IHLT Laboratory

Jordi Turmo
TALP Research Center
turmo@cs.upc.edu

Session 9

- Constituency parsing with NLTK
  - Non-probabilistic parsers
  - Probabilistic parsers
- Dependency parsing with NLTK
- Exercises

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

Main differences of non-probabilistic chart parsers:

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

Non-probabilistic parsers: Charts

Example: non-probabilistic bottom-up left corner chart parser

```
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"
  ''')

parser = ChartParser(grammar)
parse = parser.parse(['small', 'cats', 'and', 'mice'])
for tree in parse:
    print(tree)
```

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

Non-probabilistic parsers: Charts

Example: non-probabilistic bottom-up left corner chart parser

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

```
TD num edges =  28
[0:1] 'small'
[1:2] 'cats'
[2:3] 'and'
[3:4] 'mice'
[0:1] JJ -> 'small' *
[0:1] NP -> JJ * NNS
[1:2] NNS -> 'cats' *
[1:2] NP -> NNS *
[1:2] NNS -> NNS * CC NNS
[0:2] NP -> JJ NNS *
[0:2] NP -> NP * CC NP
[1:2] NP -> NP * CC NP
[2:3] CC -> 'and' *
[1:3] NNS -> NNS CC * NNS
[0:3] NP -> NP CC * NP
[1:3] NP -> NP CC * NP
[3:4] NNS -> 'mice' *
...
```

Constituency parsing with NLTK

Non-probabilistic parsers:

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

  ...

Probabilistic parsers:

- InsideChartParser, RandomChartParser, LongestChartParser
- ViterbiParser
- CoreNLPParser (third-party's parser)

  ...

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: Charts

Example: inside chart parser

```
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]
  ''')

parser = InsideChartParser(grammar)
parse = parser.parse(['small', 'cats', 'and', 'mice'])
for tree in parse:
    print(tree)
```

```
(NP (JJ small) (NNS (NNS cats) (CC and) (NNS mice))) (p=0.0019)
(NP (NP (JJ small)(NNS cats)) (CC and) (NP (NNS mice))) (p=0.0004)
```

Probabilistic parsers: Viterbi

Example: Probabilistic Viterbi parser

```
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]
  ''')
parser = ViterbiParser(grammar)
parser.trace(3)
parse, = parser.parse(['small', 'cats', 'and', 'mice'])
print("\n",parse)
```

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

Probabilistic parsers: Viterbi

Example: Probabilistic Viterbi parser

```
parser = ViterbiParser(grammar)
parser.trace(3)
parse, = parser.parse(['small', 'cats', 'and', 'mice'])
```

```
Inserting tokens into the most likely constituents table...
   Insert: |=...| small          Insert: |..=.| and
   Insert: |.=..| cats           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
...
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...
...
```

Probabilistic parsers: learn a PCFG

Example: learn a treebank grammar

```
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)
for p in grammar.productions()[10:15]:
    print(p)
```

```
S-2 -> NP-SBJ VP [0.375]
JJ -> 'sophisticated' [0.000857045]
NNP -> 'Smelting' [0.00010627]
VP -> VBD NP PP-CLR PP-CLR [6.8918e-05]
VBP -> 'skip' [0.000757002]
```

Probabilistic parsers: learn a PCFG

Example: apply the learnt PCFG to Viterbi parser

```
parser = ViterbiParser(grammar)
parse,=parser.parse(['it','is','a','small', 'group', 'of',
         'workers', 'and', 'researchers'])
print(parse,"\n")
```

```
(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)
```

Probabilistic parsers: CoreNLP parser

Example:

```
import nltk
from nltk.parse.corenlp import CoreNLPParser
parser = CoreNLPParser(url='http://localhost:9000')

parse, = parser.raw_parse('Smith jumps over lazy dogs')
print(parse)
```

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

Probabilistic parsers: CoreNLP parser

Example:

```
parse, = parser.parse(['Smith','jumps','over','lazy','dogs'])
parse.pretty_print()
```

```
             ROOT
              |
              S
   ___________|____
  |               VP
  |       ________|___
  |      |           PP
  |      |     _______|____
  NP     |    |            NP
  |      |    |        ____|___
 NNP    VBZ   IN      JJ      NNS
  |      |    |       |        |
Smith  jumps over    lazy     dogs
```

Session 9

Dependency parsing with NLTK

- CoreNLPDependencyParser (third-party's parser)
- ProjectiveDependencyParser, NonprojectiveDependencyParser, malt

Dependency parsing: CoreNLP dependency parser

Example:

```
import nltk
from nltk.parse.corenlp import CoreNLPDependencyParser
parser = CoreNLPDependencyParser(url='http://localhost:9000')

parse, = parser.raw_parse('Smith jumps over the lazy dog')

for governor, dep, dependent in parse.triples():
    print(governor, dep, dependent)
```

```
('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')
```

Exercises

A) **Constituency parsing**. Consider the following sentence: "Lazy cats play with mice". Expand the grammar in the example of non-probabilistic chart parsers in order to subsume the 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?

B) **Dependency parsing**. Consider the first three pairs of sentences from the training set of the evaluation framework of the project. Compute the Jaccard similarity of each pair using the dependency triples from CoreNLPDependencyParser.