Syntax Parsing and SMT

Lab session

In this session we will learn to use NLTK for PCFGs and Statistical Machine Translation

1. Constituent-based Syntactic Parsing

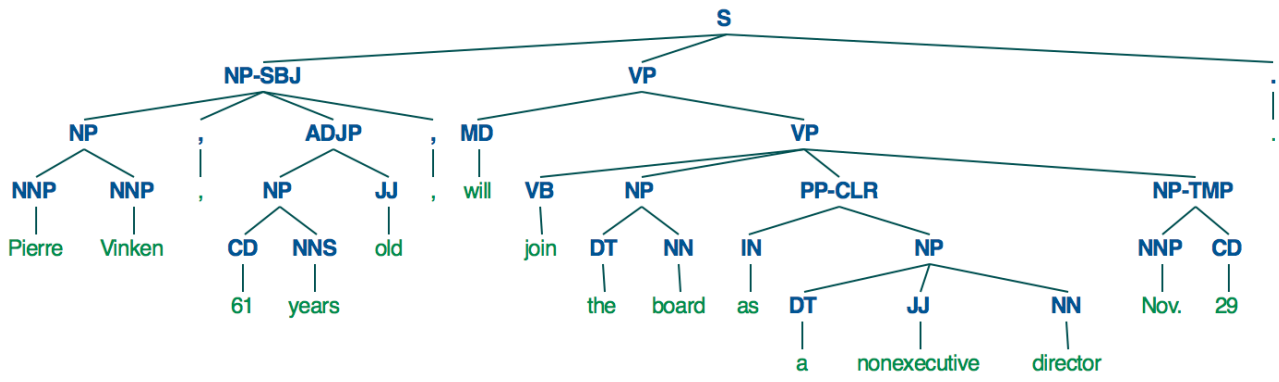
First of all, we will need to import nltk as usual and the nltk corpus of parsed WSJ treebank. This is a small portion of the WSJ treebank, but it will be enough for our lab session.

```
>>> import nltk
>>> from nltk.corpus import treebank
>>> print(treebank.parsed_sents('wsj_0001.mrg')[0])

(S
  (NP-SBJ
    (NP (NNP Pierre) (NNP Vinken))
    (, ,)
    (ADJP (NP (CD 61) (NNS years)) (JJ old))
    (, ,))
  (VP
    (MD will)
    (VP
      (VB join)
      (NP (DT the) (NN board))
      (PP-CLR (IN as) (NP (DT a) (JJ nonexecutive) (NN director)))
      (NP-TMP (NNP Nov.) (CD 29))))
  (. .))
```

You can also use the draw() method to obtain a graphical representation of the parse tree:

```
>>> treebank.parsed_sents('wsj_0001.mrg')[0].draw()
```



This method is part of the Tree NLTK class and has methods to read parse trees in bracketed format:

```
>>> from nltk import Tree
>>> s = '(S (NP (DT the) (NN cat)) (VP (VBD ate) (NP (DT a) (NN cookie))))'
>>> t = Tree.fromstring(s)
>>> t.draw() #or print(t)
```

Some useful methods for the `Tree` class:

Transforming the tree in Chomsky's Normal Form:

```
>>> t.chomsky_normal_form()
```

Obtaining the productions from a parse tree:

```
>>>t.productions()
```

For every production we can get the left and right parts of the productions:

```
>>>for p in t.productions():
    ...:     print (p.lhs()) #left part (nonterminal)
    ...:     print (p.rhs()) #right part of the rule (sequence)
```

1.1 CFG

NLTK includes several classes to encode CFG and PCFG grammars in the nltk.grammar module:

1. Nonterminal
2. Production (for rules)
3. WeightedProduction (for rules in a PCFG)
4. ContextFreeGrammar
5. PCFG

The classes include convenient parsers to convert strings into grammars. The method grammar.check_coverage(ws) determines whether the words that appear in ws can ever be parsed by the grammar.

```
>>>from nltk import nonterminals, Nonterminal, Production

>>># Create some nonterminals
>>>S, NP, VP, PP = nonterminals('S, NP, VP, PP')
>>>N, V, P, Det = nonterminals('N, V, P, Det')
>>># Create a production rule ant print it
>>> print(Production(S, [NP]))
```

Exercise 1. Using the NLTK classes to encode elements of CFG grammars, estimate the probability of observing the rule "NP -> NP PP" and the rule "VP -> VB NP"

Remember that the probability for productions is estimated as:

$$p(X \rightarrow \alpha) = \frac{C(X \rightarrow \alpha)}{C(X)}$$

The number of times the rule used in the corpus

The number of times the nonterminal X appears in the treebank

Suggestion: the treebank can be browsed as follows:

```
>>>productions=[]
>>>for item in treebank.fileids():
>>>  for tree in treebank.parsed_sents(item):
>>>    # perform optional tree transformations, e.g.:
>>>    tree.chomsky_normal_form() # Don't forget to normalize in CNF the parse tree
>>>    productions += tree.productions()
```

1.1 PCFG

NLTK can do the work of inducing a PCFG from an existing Treebank. Once we have stored in *productions* all the productions from our treebank, we can induce the PCFG as follows:

```
>>>from nltk import Nonterminal
>>>S = Nonterminal('S')
>>>grammar = induce_pcfg(S, productions)
```

Exercise 2. Compare the values obtained in exercise 1 to those calculated automatically by the induce_pcfg method. (Hint: *grammar.productions()* to navigate through the productions of the grammar).

1.2 Parsing

There are different types of parsers implemented in NLTK. One that implements the Viterbi CKY n-best parses over a PCFG is available in the *parse.viterbi* module.

```
>>>from nltk.parse import ViterbiParser

>>>s='I saw John with my eyes'
>>>tokens = s.split() #simplified – you can use a tokenizer
>>>parser=ViterbiParser(grammar) #using the grammar induced by the Treebank
>>>parses = parser.parse_all(tokens) #the resulting parse tree
```

Exercise 3. Test the above sentence and draw the resulting parse. Is this parse tree correct? Verify if the sentence 'I saw John with my telescope' is parsable. Suggest a way to deal with unknown words.

2. Statistical Machine Translation: the IBM Model 1

Objective: learn word alignments from a parallel corpus
NLTK includes an implementation of the IBM Model 1 in the package *nltk.align.ibm1* ; we will use this implementation to calculate a lexical translation dictionary.

First of all, load the parallel corpus:

```
>>>from nltk.corpus import comtrans
>>>data = comtrans.aligned_sents("alignment-en-fr.txt") #this will load the alignments
English-French
```

Now we can import the IBM Model 1 implementation:

```
>>>from nltk import IBMModel1
```

To build the model:

```
>>>model=IBMModel1(data, 30) #data is the parallel corpus, 30 is the number of
iterations for the EM algorithm
```

This will take some time; to reduce the waiting time we can work on a smaller dataset:

```
>>>model=IBMModel1(data[:10], 30)
```

At this point you should be able to look at the translation table to find the translation probabilities estimated by the IBM Model1:

```
>>> model.translation_table["the"]["le"]
>>>1.3424467563162238e-05
```

Exercise 4. Use a corpus of different sizes (10, 20, 30, 40 … 100) and max 30 iterations. Study the variation of the value of the probability of finding “le” given “the” (model.translation_table["the"]["le"]). Plot the result.
4.b. Try with a fixed size (50) while varying the number of iterations between 30 and 100 (step 10) and plot the result. What is the best way to improve the result? More iterations or more data?
4.c. Take the model built using 100 pairs of sentences and look at the values for [“dog”][“chien”] and other pairs of words. What can you deduce about the underlying model?