Introduction to Human Language Technologies 8. Parsing

Natural Language Research Group

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Course 2018/19

Outline

Constituency parsing with NLTK

Non-probabilistic parsers:

- ChartParser (default parser is BottomUpLeftCornerChartParser)
- BottomUpChartParser, LeftCornerChartParser
- TopDownChartParser, EarleyChartParser

  ...

Probabilistic parsers:

- InsideChartParser, RandomChartParser, LongestChartParser (they are bottom-up parsers)
- ViterbiParser
- CoreNLPParser (third-party's parser)

  ...

Non-probabilistic parsers: Charts

Example:

In [1]:

```
import nltk
from nltk import CFG, ChartParser

grammar = CFG.fromstring('''
  NP  -> NNS | JJ NNS | NP CC NP
  NNS -> "cats" | "dogs" | "mice" | NNS CC NNS
  JJ  -> "big" | "small"
  CC  -> "and" | "or"
  ''')

sent = ['small', 'cats', 'and', 'mice']

parser = ChartParser(grammar)
parse = parser.parse(sent)
```

In [2]:

```
ts = []
for t in parse:
    ts.append(t)
print('number of trees:', len(ts))
```

```
number of trees: 2
```

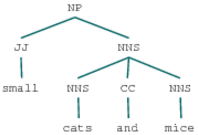
Non-probabilistic parsers: Charts

Output I:

In [3]:
```
print(ts[0])
ts[0]
```

```
(NP (JJ small) (NNS (NNS cats) (CC and) (NNS mice)))
```

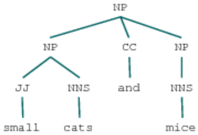
Out[3]:



In [4]:
```
print(ts[1])
ts[1]
```

```
(NP (NP (JJ small) (NNS cats)) (CC and) (NP (NNS mice)))
```

Out[4]:



Non-probabilistic parsers: Charts

Output II:

In [5]:
```
# achieve the list of applied edges
parse = parser.chart_parse(['small', 'cats', 'and', 'mice'])
print("TD num edges = ",parse.num_edges())
```

```
TD num edges =  28
```

In [6]:
```
parse.edges()
```

Out[6]:
```
[[Edge: [0:1] 'small'],
 [Edge: [1:2] 'cats'],
 [Edge: [2:3] 'and'],
 [Edge: [3:4] 'mice'],
 [Edge: [0:1] JJ -> 'small' *],
 [Edge: [0:1] NP -> JJ * NNS],
 [Edge: [1:2] NNS -> 'cats' *],
 [Edge: [1:2] NP -> NNS *],
 [Edge: [1:2] NNS -> NNS * CC NNS],
 [Edge: [0:2] NP -> JJ NNS *],
 [Edge: [0:2] NP -> NP * CC NP],
 [Edge: [1:2] NP -> NP * CC NP],
 [Edge: [2:3] CC -> 'and' *],
 [Edge: [1:3] NNS -> NNS CC * NNS],
 [Edge: [0:3] NP -> NP CC * NP],
 [Edge: [1:3] NP -> NP CC * NP],
 [Edge: [3:4] NNS -> 'mice' *],
 [Edge: [3:4] NP -> NNS *],
 [Edge: [3:4] NNS -> NNS * CC NNS]
```

Non-probabilistic parsers: Charts

Main differences of non-probabilistic chart parsers:

- BottomUpChartParser: bottom-up strategy
- BottomUpLeftCornerChartParser (ChartParser): bottom-up strategy filtering out edges without any word subsumtion (e.g., [0,0]: X→. Y Z)
- LeftCornerChartParser: bottom-up strategy filtering out edges without new word subsumptions (e.g., if we already got [0,1] Y→y. and [1,2] Z→z. then [0,1] A→Y.Z is filtered out whereas [0,2] A→Y Z. is fired)
- TopDownChartParser: top-down strategy
- EarleyChartParser: incremental top-down strategy (more efficient)

In [7]:

```
from nltk import TopDownChartParser

parser = nltk.TopDownChartParser(grammar)
parse = parser.parse(sent)
```

Mandatory Exercise

Statement:

- Consider the following sentence: *Lazy cats play with mice*.
- Expand the grammar of the example related to non-probabilistic chart parsers in order to subsume this new sentence.
- Perform the constituency parsing using a BottomUpChartParser, a BottomUpLeftCornerChartParser and a LeftCornerChartParser.
- For each one of them, provide the resulting tree, the number of edges and the list of explored edges.

- Which parser is the most efficient for parsing the sentence?
- Which edges are filtered out by each parser and why?

Probabilistic parsers: Charts

Example: inside chart parser

In [1]:

```
import nltk
from nltk.parse.pchart import PCFG, InsideChartParser

grammar = PCFG.fromstring('''
  NP  -> NNS [0.5] | JJ NNS [0.3] | NP CC NP [0.2]
  NNS -> "cats" [0.1] | "dogs" [0.2] | "mice" [0.3] | NNS CC NNS [0.4]
  JJ  -> "big" [0.4] | "small" [0.6]
  CC  -> "and" [0.9] | "or" [0.1]
  ''')

sent = ['small', 'cats', 'and', 'mice']

parser = InsideChartParser(grammar)
parse = parser.parse(sent)
```

In [2]:

```
ts = []
for t in parse:
    ts.append(t)
print('number of trees:', len(ts))
```

```
number of trees: 2
```

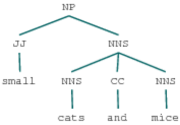
Probabilistic parsers: Charts

Output:

In [3]:
```
print(ts[0])
ts[0]
```

```
(NP (JJ small) (NNS (NNS cats) (CC and) (NNS mice))) (p=0.001944)
```

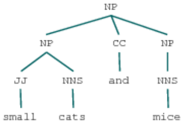
Out[3]:



In [4]:
```
print(ts[1])
ts[1]
```

```
(NP (NP (JJ small) (NNS cats)) (CC and) (NP (NNS mice))) (p=0.000486)
```

Out[4]:



Probabilistic parsers: Charts

Main differences of probabilistic chart parsers:

- They use the bottom-up strategy
- InsideChartParser: select edges in decreasing order of their trees' inside probs. p$\rightarrow$A,B $\Rightarrow$ Prob=P(p)P(A)P(B)
- RandomChartParser: select edges in random order.
- LongestChartParser: select longer edges before shorter ones.

Probabilistic parsers: Viterbi

Example: Probabilistic Viterbi parser

In [1]:

```python
import nltk
from nltk import  PCFG, ViterbiParser

grammar = PCFG.fromstring('''
  NP  -> NNS [0.5] | JJ NNS [0.3] | NP CC NP [0.2]
  NNS -> "cats" [0.1] | "dogs" [0.2] | "mice" [0.3] | NNS CC NNS [0.4]
  JJ  -> "big" [0.4] | "small" [0.6]
  CC  -> "and" [0.9] | "or" [0.1]
  ''')

sent = ['small', 'cats', 'and', 'mice']

parser = ViterbiParser(grammar)
parse = parser.parse(sent)
```

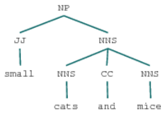
Output:

In [2]:

```python
tree = next(parse)
print(tree)
tree
```

```
(NP (JJ small) (NNS (NNS cats) (CC and) (NNS mice))) (p=0.001944)
```

Out[2]:



Probabilistic parsers: Viterbi

Trace:

In [3]:

```
parser = ViterbiParser(grammar)
parser.trace(3)
parse = parser.parse(sent)
next(parse)
```

```
Inserting tokens into the most likely constituents table...
   Insert: |=...| small
   Insert: |.=..| cats
   Insert: |..=.| and
   Insert: |...=| mice
Finding the most likely constituents spanning 1 text elements...
   Insert: |=...| JJ -> 'small' [0.6]                0.6000000000
   Insert: |.=..| NNS -> 'cats' [0.1]                0.1000000000
   Insert: |.=..| NP -> NNS [0.5]                    0.0500000000
   Insert: |..=.| CC -> 'and' [0.9]                  0.9000000000
   Insert: |...=| NNS -> 'mice' [0.3]                0.3000000000
   Insert: |...=| NP -> NNS [0.5]                    0.1500000000
Finding the most likely constituents spanning 2 text elements...
   Insert: |==..| NP -> JJ NNS [0.3]                 0.0180000000
Finding the most likely constituents spanning 3 text elements...
   Insert: |.===| NP -> NP CC NP [0.2]               0.0013500000
   Insert: |.===| NNS -> NNS CC NNS [0.4]            0.0108000000
   Insert: |.===| NP -> NNS [0.5]                    0.0054000000
  Discard: |.===| NP -> NP CC NP [0.2]               0.0013500000
  Discard: |.===| NP -> NP CC NP [0.2]               0.0013500000
Finding the most likely constituents spanning 4 text elements...
   Insert: |====| NP -> JJ NNS [0.3]                 0.0019440000
  Discard: |====| NP -> NP CC NP [0.2]               0.0004860000
  Discard: |====| NP -> NP CC NP [0.2]               0.0004860000
```

Probabilistic parsers: learn a PCFG

Example: learn a treebank grammar

In [4]:

```
import nltk
from nltk.corpus import treebank
productions = []
S = nltk.Nonterminal('S')
for f in treebank.fileids():
    for tree in treebank.parsed_sents(f):
        productions += tree.productions()
grammar = nltk.induce_pcfg(S, productions)
grammar.productions()[10:15]
```

Out[4]:

```
[NN -> 'evil' [7.59532e-05],
 NN -> 'powder' [7.59532e-05],
 SBAR -> WHNP-167 S [0.000424628],
 VBN -> 'drawn' [0.00140581],
 NN -> 'senior' [0.000151906]]
```

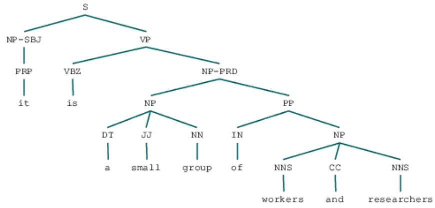
Probabilistic parsers: learn a PCFG

Example: apply the learnt PCFG to Viterbi parser

In [5]:
```
sent = ['it', 'is', 'a', 'small', 'group', 'of', 'workers', 'and', 'researchers']
parser = ViterbiParser(grammar)
parse = parser.parse(sent)
tree = next(parse)
print(tree)
tree
```

```
(S
  (NP-SBJ (PRP it))
  (VP
    (VBZ is)
    (NP-PRD
      (NP (DT a) (JJ small) (NN group))
      (PP
        (IN of)
        (NP (NNS workers) (CC and) (NNS researchers)))))) (p=2.64379e-21)
```

Out[5]:



Probabilistic parsers: CoreNLP parser

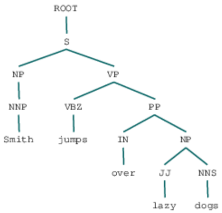
Example:

In [6]:

```
# run in a shell:
# java -mx4g -cp "*" edu.stanford.nlp.pipeline.StanfordCoreNLPServer -port 9000 -timeout 15000
import nltk
from nltk.parse.corenlp import CoreNLPParser
parser = CoreNLPParser(url='http://localhost:9000')
parse = parser.raw_parse('Smith jumps over lazy dogs')
tree = next(parse)
print(tree)
tree
```

```
(ROOT
  (S
    (NP (NNP Smith))
    (VP (VBZ jumps) (PP (IN over) (NP (JJ lazy) (NNS dogs))))))
```

Out[6]:



Outline

Dependency parsing: CoreNLP dependency parser

Example:

In [1]:
```python
import nltk
from nltk.parse.corenlp import CoreNLPDependencyParser

parser = CoreNLPDependencyParser(url='http://localhost:9000')
parse = parser.raw_parse('Smith jumps over the lazy dog')
```

In [2]:
```python
tree = next(parse)
for t in tree.triples():
    print(t)
```

```
(('jumps', 'VBZ'), 'nsubj', ('Smith', 'NNP'))
(('jumps', 'VBZ'), 'nmod', ('dog', 'NN'))
(('dog', 'NN'), 'case', ('over', 'IN'))
(('dog', 'NN'), 'det', ('the', 'DT'))
(('dog', 'NN'), 'amod', ('lazy', 'JJ'))
```

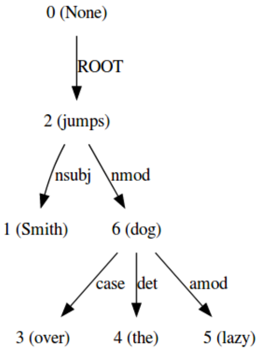
Dependency parsing: CoreNLP dependency parser

Showing the graph:

In [3]:

```
# Graphviz is needed
# sudo pip3 install graphviz
tree
```

Out[3]:



Mandatory exercise

Statement:

1. Read all pairs of sentences of the trial set within the evaluation framework of the project.
2. Compute the Jaccard similarity of each pair using the dependency triples from *CoreNLPDependencyParser*.
3. Show the results. Do you think it could be relevant to use NEs to compute the similarity between two sentences? Justify the answer.