Partial Parsing
CSCI-GA.2590 – Lecture 5A

Ralph Grishman
Road Map

- Goal: information extraction

- Paths
  - POS tags $\rightarrow$ partial parses $\rightarrow$ semantic grammar
  - POS tags $\rightarrow$ full parses $\rightarrow$ semantic interp. rules
Road Map

• Goal: information extraction

• Paths
  – POS tags → partial parses → semantic grammar
  – POS tags → full parses → semantic interp. rules
Partial Parses (Chunks)

• Strategy:
  – identify as much local syntactic structure as we can simply and deterministically
    • general (not task specific)
    • result are termed chunks or partial parses
  – build rest of structure using semantic patterns
    • task specific
Partial Parses (Chunks)

I fed the hungry young man in the house with a spoon

definitely part of the NP headed by ‘man’

attachment uncertain:
need semantics or context
Partial Parses (Chunks)

We will build and use two kinds of chunks:

• noun groups
  – head + left modifiers of an NP

• verb groups
  – head verb + auxiliaries and modals
    [“eats”, “will eat”, “can eat”, …]
    (+ embedded adverbs)

(we will use the terms ‘noun groups’ and ‘noun chunks’ interchangeably)
**Chunk patterns**

- Jet provides a regular expression language which can match specific words or parts of speech
- We will write our first chunker using these patterns
Chunk patterns: noun groups

ng := det-pos? [constit cat=adj] * [constit cat=n] | proper-noun | [constit cat=pro];

det-pos :=[constit cat=det] | [constit cat=det]? [constit cat=n number=singular] "'s";

proper-noun :=([token case=cap] | [undefinedCap])+;
Chunk patterns (verb groups)

\[ \text{vg := [constit cat=tv] | [constit cat=w] vg-inf | tv-vbe vg-ving; } \]
\[ \text{vg-inf := [constit cat=v] | "be" vg-ving; } \]
\[ \text{vg-ven := [constit cat=ven] | "been" vg-ving; } \]
\[ \text{vg-ving := [constit cat=ving]; } \]
\[ \text{tv-vbe := "is" | "are" | "was" | "were"; } \]
Assembling the pipeline

tokenizer $\rightarrow$ POS-tagger $\rightarrow$ chunker
Tipster Architecture

Want a uniform data structure for passing information from one stage of the pipeline to the next

In the Tipster architecture, basic structure is the *document* with a set of *annotations*, each consisting of

- a type
- a span
- zero or more features

Each *annotator* (tokenizer, tagger, chunker) reads current annotations and adds one or more new types of annotations
Tipster Architecture

• Offset annotation means original document is never modified
  – benefit in displaying provenance of extracted information
• Document + annotations widely used
  – JET
  – GATE (gate.ac.uk – Univ. of Sheffield)
  – UIMA (uima.apache.org)
• but not universally: NLTK Python toolkit
Adding Annotations

My cat is sleeping

<table>
<thead>
<tr>
<th>type</th>
<th>start</th>
<th>end</th>
<th>features</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Adding Annotations: tokenizer

My cat is sleeping

<table>
<thead>
<tr>
<th>type</th>
<th>start</th>
<th>end</th>
<th>features</th>
</tr>
</thead>
<tbody>
<tr>
<td>token</td>
<td>0</td>
<td>3</td>
<td>case=forcedCap</td>
</tr>
<tr>
<td>token</td>
<td>3</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>token</td>
<td>7</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>token</td>
<td>10</td>
<td>19</td>
<td></td>
</tr>
</tbody>
</table>
# Adding Annotations: POS tagger

**My cat is sleeping**

<table>
<thead>
<tr>
<th>type</th>
<th>start</th>
<th>end</th>
<th>features</th>
</tr>
</thead>
<tbody>
<tr>
<td>token</td>
<td>0</td>
<td>3</td>
<td>case=forcedCap</td>
</tr>
<tr>
<td>token</td>
<td>3</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>token</td>
<td>7</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>token</td>
<td>10</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>constit</td>
<td>0</td>
<td>3</td>
<td>cat=det</td>
</tr>
<tr>
<td>constit</td>
<td>3</td>
<td>7</td>
<td>cat=n number=singular</td>
</tr>
<tr>
<td>constit</td>
<td>7</td>
<td>10</td>
<td>cat=tv number=singular</td>
</tr>
<tr>
<td>constit</td>
<td>10</td>
<td>19</td>
<td>cat=ving</td>
</tr>
</tbody>
</table>
### Adding Annotations: chunker

**My cat is sleeping**

<table>
<thead>
<tr>
<th>type</th>
<th>start</th>
<th>end</th>
<th>features</th>
</tr>
</thead>
<tbody>
<tr>
<td>token</td>
<td>0</td>
<td>3</td>
<td>case=forcedCap</td>
</tr>
<tr>
<td>token</td>
<td>3</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>token</td>
<td>7</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>token</td>
<td>10</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>constit</td>
<td>0</td>
<td>3</td>
<td>cat=det</td>
</tr>
<tr>
<td>constit</td>
<td>3</td>
<td>7</td>
<td>cat=n number=singular</td>
</tr>
<tr>
<td>constit</td>
<td>7</td>
<td>10</td>
<td>cat=tv number=singular</td>
</tr>
<tr>
<td>constit</td>
<td>10</td>
<td>19</td>
<td>cat=ving</td>
</tr>
<tr>
<td>ng</td>
<td>0</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>vg</td>
<td>7</td>
<td>19</td>
<td></td>
</tr>
</tbody>
</table>
Jet pattern language (1)

• Matching an annotation:
  
  \[
  \text{[type feature = value ...]}
  \]
  
  – must be able to *unify* features in pattern and annotation
  – can have nested features: feature = [f1 = v1  f2 = v2]

• Matching a string

  “word”

• Optionality and repetition
  
  \[
  X ? \quad \text{(optionality)}
  \]
  
  \[
  X * \quad \text{(zero or more X’s)}
  \]
  
  \[
  X + \quad \text{(one or more X’s)}
  \]
Jet pattern language (2)

• Binding a variable to a pattern element:
  
  \[ \text{constit cat=n} : \text{Head} \]

• Adding an annotation
  
  \text{when pattern add \{ng\};}

• Writing output
  
  \text{when pattern write \"head =\", Head;}
Setting Up The Pipeline

# JET properties file to run chunk patterns
Jet.dataPath = data
Tags.fileName = pos_hmm.txt
Pattern.fileNam0 = chunkPatterns.txt
Pattern.trace = on

processSentence = tagJet, pat(chunks)

pipeline stages
(tokenization implicit for interactive use)
Processing a Document

- # JET properties file
- # apply chunkPatterns to article.txt
- Jet.dataPath       = data
- Tags.fileName      = pos_hmm.txt
- Pattern.fileName1  = chunkPatterns.txt
- Pattern.trace      = on

- JetTest.fileName1  = article.txt
- processSentence    = tokenize, tagJet, pat(chunks)
- WriteSGML.type     = ngroup

split doc into sentences, then runs processSentence on each
Viewing Annotations

• can be activated through Jet menu:

  tools : process documents and view ...

• provides color-coded display of annotations
Corpus-Trained Chunkers

• We know two ways of building a sequence classifier which can assign a tag to each token in a sequence of tokens: HMMs and TBL

• Can we use a sequence classifier to do chunking? Assign N and O tags:

\[\text{I gave a book to the new student.} \]
\[\text{N O N N O N N N N N} \]

• sequence of one or more consecutive Ns = a noun group
A Problem

• How about
  
  I gave the new student a book

  N O   N   N   N   N   N   N   N

• 2 noun groups or 3?
BIO Tags

• A solution: 3 tags
  – B: first word of a noun group
  – I: second or subsequent word of a noun group
  – O: not part of a noun group

I gave the new student a book
B 0 B I I B I

• To tag noun and verb groups, need 5 tags:
  B-N, I-N, B-V, I-V, and O