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
  - 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

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
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

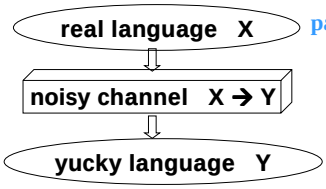
PN	Adj	Det	Noun	Prep	Noun	Prep	Det	Noun
Verb	Verb	Noun	Verb					
			Adj					
			Prep					
			...?					

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 ...

Three Finite-State Approaches

- Noisy Channel Model (statistical)



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

insert terminals

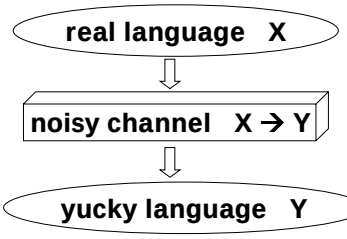
text

want to recover X from Y

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

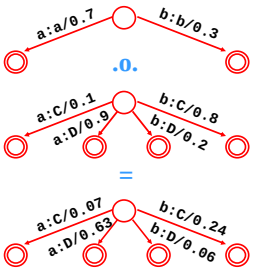
Review: Noisy Channel



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

want to recover x∈X from y∈Y
choose x that maximizes p(x | y) or equivalently p(x,y)

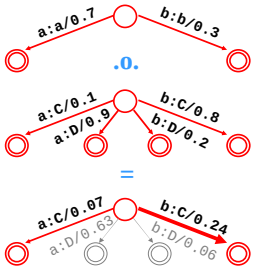
Review: Noisy Channel



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

Note p(x,y) sums to 1.
Suppose y="C"; what is best "x"?

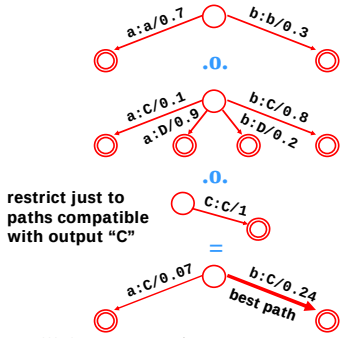
Review: Noisy Channel



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

Suppose y="C"; what is best "x"?

Review: Noisy Channel



restrict just to paths compatible with output "C"

p(X) * p(Y | X) * p(y | Y) = p(X, y)

Noisy Channel for Tagging

automaton: p(tag sequence) p(X)

"Markov Model" .o. *

transducer: tags → words p(Y | X)

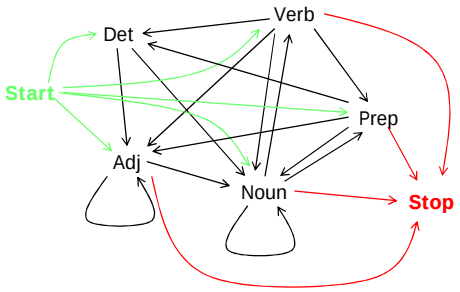
"Unigram Replacement" .o. *

automaton: the observed words p(y | Y)

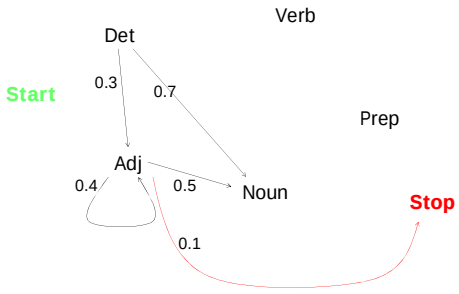
"straight line" = =

transducer: scores candidate tag seqs on their joint probability with obs words; pick best path p(X, y)

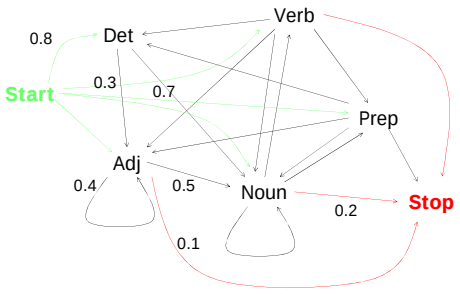
Markov Model (bigrams)



Markov Model

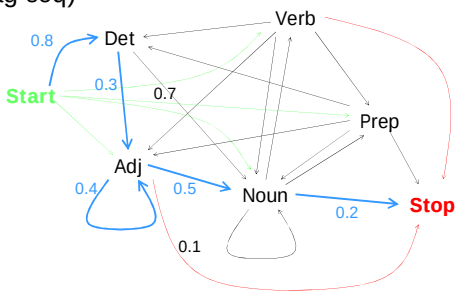


Markov Model



Markov Model

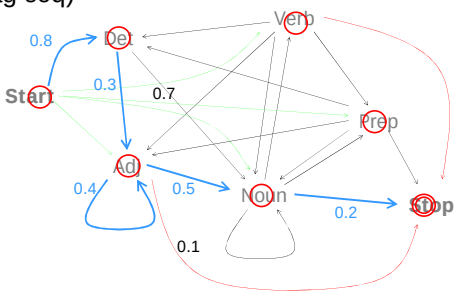
p(tag seq)



Start Det Adj Adj Noun **Stop** = 0.8 * 0.3 * 0.4 * 0.5 * 0.2

Markov Model as an FSA

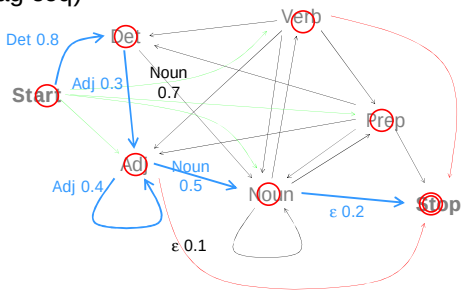
p(tag seq)



Start Det Adj Adj Noun **Stop** = 0.8 * 0.3 * 0.4 * 0.5 * 0.2

Markov Model as an FSA

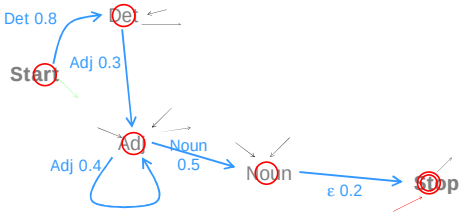
p(tag seq)



Start Det Adj Adj Noun **Stop** = 0.8 * 0.3 * 0.4 * 0.5 * 0.2

Markov Model (tag bigrams)

p(tag seq)



Start Det Adj Adj Noun **Stop** = 0.8 * 0.3 * 0.4 * 0.5 * 0.2

Noisy Channel for Tagging

automaton: p(tag sequence) — p(X)

"Markov Model"

.o. *

transducer: tags → words — p(Y | X)

"Unigram Replacement"

.o. *

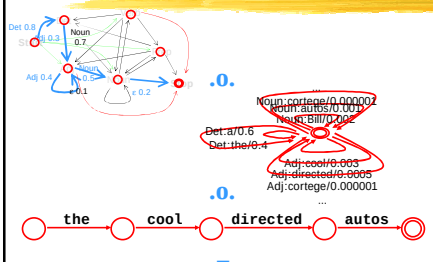
automaton: the observed words — p(y | Y)

"straight line"

= =

transducer: scores candidate tag seqs on their joint probability with obs words; pick best path — p(X, y)

Noisy Channel for Tagging

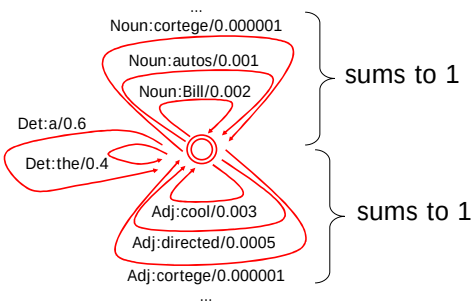


p(X) * p(Y | X) * p(y | Y) = p(X, y)

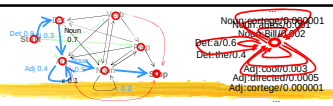
transducer: scores candidate tag seqs on their joint probability with obs words; we should pick best path

Unigram Replacement Model

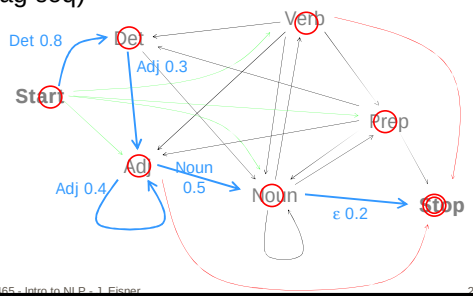
p(word seq | tag seq)



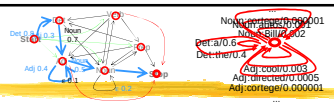
Compose



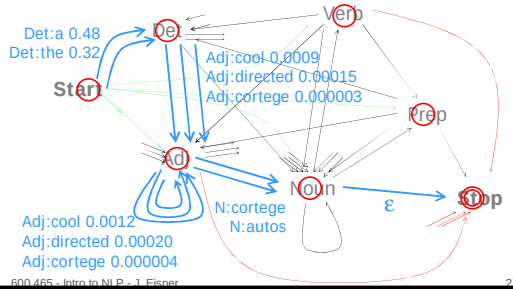
p(tag seq)



Compose

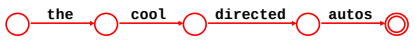


p(word seq, tag seq) = p(tag seq) * p(word seq | tag seq)

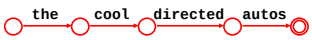


Observed Words as Straight-Line FSA

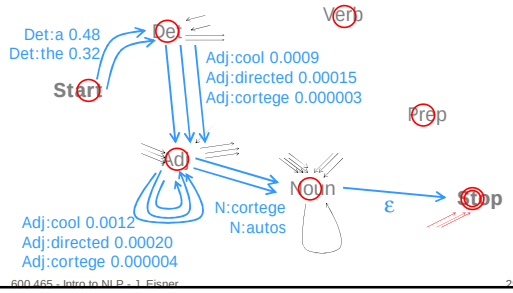
word seq



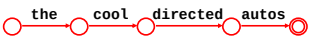
Compose with



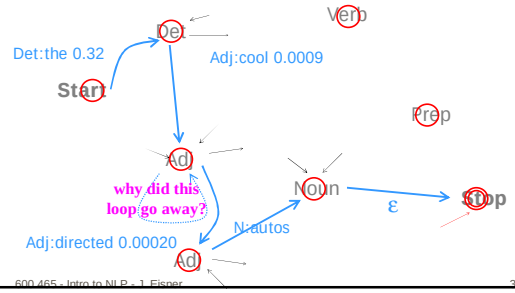
p(word seq, tag seq) = p(tag seq) * p(word seq | tag seq)



Compose with

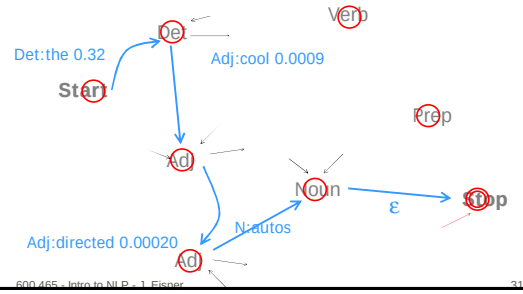


p(word seq, tag seq) = p(tag seq) * p(word seq | tag seq)



The best path:
Start Det Adj Adj Noun **Stop** = 0.32 * 0.0009 ...
the cool directed autos

p(word seq, tag seq) = p(tag seq) * p(word seq | tag seq)

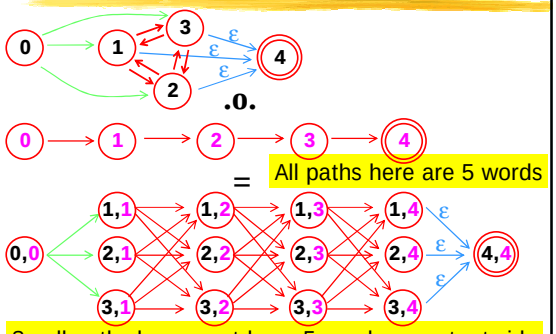


In Fact, Paths Form a "Trellis"

p(word seq, tag seq)

The best path:
Start Det Adj Adj Noun **Stop** = 0.32 * 0.0009 ...
the cool directed autos

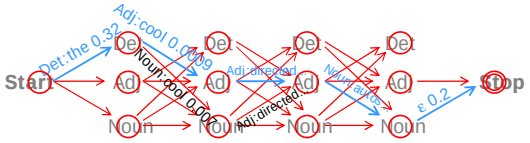
The Trellis Shape Emerges from the Cross-Product Construction



Actually, Trellis Isn't Complete

p(word seq, tag seq)

Trellis has no Det → Det or Det →Stop arcs; why?

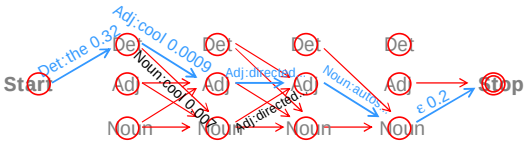


The best path:
Start Det Adj Adj Noun **Stop** = 0.32 * 0.0009 ...
the cool directed autos

Actually, Trellis Isn't Complete

p(word seq, tag seq)

Lattice is missing some other arcs; why?

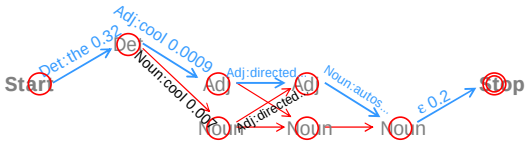


The best path:
Start Det Adj Adj Noun **Stop** = 0.32 * 0.0009 ...
the cool directed autos

Actually, Trellis Isn't Complete

p(word seq, 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(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":

probs from tag bigram model: 0.4, 0.6

Start PN Verb Det Noun Prep Noun Pr

probs from unigram replacement: 0.001

Bill directed a cortege of autos thr

- Find X that maximizes probability product

Another Viewpoint

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

p(Start PN Verb Det ... / Bill directed a ...)

= p(Start) * p(PN | Start) * p(Verb | Start PN) * p(Det | Start PN Verb) * ...
* p(Bill | Start PN Verb ...) * p(directed | Bill, Start PN Verb Det ...)
* p(a | Bill directed, Start PN Verb Det ...) * ...

Another Viewpoint

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

p(Start PN Verb Det ... / Bill directed a ...)

= p(Start) * p(PN | Start) * p(Verb | ~~Start~~ PN) * p(Det | ~~Start~~ ~~PN~~ Verb) * ...
* p(Bill | ~~Start~~ PN ~~Verb~~ ...) * p(directed | ~~Bill~~, ~~Start~~ PN Verb ~~Det~~ ...)
* p(a | ~~Bill directed,~~ ~~Start~~ PN ~~Verb~~ Det ...) * ...

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**
- **…**

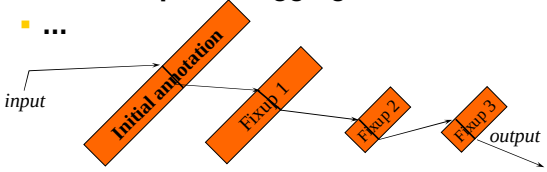


figure from Brill's thesis

Transformation-Based Tagging (Brill 1995)

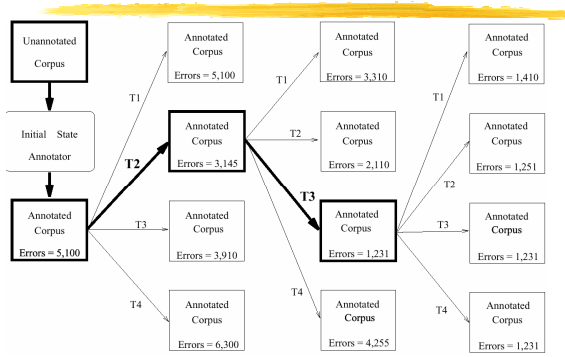


figure from Brill's thesis

Transformations Learned

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

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."

figure from Brill's thesis

Initial Tagging of OOV Words

#	Change Tag From	Change Tag To	Condition
1	NN	NNS	Has suffix **-s**
2	NN	CD	Has character **.**
3	NN	JJ	Has character **-**
4	NN	VBN	Has suffix **-ed**
5	NN	VBG	Has suffix **-ing**
6	??	RB	Has suffix **-ly**
7	??	JJ	Adding suffix **-ly** results in a word.
8	NN	CD	The word **8** can appear to the left.
9	NN	JJ	Has suffix **-al**
10	NN	VB	The word **would** can appear to the left.
11	NN	CD	Has character **0**
12	NN	JJ	The word **be** can appear to the left.
13	NNS	JJ	Has suffix **-us**
14	NNS	VBZ	The word **it** can appear to the left.
15	NN	JJ	Has suffix **-ble**
16	NN	JJ	Has suffix **-ic**
17	NN	CD	Has character **1**
18	NNS	NN	Has suffix **-ss**
19	??	JJ	Deleting the prefix **un-** results in a word
20	NN	JJ	Has suffix **-ive**

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