Parsing with Lexicalised PCFGs

ACS Introduction to NLP

Stephen Clark

Chart Parsing plus Search

$$T_{\text{best}} = \arg\max_{T} P(T, S)$$

- The number of possible parses increases exponentially with sentence length
- For a typical newspaper sentence there are far too many possible parses to enumerate (using a treebank grammar)
- Two key ideas allow the $\arg\max_T$ to be performed efficiently:
  - dynamic programming leads to an n^5 algorithm (still not efficient enough)
  - heuristic search enables efficient parsing in practice

Chart Parsing with Lexicalised PCFGs

- Use a standard bottom-up chart parser, based on CKY
- The key data structure is the *chart*
  - *chart[start, end, label]* is the set of all edges in the chart spanning words *start* to *end* inclusive, with non-terminal label *label*
- We'll look at the parser for Collins Model 1
  - Model 2 just requires some extensions to deal with the modelling of subcategorisation frames

[picture of chart]

The edge *Datatype*

`label`	`non-terminal label`
`headlabel`	`non-terminal label of head child of the edge`
`headword`	`the head word`
`headtag`	`part of speech tag of the head word`
`start`	`index of first word in edge's span`
`end`	`index of last word in edge's span`
`stop`	`TRUE if the edge has received its stop probabilities`
`prob`	`log probability of the edge`
`children`	`list of the children of the edge (left to right)`

Dynamic Programming and Packed Charts

- A key idea for efficient parsing is the following: if two edges are *equivalent* for the purpose of further parsing, then only one of them needs to be used to form future edges (as long as the other is retained - if we want to represent all parses)

- This notion of equivalence leads to a *packed chart* which can efficiently represent an exponential number of parses

[example]

Dynamic Programming and Viterbi

- If we're only looking for the highest scoring parse, no need to keep all edges in an equivalence class
- If two edges are equivalent for the purposes of future parsing, *and in terms of the probability model*, then the edge with the lowest score can be discarded
- This form of dynamic programming is the Viterbi algorithm

[example]

The Equivalence Test

```
// assume e1 and e2 have the same start and end indices

boolean edges_eqivalent(edge e1, edge e2)
{
  if(e1.label     != e2.label      OR
     e1.headlabel != e2.headlabel  OR
     e1.headword  != e2.headword   OR
     e1.headtag   != e2.headtag    OR
     e1.stop      != e2.stop)
    return FALSE;
  else
    return TRUE;
}
```

Adding Edges to the Chart

```
void add_edge(edge e, int start, int end)
{
  foreach edge x in chart[start, end, e.label]
    if(edges_equivalent(e,x))
    {
      if(e.prob > x.prob)
        replace x with e
      return;
    }

  add e to chart[start, end, e.label]
}
```

Combining Edges

```
// e1 is adjacent and to the left of e2
// e2 is a modifier of e1

void join_2_edges_follow(edge e1, edge e2)
{
  edge e3;

  e3.label      = e1.label;
  e3.headlabel  = e1.headlabel;
  e3.headword   = e1.headword;
  e3.headtag    = e1.headtag;
  e3.start      = e1.start;
  e3.end        = e2.end;
  e3.stop       = FALSE;
  e3.children   = e1.children ++ e2;
  e3.prob = e1.prob + e2.prob + log P_r(e1,e2);
  // P_r calculates the additional probability when the modifier is to the right

  add_edge(e3,e1.start,e2.end);
}
```

Combining Edges II

```
// e1 is adjacent and to the left of e2
// e1 is a modifier of e2

void join_2_edges_precede(edge e1, edge e2)
{
  edge e3;

  e3.label      = e2.label;
  e3.headlabel  = e2.headlabel;
  e3.headword   = e2.headword;
  e3.headtag    = e2.headtag;
  e3.start      = e1.start;
  e3.end        = e2.end;
  e3.stop       = FALSE;
  e3.children   = e1.children ++ e2;
  e3.prob = e1.prob + e2.prob + log P_l(e1,e2);
  // P_l calculates the additional probability when the modifier is to the left

  add_edge(e3,e1.start,e2.end);
}
```

Initialising the Chart

```
void initialise()
{
  edge e;
  for i = 1 to n  // n is number of words in input sentence
  {
    if(word_i is an ‘‘unknown’’ word)
      set X = {POS tag from tagger for word_i}
    else
      set X = {set of all tags seen for word_i in training data}

    foreach POS tag T in X
    {
      e.label     = T;    e.headword  = word_i; e.headtag   = T;
      e.stop      = TRUE; e.start     = i;      e.end       = i+1;
      e.prob      = 0;

      add_edge(e,i,i+1);
    }
  }
}
```

All Edge Combinations

```
void complete(int start, int end)
{
  for split = start to end-1
  {
    foreach edge e1 in chart[start,split] such that e1.stop == FALSE
      foreach edge e2 in chart[split+1,end] such that e2.stop == TRUE
        join_2_edges_follow(e1,e2);

    foreach edge e1 in chart[start,split] such that e1.stop == TRUE
      foreach edge e2 in chart[split+1,end] such that e2.stop == FALSE
        join_2_edges_precede(e1,e2);
  }

}
```

The Final Parsing Algorithm

```
edge parse()
{
  initialise();

  // n is the number of words in the sentence
  for span = 2 to n
    for start = 1 to n-span+1
    {
      end = start + span - 1;
      complete(start, end);
    }

  // assume TOP is the start symbol
  X = edge in chart[1,n,TOP] with highest probability;

  return X;
}
```

Parsing Complexity

- Calls to join_2_edges_[precede|follow] take $O(1)$ time
- These calls are buried within 5 loops:

Complexity	Loop
$O(n)$	for span = 2 to n
$O(n)$	for start = 1 to n-span+1
$O(n)$	for split = start to end-1
$O(n)$	foreach edge e1 in chart[start,split] s.t. e1.stop == FALSE
$O(n)$	foreach edge e2 in chart[split+1,end] s.t. e2.stop == TRUE

- Parsing algorithm is essentially an n^5 algorithm
- I've ignored some constants along the way (related to size of tag set etc)

Heuristic Beam Search

- n^5 (plus some non-neglible constants) is inefficient for practical parsing
- We need to prune low-probability constituents in the chart
- This is a "lossy" strategy since we may throw away the correct parse
  - so there are now two sources of possible error in the parser: model error and search error
  - Viterbi finds the optimal solution so does not lead to search errors
- But in practice we can obtain great increases in efficiency with very small losses in accuracy

Figures of Merit

- What score should we use for a partial parse (constituent)?
- Obvious score to use is `prob` - the (log) conditional probability of the constituent: P(subtree|label,head-word,head-tag)
- This doesn't work too well
  - problem is that the conditional probability does not account for the *prior* probability of seeing the particular (label,head-word,head-tag) triple

[example]

Prior for a Figure of Merit

$$\begin{aligned} \text{Score(subtree)} &= P(\text{subtree}|\text{label,head-tag,head-word}) \\ &\times P_{\text{prior}}(\text{label,head-tag,head-word}) \end{aligned}$$

- One way to calculate the prior (Collins):

$$\begin{aligned} P_{\text{prior}}(\text{label,head-tag,head-word}) &= P(\text{head-tag,head-word}) \\ &\times P(\text{label}|\text{head-tag,head-word}) \end{aligned}$$

- Probabilities estimated using relative frequency from counts in the corpus; second probability smoothed with interpolation

The Beam

- Let bestprob(start,end) be the highest score for any constituent spanning start..end
- Discard all constituents with span start..end and with log prob $< \alpha$ best-prob(start,end)
- α is the beam width; typical value is $\frac{1}{10000}$

Pros and Cons of the Generative Model

- Pros:
  - Conceptually easy to understand; well understood techniques
  - Estimation is easy (max. likelihood = relative frequencies)
  - Produces good results
- Cons:
  - Models S when the sentence is given
  - Independence assumptions required
  - Locality restrictions on features required for efficient estimation and decoding
  - Guarantees on estimation (e.g. soundness) only apply with unlimited training data
  - Choosing the order for the chain rule, and independence assumptions plus smoothing, something of a "black art"

References

- Appendix B, D and E of Collins' thesis
- Caraballo and Charniak (1998), New Figures of Merit for best-first probabilistic chart parsing. Computational Linguistics, 24(2), pages 275-298

all available from the web; Collins thesis from Collins' web page