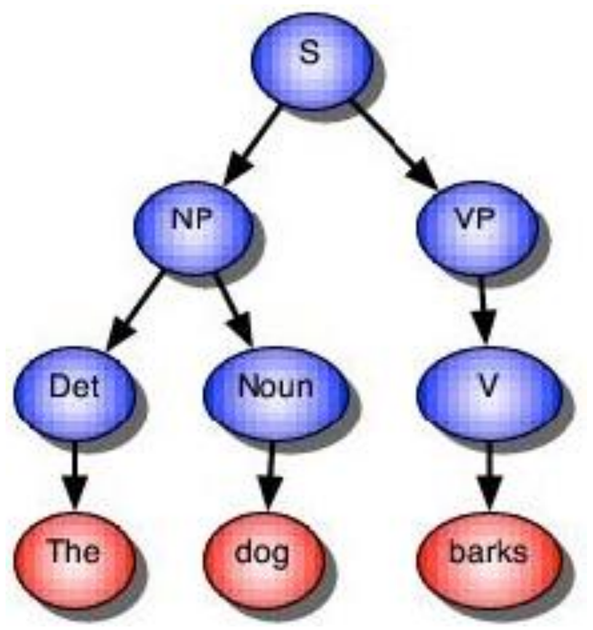


CONTEXT FREE GRAMMAR

NLP Homework 3

Komal Gujarathi
SUID - 211778351

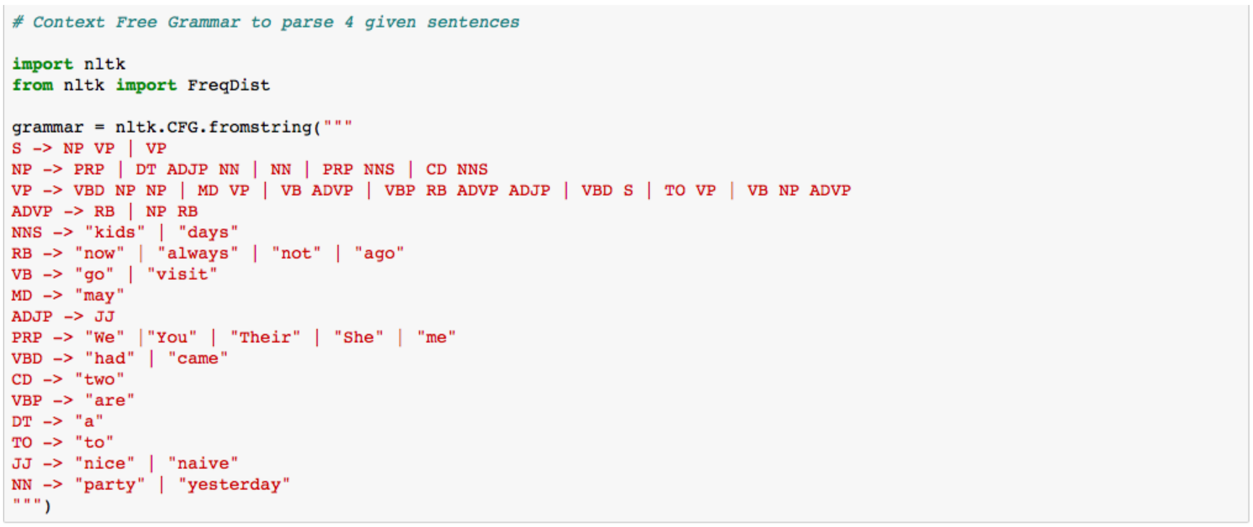
Table of Contents

1. Context Free Grammar

Develop a Context Free Grammar which can parse the following sentences. Use python environment to test your grammar to make sure it does the job and save python screen shots.

a. "We had a nice party yesterday"
b. "She came to visit me two days ago"
c. "You may go now"
d. "Their kids are not always naive"

Following is the grammar generated to parse the given sentences using recursive decent parser.



```
# Context Free Grammar to parse 4 given sentences

import nltk
from nltk import FreqDist

grammar = nltk.CFG.fromstring("""
S -> NP VP | VP
NP -> PRP | DT ADJP NN | NN | PRP NNS | CD NNS
VP -> VBD NP NP | MD VP | VB ADVP | VBP RB ADVP ADJP | VBD S | TO VP | VB NP ADVP
ADVP -> RB | NP RB
NNS -> "kids" | "days"
RB -> "now" | "always" | "not" | "ago"
VB -> "go" | "visit"
MD -> "may"
ADJP -> JJ
PRP -> "We" |"You" | "Their" | "She" | "me"
VBD -> "had" | "came"
CD -> "two"
VBP -> "are"
DT -> "a"
TO -> "to"
JJ -> "nice" | "naive"
NN -> "party" | "yesterday"
""")
```

Please find the python processing screen shots in the appendix.

2. Sample Sentences that can be parsed by this CFG

Besides these four sentences, list up to three different sentences that can be parsed by this grammar as well as the corresponding python file in the screen shot. Of these three sentences, can you generate one that does not make sense? Why can’t you have such a sentence based on your grammar?

Three different sentences which can also be parsed using this grammar are –
1. She may visit now
2. You are not always nice
3. She came two kids yesterday

I have showed the python processing screen shots in the appendix. All the three sentences can be parsed by the recursive decent parser. The third sentence is parsed even if it doesn’t make sense. This is because Context Free Grammars can represent many parts of natural language adequately, but still has the problems like Agreement, Subcategorization, Movement.

The third sentence should actually be – “She came **with** two kids yesterday”, whereas our grammar could parse it even though the preposition ‘with’ was missing from the sentence. This also happens because, since when one rule derives to multiple different rules, then our parser can find the match to any rule, even though it might not make sense. In order to parse this meaningful sentence, we might need to add more rules to our grammar to parse the sentence.

3. Probabilistic Context Free Grammar

Now, assuming that the above four sentences are your mini - mini training corpus, you will write a probabilistic context - free grammar. You will use Python environments to test your grammar and save the Python screenshot. (40%)

A Probabilistic Context-Free-Grammar is simply a CFG with probabilities assigned to the rules such that the sum of all the probabilities assigned to the rules such that the sum of all probabilities for all rules expanding the same non-terminal is equal to one.

I have created a mini corpus of given 4 sentences as shown in the python processing screenshots. After creating this mini corpus, I have calculated the frequency counts in a corpus.

From the frequency distribution obtained from the words in this corpus, we can see that each word has the equal frequency of occurring, (that is 1 for each word). Thus, each word is equally likely to appear in the corpus.

So, while creating the probabilistic grammar, I have assigned the equal frequency to each terminal symbol.
For example, NN -> “party” | “yesterday”
In this, I have assigned the probability of 0.5 and 0.5 to each of these words, because each word is equally likely to appear in the text.

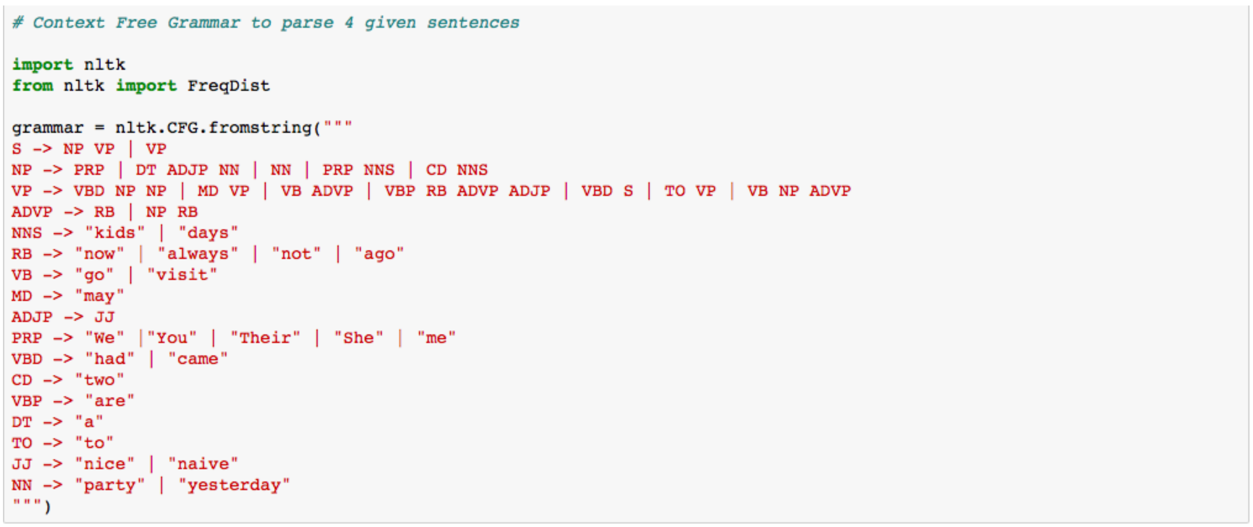
In order to decide the probability of the Non-terminal symbols we can analyze the text visually, We see that the rule S -> NP VP is used 90% of the times while parsing the sentences, and S -> VP is hardly used. Thus distribution –
S -> NP VP[0.9] | VP[0.1]

Only thing we need to make sure in this case is that the sum of the probabilities of all the non-terminal symbols must add up to 1.

Please find the Grammar and the parsed sentences in the screenshots.

4. Appendix

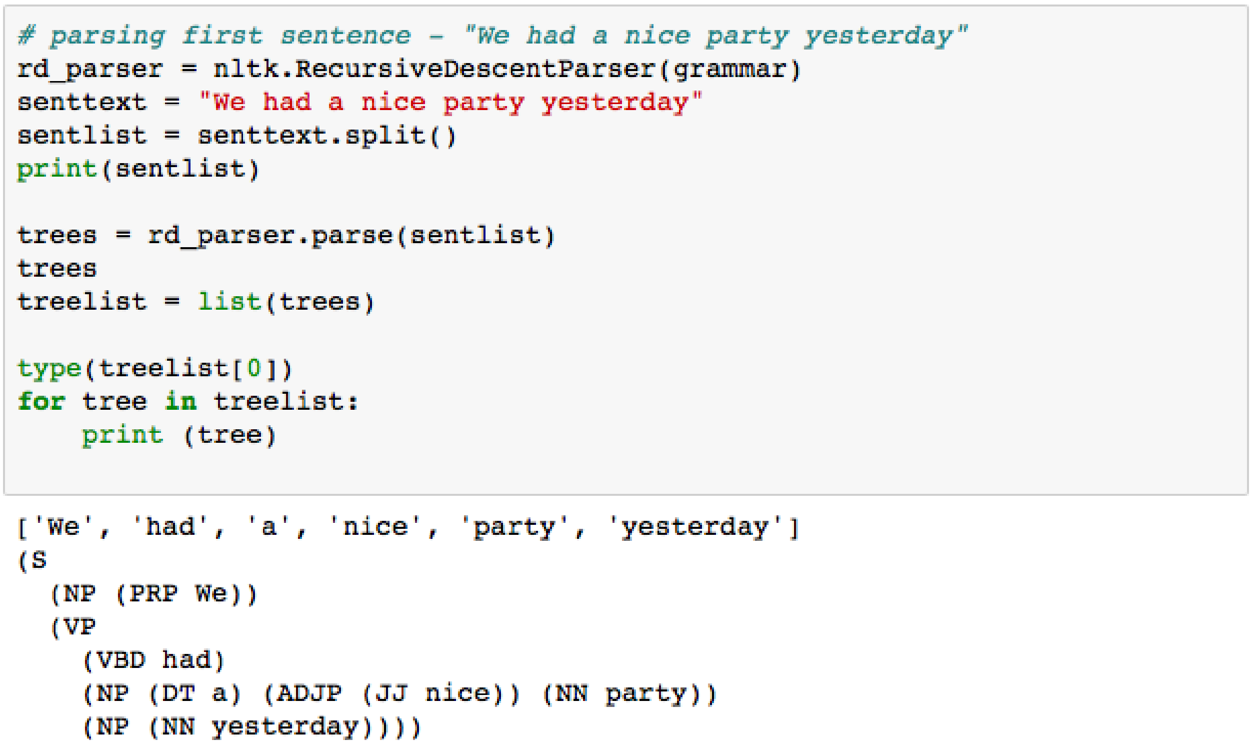
Code and Output



```
# Context Free Grammar to parse 4 given sentences

import nltk
from nltk import FreqDist

grammar = nltk.CFG.fromstring("""
S -> NP VP | VP
NP -> PRP | DT ADJP NN | NN | PRP NNS | CD NNS
VP -> VBD NP NP | MD VP | VB ADVP | VBP RB ADVP ADJP | VBD S | TO VP | VB NP ADVP
ADVP -> RB | NP RB
NNS -> "kids" | "days"
RB -> "now" | "always" | "not" | "ago"
VB -> "go" | "visit"
MD -> "may"
ADJP -> JJ
PRP -> "We" |"You" | "Their" | "She" | "me"
VBD -> "had" | "came"
CD -> "two"
VBP -> "are"
DT -> "a"
TO -> "to"
JJ -> "nice" | "naive"
NN -> "party" | "yesterday"
""")
```



```
# parsing first sentence - "We had a nice party yesterday"
rd_parser = nltk.RecursiveDescentParser(grammar)
senttext = "We had a nice party yesterday"
sentlist = senttext.split()
print(sentlist)

trees = rd_parser.parse(sentlist)
trees
treelist = list(trees)

type(treelist[0])
for tree in treelist:
    print (tree)
```

```
['We', 'had', 'a', 'nice', 'party', 'yesterday']
(S
  (NP (PRP We))
  (VP
    (VBD had)
    (NP (DT a) (ADJP (JJ nice)) (NN party))
    (NP (NN yesterday))))
```

```
# parsing second sentence - "She came to visit me two days ago
rd_parser = nltk.RecursiveDescentParser(grammar)
senttext = "She came to visit me two days ago"
sentlist = senttext.split()
print(sentlist)

trees = rd_parser.parse(sentlist)
trees
treelist = list(trees)

type(treelist[0])
for tree in treelist:
    print (tree)
```

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

```
# parsing third sentence - "You may go now"
rd_parser = nltk.RecursiveDescentParser(grammar)
senttext = "You may go now"
sentlist = senttext.split()
print(sentlist)

trees = rd_parser.parse(sentlist)
trees
treelist = list(trees)

type(treelist[0])
for tree in treelist:
    print (tree)
```

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

```
# parsing fourth sentence - "Their kids are not always naive"
rd_parser = nltk.RecursiveDescentParser(grammar)
senttext = "Their kids are not always naive"
sentlist = senttext.split()
print(sentlist)

trees = rd_parser.parse(sentlist)
trees
treelist = list(trees)

type(treelist[0])
for tree in treelist:
    print (tree)
```

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

```
# sample sentence 1
rd_parser = nltk.RecursiveDescentParser(grammar)
senttext = "She may visit now"
sentlist = senttext.split()
print(sentlist)

trees = rd_parser.parse(sentlist)
trees
treelist = list(trees)

type(treelist[0])
for tree in treelist:
    print (tree)
```

```
['She', 'may', 'visit', 'now']
(S (NP (PRP She)) (VP (MD may) (VP (VB visit) (ADVP (RB now)))))
```

```python
# sample sentence 2
rd_parser = nltk.RecursiveDescentParser(grammar)
senttext = "You are not always nice"
sentlist = senttext.split()
print(sentlist)

trees = rd_parser.parse(sentlist)
trees
treelist = list(trees)

type(treelist[0])
for tree in treelist:
    print (tree)
```

```
['You', 'are', 'not', 'always', 'nice']
(S
  (NP (PRP You))
  (VP (VBP are) (RB not) (ADVP (RB always)) (ADJP (JJ nice))))
```

```python
# sample sentence 3
rd_parser = nltk.RecursiveDescentParser(grammar)
senttext = "She came two kids yesterday"
sentlist = senttext.split()
print(sentlist)

trees = rd_parser.parse(sentlist)
trees
treelist = list(trees)

type(treelist[0])
for tree in treelist:
    print (tree)
```

```
['She', 'came', 'two', 'kids', 'yesterday']
(S
  (NP (PRP She))
  (VP (VBD came) (NP (CD two) (NNS kids)) (NP (NN yesterday))))
```

```
# Creating a mini corpus of 4 given sentences to get the probablistic
# frequency of each given word
corpus = """We had a nice party yesterday
        She came to visit me two days ago
        You may go now
        Their kids are not always naive"""
corpus_words = corpus.split()
fdist = FreqDist(corpus_words)
# fdist for each pair is 1 which means that each word occurs with the equal
fdist
```

```
FreqDist({'She': 1,
          'Their': 1,
          'We': 1,
          'You': 1,
          'a': 1,
          'ago': 1,
          'always': 1,
          'are': 1,
          'came': 1,
          'days': 1,
          'go': 1,
          'had': 1,
          'kids': 1,
          'may': 1,
          'me': 1,
          'naive': 1,
          'nice': 1,
          'not': 1,
          'now': 1,
          'party': 1,
          'to': 1,
          'two': 1,
          'visit': 1,
          'yesterday': 1})
```

```python
# Probablistic Grammar
# The probabilities for each non-terminal symbol must add up to 1

prob_grammar = nltk.PCFG.fromstring("""
S -> NP VP[0.9] | VP [0.1]
NP -> PRP [0.5]| DT ADJP NN [0.2]| NN [0.1]| PRP NNS [0.1]| CD NNS[0.1]
VP -> VBD NP NP [0.3]| MD VP [0.2]| VB ADVP[0.1] | VBP RB ADVP ADJP[0.1] | VBD S [0.1]| TO VP[0.1] | VB NP ADVP[0.1]
ADVP -> RB [0.5]| NP RB[0.5]
NNS -> "kids"[0.5] | "days"[0.5]
RB -> "now"[0.25] | "always"[0.25] | "not"[0.25] | "ago"[0.25]
VB -> "go"[0.5] | "visit"[0.5]
MD -> "may"[1.0]
ADJP -> JJ [1.0]
PRP -> "We" [0.2]|"You"[0.2] | "Their"[0.2] | "She"[0.2] | "me"[0.2]
VBD -> "had"[0.5] | "came"[0.5]
CD -> "two"[1.0]
VBP -> "are"[1.0]
DT -> "a"[1.0]
TO -> "to"[1.0]
JJ -> "nice" [0.5]| "naive"[0.5]
NN -> "party" [0.5]| "yesterday"[0.5]
""")
```

```python
viterbi_parser = nltk.ViterbiParser(prob_grammar)
```

```python
for tree in viterbi_parser.parse(['We' ,'had','a', 'nice', 'party', 'yesterday']):
    print (tree)
```

```
(S
  (NP (PRP We))
  (VP
    (VBD had)
    (NP (DT a) (ADJP (JJ nice)) (NN party))
    (NP (NN yesterday)))) (p=3.375e-05)
```

```python
for tree in viterbi_parser.parse(['She' ,'came', 'to', 'visit', 'me' ,'two' ,'days', 'ago']):
    print (tree)
```

```
(S
  (NP (PRP She))
  (VP
    (VBD came)
    (S
      (VP
        (TO to)
        (VP
          (VB visit)
          (NP (PRP me))
          (ADVP (NP (CD two) (NNS days)) (RB ago))))))) (p=1.40625e-09)
```

```python
for tree in viterbi_parser.parse(['You' ,'may', 'go' ,'now']):
    print (tree)
```

```
(S
  (NP (PRP You))
  (VP (MD may) (VP (VB go) (ADVP (RB now))))) (p=0.0001125)
```

```python
for tree in viterbi_parser.parse(['Their', 'kids', 'are', 'not' ,'always', 'naive']):
    print (tree)
```

```
(S
  (NP (PRP Their) (NNS kids))
  (VP
    (VBP are)
    (RB not)
    (ADVP (RB always))
    (ADJP (JJ naive)))) (p=1.40625e-05)
```