The Menu Bar

- Administrivia:
  - Schedule alert: Lab1b due today
  - Lab 2b released, this Weds (later today)

Agenda:
Red vs. Blue:
  - Part of speech ‘tagging’ via statistical models
  - Part of speech tagging via rules
  - Ch. 6 & 8 in Jurafsky
The Great Divide in NLP: the red pill or the blue pill?

“Knowledge Engineering” approach
Rules built by hand w/ K of Language
“Text understanding”

“Trainable Statistical” Approach
Rules inferred from lots of data (“corpora”)
“Information retrieval”

Two approaches

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

These two approaches are the guts of Lab 2 (lots of others methods: decision trees, ...)
The problem

- In unseen data, we wish to find the part of speech tags
- The set of part of speech tags are decided by experts

Noisy Channel Muddle (statistical)

- Real language $X$ to part-of-speech tags
- Noisy channel $X \rightarrow Y$
- Yucky language $Y$
- Want to recover $X$ from $Y$
A picture: the statistical, noisy channel view

Wreck a nice beach?    
Reckon eyes peach?    
Recognize speech?

x(speech) → Word model $P(x|T)$ → Bigram Tag model $P(T)$ → y(text) 

y(tags)

x(words)

Formulation, in general
General probabilistic decision problem

- E.g.: data = bunch of text
  - label = language
  - label = topic
  - label = author
- E.g. 2: (sequential prediction)
  - label = translation or summary of entire text
  - label = part of speech of current word
  - label = identity of current word (ASR) or character (OCR)

Language models – statistical view

- Application to speech recognition (and parsing, generally)
  - x = Input (speech/words)
  - y = output (text/Tags)
  - We want to find max P(y|x) Problem: we don’t know the tags – that is what we want to find!
  - Solution: We have an estimate of P(y) [the language model] and P(x|y) [the prob. of some sound/words given text/Tags = an acoustic model] or Tag model]
Finding inner form given outside:

From Bayes’ law, we have,
\[
\max P(y|x) = \max P(x|y) \cdot P(y) = \max \Pr \text{ acoustic model } \times \text{ lang model }
\]
(hold \( P(x) \) fixed, i.e., \( P(x|y) \cdot P(y) / P(x) \), but max is same for both)

So, in tagging case, we have a word model instead - so we find \( \max P(\text{tags}|w) \) from:
\[
\max P(\text{words}|\text{tags}) \cdot P(\text{tags})
\]

HMM for POS tagging

- In a Hidden Markov model, it is hypothesized that there is an underlying finite state machine (not directly observable, hence hidden) that changes state with each input element
- For us, the hidden states are the tags, and the input elements are the words
Hidden Markov Model tagging for POS

- Prob(Tag sequence) – based on n-grams: train on marked up, tagged text
- Prob(W|T) – unigram prob, based on tagged text
- Prob(T, w) computed from Viterbi trellis computation
  - max over all possible tag sequence paths, and ‘emission’ probabilities of word, tag combination
- Unseen tag sequence

Cartoon form Review

Tag sequence bigrams: P(T)

Unigram: p(W | T)

Score candidate tag seqs on their joint probability with observed words; we should pick best path

p(T, w)
**HMM construction**

- Hidden state transition model governs observed word sequences
- Transitions probabilistic
- Estimate transition probabilities from an annotated corpus state \( s' = \text{tag state} \)
  - \( P(s_j | s_{j-1}, w_j) \)
  - Based just on prior state and current word seen (hence Markovian assumption)
- At runtime, find maximum likelihood path through the network, using a max-flow algorithm (Viterbi)

---

**Cartoon form Review**

\[ P(T) \times p(W | T) \times p(w | W) = p(T, w) \]

Transducer: scores candidate tag seqs on their joint probability with obs words; we should pick best path
P(T) - Tag bigram picture

\[ p(\text{tag seq}) \]

BOS Det Adj Adj Noun EOS \[ = 0.8 \times 0.3 \times 0.4 \times 0.5 \times 0.2 \]

Unigram replacement model

\[ P(\text{word} | \text{tag}) \]

sums to 1

sums to 1
Compose with actual word seq

\[ p(\text{word seq, tag seq}) = p(\text{tag seq}) \times p(\text{word seq | tag seq}) \]

\[ 0.32 \times 0.0009 \times 0.0002 \times 0.00002 \times x.2 \approx 0.3 \times 10^{-6} \text{ total} \]

path prob, done!

Unroll the fsa - All paths together form ‘trellis’

The best path:

**BOS** Det Adj Adj Noun **EOS** = 0.32 * 0.0009 ...
the cool directed autos
Cross-product construction forms trellis

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 to find max probability path: Viterbi algorithm

- For each path reaching state s at step (word) w, we compute a path probability. We call the max of these $v_{s,w}$
- [Base step] Compute $v_{0,0}=1$
- [Induction step] Compute $v_{s',w+1}$, assuming we know $v_{s,w}$ for all s

Viterbi recursion

$$\text{path-prob}(s'|s,w) = \text{viterbi}(s,w) \cdot a[s,s']$$

$$\text{viterbi}(s',w+1) = \max_{s \in \text{STATES}} \text{path-prob}(s' | s,w)$$

Bi-gram POS probability
Method...

- This is almost correct...but again, we need to also factor in the unigram prob of a state s’ ‘emitting’ a word w given an observed surface word w, or b(o_w) at tag state s’
- So the correct formula for the path prob is:
  \[
  \text{path-prob}(s'|s,w) = \text{viterbi}(s,w) \times a[s,s'] \times b_s(o_w)
  \]

Or as in your text...p. 179 (NB: t used instead of w for index)

```python
function VITERBI(observations of len T, state-graph) returns best-path

    num-states ← NUM-OF-STATES(state-graph)
    Create a path probability matrix \( \text{viterbi}[\text{num-states}+2,T+2] \)
    viterbi[0,0] ← 1.0
    for each time step \( t \) from 0 to T do
        for each state s from 0 to num-states do
            for each transition \( s' \) from s specified by state-graph
                new-score ← viterbi[s, t] \times a[s,s'] \times b_s(o_t)
                if ((viterbi[s', t+1] = 0) || (new-score > viterbi[s', t+1]))
                    then
                        viterbi[s', t+1] ← new-score
                        back-pointer[s', t+1] ← s
                Backtrace from highest probability state in the final column of viterbi[] and return path
```
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?
- This is a “Hidden Markov Model”:

```
probs from tag bigram model
{ Start, PN, Verb, Det, Noun, Prep, Noun, Prep, Det, Noun, Stop }
```

```
probs from unigram replacement
```

```
Bill directed a cortege of autos through the dunes
```

- Find X that maximizes probability product

How much data is needed?
- System performance bears a roughly log-linear relationship to the training data quantity, at least up to about 1.2 million words
- Obtaining 1.2 million words of training data requires transcribing and annotating approximately 200 hours of broadcast news programming, or if annotating text, this would amount to approximately 1,777 average-length Wall Street Journal articles
If you think POS tagging is not relevant, then...

- Microsoft announced plans to include “Smart Tags” in its browser and other products. This is a feature that automatically inserts hyperlinks from concepts in text to related web pages chosen by Microsoft.
- The best way to make automatic hyperlinking unbiased is to base it on an unbiased source of web pages, such as Google.
How to do this?

- The main technical problem is to find pieces of text that are good concept anchors... like names!
- So: Given a text, find the starting and ending points of all the names. Depending on our specific goals, we can include the names of people, places, organizations, artifacts (such as product names), etc.
- Once we have the anchor text, we can send it to a search engine, retrieve a relevant URL (or set of URLs, once browsers can handle multi-way hyperlinks), and insert them into the original text on the fly.

Example – “Message understanding” (MUC)

SANTIAGO, 18 MAY 90 (RADIO COOPERATIVA NETWORK) — [REPORT] [JUAN ARAYA]

EDMUNDO VARGAS CARRENO, CHILEAN FOREIGN MINISTRY UNDER SECRETARY, HAS STATED THAT THE BRYANT TREATY WITH THE UNITED STATES WILL BE APPLIED IN THE LETELIER CASE ONLY TO COMPENSATE THE RELATIVES OF THE FORMER CHILEAN FOREIGN MINISTER MURDERED IN WASHINGTON AND THE RELATIVES OF HIS U.S. SECRETARY, RONNIE MOFFIT. THE CHILEAN FOREIGN UNDER SECRETARY MADE THIS STATEMENT IN REPLY TO U.S. NEWSPAPER REPORTS STATING THAT THE TREATY WOULD BE PARTIALLY RESPECTED.

FOLLOWING ARE VARGAS CARRENO’S STATEMENTS AT A NEWS CONFERENCE HE HELD IN BUENOS AIRES BEFORE CONCLUDING HIS OFFICIAL VISIT TO ARGENTINA:
<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MESSAGE: ID</td>
<td>TSTP-MUC3-0011</td>
</tr>
<tr>
<td>MESSAGE: TEMPLATE</td>
<td>1</td>
</tr>
<tr>
<td>INCIDENT: DATE</td>
<td>18 MAY 90</td>
</tr>
<tr>
<td>INCIDENT: LOCATION</td>
<td>UNITED STATES: WASHINGTON D.C. (CITY)</td>
</tr>
<tr>
<td>INCIDENT: TYPE</td>
<td>ATTACK</td>
</tr>
<tr>
<td>INCIDENT: STAGE OF EXECUTION</td>
<td>ACCOMPLISHED</td>
</tr>
<tr>
<td>INCIDENT: INSTRUMENT ID</td>
<td>-</td>
</tr>
<tr>
<td>INCIDENT: INSTRUMENT TYPE</td>
<td>STATE-SPONSORED VIOLENCE</td>
</tr>
<tr>
<td>PERP: INCIDENT CATEGORY</td>
<td>STATE-SPONSORED VIOLENCE</td>
</tr>
<tr>
<td>PERP: INDIVIDUAL ID</td>
<td>-</td>
</tr>
<tr>
<td>PERP: ORGANIZATION ID</td>
<td>&quot;CHILEAN GOVERNMENT&quot;</td>
</tr>
<tr>
<td>PERP: ORGANIZATION CONFIDENCE REPORTED AS FACT</td>
<td>&quot;CHILEAN GOVERNMENT&quot;</td>
</tr>
<tr>
<td>PHYS TGT: ID</td>
<td>-</td>
</tr>
<tr>
<td>PHYS TGT: TYPE</td>
<td>-</td>
</tr>
<tr>
<td>PHYS TGT: NUMBER</td>
<td>-</td>
</tr>
<tr>
<td>PHYS TGT: FOREIGN NATION</td>
<td>-</td>
</tr>
<tr>
<td>PHYS TGT: EFFECT OF INCIDENT</td>
<td>-</td>
</tr>
<tr>
<td>PHYS TGT: TOTAL NUMBER</td>
<td>-</td>
</tr>
<tr>
<td>HUM TGT: NAME</td>
<td>&quot;ORLANDO LETELIER&quot;</td>
</tr>
<tr>
<td>HUM TGT: DESCRIPTION</td>
<td>&quot;FORMER CHILEAN FOREIGN MINISTER&quot;: &quot;ORLANDO LETELIER&quot;</td>
</tr>
<tr>
<td>HUM TGT: TYPE</td>
<td>GOVERNMENT OFFICIAL: &quot;ORLANDO LETELIER&quot;</td>
</tr>
<tr>
<td>HUM TGT: TYPE</td>
<td>CIVILIAN: &quot;RONNIE MOFFIT&quot;</td>
</tr>
</tbody>
</table>

Recognizing domain patterns

- Match domain patterns against complex phrase heads
  - <company> <form><joint venture> with <company>
  - <company> capitalized at <currency>

"Bridgestone Sports Co. said Friday it has set up a joint venture in Taiwan with a local concern and a Japanese trading house to produce golf clubs to be shipped to Japan. "The joint venture, Bridgestone Sports Taiwan Co., capitalized at 20 million new Taiwan dollars, will start production in January 1990."

| Relationship: TIE-UP | Entities: "Bridgestone Sports Co."
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>JV Company:</td>
<td>&quot;a local concern&quot;</td>
</tr>
<tr>
<td>Capitalization:</td>
<td>&quot;a Japanese trading house&quot;</td>
</tr>
</tbody>
</table>

| Relationship: TIE-UP | Entities: "Bridgestone Sports Taiwan Co."
|----------------------|-------------------------------------------------|
| JV Company:          | "Bridgestone Sports Taiwan Co."
| Capitalization:      | "Capitalized: 20000000 TWD"                     |
What about part of speech tagging here?

- **Advantages**
  - Ambiguity can be potentially reduced (but we shall see in our laboratory if this is true)
  - Avoid errors due to incorrect categorization of rare senses e.g., “has been” as noun
- **Disadvantages**
  - Errors taggers make often those you’d most want to eliminate
  - High performance requires training on similar genre
  - Training takes time

Proper names...

- Proper names are particularly important for extraction systems
- Because typically one wants to extract events, properties, and relations about some particular object, and that object is usually identified by its name
...A challenge...

- Problems though...
  - proper names are huge classes and it is difficult, if not impossible to enumerate their members
  - Hundreds of thousands of names of locations around the world
  - Many of these names are in languages other than the one in which the extraction system is designed

How are names extracted?

- (Hidden) Markov Model
- Hypothesized that there is an underlying finite state machine (not directly observable, hence hidden) that changes state with each input element
- probability of a recognized constituent is conditioned not only on the words seen, but the state that the machine is in at that moment
- “John” followed by “Smith” is likely to be a person, while “John” followed by “Deere” is likely to be a company (a manufacturer of heavy farming and construction equipment).
HMM statistical name tagger

Method 2: Rule system (but learned)
- Error-based tagging
Another FST Paradigm: Successive Fixups

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

Brown/Upenn corpus tags

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC</td>
<td>Coord. 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>JJT</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>PPS</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>
<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, non-3sg pres</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>WPS</td>
<td>Possessive wh-</td>
<td>whose</td>
</tr>
<tr>
<td>WRB</td>
<td>Wh-adverb</td>
<td>how, where</td>
</tr>
<tr>
<td>S</td>
<td>Dollar sign</td>
<td>$</td>
</tr>
<tr>
<td>#</td>
<td>Pound sign</td>
<td>#</td>
</tr>
<tr>
<td>&quot;</td>
<td>Left quote</td>
<td>(“ or “&quot;)</td>
</tr>
<tr>
<td>'</td>
<td>Right quote</td>
<td>(’ or ’&quot;)</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 punct</td>
<td>( . ! ? )</td>
</tr>
<tr>
<td>;</td>
<td>Mid-sentence punct</td>
<td>( ; ... - )</td>
</tr>
</tbody>
</table>
Transformation-Based Tagging

(Brill 1995)

Transformations Learned

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

BaselineTag*

NN → VB // TO _
VBP → VB // ... _

etc.

Compose this cascade of FSTs.

Get a big FST that does the initial tagging and the sequence of fixups “all at once.”
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 $s$ 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>

Laboratory 2

- **Goals:**
  1. Use both HMM and Brill taggers
  2. Find errors that both make
  3. Compare performance – use of kappa & ‘confusion matrix’
  4. All the slings & arrows of corpora – use Wall Street Journal excerpts
Brill Tagger

Powered by μ TRL Technology

"Secretary is expected to race tomorrow"

Text:

- **Text:**
  - *Secretary is expected to race tomorrow*

- **Pos:**
  - *Secretary: noun*  
  - *expected: verb*  
  - *to: preposition*  
  - *race: noun*  
  - *tomorrow: noun*  

---

**Tokenization**

*Secretary is expected to race tomorrow*

**Lexical Lookup**

*Secretary/NOM ia/VBD expected/VBD to/TO race/VBN tomorrow/VBD*

**Pos:**

- *Secretary: noun*  
- *expected: verb*  
- *race: noun*  
- *tomorrow: noun*

**Guessing**

**Contextual-rule application**

Intermediate analysis:

*Secretary/NOM ia/VBD expected/VBD to/TO race/VBN tomorrow/VBD*

Applied rule:

*tag*HRVVB <- *tag*TOB[-1] .

**Analysis**

*Secretary/NOM ia/VBD expected/VBD to/TO race/VBN tomorrow/VBD*
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

Contextual Rules

- Change tag a to tag b when:
  1. The preceding (following) word is tagged z.
  2. The word two before (after) is tagged z.
  3. One of the two preceding (following) words is tagged z.
  4. One of the three preceding (following) words if tagged z.
  5. The preceding word is tagged z and the following word is tagged w.
  6. The preceding (following) word is tagged z and the word two before (after) is tagged w.
  7. The preceding (following) word x.
  ...
Lexical Rules

*Change the tag of an unknown word (from X) to Y if:*

1. Deleting the prefix (suffix) $x$, $|x| \leq 4$, results in a word ($x$ is any string of length 1 to 4).
2. The first (last) (1, 2, 3, 4) characters of the word are $x$.
3. Adding the character string $x$ as a prefix (suffix) results in a word ($|x| = \leq 4$).
4. Word $w$ ever appears immediately to the left (right) of the word.
5. Character $z$ appears in the word.

Example Lexical Rules

NN s fhassuf 1 NNS

change the tag of an unknown word from NN to NNS if it has suffix -s

\[
\text{webpages/NN} \rightarrow \text{webpages/NNS}
\]
Example 2

NN - fchar JJ
change the tag of an unknown word from NN to JJ if it has character '-'
*man-made, rule-based, three-year-old, etc.*

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 NN to VB when the previous tag is 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 -s</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 res</td>
</tr>
<tr>
<td>8</td>
<td>NN</td>
<td>CD</td>
<td>The word &amp; 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 -es</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>

OK, the proof is in the (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
  *Microsoft Windoze* & 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
Examples of Contextual Rules

- NN VB PREVTAG TO
  - change tag NN to tag VB when the preceding word is tagged TO
    to/TO run/NN
    would be changed to
    to/TO run/VB
- VBP VB PREV1OR2OR3TAG MD
- Change tag VBP(verb, non-3rd person singular present) to VB(verb, base form) when one of the three preceding words is tagged MD (modal verb)

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
   end
   best-Z ← ARGMAX_i(num-good-transforms(i) - num-bad-transforms(i))
   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”
   end
   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 TO</td>
</tr>
<tr>
<td>2</td>
<td>VBP</td>
<td>VB</td>
<td>One of the previous three tags is MD</td>
</tr>
<tr>
<td>3</td>
<td>NN</td>
<td>VB</td>
<td>One of the previous two tags is MD</td>
</tr>
<tr>
<td>4</td>
<td>VB</td>
<td>NN</td>
<td>One of the previous two tags is DT</td>
</tr>
<tr>
<td>5</td>
<td>VBD</td>
<td>VBN</td>
<td>One of the previous three tags is VBZ</td>
</tr>
<tr>
<td>6</td>
<td>VBN</td>
<td>VBD</td>
<td>Previous tag is NNP</td>
</tr>
<tr>
<td>7</td>
<td>VBN</td>
<td>VBD</td>
<td>Previous tag is NNP</td>
</tr>
<tr>
<td>8</td>
<td>VBD</td>
<td>VBN</td>
<td>Previous tag is VBD</td>
</tr>
<tr>
<td>9</td>
<td>VBP</td>
<td>VB</td>
<td>Previous tag is TO</td>
</tr>
<tr>
<td>10</td>
<td>POS</td>
<td>VB</td>
<td>Previous tag is NNP</td>
</tr>
<tr>
<td>11</td>
<td>VB</td>
<td>VBP</td>
<td>Previous tag is NNP</td>
</tr>
<tr>
<td>12</td>
<td>VBD</td>
<td>VBN</td>
<td>One of the previous three tags is VBP</td>
</tr>
<tr>
<td>13</td>
<td>IN</td>
<td>WDT</td>
<td>One of next two tags is VB</td>
</tr>
<tr>
<td>14</td>
<td>VBD</td>
<td>VBN</td>
<td>One of previous two tags is VB</td>
</tr>
<tr>
<td>15</td>
<td>VB</td>
<td>VBP</td>
<td>Previous tag is NNP</td>
</tr>
<tr>
<td>16</td>
<td>IN</td>
<td>WDT</td>
<td>Next tag is VBZ</td>
</tr>
<tr>
<td>17</td>
<td>IN</td>
<td>DT</td>
<td>Next tag is JJ</td>
</tr>
<tr>
<td>18</td>
<td>JJ</td>
<td>NNP</td>
<td>Next tag is JJ</td>
</tr>
<tr>
<td>19</td>
<td>IN</td>
<td>WDT</td>
<td>Next tag is VBD</td>
</tr>
<tr>
<td>20</td>
<td>JJR</td>
<td>RBR</td>
<td>Next tag is JJ</td>
</tr>
</tbody>
</table>

figure from Brill’s thesis

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

- Все счастливые семьи похожи друг на друга, каждая ненасчастливая семья ненасчастлива по-своему
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
Evaluation of systems

- **Recall** is the number of answers the system got right divided by the number of possible right answers
  - It measures how complete or comprehensive the system is in its extraction of relevant information

- **Precision** is the number of answers the system got right divided by the number of answers the system gave
  - It measures the system's correctness or accuracy
  - Example: there are 100 possible answers and the system gives 80 answers and gets 60 of them right, its recall is 60% and its precision is 75%.
A better measure - Kappa

- Takes baseline & complexity of task into account – if 99% of tags are Nouns, getting 99% correct no great shakes
- Suppose no “Gold Standard” to compare against?
  - $P(A)$ = proportion of times hypothesis *agrees* with standard (% correct)
  - $P(E)$ = proportion of times hypothesis and standard would be *expected* to agree by chance (computed from some other knowledge, or actual data)

Kappa [p. 315 J&M text]

- Note $K$ ranges between 0 (no agreement, except by chance; to complete agreement, 1)
- Can be used even if no ‘Gold standard’ that everyone agrees on
  - $K > 0.8$ is good
Kappa

- $A =$ actual agreement; $E =$ expected agreement
- consistency is more important than “truth”
- methods for raising consistency
  - style guides (often have useful insights into data)
  - group by task, not chronologically, etc.
  - annotator acclimatization

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!
Is that all there is?

What have we done so far?

- Only information we represent: is whether an item precedes (or follows) another
- Inventory of vocabulary items (classes)
  - = Finite state machines

- Is there anything else in language???
Motivation

- What, How, and Why
- **What**: word *chunks* behave as units, like words or endings (morphemes), like *ing*
- **How**: we have to recover these from input
- **Why**: chunks used to discover *meaning*
- Parsing: mapping from *strings* to *structured representation*

Programming languages

```c
printf ("/charset \[%s",
    (re_opcode_t) *(p - 1) == charset_not ? "^^" : "");
assert (p + *p < pend);
for (c = 0; c < 256; c++)
    if (c / 8 < *p && (p[1 + (c/8)] & (1 << (c % 8)))) {
        /* Are we starting a range? */
        if (last + 1 == c && ! inrange) {
            putchar ('-');
            inrange = 1;
        } /* Have we broken a range? */
        else if (last + 1 != c && inrange) {
            putchar (last);
            inrange = 0;
        }
    }
    if (! inrange)
        putchar (c);
    last = c;
```

- Easy to parse.
- Designed that way!
Natural languages

- No `()` `[]` to indicate scope & precedence
- Lots of overloading (arity varies)
- Grammar isn’t known in advance!
- What is the best formalism?

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)