG appropriately.

STEP 3: CONSTRUCTION OF CONTEXT-FREE GRAMMAR

Before constructing the CFG, we assume that the words in this corpus are known to us. Keeping the POS tags as the basis, I defined the CFG in the following manner –

1. Definition of the Sentence – As per the given corpus we have a sentence largely divided into noun phrase and verb phrase. Hence, the rule with S -> NP VP. For now, I have not taken the case where in a sentence contains only a verb phrase (Restricting to the current corpus of sentences).
2. Definition of the Noun Phrase – it can be observed that any noun phrase involves nouns, adjectives, adverbs and determiners.

Sl. No	Rule	Description	Example
1.	NP -> PRP	Noun phrase can contain only a personal noun/pronoun	“We”, “She”
2.	NP -> PRP ADVP	The personal noun/pronoun is in combination with adverbs	“me two days ago”. Here, ‘ago’ marks the head of the entire phrase following the noun ‘me’.
3.	NP -> PRP NN	The personal noun/pronoun in combination with another noun.	“Their kids”.
4.	NP -> Det ADP NN	The noun phrase begins with determiners and includes adjectives which are generally followed by a noun. It also handles the case wherein another noun can follow the noun in the adjective phrase.	“a nice party yesterday” where ‘yesterday’ is the noun following another noun ‘party’.

3. Definition of the Adjective phrase - mainly consists of adjectives describing the nouns.

Sl. No	Rule	Description	Example
1.	ADP -> JJ NN	The preceding adjective is used for describing that noun.	“nice party”.
2.	ADP -> JJ	Contains merely the adjective itself.	“naive”

4. Definition of Adverb phrases – which contains a variety of tags such as adverb and cardinal number.

Sl. No	Rule	Description	Example
1.	ADVP -> CD NN RB	The adverb phrase begins with a cardinal number defining the noun. This handles the special case of the adverb present far from the describing verb.	“two days ago”. Here, the adverb ‘ago’ is meant to describe the verb ‘visit’ and the presence of ‘two’, ‘days’ in the adverb phrase adds to the description.
2.	ADVP -> RB ADP	The adverb followed by an adjective phrase	“always naïve”
3.	ADVP -> RB ADVP	This covers the recursive rule of an adverb followed by another adverb phrase.	“not always naïve”
4.	ADVP -> RB	The typical case of adverb phrase consisting of only the adverb.	“now”

5. Definition of the Verb phrase – includes the presence of verbs, infinitives such as ‘To’, modal verbs and adverbs.

Sl. No	Rule	Description	Example
1.	VP -> VPS NP	The common verb phrase consisting of verb followed by another noun phrase	“had a nice party”
2.	VP -> VPS TO VP	This covers the cases in which the “to” infinitive is followed by another verb phrase	“to visit”
3.	VP -> MD VP	This handles the verb phrases with modal verbs	“may go”
4.	VP -> VPS ADVP	This is the typical case of verb phrase followed by an adverb phrase.	“go now”
5.	VP -> VBP ADVP	This covers the cases in which the simple present tense verbs are followed by the adverb phrase.	“are not” where ‘not’ is the adverb.

6. Definition of verb types – It is known that we have different verb tenses. This rule considers such forms –

Sl. No	Rule	Description	Example
1.	VPS -> VB	Refers to the verbs in base form	“Go”, “visit”
2.	VPS -> VBD	Refers to the verbs in past tense	“had”, “came”

Under this we do not include the verbs such as “are”. It is defined this way in order to have distinction between those forms of verbs that may not be combined with “to”. For example, the usage “are to party” is not valid grammatically and hence, I have defined the separate rules allowing only “VBP TO VP”. In this manner, the sentences such as “They are to party always” are not parsed by the CFG. Thus, by having separate non-terminals for these verbs the quality of parsing is improved.

7. Definition of the specific words – Once we define the non-terminals, we proceed to constructing the terminals as per the given corpus; that includes nouns, verbs, adverbs, adjectives and modals.

The complete grammar is as follows –

```
S -> NP VP
NP -> PRP |PRP ADVP | PRP NN | Det ADP NN
ADP -> JJ NN | JJ
ADVP -> CD NN RB | RB ADP | RB ADVP | RB
VP -> VPS NP | VPS TO VP | MD VP | VBP ADVP | VPS ADVP
VPS -> VB | VBD
VB -> "visit" | "go"
VBD -> "had" | "came"
VBP -> "are"
JJ -> "nice" | "naive"
Det -> "a"
NN -> "party" | "yesterday" | "days" | "kids"
PRP -> "We" | "She" | "me" | "You" | "Their"
TO -> "to"
CD -> "two"
RB -> "ago" | "now" | "not" | "always"
MD -> "may"
```

STEP 4: PARSING USING CFG

With the CFG ready, it is now required to test it. To do so, we perform “parsing’ of the corpus using RecursiveDescent Parser. This helps in assigning the proper trees from the CFG to the input strings. This step indicates to us the performance of our grammar with respect to the give corpus–

a) **Successful Parsing** - All the sentences in the corpus are parsed by the CFG and the corresponding trees are returned.

b) **Structural Ambiguity** – refers to whether the CFG was able to return non-ambiguous parsing trees for a given input sentence. The Python parsing screenshots depict that my CFG is capable of giving non-ambiguous trees for a given input.

c) **Non-restrictive grammar** – This means how well the grammar can parse sentences other than those defined in the given corpus. The Python parsing screenshots of my new example sentences show this property in my CFG.

The other three sentences generated by the CFG –

Sentence-1: She had to go now.

Sentence-2: You are always nice.

Sentence-3: We may visit a nice party.

d) **Generation of sentences with no sense** – This indicates if the grammar is parsing the sentences that actually, make no sense globally in terms of semantics. This CFG showed the parsing of non-sensical sentences.

For example, the sentence is "She visit me always". It can be observed from the screenshot that the above sentence is successfully parsed using the CFG and the corresponding tree is also shown to be consistent with the rules defined in the CFG. It is allowed to have a personal pronoun "She" followed by a verb phrase containing the verb "visit" and another noun phrase "me always" where "always" is the adverb.

However, if we observe the sentence, it makes no sense in real world. The meaning that the sentence is trying to convey is insensible.

Answering to the question – "Why can't you have such a sentence based on your grammar?"

By definition, CFG is meant to identify and define the way in which words can be arranged together. This is part of the "Syntactic Analysis" wherein we are concerned only about the syntactic structure of the sentences and not bothered about the "semantics". This is independent of the meaning of the sentence. As a result of which, such sentences are syntactically acceptable as per the rules/productions in the CFG irrespective of whether they make sense globally.

Sentence: "She visit me always"

This represents the general problems that are difficult to be represented in CFG because of which the above generation of sentence with no-sense was possible –

1. **Agreement** - In the current sentence, the noun "She" and the verb "visit" do not agree with one another. To handle this, we may be required to add additional layers of rules with agreement.

2. **Subcategorization** – Not an issue in this sentence since, the verb "visit" and its noun arguments "me" are permitted to occur together.

3. **Movement** – This is not an issue in the current sentence because the direct object argument “me” and the verb “visit” occur at right places.

STEP 5: CONSTRUCTION OF PROBABLISTIC CFG

By making use of the above generated CFG, we can define the probabilistic CFG. This is calculated as –

$$\frac{\text{Count of the number of times of usage of a production/rule}}{\text{Count of overall total of the rule in the entire corpus}}$$

For example, in case of the verb phrase rule “VP”, the production “**VP -> VPS NP**” is used 2 times and the phrase **VP** as whole is used overall 6 times. Hence, this production is assigned a probability of 0.33.

Here’s the PCFG with the assigned probabilities –

```
    S -> NP VP [1.0]
    NP -> PRP [0.5]
    NP -> PRP ADVP [0.17]
    NP -> PRP NN [0.16]
    NP -> Det ADP NN [0.17]
    ADP -> JJ NN [0.5]
    ADP -> JJ [0.5]
    ADVP -> CD NN RB [0.25]
    ADVP -> RB ADP [0.25]
    ADVP -> RB ADVP [0.25]
    ADVP -> RB [0.25]
    VP -> VPS NP [0.34]
    VP -> VPS TO VP [0.17]
    VP -> MD VP [0.17]
    VP -> VPS ADVP [0.16]
    VP -> VBP ADVP [0.16]
    VPS -> VB [0.5] | VBD [0.5]
    VB -> "visit" [0.5] | "go" [0.5]
    VBD -> "had" [0.5] | "came" [0.5]
    VBP -> "are" [1.0]
    JJ -> "nice" [0.5] | "naive" [0.5]
    Det -> "a" [1.0]
    NN -> "party"[0.25] | "yesterday" [0.25] | "days" [0.25] |
"kids" [0.25]
    PRP -> "We" [0.2] | "She" [0.2] | "me" [0.2] | "You" [0.2] |
"Their" [0.2]
    TO -> "to" [1.0]
    CD -> "two" [1.0]
    RB -> "ago" [0.25] | "now" [0.25] | "not" [0.25] | "always"
[0.25]
    MD -> "may" [1.0]
```

NOTE: In the cases wherein the probability values are recurring decimals, the rounding-off adjustments are done to ensure the total probability sums up to 1.

In this way, the probabilities are assigned to each of the rules and the PCFG is defined.

STEP 6: PARSING USING PCFG

The given 4 sentences in the corpus are parsed using the Viterbi Parser. The results are mostly similar to what we obtained in the previous parser for the CFG. Except for we obtain a probability value of the parse tree – which is the product of probabilities of the rules used in derivation.

APPENDIX

OUTPUT

CFG GRAMMAR RESULTS

Given Corpus of 4 sentences

Parsing output for first sentence – “We had a nice party yesterday”

```
(S
  (NP (PRP We))
  (VP
    (VPS (VBD had))
    (NP (Det a) (ADP (JJ nice) (NN party)) (NN yesterday))))
```

Parsing output for second sentence – “She came to visit me two days ago”.

```
(S
  (NP (PRP She))
  (VP
    (VPS (VBD came))
    (TO to)
    (VP
      (VPS (VB visit))
      (NP (PRP me) (ADVP (CD two) (NN days) (RB ago))))))
```

Parsing output for third sentence – “You may go now”.

```
(S (NP (PRP You)) (VP (MD may) (VP (VPS (VB go)) (ADVP (RB now)))))
```

Parsing output for fourth sentence – “Their kids are not always naive”.

```
(S
  (NP (PRP Their) (NN kids))
  (VP (VBP are) (ADVP (RB not) (ADVP (RB always) (ADP (JJ naive))))))
```

New sentences

Parsing output for new sentence – “She had to go now”.

```
(S
  (NP (PRP She))
  (VP (VPS (VBD had)) (TO to) (VP (VPS (VB go)) (ADVP (RB now)))))
```

Parsing output for new sentence – “You are always nice”.

```
(S (NP (PRP You)) (VP (VBP are) (ADVP (RB always) (ADP (JJ nice)))))
```

Parsing output for new sentence – “We may visit a nice party”.

```
(S
  (NP (PRP We))
  (VP
    (MD may)
    (VP (VPS (VB visit)) (NP (Det a) (ADP (JJ nice)) (NN party)))))
```

The Sentence with no sense

“She visit me always”

```
(S
  (NP (PRP She))
  (VP (VPS (VB visit)) (NP (PRP me) (ADVP (RB always)))))
```

PCFG GRAMMAR RESULTS

Given corpus of 4 sentences

Parsing output for first sentence – “We had a nice party yesterday”

```
(S
  (NP (PRP We))
  (VP
    (VPS (VBD had))
    (NP (Det a) (ADP (JJ nice) (NN party)) (NN yesterday)))) (p=2.25781e-05
)
```

Parsing output for second sentence – “She came to visit me two days ago”.

```
(S
  (NP (PRP She))
  (VP
    (VPS (VBD came))
    (TO to)
    (VP
      (VPS (VB visit))
      (NP (PRP me) (ADVP (CD two) (NN days) (RB ago)))))) (p=1.91914e-07
)
```

Parsing output for third sentence – "You may go now".

```
(S
  (NP (PRP You))
  (VP (MD may) (VP (VPS (VB go)) (ADVP (RB now))))) (p=4.25e-05)
```

Parsing output for fourth sentence – "Their kids are not always naive".

```
(S
  (NP (PRP Their) (NN kids))
  (VP
    (VBP are)
    (ADVP (RB not) (ADVP (RB always) (ADP (JJ naive)))))) (p=1.25e-06)
```

New sentences

Parsing output for new sentence – "She had to go now".

```
(S
  (NP (PRP She))
  (VP
    (VPS (VBD had))
    (TO to)
    (VP (VPS (VB go)) (ADVP (RB now))))) (p=1.0625e-05)
```

Parsing output for new sentence – "We may visit a nice party".

```
(S
  (NP (PRP You))
  (VP (VBP are) (ADVP (RB always) (ADP (JJ nice))))) (p=0.00025)
```

Parsing output for new sentence – "We may visit a nice party".

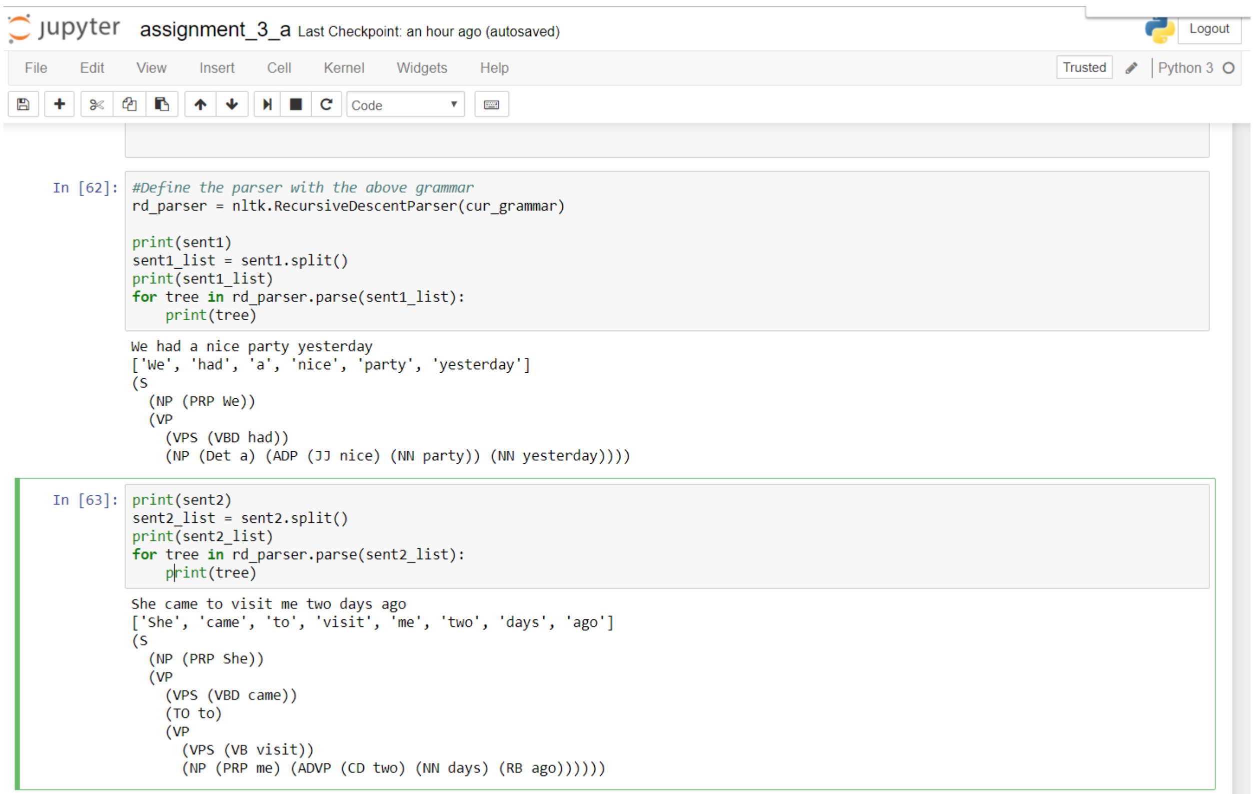
```
(S
  (NP (PRP We))
  (VP
    (MD may)
    (VP
      (VPS (VB visit))
      (NP (Det a) (ADP (JJ nice)) (NN party))))) (p=1.53531e-05)
```

Sentence with no sense

```
(S
  (NP (PRP She))
  (VP (VPS (VB visit)) (NP (PRP me) (ADVP (RB always))))) (p=1.80625e-05
)
```

PYTHON SCREENSHOTS

1. Parsing output for first and second sentence using CFG



jupyter assignment_3_a Last Checkpoint: an hour ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Code

```
In [62]: #Define the parser with the above grammar
rd_parser = nltk.RecursiveDescentParser(cur_grammar)

print(sent1)
sent1_list = sent1.split()
print(sent1_list)
for tree in rd_parser.parse(sent1_list):
    print(tree)

We had a nice party yesterday
['We', 'had', 'a', 'nice', 'party', 'yesterday']
(S
  (NP (PRP We))
  (VP
    (VPS (VBD had))
    (NP (Det a) (ADP (JJ nice) (NN party)) (NN yesterday))))

In [63]: print(sent2)
sent2_list = sent2.split()
print(sent2_list)
for tree in rd_parser.parse(sent2_list):
    print(tree)

She came to visit me two days ago
['She', 'came', 'to', 'visit', 'me', 'two', 'days', 'ago']
(S
  (NP (PRP She))
  (VP
    (VPS (VBD came))
    (TO to)
    (VP
      (VPS (VB visit))
      (NP (PRP me) (ADVP (CD two) (NN days) (RB ago))))))
```

2. Parsing output for third and fourth sentence using CFG

jupyter assignment_3_a Last Checkpoint: an hour ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
In [64]: print(sent3)
         sent3_list = sent3.split()
         print(sent3_list)
         for tree in rd_parser.parse(sent3_list):
             print(tree)

You may go now
['You', 'may', 'go', 'now']
(S (NP (PRP You)) (VP (MD may) (VP (VPS (VB go)) (ADVP (RB now)))))
```

```
In [65]: print(sent4)
         sent4_list = sent4.split()
         print(sent4_list)
         for tree in rd_parser.parse(sent4_list):
             print(tree)

Their kids are not always naive
['Their', 'kids', 'are', 'not', 'always', 'naive']
(S
  (NP (PRP Their) (NN kids))
  (VP (VBP are) (ADVP (RB not) (ADVP (RB always) (ADP (JJ naive))))))
```

3. Parsing output for two new sentences using CFG

jupyter assignment_3_a Last Checkpoint: an hour ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
In [66]: #The three other sentences that can be parsed by the above grammar

         my_sent1 = "She had to go now"
         print(my_sent1)
         my_sentlist1= my_sent1.split()
         print(my_sentlist1)
         for t in rd_parser.parse(my_sentlist1):
             print(t)

She had to go now
['She', 'had', 'to', 'go', 'now']
(S
  (NP (PRP She))
  (VP (VPS (VBD had)) (TO to) (VP (VPS (VB go)) (ADVP (RB now)))))
```

```
In [67]: my_sent2 = "You are always nice"
         print("\n"+my_sent2)
         my_sentlist2= my_sent2.split()
         print(my_sentlist2)
         for t in rd_parser.parse(my_sentlist2):
             print(t)


You are always nice
['You', 'are', 'always', 'nice']
(S (NP (PRP You)) (VP (VBP are) (ADVP (RB always) (ADP (JJ nice)))))
```

4. Parsing output for third new sentence that makes sense and another sentence with no global sense using CFG

jupyter assignment_3_a Last Checkpoint: an hour ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [68]:

```
my_sent3 = "We may visit a nice party"
print("\n"+my_sent3)
my_sentlist3= my_sent3.split()
print(my_sentlist3)
for t in rd_parser.parse(my_sentlist3):
    print(t)
```

```
We may visit a nice party
['We', 'may', 'visit', 'a', 'nice', 'party']
(S
  (NP (PRP We))
  (VP
    (MD may)
    (VP (VPS (VB visit)) (NP (Det a) (ADP (JJ nice)) (NN party)))))
```

In [75]:

```
#The sentence that does not make sense
no_sense_sent = "She visit me always"
no_sense_list= no_sense_sent.split()
print("Sentence with no sense\n")
print(no_sense_sent+"\n")
print(no_sense_list)
for t in rd_parser.parse(no_sense_list):
    print(t)
```

```
Sentence with no sense

She visit me always

['She', 'visit', 'me', 'always']
(S
  (NP (PRP She))
  (VP (VPS (VB visit)) (NP (PRP me) (ADVP (RB always)))))
```

5. Parsing output for the first and second sentences using PCFG

jupyter assignment_3_a Last Checkpoint: an hour ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help — Trusted — Python 3

In [78]:
```python
#Parse the given 4 sentence using the PCFG grammar

print(sent1)
print(sent1_list)
for tree in viterbi_parser.parse(sent1_list):
    print(tree)
```

```
We had a nice party yesterday
['We', 'had', 'a', 'nice', 'party', 'yesterday']
(S
  (NP (PRP We))
  (VP
    (VPS (VBD had))
    (NP (Det a) (ADP (JJ nice) (NN party)) (NN yesterday)))) (p=2.25781e-05)
```

In [79]:
```python
print(sent2)
print(sent2_list)
for tree in viterbi_parser.parse(sent2_list):
    print(tree)
```

```
She came to visit me two days ago
['She', 'came', 'to', 'visit', 'me', 'two', 'days', 'ago']
(S
  (NP (PRP She))
  (VP
    (VPS (VBD came))
    (TO to)
    (VP
      (VPS (VB visit))
      (NP (PRP me) (ADVP (CD two) (NN days) (RB ago)))))) (p=1.91914e-07)
```

6. Parsing output for third and fourth sentences using PCFG

jupyter assignment_3_a Last Checkpoint: an hour ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help — Trusted — Python 3

In [80]:
```python
print(sent3)
print(sent3_list)
for tree in viterbi_parser.parse(sent3_list):
    print(tree)
```

```
You may go now
['You', 'may', 'go', 'now']
(S
  (NP (PRP You))
  (VP (MD may) (VP (VPS (VB go)) (ADVP (RB now))))) (p=4.25e-05)
```

In [81]:
```python
print(sent4)
print(sent4_list)
for tree in viterbi_parser.parse(sent4_list):
    print(tree)
```

```
Their kids are not always naive
['Their', 'kids', 'are', 'not', 'always', 'naive']
(S
  (NP (PRP Their) (NN kids))
  (VP
    (VBP are)
    (ADVP (RB not) (ADVP (RB always) (ADP (JJ naive)))))) (p=1.25e-06)
```

7. Parsing output for new sentences using PCFG

jupyter assignment_3_a Last Checkpoint: an hour ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
In [82]: #Parsing the other 3 sentences of my own using the PCFG
print(my_sent1)
print(my_sentlist1)
for t in viterbi_parser.parse(my_sentlist1):
    print(t)
```

```
She had to go now
['She', 'had', 'to', 'go', 'now']
(S
  (NP (PRP She))
  (VP
    (VPS (VBD had))
    (TO to)
    (VP (VPS (VB go)) (ADVP (RB now))))) (p=1.0625e-05)
```

```
In [83]: print(my_sent2)
print(my_sentlist2)
for t in viterbi_parser.parse(my_sentlist2):
    print(t)
```

```
You are always nice
['You', 'are', 'always', 'nice']
(S
  (NP (PRP You))
  (VP (VBP are) (ADVP (RB always) (ADP (JJ nice))))) (p=0.00025)
```

8. Parsing output for new sentence with sense and another sentence with no global sense using the PCFG

jupyter assignment_3_a Last Checkpoint: an hour ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
In [84]: print(my_sent3)
print(my_sentlist3)
for t in viterbi_parser.parse(my_sentlist3):
    print(t)
```

```
We may visit a nice party
['We', 'may', 'visit', 'a', 'nice', 'party']
(S
  (NP (PRP We))
  (VP
    (MD may)
    (VP
      (VPS (VB visit))
      (NP (Det a) (ADP (JJ nice)) (NN party))))) (p=1.53531e-05)
```

```
In [85]: #Parsing the sentence that does not make sense but parsed by the grammar using PCFG

print(no_sense_sent)
print(no_sense_list)
for t in viterbi_parser.parse(no_sense_list):
    print(t)
```

```
She visit me always
['She', 'visit', 'me', 'always']
(S
  (NP (PRP She))
  (VP (VPS (VB visit)) (NP (PRP me) (ADVP (RB always))))) (p=1.80625e-05)
```