Part-of-Speech Tagging

A Canonical Finite-State Task
The Tagging Task

Input: the lead paint is unsafe
Output: the/Det lead/N paint/N is/V unsafe/Adj

- Uses:
  - text-to-speech (how do we pronounce “lead”?)
  - can write regexps like (Det) Adj* N+ over the output
  - preprocessing to speed up parser (but a little dangerous)
  - if you know the tag, you can back off to it in other tasks
Why Do We Care?

The first statistical NLP task

Been done to death by different methods

Easy to evaluate (how many tags are correct?)

Canonical finite-state task

- Can be done well with methods that look at local context
- Though should “really” do it by parsing!

Input: the lead paint is unsafe
Output: the/Det lead/N paint/N is/V unsafe/Adj
Degree of Supervision

- **Supervised**: Training corpus is tagged by humans
- **Unsupervised**: Training corpus isn’t tagged
- **Partly supervised**: Training corpus isn’t tagged, but you have a dictionary giving possible tags for each word

- We’ll start with the supervised case and move to decreasing levels of supervision.
Current Performance

Input: the lead paint is unsafe
Output: the/Det lead/N paint/N is/V unsafe/Adj

- How many tags are correct?
  - About 97% currently
  - But baseline is already 90%
    - Baseline is performance of stupidest possible method
    - Tag every word with its most frequent tag
    - Tag unknown words as nouns
What Should We Look At?

**correct tags**

Bill directed a cortege of autos through the dunes

Each unknown tag is **constrained** by its word and by the tags to its immediate left and right. But those tags are unknown too ...
What Should We Look At?

correct tags

PN  Verb  Det  Noun  Prep  Noun  Prep  Det  Noun
Bill  directed  a  cortege  of  autos  through  the  dunes

PN  Adj  Det  Noun  Prep  Noun  Prep  Det  Noun
Verb  Verb  Noun  Verb
Adj
Prep
...

some possible tags for each word (maybe more)

Each unknown tag is constrained by its word and by the tags to its immediate left and right. But those tags are unknown too ...
What Should We Look At?

**correct tags**

PN   Verb   Det   Noun   Prep   Noun   Prep   Det   Noun
Bill directed a cortege of autos through the dunes

PN   Adj   Det   Noun   Prep   Noun   Prep   Det   Noun
Adj
Verb   Verb   Noun   Verb
Prep
...

Each unknown tag is constrained by its word and by the tags to its immediate left and right. But those tags are unknown too ...
Three Finite-State Approaches

- Noisy Channel Model (statistical)

\[
\text{real language } X \xrightarrow{\text{noisy channel}} Y \xrightarrow{\text{part-of-speech tags}} \text{yucky language } Y
\]

want to recover X from Y

replace tags with words

part-of-speech tags (n-gram model)

text
Three Finite-State Approaches

1. Noisy Channel Model (statistical)

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

3. Nondeterministic tagger composed with a cascade of finite-state automata that act as filters
Review: Noisy Channel

real language $X$ \[\rightarrow\] noisy channel $X \rightarrow Y$ \[\rightarrow\] yucky language $Y$

$p(X)$  
$p(Y | X)$  
$= p(X,Y)$

want to recover $x \in X$ from $y \in Y$  
choose $x$ that maximizes $p(x | y)$ or equivalently $p(x,y)$
Review: Noisy Channel

\[ p(X) \]
\[ \ast \]
\[ p(Y \mid X) \]
\[ = \]
\[ p(X,Y) \]

Note \( p(x,y) \) sums to 1.

Suppose \( y=\text{“C”} \); what is best “x”?
Review: Noisy Channel

Suppose y=“C”; what is best “x”?

\[
p(X) \ast p(Y \mid X) = p(X,Y)
\]
Review: Noisy Channel

\[ p(X) \]
\[ \ast \]
\[ p(Y \mid X) \]
\[ \ast \]
\[ (Y = y)? \]
\[ = \]
\[ p(X, y) \]

restrict just to paths compatible with output “C”
Noisy Channel for Tagging

acceptor: \( p(\text{tag sequence}) \) 

“Markov Model”

\( \ast \)

transducer: tags \( \rightarrow \) words

“Unigram Replacement”

\( \ast \)

acceptor: the observed words

“straight line”

transducer: scores candidate tag seqs on their joint probability with obs words; pick best path

\[ p(X) \ast \]

\[ p(Y \mid X) \ast \]

\[ (Y = y)? \]

\[ p(X, y) \]
Markov Model (bigrams)
Markov Model

Start

Det

0.3

0.7

Adj

0.4

0.5

Noun

Verb

Prep

Stop

0.1
Markov Model

Start → Det (0.8) → Verb → Prep → Stop

Start → Adj (0.3) → Verb → Prep → Stop

Start → Noun (0.7) → Verb → Prep → Stop

Start → Noun (0.4) → Verb → Prep → Stop

Start → Noun (0.5) → Verb → Prep → Stop

Start → Noun (0.1) → Verb → Prep → Stop

0.8
0.3
0.7
0.4
0.5
0.1
0.2
Markov Model

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

\[
\text{Start} \xrightarrow{0.8} \text{Det} \xrightarrow{0.3} \text{Adj} \xrightarrow{0.4} \text{Stop} = 0.8 \times 0.3 \times 0.4 \times 0.5 \times 0.2
\]
Markov Model as an FSA

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

\[ \text{Start} \quad \text{Det} \quad \text{Adj} \quad \text{Adj} \quad \text{Noun} \; \text{Stop} = 0.8 \times 0.3 \times 0.4 \times 0.5 \times 0.2 \]
Markov Model as an FSA

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

\[
\text{Start} \xrightarrow{\text{Det} \ 0.8} \text{Det} \xrightarrow{\text{Adj} \ 0.3} \text{Adj} \xrightarrow{\text{Adj} \ 0.4} \text{Adj} \xrightarrow{\text{Noun} \ 0.5} \text{Noun} \xrightarrow{\varepsilon \ 0.1} \text{Det} \xrightarrow{\varepsilon \ 0.2} \text{Stop}
\]

\[
\text{Start} \xrightarrow{\text{Det Adj Adj Noun Stop}} = 0.8 \times 0.3 \times 0.4 \times 0.5 \times 0.2
\]
Markov Model (tag bigrams)

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

\[ \text{Start} \xrightarrow{\text{Det} 0.8} \text{Det} \xrightarrow{\text{Adj} 0.3} \text{Adj} \xrightarrow{\text{Adj} 0.4} \text{Adj} \xrightarrow{\text{Noun} 0.5} \text{Noun} \xrightarrow{\epsilon 0.2} \text{Stop} \]

\[ \text{Start Det Adj Adj Noun Stop} = 0.8 \times 0.3 \times 0.4 \times 0.5 \times 0.2 \]
Noisy Channel for Tagging

automaton: \( p(\text{tag sequence}) \)

“Markov Model”

\[ p(X) \]

\[ \ast \]

transducer: tags \( \rightarrow \) words

“Unigram Replacement”

\[ p(Y \mid X) \]

\[ \ast \]

automaton: the observed words

“straight line”

\[ p(y \mid Y) \]

\[ = \]

transducer: scores candidate tag seqs on their joint probability with obs words;

pick best path

\[ p(X, y) \]
Noisy Channel for Tagging

transducer: scores candidate tag seqs on their joint probability with obs words; we should pick best path
Unigram Replacement Model

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

... \[ \text{Noun: cortege/0.000001} \]

Noun: autos/0.001

Noun: Bill/0.002

... \[ \text{Adj: cool/0.003} \]

Adj: directed/0.0005

Adj: cortege/0.000001

... sums to 1

Det: a/0.6

Det: the/0.4

sums to 1
Compose

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

\[ p(\text{word seq, tag seq}) = p(\text{tag seq}) \times p(\text{word seq | tag seq}) \]
Observed Words as Straight-Line FSA

word seq

the → cool → directed → autos →
Compose with

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

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

Start Det Adj Adj Noun Stop = 0.32 * 0.0009 ...
the cool directed autos

\[ p(\text{word seq, tag seq}) = p(\text{tag seq}) \times p(\text{word seq | tag seq}) \]
In Fact, Paths Form a “Trellis”

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

The best path:

**Start** Det Adj Adj Noun **Stop** = 0.32 * 0.0009 ...

the cool directed autos
The Trellis Shape Emerges from the Cross-Product Construction for Finite-State Composition

All paths here are 4 words

So all paths here must have 4 words on output side
Actually, Trellis Isn’t Complete

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

Trellis has no Det \rightarrow Det or Det \rightarrow \text{Stop} arcs; why?

The best path:

\[ \text{Start} \quad \text{Det} \quad \text{Adj} \quad \text{Adj} \quad \text{Noun} \quad \text{Stop} = 0.32 \times 0.0009 \ldots \]

the cool directed autos
Actually, Trellis Isn’t Complete

p(word seq, tag seq)

Lattice is missing some other arcs; why?

The best path:

Start Det Adj Adj Noun Stop = 0.32 * 0.0009 ...

the cool directed autos
Actually, Trellis Isn’t Complete

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

Lattice is missing some states; why?

The best path:

\[
\begin{align*}
\text{Start} & \quad \text{Det:the} \quad 0.32 \quad \text{Det} \quad \text{Adj:cool} \quad 0.0009 \\
& \quad \text{Noun:cool} \quad 0.007 \\
& \quad \text{Adj:directed} \\
& \quad \text{Noun} \rightarrow \text{Noun} \rightarrow \text{Noun} \\
& \quad \epsilon \quad 0.2 \\
& \quad \text{Stop}
\end{align*}
\]

\[ \text{the cool directed autos} = 0.32 \times 0.0009 \ldots \]
Find best path from Start to Stop

- Use dynamic programming – like prob. parsing:
  - 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 alg.
- Faster if some arcs/states are absent
In Summary

- We are modeling $p(\text{word seq}, \text{tag seq})$
- 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”:

```
Start
PN   Verb    Det     Noun  Prep Noun   Prep     Det  Noun
Bill  directed   a    cortege  of   autos  through  the  dunes
```

- Find $X$ that maximizes probability $\text{product}$
Another Viewpoint

- We are modeling $p(\text{word seq}, \text{tag seq})$
- Why not use chain rule + some kind of backoff?
- Actually, we are!

$$p(\text{Start PN Verb Det ... Bill directed a ...})$$

$$= p(\text{Start}) \cdot p(\text{PN | Start}) \cdot p(\text{Verb | Start PN}) \cdot p(\text{Det | Start PN Verb}) \cdot ...$$

$$\cdot p(\text{Bill | Start PN Verb ...}) \cdot p(\text{directed | Bill, Start PN Verb Det ...})$$

$$\cdot p(\text{a | Bill directed, Start PN Verb Det ...}) \cdot ...$$
Another Viewpoint

- We are modeling $p(\text{word seq, tag seq})$
- Why not use chain rule + some kind of backoff?
- Actually, we are!

$$p(\text{Start PN Verb Det ...}) = p(\text{Start}) \times p(\text{PN | Start}) \times p(\text{Verb | Start PN}) \times p(\text{Det | Start PN Verb}) \times ... \times p(\text{Bill | Start PN Verb ...}) \times p(\text{directed | Bill, Start PN Verb Det ...}) \times p(\text{a | Bill directed, Start PN Verb Det ...}) \times ...$$

Start PN Verb Det Noun Prep Noun Prep Det Noun Stop
Bill directed a cortege of autos through the dunes
Three Finite-State Approaches

1. Noisy Channel Model (statistical)

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

3. Nondeterministic tagger composed with a cascade of finite-state automata that act as filters
Another FST Paradigm: Successive Fixups

- Like successive markups but *alter*
- Morphology
- Phonology
- Part-of-speech tagging
- ...
Transformation-Based Tagging
(Brill 1995)

Unannotated Corpus

Annotated Corpus
Errors = 5,100

Annotated Corpus
Errors = 3,145

Annotated Corpus
Errors = 3,910

Annotated Corpus
Errors = 1,231

Annotated Corpus
Errors = 6,300

Annotated Corpus
Errors = 4,255

Annotated Corpus
Errors = 1,231

Annotated Corpus
Errors = 1,251

Annotated Corpus
Errors = 1,231

Annotated Corpus
Errors = 1,410

Initial State Annotator

figure from Brill’s thesis
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 PRP</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>VBZ</td>
<td>Previous tag is PRP</td>
</tr>
<tr>
<td>11</td>
<td>VB</td>
<td>VBP</td>
<td>Previous tag is NNS</td>
</tr>
<tr>
<td>12</td>
<td>VBD</td>
<td>VBN</td>
<td>One of 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 PRP</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 VBZ</td>
</tr>
<tr>
<td>18</td>
<td>JJ</td>
<td>NNP</td>
<td>Next tag is NNP</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>

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

Compose this cascade of FSTs.

Gets 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 $ 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>

*Figure from Brill’s thesis*
Three Finite-State Approaches

1. Noisy Channel Model (statistical)

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

3. Nondeterministic tagger composed with a cascade of finite-state automata that act as filters
Variations

- Multiple tags per word
  - Transformations to knock some of them out
- \textit{How to encode multiple tags and knockouts?}

- Use the above for \textit{partly supervised} learning
  - \textbf{Supervised:} You have a tagged training corpus
  - \textbf{Unsupervised:} You have an untagged training corpus
  - \textbf{Here:} You have an untagged training corpus and a dictionary giving possible tags for each word