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?

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

- 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!
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 (for English)
  - 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

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 ...

600.465 - Intro to NLP - J. Eisner
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 ...
Three Finite-State Approaches

- Noisy Channel Model (statistical)

  real language $Y$

  noisy channel $Y \rightarrow X$

  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** $X$

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

Choose $y$ that maximizes $p(y \mid x)$ or equivalently $p(x,y)$.

Mathematical expressions:

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

Note $p(x,y)$ sums to 1.
Suppose $x=“C”$; what is best “y”?
Review: Noisy Channel

\[ p(Y) \ast \]

\[ p(X \mid Y) \]

\[ = \]

\[ p(X, Y) \]

Suppose x = “C”; what is best “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"

\( p(Y) \)

\( * \)

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

"Unigram Replacement"

\( p(X | Y) \)

\( * \)

**acceptor:** the observed words

"straight line"

\( (X = x)? \)

\( = \)

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

\( p(x, Y) \)
Markov Model (bigrams)
Markov Model

Start

Det

Adj

Noun

Verb

Prep

Stop

Probabilities:

- Start to Det: 0.3
- Start to Adj: 0.7
- Det to Adj: 0.4
- Adj to Noun: 0.5
- Noun to Verb: 0.1
Markov Model

Start

Det

Verb

Prep

Noun

Adj

Stop

0.8

0.3

0.7

0.4

0.5

0.1

0.2
Markov Model

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

\[ \text{Start} \rightarrow \text{Det} \rightarrow \text{Adj} \rightarrow \text{Noun} \rightarrow \text{Verb} \rightarrow \text{Prep} \rightarrow \text{Stop} \]

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

$p(\text{tag seq})$

$p(\text{Start Det Adj Adj Noun Stop}) = 0.8 \times 0.3 \times 0.4 \times 0.5 \times 0.2$
Markov Model as an FSA

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

\[
\text{Start} \xrightarrow{\text{Det} 0.8} \xrightarrow{\text{Adj} 0.3} \xrightarrow{\text{Adj} 0.4} \text{Adj} \xrightarrow{\text{Noun} 0.5} \text{Noun} \xrightarrow{\text{Noun} 0.7} \text{Verb} \xrightarrow{\epsilon 0.2} \text{Stop} \]

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

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

\[ \text{Start} \rightarrow \text{Det} \rightarrow \text{Adj} \rightarrow \text{Adj} \rightarrow \text{Noun} \rightarrow \text{Stop} \]

\[ p(\text{tag seq}) = 0.8 \times 0.3 \times 0.4 \times 0.5 \times 0.2 \]
Noisy Channel for Tagging

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

"Markov Model"

\[ p(Y) \]

\[ \ast \]

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

"Unigram Replacement"

\[ p(X \mid Y) \]

\[ \ast \]

**automaton:** the observed words

"straight line"

\[ p(x \mid X) \]

\[ = \]

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

\[ p(x, Y) \]
Noisy Channel for Tagging

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

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

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

Diagram:

- Det: the / 0.4
- Det: a / 0.6
- Noun: Bill / 0.002
- Adjective: cool / 0.003
- Adjective: directed / 0.0005
- Adjective: cortege / 0.000001
- Noun: autos / 0.001
- Noun: cortege / 0.000001
- \[ \ldots \]

The probabilities sum to 1:

- \[ \text{Det: the} / 0.4 \]
- \[ \text{Det: a} / 0.6 \]
- \[ \text{Noun: Bill} / 0.002 \]
- \[ \text{Adj: cool} / 0.003 \]
- \[ \text{Adj: directed} / 0.0005 \]
- \[ \text{Adj: cortege} / 0.000001 \]
- \[ \ldots \]
Compose

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

word seq

the cool directed autos
Compose with the cool directed autos

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

Composition diagram with:
- **Det**: the 0.32
- **Adj**: cool 0.0009
- **Adj**: directed 0.00020
- **Noun**: autos
- **Verb**
- **Prep**
- **Stop**

Question: why did this loop go away?
The best path:

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

\[ \text{the} \quad \text{cool} \quad \text{directed} \quad \text{autos} \]

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

So all paths here must have 4 words on output side

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** = \( 0.32 \times 0.0009 \cdots \)

the cool directed autos
Actually, Trellis Isn’t Complete

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

Lattice is missing some other arcs; why?

The best path:

\textbf{Start} Det Adj Adj Noun \textbf{Stop} = 0.32 \times 0.0009 ...
Actually, Trellis Isn’t Complete

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

Lattice is missing some states; 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
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”:

$$\text{Bill directed a cortege of autos through the dunes}$$

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!

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

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

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

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

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

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

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
- ...

Diagram:

1. Initial annotation
2. Fixup 1
3. Fixup 2
4. Fixup 3

Input: Initial annotation

Output: Final output
Transformation-Based Tagging (Brill 1995)

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 VBP</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 NN</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 @→ VB // TO _
- VBP @→ 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>
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