Part-of-Speech Tagging & Parsing

Chloé Kiddon

(slides adapted and/or stolen outright from Andrew McCallum, Christopher Manning, and Julia Hockenmaier)

Part-of-speech tagging

• Often want to know what part of speech (POS) or word class (noun, verb, ...) should be assigned to words in a piece of text

• **Part-of-speech tagging** assigns POS labels to words

```
JJ  JJ  NNS  VBP  RB
Colorless green ideas sleep furiously.
```
Why do we care?

• Parsing (come to later)

• Speech synthesis
  ▫ Insult or inSULT, overFLOW or OVERflow, REad or reAD

• Information extraction: entities, relations
  ▫ Romeo loves Juliet vs. lost loves found again

• Machine translation

Penn Treebank Tagset

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC</td>
<td>Coordinating conjunction</td>
</tr>
<tr>
<td>CD</td>
<td>Cardinal number</td>
</tr>
<tr>
<td>DT</td>
<td>Determiner</td>
</tr>
<tr>
<td>EX</td>
<td>Existential there</td>
</tr>
<tr>
<td>FW</td>
<td>Foreign word</td>
</tr>
<tr>
<td>IN</td>
<td>Preposition or subordinating</td>
</tr>
<tr>
<td>JJ</td>
<td>Adjective</td>
</tr>
<tr>
<td>JJR</td>
<td>Adjective, comparative</td>
</tr>
<tr>
<td>JJS</td>
<td>Adjective, superlative</td>
</tr>
<tr>
<td>LS</td>
<td>List item marker</td>
</tr>
<tr>
<td>MD</td>
<td>Modal</td>
</tr>
<tr>
<td>NN</td>
<td>Noun, singular or mass</td>
</tr>
<tr>
<td>NNS</td>
<td>Noun, plural</td>
</tr>
<tr>
<td>NP</td>
<td>Proper noun, singular</td>
</tr>
<tr>
<td>NPS</td>
<td>Proper noun, plural</td>
</tr>
<tr>
<td>PDT</td>
<td>Predeterminer</td>
</tr>
<tr>
<td>POS</td>
<td>Possessive ending</td>
</tr>
<tr>
<td>PP</td>
<td>Personal pronoun</td>
</tr>
<tr>
<td>PPS</td>
<td>Personal pronoun</td>
</tr>
</tbody>
</table>

Ambiguity

Buffalo buffalo buffalo.

How many words are ambiguous?

<table>
<thead>
<tr>
<th>Tag Set</th>
<th>Unambiguous (1 tag)</th>
<th>Ambiguous (2–7 tags)</th>
</tr>
</thead>
<tbody>
<tr>
<td>87-tag Original Brown</td>
<td>44,619</td>
<td>5,490</td>
</tr>
<tr>
<td>45-tag Treebank Brown</td>
<td>38,887</td>
<td>8,844</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Details</th>
<th>Count (87-tag)</th>
<th>Count (45-tag)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 tags</td>
<td>4,967</td>
<td>6,731</td>
</tr>
<tr>
<td>3 tags</td>
<td>411</td>
<td>1,621</td>
</tr>
<tr>
<td>4 tags</td>
<td>91</td>
<td>357</td>
</tr>
<tr>
<td>5 tags</td>
<td>17</td>
<td>90</td>
</tr>
<tr>
<td>6 tags</td>
<td>2 (well, best)</td>
<td>32</td>
</tr>
<tr>
<td>7 tags</td>
<td>2 (still, down)</td>
<td></td>
</tr>
<tr>
<td>8 tags</td>
<td>4 (well, set, round, open, fit, down)</td>
<td></td>
</tr>
<tr>
<td>9 tags</td>
<td>3 (chat, more, in)</td>
<td></td>
</tr>
</tbody>
</table>
Naïve approach!

- Pick the most common tag for the word

<table>
<thead>
<tr>
<th>Word</th>
<th>POS listings in Brown</th>
</tr>
</thead>
<tbody>
<tr>
<td>heat</td>
<td>noun/89 verb/5</td>
</tr>
<tr>
<td>oil</td>
<td>noun/87</td>
</tr>
<tr>
<td>in</td>
<td>prep/20731 noun/1 adv/462</td>
</tr>
<tr>
<td>a</td>
<td>det/22943 noun/50 noun-proper/30</td>
</tr>
<tr>
<td>large</td>
<td>adj/354 noun/2 adv/5</td>
</tr>
<tr>
<td>pot</td>
<td>noun/27</td>
</tr>
</tbody>
</table>

- 91% success rate!

We have more information

- We are not just tagging words, we are tagging sequences of words

For a sequence of words $W$:

$$W = w_1w_2w_3...w_n$$

We are looking for a sequence of tags $T$:

$$T = t_1t_2t_3...t_n$$

where $P(T|W)$ is maximized

Bayes’ Rule

- To find $P(T|W)$, use Bayes’ Rule:

$$P(T|W) \propto P(W|T) \times P(T)$$

- We can maximize $P(T|W)$ by maximizing $P(W|T)P(T)$

In an ideal world...

- Find all instances of a sequence in the dataset and pick the most common sequence of tags

  - Count("heat oil in a large pot") = 0 ????
  - Uhh...

  - Spare data problem
  - Most sequences will never occur, or will occur too few times for good predictions
Finding P(T)

- Generally,
  \[ P(t_1, t_2, \ldots, t_n) = P(t_1) \times P(t_2 | t_1) \times \cdots \times P(t_n | t_{n-1}) \]

- Usually not feasible to accurately estimate more than tag bigrams (possibly trigrams)

\[ P(t_1, t_2, \ldots, t_n) = \prod_j P(t_j | t_1, \ldots, t_{j-1}) \]

Markov assumption

- Assume that the probability of a tag only depends on the tag that came directly before it
  \[ P(t_i | t_1, t_2, \ldots, t_{i-1}) = P(t_i | t_{i-1}) \]

- Then,
  \[ P(t_1, t_2, \ldots, t_n) = \prod_i P(t_i | t_{i-1}) \times \prod_i P(w_i | t_i) \]

- Only need to count tag bigrams.

Putting it all together

- We can similarly assume
  \[ P(w_i | t_1, \ldots, t_n) = P(w_i | t_i) \]

- So:
  \[ P(w_1, \ldots, w_n | t_1, t_2, \ldots, t_n) = P(w_1 | t_1) \times P(w_2 | t_2) \times \cdots \times P(w_n | t_n) \]

- And the final equation becomes:
  \[ P(W | T) \times P(T) = P(w_1 | t_1) \times P(w_2 | t_2) \times \cdots \times P(w_n | t_n) \times P(t_1) \times P(t_2 | t_1) \times \cdots \times P(t_n | t_{n-1}) \]

Process as an HMM

- Start in an initial state \( t_0 \) with probability \( \pi(t_0) \)
- Move from state \( t_i \) to \( t_j \) with transition probability \( a(t_j | t_i) \)
- In state \( t_i \), emit symbol \( w_k \) with emission probability \( b(w_k | t_i) \)
Three Questions for HMMs

1. **Evaluation** - Given a sequence of words \( W = w_1, w_2, w_3, ..., w_n \) and an HMM model \( \Theta \), what is \( P(W \mid \Theta) \)?

2. **Decoding** - Given a sequence of words \( W \) and an HMM model \( \Theta \), find the most probable parse \( T = t_1, t_2, t_3, ... , t_n \).

3. **Learning** - Given a tagged (or untagged) dataset, find the HMM \( \Theta \) that maximizes the data.

---

**Tagging**

- Need to find the most likely tag sequence given a sequence of words
- maximizes \( P(W \mid T) \cdot P(T) \) and thus \( P(T \mid W) \)
- Use Viterbi!

---

**Trellis**

- **Evaluation Task:** \( P(w_1, w_2, ..., w_i) \) given in \( t_j \) at time \( i \)
- **Decoding Task:** \( \max P(w_i, w_{i+1}, ..., w_n) \) given in \( t_j \) at time \( i \)
Evaluation Task: $P(w_1, w_2, \ldots, w_i)$ given in $t_j$ at time $i$

Decoding Task: $\max \log P(w_i, w_{i-1}, \ldots, w_1)$ given in $t_j$ at time $i$

Tagging initialization

$= \log P(w_i | t_j) + \log P(t_j)$

Tagging recursive step

$= \arg \max_k \log P(t_j | t_k) + \text{trellis}[w_1][t_k]$
Learning a POS-tagging HMM

- Estimate the parameters in the model using counts

\[ P(t_i | t_{i-1}) \rightarrow \]

\[ P(w_j | t_i) \rightarrow \]

- With smoothing, this model can get 95-96% correct tagging

Problem with supervised learning

- Requires a large hand-labeled corpus
  - Doesn’t scale to new languages
  - Expensive to produce
  - Doesn’t scale to new domains

- Instead, apply unsupervised learning with Expectation Maximization (EM)
  - Expectation step: calculate probability of all sequences using set of parameters
  - Maximization step: re-estimate parameters using results from E-step
Lots of other techniques!

- Trigram models (more common)
- Text normalization
- Error-based transformation learning ("Brill learning")
  - Rule-based system
    - Calculate initial states: proper noun detection, tagged corpus
    - Acquire transformation rules
    - Change VB to NN when prev word was adjective
    - The long race finally ended
- Minimally supervised learning
  - Unlabeled data but have a dictionary

Seems like POS-tagging is solved

- Penn Treebank POS-tagging accuracy ≈ human ceiling
  - Human agreement 97%

  - In other languages, not so much

So now we are HMM Masters

- We can use HMMs to...
  - Tag words in a sentence with their parts of speech
  - Extract entities and other information from a sentence

- Can we use them to determine syntactic categories?

Syntax

- Refers to the study of the way words are arranged together, and the relationship between them.

- Prescriptive vs. Descriptive

- Goal of syntax is to model the knowledge of that people unconsciously have about the grammar of their native language

- Parsing extracts the syntax from a sentence
Parsing applications

- High-precision Question-Answering systems
- Named Entity Recognition (NER) and information extraction
- Opinion extraction in product reviews
- Improved interaction during computer applications/games

Basic English sentence structure

- Ike eats cake.
- Ike sleeps cake.
- Ike gives you cake.
- Ike gives cake. Hmm...
- Ike eat cake???

Can we build an HMM?

- Ike, dogs, ...
- eat, sleep, ...
- cake, science, ...

Words take arguments

- Subcategorization
  - Intransitive verbs: take only a subject
  - Transitive verbs: take a subject and an object
  - Ditransitive verbs: take a subject, object, and indirect object
- Selectional preferences
  - The object of eat should be edible
A better model

Language has recursive properties

HMMs can’t generate hierarchical structure

Words work in groups

- Constituents - words or groupings of words that function as single units
  - Noun phrases (NPs)
  - The computer science class
  - Peter, Paul, and Mary
  - PAC10 Schools, such as UW,
  - He
  - The reason I was late
Words work in groups

- Constituents - words or groupings of words that function as single units
- Noun phrases (NPs)
  - The computer science class listened.
  - PAC10 Schools, such as UW, dominate...
  - He juggled...
  - The reason I was late was...
  - *such sing
  - *late was

NPs can appear before a verb.

Many different constituents

1. S - simple declarative clause
2. SBAR - Clause introduced by a possibly empty subordinating conjunction
3. SBARQ - Direct question introduced by a wh-word or a wh-phrase
4. SQ - Inverted yes/no question, or main clause of a wh-question
5. ADJP - Adjective Phrase
6. ADVP - Adverb Phrase
7. CONJP - Conjunction Phrase
8. FRAG - Fragment
9. INTJ - Interjection
10. LST - List marker
11. NP - Noun Phrase
12. RR - Used within certain complex NPs to mark the head of the NP: Corresponds very roughly to X-bar level but used quite differently.
13. PP - Prepositional Phrase
14. PRN - Parenthetical
15. PRT - Particle
16. RB - Adverb
17. RBR - Adverb, comparative
18. RBS - Adverb, superlative
19. RP - Particle
20. SYM - Symbol
21. TO - to
22. UH - Interjection
23. VB - Verb, base form
24. VBD - Verb, past tense
25. VBG - Verb, gerund or present participle
26. VBN - Verb, past participle
27. VBP - Verb, non-3rd person singular present
28. VBZ - Verb, 3rd person singular present
29. WDT - Wh-determiner
30. WP - Wh-pronoun
31. WRB - Wh-adverb

Attachment ambiguities

- Teacher Strikes Idle Kids
- Squad Helps Dog Bite Victim
- Complaints About NBA Referees Getting Ugly
- Soviet Virgin Lands Short of Goal Again
- Milk Drinkers are Turning to Powder
Attachment ambiguities

• The key parsing decision: How do we ‘attach’ various kinds of constituents - PPs, adverbial or participial phrases, coordinations, etc.
• Prepositional phrase attachment
  ▫ I saw the man with the telescope.
• What does with a telescope modify?
  ▫ The verb saw?
  ▫ The noun man?
• Very hard problem. AI Complete.

Parsing

• We want to run a grammar backwards to find possible structures for a sentence

  • Parsing can be viewed as a search problem

  • Parsing is a hidden data problem

Context-free grammars (CFGs)

• Specifies a set of tree structures that capture constituency and ordering in language

  • A noun phrase can come before a verb phrase

    \[ S \rightarrow NP \; VP \]

Phrase structure grammars = Context-free grammars

• \( G = (T, N, S, R) \)
  ▫ \( T \) is the set of terminals (i.e. words)
  ▫ \( N \) is the set of non-terminals
  ▫ Usually separate the set \( P \) of preterminals (POS tags) from the rest of the non-terminals
  ▫ \( S \) is the start symbol
  ▫ \( R \) is the set of rules/productions of the form \( X \rightarrow \gamma \) where \( X \) is a nonterminal and \( \gamma \) is a sequence of terminals and nonterminals (possibly empty)
  ▫ A grammar \( G \) generates a language \( L \)
A phrase structure grammar

• By convention, S is the start symbol
  - S → NP VP
  - NP → DT NN
  - NP → NNS
  - VP → V NP
  - VP → V
  - ... DT → a

But since a sentence can have more than one parse...

Probabilistic context-free grammars (PCFGs)

• G = (T, N, S, R, P)
  - T is the set of terminals (i.e. words)
  - N is the set of non-terminals
    - Usually separate the set P of preterminals (POS tags) from the rest of the non-terminals
  - S is the start symbol
  - R is the set of rules/productions of the form X → γ
    - X is a nonterminal
    - γ is a sequence of terminals and nonterminals (possibly empty)
  - P(R) gives the probability of each rule

• A grammar G generates a language L

How to parse

• Top-down: Start at the top of the tree with an S node, and work your way down to the words.

• Bottom-up: Look for small pieces that you know how to assemble, and work your way up to larger pieces.

Given a sentence S...

• We want to find the most likely parse τ
  \[ \arg\max_{\tau} P(\tau \mid S) = \frac{\arg\max_{\tau} P(\tau, S)}{P(S)} \]
  \[ = \arg\max_{\tau} P(\tau, S) \]
  \[ = \arg\max_{\tau} P(\tau) \quad \text{if } S = \text{yield}(\tau) \]

• How are we supposed to find P(τ)?
• Infinitely many trees in the language!
Finding $P(\tau)$

- Define probability distributions over the rules in the grammar
- Context free!

<table>
<thead>
<tr>
<th>Rule</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S \rightarrow NP \ VP$</td>
<td>0.8</td>
</tr>
<tr>
<td>$S \rightarrow S \ conj \ S$</td>
<td>0.2</td>
</tr>
<tr>
<td>$NP \rightarrow Noun$</td>
<td>0.2</td>
</tr>
<tr>
<td>$NP \rightarrow Det \ Noun$</td>
<td>0.4</td>
</tr>
<tr>
<td>$NP \rightarrow NP \ PP$</td>
<td>0.2</td>
</tr>
<tr>
<td>$NP \rightarrow NP \ conj \ NP$</td>
<td>0.2</td>
</tr>
<tr>
<td>$VP \rightarrow Verb$</td>
<td>0.4</td>
</tr>
<tr>
<td>$VP \rightarrow Verb \ NP$</td>
<td>0.3</td>
</tr>
<tr>
<td>$VP \rightarrow Verb \ NP \ NP$</td>
<td>0.1</td>
</tr>
<tr>
<td>$VP \rightarrow VP \ PP$</td>
<td>0.2</td>
</tr>
<tr>
<td>$PP \rightarrow P \ NP$</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Finding $P(\tau)$

- The probability of a tree is the product of the probability of the rules that created it

<table>
<thead>
<tr>
<th>Rule</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S \rightarrow NP \ VP$</td>
<td>0.8</td>
</tr>
<tr>
<td>$S \rightarrow S \ conj \ S$</td>
<td>0.2</td>
</tr>
<tr>
<td>$NP \rightarrow Noun$</td>
<td>0.2</td>
</tr>
<tr>
<td>$NP \rightarrow Det \ Noun$</td>
<td>0.4</td>
</tr>
<tr>
<td>$NP \rightarrow NP \ PP$</td>
<td>0.2</td>
</tr>
<tr>
<td>$NP \rightarrow NP \ conj \ NP$</td>
<td>0.2</td>
</tr>
<tr>
<td>$VP \rightarrow Verb$</td>
<td>0.4</td>
</tr>
<tr>
<td>$VP \rightarrow Verb \ NP$</td>
<td>0.3</td>
</tr>
<tr>
<td>$VP \rightarrow VP \ PP$</td>
<td>0.2</td>
</tr>
<tr>
<td>$PP \rightarrow P \ NP$</td>
<td>1.0</td>
</tr>
</tbody>
</table>

$P(\tau) = 0.8 \times 0.3 \times 0.2 \times 1.0 \times 0.2^2 = 0.00384$

Parsing - Cocke-Kasami-Younger (CKY)

- Like Viterbi but for trees
- Guaranteed to find the most likely parse

Chomsky Normal Form

- All rules are of the form $X \rightarrow Y \ Z$ or $X \rightarrow w$.
- $n$-ary rules introduce new nonterminals ($n > 2$)
  - $VP \rightarrow V \ NP \ PP$ becomes:
    - $VP \rightarrow V \ @VP-V$ and
    - $@VP-V \rightarrow NP \ PP$
CKY Example

Input: POS-tagged sentence
John_N eats_V pie_N with_P cream_N

Estimating $P(X \rightarrow \alpha)$

- **Supervised**
  - Relative frequency estimation
  - Count what is seen in a treebank corpus
  
  \[
  P(X \rightarrow \alpha) = \frac{C(X \rightarrow \alpha)}{C(X)}
  \]

- **Unsupervised**
  - Expected relative frequency estimation
  - Use Inside-Outside Algorithm (EM variant)
  
  \[
  P(X \rightarrow \alpha) = \frac{E[C(X \rightarrow \alpha)]}{E[C(X)]}
  \]

How well do PCFGs perform?

- Runtime - supercubic!

- Robust to variations in language
- Strong independence assumptions
- WSJ parsing accuracy: about 73% LP/LR F1

- Lack of lexicalization
  - A PCFG uses the actual words only to determine the probability of parts-of-speech (the preterminals)
  - I like to eat cake with white frosting.
  - I like to eat cake with a spork.
Lexicalization

- Lexical heads are important for certain classes of ambiguities (e.g., PP attachment):
  - Lexicalizing grammar creates a much larger grammar.
    - Sophisticated smoothing needed
    - Smarter parsing algorithms needed
    - More DATA needed

Huge area of research

- Coarse-to-fine parsing
  - Parse with a simpler grammar
  - Refine with a more complex one
- Dependency parsing
  - A sentence is parsed by relating each word to other words in the sentence which depend on it.
- Discriminative parsing
  - Given training examples, learn a function that classifies a sentence with its parse tree
  - and more!

The good news!

- Part of speech taggers and sentence parsers are freely available!

- So why did we sit through this lecture?
  - Maybe you’ll be interested in this area
  - Useful ideas to be applied elsewhere
    - Write a parser to parse web tables
    - PCFGs for information extraction
  - Like to know how things work

It’s over!

- Thanks!