Lecture 6: part-of-speech tagging to parsing

Instructor: Robert C. Berwick
berwick@ai.mit.edu
The Menu Bar

• Administrivia:
  • Schedule alert: Lab1 due next today Lab 2, posted Feb 24; due the Weds after this – March 5 (web only – can post pdf)

• Agenda:
  • Finish up POS tagging – Brill method
  • From tagging to parsing: from linear representations to hierarchical representations
Two approaches

1. **Noisy Channel Model (statistical)** –
2. Deterministic baseline tagger composed with a cascade of fixup transducers

These two approaches will be the guts of Lab 2 (lots of others: decision trees, ...)
Summary

• We are modeling $p(\text{word seq, tag seq})$
• The tags are hidden, but we see the words
• Is tag sequence $X$ likely with these words?
• Noisy channel model is a “Hidden Markov Model”:

- Find $X$ that maximizes probability product

Noisy channel model is a “Hidden Markov Model”:

- Find $X$ that maximizes probability product
Finding the best path from start to stop

- Use dynamic programming
- What is best path from Start to each node?
  - Work from left to right
  - Each node stores its best path from Start (as probability plus one backpointer)
- Special acyclic case of Dijkstra’s shortest-path algorithm
- Faster if some arcs/states are absent
Method: Viterbi algorithm

- For each path reaching state $s$ at step (word) $t$, we compute a path probability. We call the max of these $\text{viterbi}(s,t)$.

- [Base step] Compute $\text{viterbi}(0,0)=1$

- [Induction step] Compute $\text{viterbi}(s',t+1)$, assuming we know $\text{viterbi}(s,t)$ for all $s$
Viterbi recursion

\[
\text{path-prob}(s'|s,t) = \text{viterbi}(s,t) \times a[s,s']
\]

probability of path to s’ through s  
max path score * transition probability for state s at time t  
s \rightarrow s’

\[
viterbi(s',t+1) = \max_{s \in \text{STATES}} \text{path-prob}(s' | s,t)
\]
Viterbi Method...

- This is almost correct...but again, we need to factor in the unigram prob of a state $s'$ emitting a particular word $w$ given an observation of that surface word $w$.
- So the correct formula for the path prob to $s'$ from $s$ is:

$$\text{path-prob}(s'|s,t) = \text{viterbi}(s,t) \times a[s,s'] \times b_{s'}(o_t)$$
Finally...

- As before, we want to find the max path probability, over all states $s$:

$$\max_{s \in \text{STATES}} \text{path-prob}(s' \mid s,t)$$
function \textsc{Viterbi}(observations of len \textit{T}, state-graph) \textbf{returns} best-path

\textit{num-states} $\leftarrow$ \textsc{Num-of-States}(state-graph)
Create a path probability matrix \textit{viterbi}[num-states+2, T+2]
\textit{viterbi}[0, 0] $\leftarrow$ 1.0
for each time step \textit{t} from 0 to \textit{T} do
  for each state \textit{s} from 0 to num-states do
    for each transition \textit{s'} from \textit{s} specified by state-graph
      \textit{new-score} $\leftarrow$ \textit{viterbi}[s, t] $\times$ \textit{a}[s, s'] $\times$ \textit{b}_{s'}(\textit{o}_t)
      \textbf{Find the path probability}
      if \((\textit{viterbi}[s', t+1] = 0) \ | | (\textit{new-score} \ > \ \textit{viterbi}[s', t+1]))
        \textbf{Find the max so far}
        then
          \textit{viterbi}[s', t+1] $\leftarrow$ \textit{new-score}
          \textit{back-pointer}[s', t+1] $\leftarrow$ \textit{s}
Backtrace from highest probability state in the final column of \textit{viterbi}[\textit{t}] and
return path

6.863J/9.611J Lecture 6 Sp03
Two approaches

1. Noisy Channel Model (statistical) – what’s that?? (we will have to learn some statistics)

2. Deterministic baseline tagger composed with a cascade of fixup transducers

These two approaches will be the guts of Lab 2 (lots of others: decision trees, ...)
Fixup approach: Brill tagging (a kind of transformation-based learning)
Another FST Paradigm: Successive Fixups

- Like successive markups but alter
- Morphology
- Phonology
- Part-of-speech tagging
- ...

Initial annotation → Fixup 1 → Fixup 2 → Fixup 3 → output
Transformation-Based Tagging
(Brill 1995)

figure from Brill’s thesis
Brill Tagger

Powered by μ-TBL Technology

☐ Swedish ☑ English ☐ Russian

Text:

Secretariat is expected to race tomorrow

☒ Trace ☑ Analyze
Tokenization

Secretariat is expected to race tomorrow

Lexical lookup

Secretariat/NNP is/VBZ expected/VBN to/TO race/NN tomorrow/NN

Guessing

Contextual-rule application

Intermediate analysis:

Secretariat/NNP is/VBZ expected/VBN to/TO race/NN tomorrow/NN

Applied rule:

tag:NN>VB <- tag:TO[−1].

Analysis

Secretariat/NNP is/VBZ expected/VBN to/TO race/VB tomorrow/NN
Transformation based tagging

• Combines symbolic and stochastic approaches: uses machine learning to refine its tags, via several passes

• Analogy: painting a picture, use finer and finer brushes - start with broad brush that covers a lot of the canvas, but colors areas that will have to be repainted. Next layer colors less, but also makes fewer mistakes, and so on.

• Similarly: tag using broadest (most general) rule; then an narrower rule, that changes a smaller number of tags, and so on. (We haven’t said how the rules are learned)

• First we will see how the TBL rules are applied
Applying the rules

1. First label every word with its most-likely tag (as we saw, this gets 90% right...!) for example, in Brown corpus, race is most likely to be a Noun:
   \[ P(\text{NN}|\text{race}) = 0.98 \]
   \[ P(\text{VB}|\text{race}) = 0.02 \]

2. ...expected/VBZ to/T TO race/VB tomorrow/NN
   ...the/DT race/NN for/IN outer/JJ space/NN

3. Use transformational (learned) rules to change tags:
   Change \textbf{NN} to \textbf{VB} when the previous tag is \textbf{TO}
## Initial Tagging of OOV Words

<table>
<thead>
<tr>
<th>#</th>
<th>From</th>
<th>To</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NN</td>
<td>NNS</td>
<td>Has suffix -s</td>
</tr>
<tr>
<td>2</td>
<td>NN</td>
<td>CD</td>
<td>Has character .</td>
</tr>
<tr>
<td>3</td>
<td>NN</td>
<td>JJ</td>
<td>Has character -</td>
</tr>
<tr>
<td>4</td>
<td>NN</td>
<td>VBN</td>
<td>Has suffix -ed</td>
</tr>
<tr>
<td>5</td>
<td>NN</td>
<td>VBG</td>
<td>Has suffix -ing</td>
</tr>
<tr>
<td>6</td>
<td>??</td>
<td>RB</td>
<td>Has suffix -ly</td>
</tr>
<tr>
<td>7</td>
<td>??</td>
<td>JJ</td>
<td>Adding suffix -ly results in a word.</td>
</tr>
<tr>
<td>8</td>
<td>NN</td>
<td>CD</td>
<td>The word $ can appear to the left.</td>
</tr>
<tr>
<td>9</td>
<td>NN</td>
<td>JJ</td>
<td>Has suffix -al</td>
</tr>
<tr>
<td>10</td>
<td>NN</td>
<td>VB</td>
<td>The word would can appear to the left.</td>
</tr>
<tr>
<td>11</td>
<td>NN</td>
<td>CD</td>
<td>Has character 0</td>
</tr>
<tr>
<td>12</td>
<td>NN</td>
<td>JJ</td>
<td>The word be can appear to the left.</td>
</tr>
<tr>
<td>13</td>
<td>NNS</td>
<td>JJ</td>
<td>Has suffix -us</td>
</tr>
<tr>
<td>14</td>
<td>NNS</td>
<td>VBZ</td>
<td>The word it can appear to the left.</td>
</tr>
<tr>
<td>15</td>
<td>NN</td>
<td>JJ</td>
<td>Has suffix -ble</td>
</tr>
<tr>
<td>16</td>
<td>NN</td>
<td>JJ</td>
<td>Has suffix -ic</td>
</tr>
<tr>
<td>17</td>
<td>NN</td>
<td>CD</td>
<td>Has character 1</td>
</tr>
<tr>
<td>18</td>
<td>NNS</td>
<td>NN</td>
<td>Has suffix -ss</td>
</tr>
<tr>
<td>19</td>
<td>??</td>
<td>JJ</td>
<td>Deleting the prefix un- results in a word</td>
</tr>
<tr>
<td>20</td>
<td>NN</td>
<td>JJ</td>
<td>Has suffix -ive</td>
</tr>
</tbody>
</table>
(supervised) learning pudding - How?

- 3 stages
1. Start by labeling every word with most-likely tag
2. Then examine every possible transformation, and selects one that results in most improved tagging
3. Finally, re-tags data according to this rule
4. Repeat 1-3 until some stopping criterion (no new improvement, or small improvement)

- Output is ordered list of transformations that constitute a tagging procedure
How this works

- Set of possible ‘transforms’ is infinite, e.g., “transform NN to VB if the previous word was MicrosoftWindoze & word braindead occurs between 17 and 158 words before that”
- To limit: start with small set of abstracted transforms, or templates
Templates used: Change a to b when...

The preceding (following) word is tagged z.
The word two before (after) is tagged z.
One of the two preceding (following) words is tagged z.
One of the three preceding (following) words is tagged z.
The preceding word is tagged z and the following word is tagged w.
The preceding (following) word is tagged z and the word
two before (after) is tagged w.

Variables $a, b, z, w$, range over parts of speech
Method

1. Call *Get-best-transform* with list of potential templates; this calls
2. *Get-best-instance* which instantiates each template over all its variables (given specific values for where we are)
3. Try it out, see what score is (improvement over known tagged system -- supervised learning); pick best one locally
function TBL(corpus) returns transforms-queue
    INITIALIZE-WITH-MOST-LIKELY-TAGS(corpus)
    until end condition is met do
        templates ← GENERATE-POTENTIAL-RELEVANT-TEMPLATES
        best-transform ← GET-BEST-TRANSFORM(corpus, templates)
        APPLY-TRANSFORM(best-transform, corpus)
        ENQUEUE(best-transform-rule, transforms-queue)
    end
    return(transforms-queue)

function GET-BEST-TRANSFORM(corpus, templates) returns transform
    for each template in templates
        (instance, score) ← GET-BEST-INSTANCE(corpus, template)
        if (score > best-transform.score) then best-transform ← (instance, score)
    return(best-transform)
function GET-BEST-INSTANCE(corpus, template) returns transform

for from-tag ← from tag − 1 to tag − n do
  for to-tag ← from tag − 1 to tag − n do
    for pos ← from 1 to corpus-size do
      if (correct-tag(pos) == to-tag && current-tag(pos) == from-tag)
        num-good-transforms(current-tag(pos−1))++
      elseif (correct-tag(pos) == from-tag && current-tag(pos) == from-tag)
        num-bad-transforms(current-tag(pos−1))++
    end
  end
best-Z ← ARGMAX_{t}(num-good-transforms(t) - num-bad-transforms(t),
if(num-good-transforms(best-Z) - num-bad-transforms(best-Z) > best-instance.Z) then
  best-instance ← “Change tag from from-tag to to-tag
  if previous tag is best-Z”
return(best-instance)

procedure APPLY-TRANSFORM(transform, corpus)
for pos ← from 1 to corpus-size do
  if (current-tag(pos) == best-rule-from)
    && (current-tag(pos−1) == best-rule-prev))
    current-tag(pos) = best-rule-to
nonlexicalized rules learned by TBL tagger

<table>
<thead>
<tr>
<th>#</th>
<th>From</th>
<th>To</th>
<th>Condition</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NN</td>
<td>VB</td>
<td>Previous tag is TO</td>
<td>to/TO race/NN → VB</td>
</tr>
<tr>
<td>2</td>
<td>VBP</td>
<td>VB</td>
<td>One of the previous 3 tags is MD</td>
<td>might/MD vanish/VBP → VB</td>
</tr>
<tr>
<td>3</td>
<td>NN</td>
<td>VB</td>
<td>One of the previous 2 tags is MD</td>
<td>might/MD not reply/NN → VB</td>
</tr>
<tr>
<td>4</td>
<td>VB</td>
<td>NN</td>
<td>One of the previous 2 tags is DT</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>VBD</td>
<td>VBN</td>
<td>One of the previous 3 tags is VBZ</td>
<td></td>
</tr>
</tbody>
</table>
Transformations Learned

<table>
<thead>
<tr>
<th>#</th>
<th>From</th>
<th>To</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NN</td>
<td>VB</td>
<td>Previous tag is <em>TO</em></td>
</tr>
<tr>
<td>2</td>
<td>VBP</td>
<td>VB</td>
<td>One of the previous three tags is <em>MD</em></td>
</tr>
<tr>
<td>3</td>
<td>NN</td>
<td>VB</td>
<td>One of the previous two tags is <em>MD</em></td>
</tr>
<tr>
<td>4</td>
<td>VB</td>
<td>NN</td>
<td>One of the previous two tags is <em>DT</em></td>
</tr>
<tr>
<td>5</td>
<td>VBD</td>
<td>VBN</td>
<td>One of the previous three tags is <em>VBZ</em></td>
</tr>
<tr>
<td>6</td>
<td>VBN</td>
<td>VBD</td>
<td>Previous tag is <em>PRP</em></td>
</tr>
<tr>
<td>7</td>
<td>VBN</td>
<td>VBD</td>
<td>Previous tag is <em>NNP</em></td>
</tr>
<tr>
<td>8</td>
<td>VBD</td>
<td>VBN</td>
<td>Previous tag is <em>VBD</em></td>
</tr>
<tr>
<td>9</td>
<td>VBP</td>
<td>VB</td>
<td>Previous tag is <em>TO</em></td>
</tr>
<tr>
<td>10</td>
<td>POS</td>
<td>VBZ</td>
<td>Previous tag is <em>PRP</em></td>
</tr>
<tr>
<td>11</td>
<td>VB</td>
<td>VBP</td>
<td>Previous tag is <em>NNS</em></td>
</tr>
<tr>
<td>12</td>
<td>VBD</td>
<td>VBN</td>
<td>One of previous three tags is <em>VBP</em></td>
</tr>
<tr>
<td>13</td>
<td>IN</td>
<td>WDT</td>
<td>One of next two tags is <em>VB</em></td>
</tr>
<tr>
<td>14</td>
<td>VBD</td>
<td>VBN</td>
<td>One of previous two tags is <em>VB</em></td>
</tr>
<tr>
<td>15</td>
<td>VB</td>
<td>VBP</td>
<td>Previous tag is <em>PRP</em></td>
</tr>
<tr>
<td>16</td>
<td>IN</td>
<td>WDT</td>
<td>Next tag is <em>VBZ</em></td>
</tr>
<tr>
<td>17</td>
<td>IN</td>
<td>DT</td>
<td>Next tag is <em>NN</em></td>
</tr>
<tr>
<td>18</td>
<td>JJ</td>
<td>NNP</td>
<td>Next tag is <em>NNP</em></td>
</tr>
<tr>
<td>19</td>
<td>IN</td>
<td>WDT</td>
<td>Next tag is <em>VBD</em></td>
</tr>
<tr>
<td>20</td>
<td>JJR</td>
<td>RBR</td>
<td>Next tag is <em>JJ</em></td>
</tr>
</tbody>
</table>

BaselineTag*

NN $\rightarrow$ VB // TO _
VBP $\rightarrow$ VB // ... _ etc.

Compose this cascade of FSTs.

Get a big FST that does the initial tagging and the sequence of fixups “all at once.”
Error analysis: what’s hard for taggers

• Common errors (> 4%)
  • NN vs .NNP (proper vs. other nouns) vs. JJ (adjective): hard to distinguish prenominally; important to distinguish esp. for information extraction
  • RP vs. RB vs IN: all can appear in sequences immed. after verb
  • VBD vs. VBN vs. JJ: distinguish past tense, past participles (raced vs. was raced vs. the out raced horse)
What’s hard

• Unknown words
  • Order 0 idea: equally likely over all parts of speech
  • Better idea: same distribution as ‘Things seen once’ estimator of ‘things never seen’ - theory for this done by Turing (again!)
  • Hapax legomenon
  • Assume distribution of unknown words is like this
  • But most powerful methods make use of how word is spelled

• See file in the course tagging dir on this
Or unknown language

- Vse schastlivye sen’i pokhozhi brug na druga, kazhdaja neschastlivaja sem’ja neschastliva po-svoemu
Brill Tagger

Powered by μ-TBL Technology

○ Swedish ○ English ○ Russian

Text:

"Ive schastlivye seni pokhozhi brug na druga, kazhdaja neschastlivaja semja neschastlivaja po"

[Trace] [Analyze]
Most powerful unknown word detectors

- 3 inflectional endings (-ed, -s, -ing); 32 derivational endings (-ion, etc.); capitalization; hyphenation
- More generally: should use morphological analysis! (and some kind of machine learning approach)
- How hard is this? We don’t know - we actually don’t know how children do this, either (they make mistakes)
Laboratory 2

- **Goals:**
  1. Use both HMM and Brill taggers
  2. Find errors that both make, relative to genre
  3. Compare performance – use of kappa & ‘confusion matrix’
  4. All the slings & arrows of corpora – use Wall Street Journal excerpts, as well as ‘switchboard’ corpus
Brown/Upenn corpus tags

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC</td>
<td>Coordin. Conjunction</td>
<td>and, but, or</td>
</tr>
<tr>
<td>CD</td>
<td>Cardinal number</td>
<td>one, two, three</td>
</tr>
<tr>
<td>DT</td>
<td>Determiner</td>
<td>a, the</td>
</tr>
<tr>
<td>EX</td>
<td>Existential ‘there’</td>
<td>there</td>
</tr>
<tr>
<td>FW</td>
<td>Foreign word</td>
<td>mea culpa</td>
</tr>
<tr>
<td>IN</td>
<td>Preposition/sub-conj</td>
<td>of, in, by</td>
</tr>
<tr>
<td>JJ</td>
<td>Adjective</td>
<td>yellow</td>
</tr>
<tr>
<td>JJR</td>
<td>Adj., comparative</td>
<td>bigger</td>
</tr>
<tr>
<td>JJS</td>
<td>Adj., superlative</td>
<td>wildest</td>
</tr>
<tr>
<td>LS</td>
<td>List item marker</td>
<td>1, 2, One</td>
</tr>
<tr>
<td>MD</td>
<td>Modal</td>
<td>can, should</td>
</tr>
<tr>
<td>NN</td>
<td>Noun, sing. or mass</td>
<td>llama</td>
</tr>
<tr>
<td>NNS</td>
<td>Noun, plural</td>
<td>llamas</td>
</tr>
<tr>
<td>NNP</td>
<td>Proper noun, singular</td>
<td>IBM</td>
</tr>
<tr>
<td>NNPS</td>
<td>Proper noun, plural</td>
<td>Carolinas</td>
</tr>
<tr>
<td>PDT</td>
<td>Predeterminer</td>
<td>all, both</td>
</tr>
<tr>
<td>POS</td>
<td>Possessive ending</td>
<td>’s</td>
</tr>
<tr>
<td>PP</td>
<td>Personal pronoun</td>
<td>I, you, he</td>
</tr>
<tr>
<td>PP$</td>
<td>Possessive pronoun</td>
<td>your, one’s</td>
</tr>
<tr>
<td>RB</td>
<td>Adverb</td>
<td>quickly, never</td>
</tr>
<tr>
<td>RBR</td>
<td>Adverb, comparative</td>
<td>faster</td>
</tr>
<tr>
<td>RBS</td>
<td>Adverb, superlative</td>
<td>fastest</td>
</tr>
<tr>
<td>RP</td>
<td>Particle</td>
<td>up, off</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYM</td>
<td>Symbol</td>
<td>+, %, &amp;</td>
</tr>
<tr>
<td>TO</td>
<td>“to”</td>
<td>to</td>
</tr>
<tr>
<td>UH</td>
<td>Interjection</td>
<td>ah, oops</td>
</tr>
<tr>
<td>VB</td>
<td>Verb, base form</td>
<td>eat</td>
</tr>
<tr>
<td>VBD</td>
<td>Verb, past tense</td>
<td>ate</td>
</tr>
<tr>
<td>VBG</td>
<td>Verb, gerund</td>
<td>eating</td>
</tr>
<tr>
<td>VBN</td>
<td>Verb, past participle</td>
<td>eaten</td>
</tr>
<tr>
<td>VBP</td>
<td>Verb, past participle</td>
<td>eat</td>
</tr>
<tr>
<td>VBZ</td>
<td>Verb, 3sg pres</td>
<td>eats</td>
</tr>
<tr>
<td>WDT</td>
<td>Wh-determiner</td>
<td>which, that</td>
</tr>
<tr>
<td>WP</td>
<td>Wh-pronoun</td>
<td>what, who</td>
</tr>
<tr>
<td>WP$</td>
<td>Possessive wh-</td>
<td>whose</td>
</tr>
<tr>
<td>WRB</td>
<td>Wh-adverb</td>
<td>how, where</td>
</tr>
<tr>
<td>$</td>
<td>Dollar sign</td>
<td>$</td>
</tr>
<tr>
<td>#</td>
<td>Pound sign</td>
<td>#</td>
</tr>
<tr>
<td>”</td>
<td>Left quote</td>
<td>(‘ or “)</td>
</tr>
<tr>
<td>”</td>
<td>Right quote</td>
<td>(’ or ”)</td>
</tr>
<tr>
<td>(</td>
<td>Left parenthesis</td>
<td>[, (, {, &lt;)</td>
</tr>
<tr>
<td>)</td>
<td>Right parenthesis</td>
<td>[), }, &gt;)</td>
</tr>
<tr>
<td>,</td>
<td>Comma</td>
<td>,</td>
</tr>
<tr>
<td>:</td>
<td>Sentence-final punc</td>
<td>(. ! ?)</td>
</tr>
<tr>
<td>;</td>
<td>Mid-sentence punc</td>
<td>(: ; ... – -)</td>
</tr>
</tbody>
</table>
Coda on kids

C: “Mommy, nobody don’t like me”

A: No, say, “nobody likes me”

C: Nobody don’t likes me

A: Say, “nobody likes me”

C: Nobody don’t likes me

[7 repetitions]

C: Oh! Nobody don’t like me!
Parsing words - review

- We are mapping between surface, underlying forms
- Sometimes, information is ‘invisible’ (i.e., erased e, or an underlying/surface 0)
- There is ambiguity (more than one parse)
From lines to hierarchical representations...

- From this: **morph-ology**
- To this:

\[
\text{VP [head=vouloir, ...]}
\]
\[
\text{V [head=vouloir, ...}
\text{tense=Present, num=SG, person=P3]}
\]
\[
\text{veut}
\]

the problem of **morphology** ("word shape") - an area of linguistics
What can’t linear relations represent?

- wine dark sea → (wine (dark sea)) or ((wine dark) sea) ?

- deep blue sky

- Can fsa’s represent this?
  - Not really: algebraically, defined as being associative (doesn’t matter about concatenation order)
So, from linear relations... to hierarchies
The plan to swallow Wanda has been thrilling Otto.
Examples

Verb $\rightarrow$ thrills
VP $\rightarrow$ Verb NP
S $\rightarrow$ NP VP

A roller coaster thrills every teenager
Parsing for fsa’s: keep track of what ‘next state’ we could be in at each step

NB: ambiguity = > 1 path through network
= > 1 sequence of states (‘parses’)
= > 1 ‘syntactic rep’ = > 1 ‘meaning’
Brill Tagger

Powered by μ-TBL Technology

- Swedish  English  Russian

Text:

```
```

Tokenization

fruit flies like a banana

Lexical lookup

```
fruit/MN flies/VBZ like/IN a/DT banana/NN
```

Guessing

Contextual-rule application
FSA Terminology

- Transition function: next state unique = deterministic fsa
- Transition relation: > 1 next state = nondeterministic fsa

fruit flies like a banana

fruit flies like a banana
Methods for parsing

• How do we handle ambiguity?
• Methods:
  1. Backtrack
  2. Convert to deterministic machine (ndfsa → dfsa): offline compilation
  3. Pursue all paths in parallel: online computation ("state set" method)
  4. Use lookahead

  We will use all these methods for more complex machines/language representations
FSA terminology

- Input alphabet, $\Sigma$; transition mapping, $\delta$; finite set of states, $Q$; start state $q_0$; set of final states, $q_f$
- $\delta(q, s) \rightarrow q'$
- Transition function: next state unique = deterministic fsa
- Transition relation: $> 1$ next state = nondeterministic fsa
State-set method: simulate a nondeterministic fsa

- Compute all the possible next states the machine can be in at a step = state-set
- Denote this by $S_i = \text{set of states machine can be in after analyzing } i \text{ tokens}$
- Algorithm has 3 parts: (1) Initialize; (2) Loop; (3) Final state?
  - Initialize: $S_0$ denotes initial set of states we’re in, before we start parsing, that is, $q_0$
  - Loop: We must compute $S_i$, given $S_{i-1}$
  - Final?: $S_f = \text{set of states machine is in after reading all tokens; we want to test if there is a final state in there}$
State-set parsing

Initialize: Compute initial state set, $S_0$

1. $S_0 \leftarrow q_0$
2. $S_0 \leftarrow \varepsilon$–closure($S_0$)

Loop: Compute $S_i$ from $S_{i-1}$

1. For each word $w_i$, $i=1,2,\ldots,n$
2. $S_i \leftarrow \bigcup_{q \in S_{i-1}} \delta(q, w_i)$
3. $S_i \leftarrow \varepsilon$–closure($S_i$)
4. if $S_i = \emptyset$ then halt & reject else continue

Final: Accept/reject

1. If $q_f \in S_i$ then accept else reject
What’s the minimal data structure we need for this?

- $[S, i]$ where $S =$ denotes set of states we could be in; $i$ denotes current point we’re at in sentence

- As we’ll see, we can use this same representation for parsing w/ more complex networks (grammars) - we just need to add one new piece of information for state names

- In network form: $q_i \xrightarrow{\alpha \cdot \beta} q_k$

- In rule form: $q_i \rightarrow t \cdot \beta \ q_f$ where $t =$ some token of the input, and $\beta =$ remainder (so ‘dot’ represents how far we have traveled)
Example

fruit flies like a banana

0 1 2 3 4 5

banana
Use backpointers to keep track of the different paths (parses):

State set 0

S0:[0]  S1:[0, 1]  S2:[1, 2, 3]  S3:[2, 3]  S4:[4]  S5:[5]

State set f
When is it better to convert at compile time vs. run time? (for fsa)

- Run time: compute next state set on the fly
- Compile time: do it once and for all
- When would this difference show up in natural languages (if at all)?
Where do the fsa states come from?

- States are equivalence classes of words (tokens) under the operation of substitution.

- Linguistic formulation (Wells, 1947, pp. 81-82): “A word A belongs to the class determined by the environment ___X if AX is either an utterance or occurs as a part of some utterance” (distributional analysis).

- This turns out to be algebraically correct.
- Can be formalized - the notion of syntactic equivalence.
X-files: fragments from an alien language

1. Kerry lost the election
2. Gore will lose the election
3. Gore could lose the election
4. Gore should lose the election
5. Gore did lose the election
6. Gore could have lost the election
7. Gore should have lost the election
8. Gore will have lost the election
9. Gore could have been losing the election
10. Gore should have been losing the election
11. Gore will have been losing the election
12. Gore has lost the election
More X-files

14. Bush lost the election
15. Bush will lose the election
16. Bush could lose the election
17. Bush should lose the election
18. Bush did lose the election
19. Bush could have lost the election
20. Bush should have lost the election
21. Bush will have lost the election
22. Bush could have been losing the election
23. Bush should have been losing the election
24. Bush will have been losing the election
25. Bush has lost the election
26. Bush has been losing the election
Formally...

- **Definition.** A binary relation between sets $A, B$, is a subset (possibly empty) of $A \times B$

- **Definition.** Strings $k, r$ are left-substitutable in a language $L$, if, for all strings $w$ defined over $\Sigma^*$, $kw \in L$ iff $rw \in L$

- **Fact.** Left-substitutability is an equivalence relation (reflexive, transitive, symmetric)

- **Definition.** An equivalence relation over $\Sigma$ is finite rank if it divides $\Sigma$ into finitely many equivalence classes

- **Definition.** A binary relation $R$ is called right-invariant if, for all $p, r \in \Sigma^*$, $pRr \Rightarrow pwRrw$
And formally...

• Fact. A right-invariant relation $R$ is an equivalence relation
• **Theorem** (Myhill-Nerode, 1956)
Theorem (Myhill-Nerode, 1956).

- Let $L \subseteq \Sigma^*$. Then the following 3 propositions are equivalent:

1. $L$ is generated (accepted) by some finite-state automaton (finite transition network);

2. $L$ is the union of certain equivalence classes of a right-invariant equivalence relation of finite rank.

3. Let the equivalence relation $R$ be defined as follows: $xRy$ iff $x$ and $y$ are left-substitutable in $L$. Then this relation $R$ is of finite-rank and is right-invariant [this is Wells’ definition].
Finite # of bins = finite state

• Gives easy way to show what is not finite-state
• Eg, $a^n c b^n$, for all $n > 0$
• Proof by contradiction.

Suppose there was such an FSA. By the theorem, this FSA is of finite rank, and classifies all strings in $\Sigma^*$ into one of a finite number of classes.

By the pigeonhole principle, there must exist some string $a^i$ s.t. $a^j$ with $j \neq i$ is in the same equivalence class as $a^i$. But then the fsa must recognize both $a^i c a^j$ and $a^i c a^i$, a contradiction.
Why not fsa’s forever?

• Can’t yield the right set of strings = weak generative capacity (antiantimissle...)
• Can’t yield the right set of structures = strong generative capacity (dark blue sky)
• How do these failures show up?
A more complex fsa

0 1

4 5 6 7

2 3 8 9 10 11

the guy have will, can have has, have been has, have been is, are been is, are been been being be

eat be eating ice-cream eats eaten

6.863J/9.611J Lecture 6 Sp03
Conversion to deterministic machine

```
0  the  1  guy
   ↓       ↓
  2,3
   ↓       ↓
  5,9
   ↓       ↓
10,11
```

```
7  eats  eat
  ▼    ▼
11  being  being
     ▼  ▼
12  eaten  eaten
```

```
6, 7, 10, 11  eating  being
            ▲    ▲
6,10  being  eating
        ▲    ▲
4,8  being  eating
               ▲  ▲
3,2  being  eating
```

```
5,9  being  eating
     ▲    ▲
0,1  being  eating
```

```
11  eaten  eaten
       ▲    ▲
12  eaten  eaten
```

```
12  ice-cream
```

```
6,7,10,11  being  eating
```

```
6,10  being  eating
```

```
4,8  being  eating
```

```
3,2  being  eating
```

```
0,1  being  eating
```

```
5,9  being  eating
```

```
2,3  being  eating
```

```
1  guy
```

```
the  1
```

```
0
```

```
the  1
```

```
the  1
```

```
the  1
```
What are we missing here?
We are missing the symmetry
Having a poor representation...

- Shows up in having duplicated states (with no other connection to each other)
- System would be ‘just as complex’= have the same size (what is size of automaton?) even if the network were not symmetric
- So we have failed to capture this regularity & the network could be compressed
- How?
Compressability reveals redundancy (pattern) that we have missed

Active: + Rule that flips network =

Passive:

Aka “transformational grammar”
But it’s worse than that... more redundancy even so

So, obvious programming approach: use a subroutine
Subnetworks as subroutines, to compress the description

Sentence: the guy saw the guy

Noun phrase: the guy

“splice out” common subnets
Could be worse...

Could be raining...

Noun “specifiers”
It could be even worse...

Noun specifiers
Examples

Verb → thrills
VP → Verb NP
S → NP VP

A roller coaster thrills every teenager
The notion of a **common subnetwork**

- Equivalent to the notion of a **phrase**
- A **Noun Phrase** (NP)
- Defined by substitution class of a sequence of words (aka “a **constituent**”) - extension beyond substitution of single words
- A phrase iff we can interchangeably substitute that sequence of words regardless of context
- So also gives us the notion of a **context-free grammar** (CFG)
Constituents, aka phrases

• Building blocks that are units of words concatenated together
• Why?
• Ans:
  1. They act together (i.e., behave alike under operations) - what operations?
  2. Succinctness
  3. (Apparently) nonadjacent constraints
The deepest lesson

- Claim: all apparently nonadjacent relationships in language can be reduced to adjacent ones via projection to a new level of representation
- (In one sense, vacuous; in another, deep)
- Example: Subject-Verb agreement (agreement generally)
- Example: so-called wh-movement
Gaps ("deep" grammar!)

- Pretend "kiss" is a pure transitive verb.
- Is "the president kissed" grammatical?
  - If so, what type of phrase is it?

- the sandwich that
- I wonder what
- What else has

- the president kissed e
- Sally said the president kissed e
- Sally consumed the pickle with e
- Sally consumed e with the pickle
Examples

• The guy that we know in Somerville likes ice-cream
• Who did the guy who lives in Somerville see __?

[Diagram]

6.863J/9.611J Lecture 6 Sp03
The deep reason why

- Machinery of the mind: based only on concatenation of adjacent elements - not on ‘counting’ eg., “take the 7th element & move it…”
- Runs through all of linguistic representations (stress, metrical patterns, phonology, syntax, …)
- Strong constraint on what we have to represent
Constituents

• Basic ‘is-a’ relation
• Act as ‘whole units’ -
  • I want this student to solve the problem
  • ?? Student, I want this to solve the problem
  • This student, I want to solve the problem
• Sometimes, we don’t see whole constituents...book titles (claimed as objection to constituency):
  • Sometimes a Great Notion
  • The Fire Next Time
• Why might that be?