ACS Syntax and Semantics of Natural Language

Lecture 8: Statistical Parsing Models for CCG

Stephen Clark
Natural Language and Information Processing (NLIP) Group
sc609@cam.ac.uk
• The parsing models that have been developed for CCG are not CCG-specific, but general models for structural prediction problems.

• Generative models can be applied to CCG derivations, generating the tree top-down using a similar process to the Collins model (Hockemaier, 2003).

• Feature-based models have been successfully applied to CCG, including log-linear models and the generalized perceptron.
\[ d_{\text{max}}_{\Phi,\Lambda}(S) = \arg \max_d \text{Score}(\Phi(d, S), \Lambda) \]

- \(d_{\text{max}}\) is the highest scoring derivation for sentence \(S\), according to a feature representation function \(\Phi\) and a model (set of weights) \(\Lambda\)
- The algorithm which implements the \(\arg \max\) is the decoder - let’s worry about that later, but bearing in mind that how we choose to define \(\Phi\) is likely to impact on the efficiency of the decoder
- \(\Phi(d, S)\) takes a derivation \(d\) for sentence \(S\) and returns the features for that derivation - how we choose to break \(d\) into features will be crucial for accuracy
- \(\Lambda\) is the set of weights corresponding to the complete set of features - \(\Lambda\) will be learned from annotated data (CCGbank)
- The Score function relates \(\Phi\) and \(\Lambda\) and assigns a real-valued score to a derivation - we’ll be using log-linear and linear formulations
The Feature Representation

• Features will be defined *locally* in terms of rule instantiations
  – by *rule instantiation* I just mean the subtree consisting of a parent and children (one or two children in CCG’s case)

• We could extend the feature range, and this may increase the discriminatory power of the model, but the efficiency of the decoding and estimation algorithms will decrease

• Mathematically, features are functions from derivations onto integers (i.e. counts)
  – so extensions of the binary indicator functions we saw for tagging
The Features

<table>
<thead>
<tr>
<th>Feature type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>LexCat + Word</td>
<td>((S/S)/NP + \text{Before})</td>
</tr>
<tr>
<td>LexCat + POS</td>
<td>((S/S)/NP + \text{IN})</td>
</tr>
<tr>
<td>RootCat</td>
<td>(S[dcl])</td>
</tr>
<tr>
<td>RootCat + Word</td>
<td>(S[dcl] + \text{was})</td>
</tr>
<tr>
<td>RootCat + POS</td>
<td>(S[dcl] + \text{VBD})</td>
</tr>
<tr>
<td>Rule</td>
<td>(S[dcl]\to NP S[dcl]\backslash NP)</td>
</tr>
<tr>
<td>Rule + Word</td>
<td>(S[dcl]\to NP S[dcl]\backslash NP + \text{bought})</td>
</tr>
<tr>
<td>Rule + POS</td>
<td>(S[dcl]\to NP S[dcl]\backslash NP + \text{VBD})</td>
</tr>
<tr>
<td>Word-Word</td>
<td>((\text{company}, S[dcl]\to NP S[dcl]\backslash NP, \text{bought}))</td>
</tr>
<tr>
<td>Word-POS</td>
<td>((\text{company}, S[dcl]\to NP S[dcl]\backslash NP, \text{VBD}))</td>
</tr>
<tr>
<td>POS-Word</td>
<td>((\text{NN}, S[dcl]\to NP S[dcl]\backslash NP, \text{bought}))</td>
</tr>
<tr>
<td>POS-POS</td>
<td>((\text{NN}, S[dcl]\to NP S[dcl]\backslash NP, \text{VBD}))</td>
</tr>
<tr>
<td>Word + Distance(words)</td>
<td>((\text{bought}, S[dcl]\to NP S[dcl]\backslash NP) + &gt; 2)</td>
</tr>
<tr>
<td>Word + Distance(punct)</td>
<td>((\text{bought}, S[dcl]\to NP S[dcl]\backslash NP) + 2)</td>
</tr>
<tr>
<td>Word + Distance(verbs)</td>
<td>((\text{bought}, S[dcl]\to NP S[dcl]\backslash NP) + 0)</td>
</tr>
<tr>
<td>POS + Distance(words)</td>
<td>((\text{VBD}, S[dcl]\to NP S[dcl]\backslash NP) + &gt; 2)</td>
</tr>
<tr>
<td>POS + Distance(punct)</td>
<td>((\text{VBD}, S[dcl]\to NP S[dcl]\backslash NP) + 2)</td>
</tr>
<tr>
<td>POS + Distance(verbs)</td>
<td>((\text{VBD}, S[dcl]\to NP S[dcl]\backslash NP) + 0)</td>
</tr>
</tbody>
</table>

- This basic feature set results in a few hundred thousand features and leads to good parsing performance
Log-Linear Model

\[ P(d|S) = \exp(\sum_i \lambda_i f_i(d, S))/Z(S) \]

where \( d \) is a derivation for sentence \( S \) and \( Z(S) \) is a normalisation constant

- Note that this is a *global* model, assigning a probability to the complete parse tree
- Can also be thought of as a maximum entropy model, as for the local maxent models we used for tagging
- To estimate the model parameters (weights, \( \lambda \)'s), we need to calculate feature expectations, as before
- Calculating feature expectations requires a sum over *all* derivations for each sentence in the training data
  - sum can be performed efficiently using a dynamic programming algorithm (the inside-outside algorithm) over a packed chart
• We can build a packed chart for CCG as we did for a PCFG in the Intro to NLP module

• Categories with the same CCG type, same span in the sentence, and same headword, are grouped together into an equivalence class
  – definition of equivalence depends on the feature range

• The Viterbi algorithm runs recursively top-down over the chart, choosing the highest scoring category in each equivalence class

• The Viterbi algorithm is optimal and operates in polynomial time (with respect to sentence length)
Score\((d, S) = \sum \lambda_i f_i(d, S)\)

- An alternative to the log-linear model which is trained using a simple parameter update which aims to maximise accuracy on the training data
- Performs surprisingly well given simple nature of the update
- Also has some theoretical guarantees
- See Collins (2002) for application to tagging
Perceptron Training

\[ \text{Score}(d, s) = \sum_i \lambda_i \cdot f_i(d, s) = \bar{\lambda} \cdot \bar{f}(d) \]

**Inputs:** training examples \((s_j, d_j)\)

**Initialisation:** set \(\bar{\lambda} = 0\)

**Algorithm:**
- for \(t = 1..T, j = 1..N\)
  - calculate \(d_{\text{max}} = \arg \max_d \bar{\lambda} \cdot \bar{f}(d)\)
  - if \(d_{\text{max}} \neq d_j\)
    - \(\bar{\lambda} = \bar{\lambda} + \bar{f}(d_j) - \bar{f}(d_{\text{max}})\)

**Outputs:** \(\bar{\lambda}\)

[see JHU tutorial slides for animated description of online update]
References

