

# Chunk Parsing

Steven Bird   Ewan Klein   Edward Loper

University of Melbourne, AUSTRALIA

University of Edinburgh, UK

University of Pennsylvania, USA

March 1, 2012

- **chunk parsing:**
  - efficient and robust approach to parsing natural language
  - a popular alternative to the full parsing
- **chunks:**
  - non-overlapping regions of text
  - contain a head word (e.g. noun)
  - adjacent modifiers and function words
- **motivations:**
  - extract information
  - ignore information

- chunk parsing:
  - efficient and robust approach to parsing natural language
  - a popular alternative to the full parsing
- chunks:
  - non-overlapping regions of text
  - contain a head word (e.g. noun)
  - adjacent modifiers and function words
- motivations:
  - extract information
  - ignore information

- chunk parsing:
  - efficient and robust approach to parsing natural language
  - a popular alternative to the full parsing
- chunks:
  - non-overlapping regions of text
  - contain a head word (e.g. noun)
  - adjacent modifiers and function words
- motivations:
  - extract information
  - ignore information

- chunk parsing:
  - efficient and robust approach to parsing natural language
  - a popular alternative to the full parsing
- chunks:
  - non-overlapping regions of text
  - contain a head word (e.g. noun)
  - adjacent modifiers and function words
- motivations:
  - extract information
  - ignore information

- chunk parsing:
  - efficient and robust approach to parsing natural language
  - a popular alternative to the full parsing
- chunks:
  - non-overlapping regions of text
  - contain a head word (e.g. noun)
  - adjacent modifiers and function words
- motivations:
  - extract information
  - ignore information

- chunk parsing:
  - efficient and robust approach to parsing natural language
  - a popular alternative to the full parsing
- chunks:
  - non-overlapping regions of text
  - contain a head word (e.g. noun)
  - adjacent modifiers and function words
- motivations:
  - extract information
  - ignore information

- chunk parsing:
  - efficient and robust approach to parsing natural language
  - a popular alternative to the full parsing
- chunks:
  - non-overlapping regions of text
  - contain a head word (e.g. noun)
  - adjacent modifiers and function words
- motivations:
  - extract information
  - ignore information



- chunk parsing:
  - efficient and robust approach to parsing natural language
  - a popular alternative to the full parsing
- chunks:
  - non-overlapping regions of text
  - contain a head word (e.g. noun)
  - adjacent modifiers and function words
- motivations:
  - extract information
  - ignore information

- chunk parsing:
  - efficient and robust approach to parsing natural language
  - a popular alternative to the full parsing
- chunks:
  - non-overlapping regions of text
  - contain a head word (e.g. noun)
  - adjacent modifiers and function words
- motivations:
  - extract information
  - ignore information

- chunk parsing:
  - efficient and robust approach to parsing natural language
  - a popular alternative to the full parsing
- chunks:
  - non-overlapping regions of text
  - contain a head word (e.g. noun)
  - adjacent modifiers and function words
- motivations:
  - extract information
  - ignore information

# Extracting Information: Coreference Annotation

```
<COREF ID="2" MIN="woman">
  This woman
</COREF>
receives three hundred dollars a
month under
<COREF ID="5">
  General Relief
</COREF>
, plus
<COREF ID="16"
  MIN="four hundred dollars">
  four hundred dollars a month in
<COREF ID="17"
  MIN="benefits" REF="16">
  A.F.D.C. benefits
</COREF>
</COREF>
for
<COREF ID="9" MIN="son">
  <COREF ID="3" REF="2">
    her
  </COREF>
  son
</COREF>
, who is
```

```
<COREF ID="10" MIN="citizen" REF="9">
  a U.S. citizen
</COREF>.
<COREF ID="4" REF="2">
  She
</COREF>
's among
<COREF ID="18" MIN="aliens">
  an estimated five hundred illegal
  aliens on
  <COREF ID="6" REF="5">
    General Relief
  </COREF>
  out of
  <COREF ID="11" MIN="population">
    <COREF ID="13" MIN="state">
      the state
    </COREF>
    's total illegal immigrant
    population of
    <COREF ID="12" REF="11">
      one hundred thousand
    </COREF>
  </COREF>
</COREF>
```

```
.
<COREF ID="7" REF="5">
  General Relief
</COREF>
is for needy families and unemployable
adults who don't qualify for other public
assistance. Welfare Department spokeswoman
Michael Reganburg says
<COREF ID="15" MIN="state" REF="13">
  the state
</COREF>
will save about one million dollars a year if
<COREF ID="20" MIN="aliens" REF="18">
  illegal aliens
</COREF>
are denied
<COREF ID="8" REF="5">
  General Relief
</COREF>
.
```

# Extracting Information: Message Understanding

- 0. Message: ID
- 3. Incident: Location
- 4. Incident: Type
- 6. Incident: Instrument ID
- 9. Perp: Individual ID
- 12. Phys Tgt: ID
- 18. Hum Tgt: Name
- 23. Hum Tgt: Effect of Incident

TST2-MUC4-0048

El Salvador: San Salvador (City)

Bombing

"bomb"

"urban guerrillas"

"vehicle"

"Garcia Alvarado"

Death: "Garcia Alvarado"

Garcia Alvarado, 56, was killed when a bomb placed by urban guerrillas on his vehicle exploded as it came to a halt at an intersection in downtown San Salvador.

# Ignoring Information: Lexical Acquisition

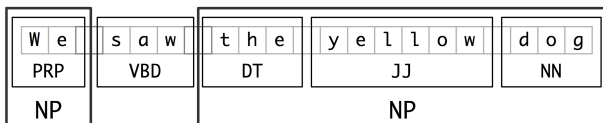
- studying syntactic patterns, e.g. finding verbs in a corpus, displaying possible arguments
- e.g. *gave*, in 100 files of the Penn Treebank corpus
- replaced internal details of each noun phrase with NP

```
gave NP
gave up NP in NP
gave NP up
gave NP help
gave NP to NP
```

- use in lexical acquisition, grammar development

# Analogy with Tokenization and Tagging

- fundamental in NLP: segmentation and labelling
- tokenization and tagging



- other similarities: skipping material; finite-state; application specific

# Chunking vs Parsing

## 1 Parsing

```
[  
  [ G.K. Chesterton ],  
  [  
    [ author ] of  
    [  
      [ The Man ] who was  
      [ Thursday ]  
    ]  
  ]  
]
```

## 2 Chunking:

```
[ G.K. Chesterton ],  
[ author ] of  
[ The Man ] who was  
[ Thursday ]
```



# Perfection is unattainable

1. Prepositional phrase:

```
[  
  [ I ]  
  [ turned ]  
  [ off the spectroroute ]  
]
```

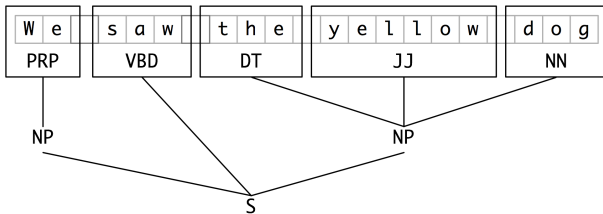
2. Verb-particle construction:

```
[  
  [ I ]  
  [ turned off ]  
  [ the spectroroute ]  
]
```

# Tag Representation

W	e	s	a	w	t	h	e	y	e	l	l	o	w	d	o	g
PRP		VBD			DT			JJ						NN		
B-NP		0			B-NP			I-NP						I-NP		

# Tree Representation



# Chunk Structures

```
(S: (NP: 'I')
     'saw'
     (NP: 'the' 'big' 'dog')
     'on'
     (NP: 'the' 'hill'))
```

- Demonstration: reading chunk structures from Treebank and CoNLL-2000 corpora

# Chunk Parsing

- regular expressions over part-of-speech tags:  
`parse.RegexpChunk`
- *Tag string*: a string consisting of tags delimited with angle-brackets, e.g. `<DT><JJ><NN><VBD><DT><NN>`
- *Tag pattern*: regular expression over tag strings
  - `<DT><JJ>?<NN>`
  - `<NN|JJ>+`
  - `<NN.*>`
- chunk a sequence of words matching a tag pattern:  
`parse.ChunkRule`

```
>>> grammar = "NP: {<DT|NN>+} # Chunk sequences
```

# A Simple NP Chunker

```
import nltk
grammar = r"""
NP:
    {<DT>?<JJ>*<NN>}      # chunk determiners, adjectives and nouns
    {<NNP>+}                # chunk sequences of proper nouns
"""
cp = nltk.RegexpParser(grammar)
tagged_tokens = [("the", "DT"), ("little", "JJ"), ("cat", "NN"),
                 ("sat", "VBD"), ("on", "IN"), ("the", "DT"), ("mat", "NN")]

>>> print cp.parse(tagged_tokens)
(S (NP the/DT little/JJ cat/NN) sat/VBD on/IN (NP the/DT mat/NN))
```

# Developing Chunkers

```
cp1 = nltk.RegexpParser(r"""
NP: {<DT><JJ><NN>}      # Chunk det+adj+noun
    {<DT|NN>+}          # Chunk sequences of NN and DT
""")
cp2 = nltk.RegexpParser(r"""
NP: {<DT|NN>+}          # Chunk sequences of NN and DT
    {<DT><JJ><NN>}      # Chunk det+adj+noun
""")

>>> print cp1.parse(tagged_tokens, trace=1)
# Input:
<DT> <JJ> <NN> <VBD> <IN> <DT> <NN>
# Chunk det+adj+noun:
{<DT> <JJ> <NN>} <VBD> <IN> <DT> <NN>
# Chunk sequences of NN and DT:
{<DT> <JJ> <NN>} <VBD> <IN> {<DT> <NN>}
```

- tracing; rule ordering; overlapping contexts

# More Chunking Rules: Chinking

- *chink*: sequence of stopwords
- *chinking*: process of removing tokens from a chunk

	<b>Entire chunk</b>	<b>Middle of a chunk</b>	<b>End of a chunk</b>
<b>Input:</b>	[a/DT big/JJ cat/NN]	[a/DT big/JJ cat/NN]	[a/DT big/JJ cat/NN]
<b>Operation:</b>	Chink a/DT big/JJ cat/NN	Chink big/JJ	Chink cat/NN
<b>Output:</b>	a/DT big/JJ cat/NN	[a/DT] big/JJ [cat/NN]	[a/DT big/JJ] cat/NN



# Chinking Example

```
>>> grammar = r"""
... NP:
...     {<.*>+}           # Chunk everything
...     }<VBD|IN>+{       # Chink sequences of VBD and IN
...     """
>>> cp = nltk.RegexpParser(grammar)
>>> print cp.parse(tagged_tokens)
(S:
  (NP: ('the', 'DT') ('little', 'JJ') ('cat', 'NN'))
  ('sat', 'VBD')
  ('on', 'IN')
  (NP: ('the', 'DT') ('mat', 'NN')))
>>> print nltk.chunk.accuracy(cp, nltk.corpus.conll2000.chunked_se
0.581041433607
```

# Evaluating Chunk Parsers

- **Process:**

- 1 take some already chunked text
- 2 strip off the chunks
- 3 rechunk it
- 4 compare the result with the original chunked text

- `ChunkScore.score()`

- *precision*: what fraction of the returned chunks were correct?
- *recall*: what fraction of correct chunks were returned?

# Evaluating Chunk Parsers

- Process:
  - 1 take some already chunked text
  - 2 strip off the chunks
  - 3 rechunk it
  - 4 compare the result with the original chunked text
- `ChunkScore.score()`
  - *precision*: what fraction of the returned chunks were correct?
  - *recall*: what fraction of correct chunks were returned?

# Evaluating Chunk Parsers

- Process:
  - 1 take some already chunked text
  - 2 strip off the chunks
  - 3 rechunk it
  - 4 compare the result with the original chunked text
- `ChunkScore.score()`
  - *precision*: what fraction of the returned chunks were correct?
  - *recall*: what fraction of correct chunks were returned?

# Evaluating Chunk Parsers

- Process:
  - 1 take some already chunked text
  - 2 strip off the chunks
  - 3 rechunk it
  - 4 compare the result with the original chunked text
- `ChunkScore.score()`
  - *precision*: what fraction of the returned chunks were correct?
  - *recall*: what fraction of correct chunks were returned?

# Evaluating Chunk Parsers

- Process:
  - 1 take some already chunked text
  - 2 strip off the chunks
  - 3 rechunk it
  - 4 compare the result with the original chunked text
- `ChunkScore.score()`
  - *precision*: what fraction of the returned chunks were correct?
  - *recall*: what fraction of correct chunks were returned?

# Evaluating Chunk Parsers

- Process:
  - ① take some already chunked text
  - ② strip off the chunks
  - ③ rechunk it
  - ④ compare the result with the original chunked text
- `ChunkScore.score()`
  - *precision*: what fraction of the returned chunks were correct?
  - *recall*: what fraction of correct chunks were returned?

# Evaluating Chunk Parsers

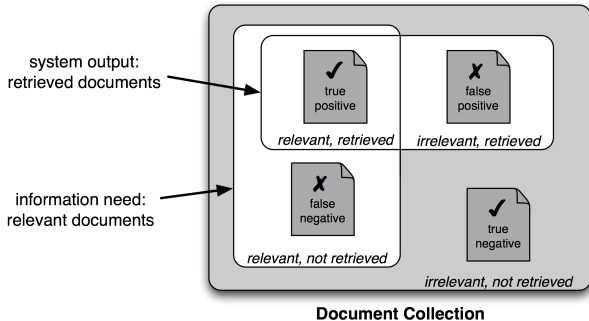
- Process:
  - ① take some already chunked text
  - ② strip off the chunks
  - ③ rechunk it
  - ④ compare the result with the original chunked text
- `ChunkScore.score()`
  - *precision*: what fraction of the returned chunks were correct?
  - *recall*: what fraction of correct chunks were returned?



# Evaluating Chunk Parsers

- Process:
  - ① take some already chunked text
  - ② strip off the chunks
  - ③ rechunk it
  - ④ compare the result with the original chunked text
- `ChunkScore.score()`
  - *precision*: what fraction of the returned chunks were correct?
  - *recall*: what fraction of correct chunks were returned?

# Precision and Recall



# Chunker evaluation in NLTK

```
>>> chunkscore = nltk.chunk.ChunkScore()
>>> for chunk_struct in nltk.corpus.treebank.chunked_sents()[ :10]:
...     test_sent = cp.parse(chunk_struct.leaves())
...     chunkscore.score(chunk_struct, test_sent)
>>> print chunkscore
ChunkParse score:
Precision:  48.6%
Recall:     34.0%
F-Measure: 40.0%
```

## Error Analysis: Missed Chunks

```
>>> from random import randint
>>> missed = chunkscore.missed()
>>> for i in range(15):
...     print missed[randint(0,len(missed)-1)]
(('A', 'DT'), ('Lorillard', 'NNP'), ('spokewoman', 'NN'))
(('it', 'PRP'),)
(('symptoms', 'NNS'),)
(('even', 'RB'), ('brief', 'JJ'), ('exposures', 'NNS'))
(('its', 'PRP$'), ('Micronite', 'NN'), ('cigarette', 'NN'), ('filters'
(('30', 'CD'), ('years', 'NNS'))
(('workers', 'NNS'),)
(('preliminary', 'JJ'), ('findings', 'NNS'))
(('Medicine', 'NNP'),)
(('cancer', 'NN'), ('deaths', 'NNS'))
(('Consolidated', 'NNP'), ('Gold', 'NNP'), ('Fields', 'NNP'), ('PLC',
(('Medicine', 'NNP'),)
(('its', 'PRP$'), ('Micronite', 'NN'), ('cigarette', 'NN'), ('filters'
(('a', 'DT'), ('forum', 'NN'))
(('researchers', 'NNS'),)
```

## Error Analysis: Incorrect Chunks

```
>>> incorrect = chunkscore.incorrect()
>>> for i in range(15):
...     print incorrect[randint(0, len(incorrect)-1)]
(('New', 'JJ'), ('York-based', 'JJ'))
(('Micronite', 'NN'), ('cigarette', 'NN'))
(('a', 'DT'), ('forum', 'NN'), ('likely', 'JJ'))
(('later', 'JJ'),)
(('later', 'JJ'),)
(('brief', 'JJ'),)
(('preliminary', 'JJ'),)
(('New', 'JJ'), ('York-based', 'JJ'))
(('resilient', 'JJ'),)
(('group', 'NN'),)
(('cancer', 'NN'),)
(('the', 'DT'),)
(('cancer', 'NN'),)
(('Micronite', 'NN'), ('cigarette', 'NN'))
(('A', 'DT'),)
```

# Development Methodology

- approaches
  - different rules and combinations
  - hand-crafted vs automatic
- focus on diagnosis:
  - manual
  - utility functions
  - error analysis
  - evaluation

# Development Methodology

- approaches
  - different rules and combinations
    - hand-crafted vs automatic
- focus on diagnosis:
  - manual
  - utility functions
  - error analysis
  - evaluation

# Development Methodology

- approaches
  - different rules and combinations
  - hand-crafted vs automatic
- focus on diagnosis:
  - manual
  - utility functions
  - error analysis
  - evaluation



# Development Methodology

- approaches
  - different rules and combinations
  - hand-crafted vs automatic
- focus on diagnosis:
  - manual
  - utility functions
  - error analysis
  - evaluation

# Development Methodology

- approaches
  - different rules and combinations
  - hand-crafted vs automatic
- focus on diagnosis:
  - manual
  - utility functions
  - error analysis
  - evaluation

# Development Methodology

- approaches
  - different rules and combinations
  - hand-crafted vs automatic
- focus on diagnosis:
  - manual
  - utility functions
  - error analysis
  - evaluation

# Development Methodology

- approaches
  - different rules and combinations
  - hand-crafted vs automatic
- focus on diagnosis:
  - manual
  - utility functions
  - error analysis
  - evaluation

# Development Methodology

- approaches
  - different rules and combinations
  - hand-crafted vs automatic
- focus on diagnosis:
  - manual
  - utility functions
  - error analysis
  - evaluation