Today 3/11

• Review
  • Partial Parsing & Chunking
    • Sequence classification
  • Statistical Parsing

Back to Sequences

\[ T = \arg \max_{T} P(T | W) \]
\[ = \arg \max_{T} P(W | T, P(T) \prod_{i} P(\text{word} | \text{name}) \prod_{i} P(\text{name} | \text{name}_{-1}) \]

\[ \hat{T} = \arg \max_{\hat{T}} P(T | W) \]
\[ = \arg \max_{\hat{T}} \prod_{i} P(\text{name} | \text{word}, \text{name}_{-1}) \]

• HMMs

• MEMMs

And whatever other features you choose to use!
The value for a cell is found by examining all the cells in the previous column and multiplying by the posterior for the current column (which incorporates the transition as a factor, along with any other features you like).

\[ v_j(j) = \max_{1 \leq i \leq N-1} v_{i-1}(j) P(s_j|s_i, a_i); \quad 1 < j < N, 1 < t < T \]

HMMs vs. MEMMs

\[ P(T|W) = \prod P(t_i|t_{i-1}, w_i) \]
HMMs vs. MEMMs

\[
P(T|W) = \prod P(t_i|t_{i-1}, w_i, f_i)
\]

Dynamic Programming Parsing Approaches

- Earley
  - Top-down, no filtering, no restriction on grammar form
- CYK
  - Bottom-up, no filtering, grammars restricted to Chomsky-Normal Form (CNF)
- Details are not important...
  - Bottom-up vs. top-down
  - With or without filters
  - With restrictions on grammar form or not

Back to Ambiguity
Disambiguation

• Of course, to get the joke we need both parses.
• But in general we’ll assume that there’s one right parse.
• To get that we need knowledge: world knowledge, knowledge of the writer, the context, etc…
• Or maybe not..

Disambiguation

• Instead let’s make some assumptions and see how well we do…

Example
Probabilistic CFGs

- The probabilistic model
  - Assigning probabilities to parse trees
  - Getting the probabilities for the model
- Parsing with probabilities
  - Slight modification to dynamic programming approach
  - Task is to find the max probability tree for an input

Probability Model

- Attach probabilities to grammar rules
- The expansions for a given non-terminal sum to 1
  - VP -> Verb .55
  - VP -> Verb NP .40
  - VP -> Verb NP NP .05
  - Read this as P(Specific rule | LHS)

Probability Model (1)

- A derivation (tree) consists of the bag of grammar rules that are in the tree
- The probability of a tree is just the product of the probabilities of the rules in the derivation.
  \[ P(T,S) = \prod_{node \in T} P(\text{rule}(\mu)) \]
Probability Model (1.1)

- The probability of a word sequence (sentence) is the probability of its tree in the unambiguous case.
- It’s the sum of the probabilities of the trees in the ambiguous case.
- Since we can use the probability of the tree(s) as a proxy for the probability of the sentence…
- PCFGs give us an alternative to N-Gram models as a kind of language model.

Example

```
S  
  
  VP
  
  Verb
  
  Book
  
  Det
  
  Nominal
  
  the
  
  Nominal
  
  flight
  
  Noun
  
  dinner

S  
  
  VP
  
  Verb
  
  Book
  
  Det
  
  Nominal
  
  the
  
  Nominal
  
  dinner
  
  Noun
  
  flight
```

Rule Probabilities

<table>
<thead>
<tr>
<th>Rules</th>
<th>P</th>
<th>Rules</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>S → VP</td>
<td>0.05</td>
<td>S → VP</td>
<td>0.05</td>
</tr>
<tr>
<td>VP → Verb NP</td>
<td>0.20</td>
<td>VP → Verb NP</td>
<td>0.10</td>
</tr>
<tr>
<td>NP → Det Nominal</td>
<td>0.20</td>
<td>NP → Det Nominal</td>
<td>0.20</td>
</tr>
<tr>
<td>Nominal → Nominal</td>
<td>0.75</td>
<td>Nominal → Nominal</td>
<td>0.75</td>
</tr>
<tr>
<td>Nominal → Noun</td>
<td>0.75</td>
<td>Nominal → Noun</td>
<td>0.75</td>
</tr>
<tr>
<td>Verb → book</td>
<td>0.30</td>
<td>Verb → book</td>
<td>0.30</td>
</tr>
<tr>
<td>Det → the</td>
<td>0.60</td>
<td>Det → the</td>
<td>0.60</td>
</tr>
<tr>
<td>Noun → dinner</td>
<td>0.10</td>
<td>Noun → dinner</td>
<td>0.10</td>
</tr>
<tr>
<td>Noun → flights</td>
<td>0.40</td>
<td>Noun → flights</td>
<td>0.40</td>
</tr>
</tbody>
</table>

2.2 * 10^{-6}  
6.1 * 10^{-7}
Getting the Probabilities

- From an annotated database (a treebank)
  - So for example, to get the probability for a particular VP rule just count all the times the rule is used and divide by the number of VPs overall.

\[ P(\alpha \rightarrow \beta|\alpha) = \frac{\text{Count}(\alpha \rightarrow \beta)}{\sum \text{Count}(\alpha \rightarrow \gamma)} = \frac{\text{Count}(\alpha \rightarrow \beta)}{\text{Count}(\alpha)} \]

Smoothing

- Using this method do we need to worry about smoothing these probabilities?

Inside/Outside

- If we don’t have a treebank, but we do have a grammar can we get reasonable probabilities?
- Yes. Use a prob parser to parse a large corpus and then get the counts as above.
- But
  - In the unambiguous case we’re fine
  - In ambiguous cases, weight the counts of the rules by the probabilities of the trees they occur in.
Inside/Outside

• But…
• Where do those probabilities come from?
• Make them up. And then re-estimate them.
• This sounds a lot like…

Assumptions

• We’re assuming that there is a grammar to be used to parse with.
• We’re assuming the existence of a large robust dictionary with parts of speech
• We’re assuming the ability to parse (i.e. a parser)
• Given all that… we can parse probabilistically

Typical Approach

• Use CKY as the backbone of the algorithm
• Assign probabilities to constituents as they are completed and placed in the table
• Use the max probability for each constituent going up
What does that last bullet mean?

- Say we’re talking about a final part of a parse
  - S→NPjVPj

  The probability of this S is...
  \[ P(S→NPjVPj)\times P(NPj)\times P(\text{VP}) \]

  The green stuff is already known if we’re using some kind of sensible DP approach.

Max

- I said the P(NP) is known.
- What if there are multiple NPs for the span of text in question (0 to i)?
- Take the max (where?)

CKY

Where does the max go?

function CKY-FARMLH[\text{word, grammar}] returns table
for i ← 1 to \text{LEN(GOAL)} do
  \text{node[i]} ← \{ \text{if } i \rightarrow \text{word[i]} \scope{grammar} \}
for i ← 1 to \text{LEN(GOAL)} do
  for j ← 1 to \text{LEN(GOAL)} do
    \text{node[i]} ← \{ \text{if } i \rightarrow j, \text{node[i]}, \text{node[j]} \scope{grammar} \}
}

Where does the max go?
**Probs CKY**

```plaintext
function PROBABLISTIC_CKY(words, grammar) returns most probable parse
    and its probability
for j from 1 to LENGTH(words) do
    for all [ A | A → words[j] in grammar ]
        table[j][j] = P(A → words[j])
for j from 1 to LENGTH(words) do
    for i from j-1 to 1 do
        for all [ A | A → B C in grammar ]
            if table[i][j] > 0 and table[i][k] > 0 then
                table[i][j] = table[i][k] * table[k][j]
        return BUILD_TREE(words, 1, LENGTH(words), 1, table, 1, LENGTH(words), 1)
```

**Break**

- Next assignment details have been posted. See the course web page. It’s due March 20.
- Quiz is a week from today.

**Problems with PCFGs**

- The probability model we’re using is just based on the rules in the derivation...
  - Doesn’t use the words in any real way
  - Doesn’t take into account where in the derivation a rule is used
  - Doesn’t really work (shhh)
    - Most probable parse isn’t usually the right one (the one in the treebank test set).
Solution 1

- Add lexical dependencies to the scheme...
  - Integrate the preferences of particular words into the probabilities in the derivation
  - I.e. Condition the rule probabilities on the actual words

Heads

- To do that we’re going to make use of the notion of the head of a phrase
  - The head of an NP is its noun
  - The head of a VP is its verb
  - The head of a PP is its preposition
  (It’s really more complicated than that but this will do.)

Example (right)
Example (wrong)

How?

• We used to have
  • VP -> V NP PP \( P(\text{rule|VP}) \)
    • That’s the count of this rule divided by the number of VPs in a treebank
  • Now we have
    • VP(dumped) -> V(dumped) NP(sacks)PP(in)
    • \( P(r|VP \land \text{dumped} \land \text{sacks}) \) is the count of this rule divided by the number of VPs in a treebank
    • \( P(r|VP \land \text{dumped} \land \text{sacks} \land \text{in}) \) is the count of this rule divided by the number of VPs in a treebank
    • Not likely to have significant counts in any treebank

Declare Independence

• When stuck, exploit independence and collect the statistics you can…
• We’ll focus on capturing two things
  • Verb subcategorization
    • Particular verbs have affinities for particular VP rules
  • Objects affinities for their predicates (mostly their mothers and grandmothers)
    • Some objects fit better with some predicates than others
Subcategorization

- Condition particular VP rules on their head... so
  \[ r: \text{VP} \rightarrow \text{V NP PP} \]
  \[ P(r|\text{VP}) \]
  Becomes
  \[ P(r|\text{VP} \land \text{dumped}) \]

  What's the count?
  How many times was this rule used with dump, divided
  by the number of VPs that dump appears in total

Preferences

- Subcat captures the affinity between VP heads (verbs) and the VP rules they go
  with.
- What about the affinity between VP heads and the heads of the other daughters of
  the VP
- Back to our examples...

Example (right)
Preferences

• The issue here is the attachment of the PP. So the affinities we care about are the ones between dumped and into vs. sacks and into.
• So count the places where dumped is the head of a constituent that has a PP daughter with into as its head and normalize
• Vs. the situation where sacks is a constituent with into as the head of a PP daughter.

Preferences (2)

• Consider the VPs
  • Ate spaghetti with gusto
  • Ate spaghetti with marinara
• The affinity of gusto for eat is much larger than its affinity for spaghetti
• On the other hand, the affinity of marinara for spaghetti is much higher than its affinity for ate
Preferences (2)

- Note the relationship here is more distant and doesn't involve a headword since gusto and marinara aren't the heads of the PPs.

Next Time

- Finish up 14
  - Rule re-writing approaches
  - Evaluation