Natural Language Processing

Lecture 16, 17, 18.1—3/7/2015 – 3/12/2015
Martha Palmer

Today

- Review CKY
- Earley
- Partial parsing
  - Finite-state methods
  - Chunking
    - Sequence labeling methods

---

CKY Algorithm

function CKY-Parse(words, grammar) returns table

for \( j \leftarrow \text{from} 1 \text{ to } \text{LENGTH(words)} \text{ do} \)  
   Filling the bottom cell
   \( \text{table}[j - 1, j] \leftarrow \{ A \mid A \rightarrow \text{words}[j] \in \text{grammar} \} \)

for \( i \leftarrow \text{from} j - 2 \text{ downto} 0 \text{ do} \)  
   Filling row \( i \) in column \( j \)
   \( \text{table}[i, j] \leftarrow \text{table}[i, j] \cup \)  
   Looping over the possible split locations between \( i \) and \( j \).

- \( \{ A \mid A \rightarrow BC \in \text{grammar}, B \in \text{table}[i, k], C \in \text{table}[k, j] \} \)

---

Example

[Diagram showing the CKY algorithm process with a matrix and parsed structures.]
Note

* An alternative is to fill a diagonal at a time.
  * That still satisfies our requirement that the component parts of each constituent/cell will already be available when it is filled in.

CKY Notes

* Since it’s bottom up, CKY populates the table with a lot of phantom constituents.
  * Segments that by themselves are constituents but cannot really occur in the context in which they are being suggested.
  * To avoid this we can switch to a top-down control strategy
  * Or we can add some kind of filtering that blocks constituents where they can not happen in a final analysis.

Earley Parsing

* Allows arbitrary CFGs
* Top-down control
* Fills a table in a single sweep over the input
  * Table is length N+1; N is number of words
  * Table entries represent
    * Completed constituents and their locations
    * In-progress constituents
    * Predicted constituents
The table-entries are called states and are represented with dotted-rules.

- **S**: The start symbol, representing an incomplete state. A VP is predicted.
- **NP**: The noun phrase, currently in progress. The Det goes from 1 to 2.
- **VP**: The verb phrase, complete and found. A VP has been found.

### Earley

As with most dynamic programming approaches, the answer is found by looking in the table in the right place. In this case, there should be an S state in the final column that spans from 0 to N and is complete. That is,

- **S**: The start symbol, representing a complete state.
- **NP**: The noun phrase, currently in progress. The Det goes from 1 to 2.
- **VP**: The verb phrase, complete and found. A VP has been found starting at 0 and ending at 3.

So sweep through the table from 0 to N...

- New predicted states are created by starting top-down from S.
- New incomplete states are created by advancing existing states as new constituents are discovered.
- New complete states are created in the same way.
Upfront Earley

- More specifically...
  1. **Predict** all the states you can upfront
  2. Read a word
     1. Extend states based on matches
     2. Generate new predictions
     3. Go to step 2
  3. When you’re out of words, look at the chart to see if you have a winner

Earley Code

```
procedure PREDICTOR(A → α • B β, [i, j])
  for each (B → γ) in GRAMMAR-RULES-FOR(B, grammar) do
    ENQUEUE(B → γ, [j, j], chart[j])
  end

procedure SCANNER(A → α • B β, [i, j])
  if B ⊂ PARTS-OF-SPEECH(word[j]) then
    ENQUEUE(B → word[j], [j, j+i]), chart[j+1])
  end

procedure COMPLETER(B → γ •, [j, k])
  for each (A → α • B β, [i, j]) in chart[j] do
    ENQUEUE((A → α B β, [i, k]), chart[k])
  end
```

Example

- Book that flight
- We should find... an S from 0 to 3 that is a completed state...
Note that given a grammar, these entries are the same for all inputs; they can be pre-loaded.

For such a simple example, there seems to be a lot of useless stuff in there.

Why?

- It's predicting things that aren't consistent with the input
- That's the flipside to the CKY problem.
Details

- As with CKY that isn’t a parser until we add the backpointers so that each state knows where it came from.

Ambiguity

- No...
  - Both CKY and Earley will result in multiple S structures for the \([0,N]\) table entry.
  - They both efficiently store the sub-parts that are shared between multiple parses.
  - And they obviously avoid re-deriving those sub-parts.
  - But neither can tell us which one is right.

Back to Ambiguity

- Did we solve it?

Ambiguity

- In most cases, humans don’t notice incidental ambiguity (lexical or syntactic). It is resolved on the fly and never noticed.
- We’ll try to model that with probabilities.
Full Syntactic Parsing

- Probably necessary for deep semantic analysis of texts (as we'll see in a couple of weeks).
- Probably not practical for many applications (given typical resources)
  - $O(n^3)$ for straight parsing
  - $O(n^5)$ for probabilistic versions
  - Too slow for applications that need to process texts in real time (search engines)
  - Or that need to deal with large volumes of new material over short periods of time

Two Alternatives

- Partial parsing
  - Approximate phrase-structure parsing with finite-state and statistical approaches
- Dependency parsing
  - Change the underlying grammar formalism
- Both of these approaches give up something (syntactic structure) in return for more robust and efficient parsing

Partial Parsing

- For many applications you don’t really need a full-blown syntactic parse. You just need a good idea of where the base syntactic units are.
  - Often referred to as chunks.
- For example, if you’re interested in locating all the people, places and organizations in an English text it can be useful to know where all the NPs are
  - Because that’s where you’ll find the people, places and things

Examples

- [NP The morning flight] [PP from] [NP Denver] [VP has arrived.]
- [NP a flight] [PP from] [NP Indianapolis] [PP to] [NP Houston] [PP on] [NP TWA]
  - [NP The morning flight] from [NP Denver] has arrived.
- The first two are examples of full partial parsing or chunking. All of the elements in the text are part of a chunk. And the chunks are non-overlapping.
- Note how the second example has no hierarchical structure.
- The last example illustrates base-NP chunking. Ignore anything that isn’t in the kind of chunk you’re looking for.
Rule-Based Partial Parsing

- Restrict the form of rules to exclude recursion
- Group and order the rules so that the RHS of the rules can refer to non-terminals introduced in earlier transducers, but not later ones.
- Combine the rules in a group in the same way we did with the rules for spelling changes.
- Combine the groups into a cascade...
- Then compose, determinize and minimize the whole thing (optional).

Typical Architecture

- Phase 1: Part of speech tags
- Phase 2: Base syntactic phrases
- Phase 3: Larger verb and noun groups
- Phase 4: Sentential level rules

Partial Parsing

\[ NP \rightarrow (Dei) \text{Noun}^* \text{Noun} \]
\[ NP \rightarrow \text{Proper-Noun} \]
\[ VP \rightarrow \text{Verb} \]
\[ VP \rightarrow \text{Aux Verb} \]

Cascaded Transducers

- No direct or indirect recursion allowed in these rules.
- That is, you can’t directly or indirectly reference the LHS of the rule on the RHS.

The morning flight from Denver has arrived
Partial Parsing

- This cascaded approach can be used to find the sequence of flat chunks you’re interested in.
- Or it can be used to approximate the kind of hierarchical trees you get from full parsing with a CFG.

The Other Way

- An alternative approach is to use statistical machine learning methods to do partial parsing
  - Analogous to the same situation with part-of-speech tagging
    - Rules vs. HMMs

Statistical Sequence Labeling

- As with POS tagging, we can use rules to do partial parsing or we can train systems to do it for us. To do that we need training data and a way to view the problem as a classification problem
  - Training data
    - Hand tag a bunch of data (as with POS tagging)
    - Or even better, extract partial parse bracketing information from a treebank.

Encoding

- With the right encoding you can turn any labeled bracketing task into a tagging task. And then proceed exactly as we did with POS Tagging.
- We’ll use what’s called IOB labeling to do this
  - I -> Inside
  - O -> Outside
  - B -> Begin
IOB encoding

The morning flight from Denver has arrived.
B_NP L_NP L_NP O B_NP O O

- This example shows the encoding for just base-NPs. There are 3 tags in this scheme.

The morning flight from Denver has arrived
B_NP L_NP L_NP B_PP B_NP B_VP L_VP

- This example shows full coverage. In this scheme there are 2*N+1 kinds of tags. Where N is the number of constituents in your set.

Different encodings

- Voting between multiple data representations for text chunking
  Hong Shen, Anoop Sarkar, In Canadian AAI, 2005
  Added S for Singleton tag, increase from 94.22 to 95.23 F1 score on base NPs.

Methods

- Argmax P(Tags|Words)
  - HMMs
  - Discriminative Sequence Classification
    - Using any kind of standard ML-based classifier.

HMM Tagging

- Same as we did with POS tagging
  - Argmax P(T|W) = P(W|T)P(T)
  - The tags are the hidden states
  - Works ok, but has one significant shortcoming
    - The typical kinds of things that we might think would be useful in this task aren't easily squeezed into the HMM model
  - We'd like to be able to make arbitrary features available for the statistical inference being made.
  - For that we'll turn to classifiers created using classical machine learning techniques
Supervised Classification

- Training a system to take an object represented as a set of features and apply a label to that object.

- Methods typically include
  - Naïve Bayes
  - Decision Trees
  - Logistic regression (maximum entropy)
  - Support Vector Machines
  - ...

From Classification to Sequence Processing

- Applying this to tagging...
  - The object to be tagged is a word in the sequence
  - The features are
    - features of the word,
    - features of its immediate neighbors,
    - and features derived from the entire context
  - Sequential tagging means sweeping a classifier across the input assigning tags to words as you proceed.

Typical Features

- Typical setup involves
  - A small sliding window around the object being tagged
  - Features extracted from the window
    - Current word token
    - Previous/next N word tokens
    - Current word POS
    - Previous/next POS
    - Previous N chunk labels
    - Capitalization information
    - ...

Statistical Sequence Labeling

Classifier

The morning flight from Denver has arrived.

Corresponding feature representation
Evaluation

- Suppose you employ this IOB scheme. What’s the best way to measure performance.
- Probably not the per-tag accuracy we used for POS tagging.
  - Why?
    • It’s not measuring what we care about
    • We need a metric that looks at the chunks not the tags

Example

- Suppose we were looking for PP chunks for some reason.
- If the system simply said O all the time it would do pretty well on a per-label basis since most words reside outside any PP.

Precision/Recall/F

- Precision:
  - The fraction of chunks the system returned that were right
    • “Right” means the boundaries and the label are correct given some labeled test set.
- Recall:
  - The fraction of the chunks that system got from those that it should have gotten.
- F: Simple harmonic mean of those two numbers.