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

*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*

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

Bill directed a cortege of autos through the dunes

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 ...
Three Finite-State Approaches

- Noisy Channel Model (statistical)

\[ \text{real language } Y \rightarrow \text{noisy channel } Y \rightarrow X \rightarrow \text{observed string } X \]

- want to recover \( Y \) from \( X \)

- part-of-speech tags (n-gram model)
- replace tags with words
- 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 $Y$
- noisy channel $Y \rightarrow X$
- observed string $X$

$p(Y)$

$\ast$

$p(X \mid Y)$

$=$

$p(X,Y)$

want to recover $y \in Y$ from $x \in X$

choose $y$ that maximizes $p(y \mid x)$ or equivalently $p(x,y)$
Review: Noisy Channel

Note $p(x,y)$ sums to 1. Suppose $x=\text{“C”};$ what is best “$y$”? 

$p(Y) \ast \quad p(X \mid Y) \quad = \quad p(X,Y)$
Review: Noisy Channel

Suppose $x = \text{“C”}$; what is best $\text{“y”}$?

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

\[ p(Y) \]

\[ p(X \mid Y) \]

\[ (X = x) \]

\[ = \]

\[ p(x, Y) \]

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

acceptor: \( p(\text{tag sequence}) \)  
“Markov Model”  

transducer: tags \( \rightarrow \) words  
“Unigram Replacement”  

acceptor: the observed words  
“straight line”  

transducer: scores candidate tag seqs on their joint probability with obs words; pick best path
Markov Model (bigrams)
Markov Model

Start

- Det (0.3)
- Adj (0.4)

Verb

Prep

Stop

Noun

- Verb (0.7)
- Prep (0.5)
- Noun (0.1)
Markov Model

- Start
- Det
- Adj
- Verb
- Prep
- Noun
- Stop

Transition probabilities:
- Start to Det: 0.8
- Start to Adj: 0.3
- Start to Verb: 0.7
- Det to Verb: 0.4
- Det to Prep: 0.5
- Adj to Noun: 0.5
- Verb to Prep: 0.2
- Prep to Noun: 0.1
- Noun to Stop: 0.2
Markov Model

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

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

$p(\text{tag seq})$

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

$p(\text{tag seq})$

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

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

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

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

"Markov Model"

\[ p(Y) \cdot * \]

**transducer:** tags \( \rightarrow \) words

"Unigram Replacement"

\[ p(X \mid Y) \cdot * \]

**automaton:** the observed words

"straight line"

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

**transducer:** scores candidate tag seqs on their joint probability with obs words; pick best path
Noisy Channel for Tagging

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

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

```
Noun: Bill/0.002
Noun: autos/0.001
Noun: cortege/0.000001
Adj: cool/0.003
Adj: directed/0.0005
Adj: cortege/0.000001
...
Det: the/0.4
Det: a/0.6
...
```

sums to 1

sums to 1
Compose

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

word seq

the → cool → directed → autos
p(word seq, tag seq) = p(tag seq) * p(word seq | tag seq)

Det:a 0.48
Det:the 0.32

Adj:cool 0.0009
Adj:directed 0.00015
Adj:cortege 0.000003

Verb

Prep

N:cortege
N:autos

Start

Compose with

the cool directed autos

Adj:cool 0.0012
Adj:directed 0.00020
Adj:cortege 0.000004
p(word seq, tag seq) = p(tag seq) * p(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 \times 0.0009 \ldots

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

$p(\text{word seq, tag seq})$

Trellis has no Det $\rightarrow$ Det or Det $\rightarrow$ Stop arcs; why?

The best path:

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

the cool directed autos
Actually, Trellis Isn’t Complete

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

Lattice is missing some other arcs; why?

The best path:

\bf{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

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

Lattice is missing some states; why?

The best path:

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

the cool directed autos
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 $p(\text{word seq, tag seq})$
- 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
Another Viewpoint

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

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

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

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

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

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

$$p(\quad ) = p(\text{Start}) \times p(\text{PN} \mid \text{Start}) \times p(\text{Verb} \mid \text{Start PN}) \times p(\text{Det} \mid \text{Start PN Verb}) \times \ldots$$
$$\times p(\text{Bill} \mid \text{Start PN Verb} \ldots) \times p(\text{directed} \mid \text{Bill, Start PN Verb Det} \ldots)$$
$$\times p(\text{a} \mid \text{Bill directed, Start PN Verb Det} \ldots) \times \ldots$$

Start PN Verb Det ... 
Bill directed a ...

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 \textit{alter}
- Morphology
- Phonology
- Part-of-speech tagging
- ...

\textbf{Initial annotation} \quad \textbf{Fixup 1} \quad \textbf{Fixup 2} \quad \textbf{Fixup 3}
Transformation-Based Tagging
(Brill 1995)

figure from Brill’s thesis

600.465 - Intro to NLP - J. Eisner
## 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 <code>TO</code></td>
</tr>
<tr>
<td>2</td>
<td>VBP</td>
<td>VB</td>
<td>One of the previous three tags is <code>MD</code></td>
</tr>
<tr>
<td>3</td>
<td>NN</td>
<td>VB</td>
<td>One of the previous two tags is <code>MD</code></td>
</tr>
<tr>
<td>4</td>
<td>VB</td>
<td>NN</td>
<td>One of the previous two tags is <code>DT</code></td>
</tr>
<tr>
<td>5</td>
<td>VBD</td>
<td>VBN</td>
<td>One of the previous three tags is <code>VBZ</code></td>
</tr>
<tr>
<td>6</td>
<td>VBN</td>
<td>VBD</td>
<td>Previous tag is <code>PRP</code></td>
</tr>
<tr>
<td>7</td>
<td>VBN</td>
<td>VBD</td>
<td>Previous tag is <code>NNP</code></td>
</tr>
<tr>
<td>8</td>
<td>VBD</td>
<td>VBN</td>
<td>Previous tag is <code>VBD</code></td>
</tr>
<tr>
<td>9</td>
<td>VBP</td>
<td>VB</td>
<td>Previous tag is <code>TO</code></td>
</tr>
<tr>
<td>10</td>
<td>POS</td>
<td>VBZ</td>
<td>Previous tag is <code>PRP</code></td>
</tr>
<tr>
<td>11</td>
<td>VB</td>
<td>VBP</td>
<td>Previous tag is <code>NNS</code></td>
</tr>
<tr>
<td>12</td>
<td>VBD</td>
<td>VBN</td>
<td>One of previous three tags is <code>VBP</code></td>
</tr>
<tr>
<td>13</td>
<td>IN</td>
<td>WDT</td>
<td>One of next two tags is <code>VB</code></td>
</tr>
<tr>
<td>14</td>
<td>VBD</td>
<td>VBN</td>
<td>One of previous two tags is <code>VB</code></td>
</tr>
<tr>
<td>15</td>
<td>VB</td>
<td>VBP</td>
<td>Previous tag is <code>PRP</code></td>
</tr>
<tr>
<td>16</td>
<td>IN</td>
<td>WDT</td>
<td>Next tag is <code>VBZ</code></td>
</tr>
<tr>
<td>17</td>
<td>IN</td>
<td>DT</td>
<td>Next tag is <code>NN</code></td>
</tr>
<tr>
<td>18</td>
<td>JJ</td>
<td>NNP</td>
<td>Next tag is <code>NNP</code></td>
</tr>
<tr>
<td>19</td>
<td>IN</td>
<td>WDT</td>
<td>Next tag is <code>VBD</code></td>
</tr>
<tr>
<td>20</td>
<td>JJR</td>
<td>RBR</td>
<td>Next tag is <code>JJ</code></td>
</tr>
</tbody>
</table>

BaselineTag*

- `NN @→ VB // TO _`
- `VBP @→ VB // ... _ _ _`

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>
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
- *How to encode multiple tags and knockouts?*

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